เริ่มเรียน เขียนโปรแกรม

ฉบับวาจาจาวา

```
import jlab.graphics.*;

public class Tree {
  public static void main(String[] args) {
    int width = 140, height = 300;
    DWindow w = new DWindow(width, height);
    drawTree(w, width / 2, height, height * 2 / 3, 90, 4);
  }
  public static void drawTree(DWindow w, double x0, double y0,
                              double d, double a, int depth) {
    double x1 = x0 + d * cos(a), y1 = y0 - d * sin(a);
    w.add(new DLine(x0, y0, x1, y1));
    if (depth <= 0) return;
    drawTree(w, x1, y1, 0.3 * d, a, depth - 1);
    double dx = d * 0.25;
    for (double d1 = dx; d1 < d; d1 += dx) {
      x1 = x0 + d1 * cos(a);
      y1 = y0 - d1 * sin(a);
      drawTree(w, x1, y1, 0.3 * d, a + 35, depth - 1);
    }
    for (double d1 = 1.5*dx; d1 < d; d1 += dx) {
      x1 = x0 + d1 * cos(a);
      y1 = y0 - d1 * sin(a);
      drawTree(w, x1, y1, 0.3 * d, a - 35, depth - 1);
    }
  }
  public static double sin(double a) {
    return Math.sin(Math.toRadians(a));
  }
  public static double cos(double a) {
    return Math.cos(Math.toRadians(a));
  }
}
```

สมชาย ประสิทธิ์จูตระกูล

v3.0.1

คำนำ

ฉบับ e-book

หนังสืออิเล็กทรอนิกส์ได้รับความนิยมเพิ่มมากขึ้นในปัจจุบัน เนื่องด้วยความสะดวกด้านการพกพา การค้น การเข้าถึงเนื้อหาที่มีความสัมพันธ์กันภายในเล่ม การเชื่อมโยงเนื้อหาภายในเล่มกับเนื้อหาเสริมอื่น ๆ นอกเล่ม และอื่น ๆ จึงเป็นที่มาของการจัดเตรียมหนังสือ “เริ่มเรียนเขียนโปรแกรม ฉบับวาจาจาวา” ให้เป็นหนังสืออิเล็กทรอนิกส์ ที่เพิ่มตัวเชื่อมโยงเนื้อหาภายในเล่ม และตัวเชื่อมไปยังภาพยนตร์บรรยายของผู้เขียนเองที่ได้จัดเก็บใน YouTube โดยจะมีรูป  กำกับข้าง ๆ เนื้อหาที่ผู้อ่านสามารถคลิกเพื่อชมภาพยนตร์ (ผู้อ่านจำเป็นต้องต่อกับอินเทอร์เน็ตเพื่อชมภาพยนตร์ดังกล่าว) จึงต้องขออนุญาตไม่มีแผ่นซีดีรอมแนบให้กับหนังสือ เหมือนในกรณีของหนังสือกระดาษ สำหรับซอฟต์แวร์ JLab ที่ใช้ประกอบการเขียนโปรแกรม ผู้อ่านสามารถดาวน์โหลดและอ่านวิธีติดตั้งได้จาก http://www.cp.eng.chula.ac.th/~somchai/JLab

สมชาย ประสิทธิ์จูตระกูล
ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย
somchaip@chula.ac.th
๕ กรกฎาคม ๒๕๕๕

คำนำ

คอมพิวเตอร์เป็นอุปกรณ์อิเล็กทรอนิกส์ที่แปลกไม่เหมือนใคร คอมพิวเตอร์ทำงานรวดเร็ว จำข้อมูลได้แม่นและจำได้นาน ไม่บ่นถึงแม้ต้องทำแต่เรื่องซ้ำๆ ซากๆ และที่เด่นมากคือ ซื่อสัตย์ สั่งให้ทำอะไรก็ทำตามนั้น คิดและนัท พี่น้องสองคนมีคอมพิวเตอร์ไว้ใช้ที่บ้านหนึ่งเครื่อง คิดใช้คอมพิวเตอร์เป็นเครื่องเล่นเกมเพื่อความบันเทิง ในขณะที่นัทใช้คอมพิวเตอร์ตัวเดียวกันนี้แทนเครื่องพิมพ์ดีดเพื่อพิมพ์รายงานส่งคุณครู การที่เราสามารถเปลี่ยนพฤติกรรมของคอมพิวเตอร์ให้เป็นเครื่องเล่นเกมสำหรับการใช้งานหนึ่ง และเปลี่ยนเป็นเครื่องพิมพ์ดีดสำหรับอีกงานหนึ่งได้นี่เองเป็นเรื่องมหัศจรรย์นัก คอมพิวเตอร์จะมีพฤติกรรมการทำงานตามคำสั่งที่ทำ ถ้าผู้ใช้เปลี่ยนชุดคำสั่งของเครื่องได้ เครื่องก็เปลี่ยนพฤติกรรมตามขั้นตอนการทำงานของชุดคำสั่งที่ได้รับ ชุดคำสั่งที่กำหนดพฤติกรรมการทำงานของคอมพิวเตอร์นี่เองเรียกว่า "*โปรแกรมคอมพิวเตอร์*"

โปรแกรมคอมพิวเตอร์ (หรือเรียกสั้น ๆ ว่า *โปรแกรม*) ไม่มีตัวตนจับต้องไม่ได้ แต่เราสร้างได้ด้วย*การเขียนโปรแกรม* (*programming*) † การเขียนโปรแกรมคอมพิวเตอร์มีกระบวนการคล้ายกับการเขียนหนังสือ หนังสือเล่มหนึ่งกว่าจะเขียนเรียบร้อยพร้อมส่งโรงพิมพ์ ต้องเริ่มด้วยการกำหนดวัตถุประสงค์ของหนังสือ คิดหาแนวทางการนำเสนอ เขียนโครงร่าง ลงมือเขียนในรายละเอียดให้ตรงตามขอบเขตและโครงร่างที่กำหนดไว้ เขียนเสร็จก็ให้เพื่อนอ่าน ให้บรรณาธิการอ่าน เพื่อตรวจสอบหลักไวยากรณ์ การใช้ภาษา และเนื้อหา นำข้อแนะนำคำติติงกลับมาแก้ไข ทำเป็นวงวนจนแน่ใจว่า ผลที่ได้คือหนังสือที่ตั้งใจไว้ตอนต้น แล้วก็ส่งโรงพิมพ์ หรือทำเป็นหนังสืออิเล็กทรอนิกส์เพื่อเผยแพร่ต่อไป

สำหรับการเขียนโปรแกรม เริ่มด้วยการเขียนข้อกำหนดว่า ต้องการให้ทำอะไร ออกแบบขั้นตอนวิธีการทำงาน ลงมือเขียนรหัสคำสั่งเพื่อบรรยายขั้นตอนที่ออกแบบไว้ จากนั้นใช้ตัวแปลโปรแกรมตรวจสอบหลักไวยากรณ์ของโปรแกรม ถ้าผิดก็แก้ไข ถูกต้องก็สร้างเป็นรหัสเครื่อง เพื่อสั่งทำงานและทดสอบว่า โปรแกรมทำงานตามข้อกำหนดหรือไม่ ถ้าไม่ ก็เข้าสู่วงวนของการแก้ไข แปล ทดสอบ จนกว่าจะสมบูรณ์ จึงเข้าสู่กระบวนการเผยแพร่โปรแกรมที่ได้พัฒนาขึ้น

---

† บางคนเรียกว่า การสร้างโปรแกรม, การทำโปรแกรม, หรือการโปรแกรม

อย่างไรก็ตามการเขียนโปรแกรมก็มีหลายประเด็นที่ต่างจากการเขียนหนังสือ เราเขียนหนังสือให้คนอ่าน ในขณะที่โปรแกรมถูกเขียนขึ้นให้คนอ่านเพื่อปรับปรุงแก้ไข และให้คอมพิวเตอร์อ่านไปทำงานตามคำสั่งที่เขียน แต่เนื่องจากความไม่ค่อยฉลาดของคอมพิวเตอร์ ไวยากรณ์ของ*ภาษาโปรแกรม* (*programming language*) จึงมีกฎเกณฑ์จู้จี้จุกจิก เขียนผิดนิดผิดหน่อยก็ไม่ได้ ต้องเขียนถูกหลักภาษา 100% อีกทั้งคำสั่งต่าง ๆ ก็เป็นคำสั่งพื้นฐานมาก ๆ ที่เกี่ยวกับการคำนวณ การตรวจสอบ การเลือกทำ และการทำงานซ้ำ ๆ ที่คอมพิวเตอร์ (ฮาร์ดแวร์) ถนัด จึงเป็นหน้าที่ของ*นักเขียนโปรแกรม* (*programmer*) ที่ต้องนำคำสั่งพื้นฐานเหล่านี้มาประกอบกันเป็นชุดคำสั่งตามกฎเกณฑ์ของภาษาเพื่อให้ทำงานได้ตามข้อกำหนด

จำนวนคำสั่งที่เขียนประกอบกันเป็นโปรแกรมหนึ่ง ๆ จะมีจำนวนมากหรือน้อย ขึ้นกับความซับซ้อนของข้อกำหนดที่ต้องการ โปรแกรมเล็กอาจมีจำนวนบรรทัดคำสั่งไม่ถึง 10 บรรทัด เพื่อแสดงข้อความทักทายผู้ใช้งานทางจอภาพ หรือโปรแกรมใหญ่อาจมีเป็นล้านบรรทัดเพื่อรองรับงานที่ซับซ้อน เช่น ชุดไมโครซอฟต์ออฟฟิศรุ่น 2003 มีจำนวนบรรทัดคำสั่งประมาณ 25 ล้านบรรทัด ซึ่งแน่นอนว่า คงไม่ได้เขียนด้วยนักเขียนโปรแกรมหนึ่งคนเป็นแน่ การพัฒนาโปรแกรมขนาดใหญ่เช่นนี้อาศัยหลักการทางวิศวกรรมซอฟต์แวร์ที่มีเนื้อหามากเกินกว่าการเริ่มเรียนเขียนโปรแกรม

หนังสือเล่มนี้นำเสนอหลักการเขียนโปรแกรมเบื้องต้นสำหรับผู้เริ่มเรียนเขียนโปรแกรม โดยเน้นการนำเสนอแนวคิดของคำสั่งที่ใช้งานบ่อย ไม่ครอบคลุมไวยากรณ์และคำสั่งทั้งหมดของภาษา เพราะมีเรื่องจุกจิกมากมาย เรื่องใดใช้น้อย ขอไม่นำเสนอ หรือไม่ก็ย้ายไปไว้ที่หัวข้อ “เพิ่มเติม” ท้ายบท ตามด้วยตัวอย่างโปรแกรมหลากหลายประกอบการนำเสนอจำนวนมาก เช่น โปรแกรมประมวลผลจำนวน ประมวลผลภาพนิ่งและภาพเคลื่อนไหว ประมวลผลข้อความทั้งที่ได้จากแฟ้มและจากอินเทอร์เน็ต เป็นต้น เพื่อให้ผู้อ่านเห็นความหลากหลายในการแปลงข้อกำหนดที่ต้องการให้คอมพิวเตอร์ทำ มาเป็นโปรแกรม ผู้อ่านต้องตระหนักว่า จะเริ่มเรียนเขียนโปรแกรมให้ได้ผล ต้องหมั่นลงมือปฏิบัติจริง ฝึกอ่าน ฝึกคิดตาม ฝึกคิดเอง ฝึกเขียน (ฝึกพิมพ์) ฝึกทดสอบ และฝึกแก้ปัญหา เสมือนการเริ่มเรียนภาษาต่างประเทศที่ต้องฝึกอ่าน ฝึกฟัง ฝึกพูดในสถานการณ์จริง ย่อมได้ผลมากกว่าการเริ่มเรียนแต่ไวยากรณ์ของภาษาต่างประเทศนั้น จึงขอเน้นอีกครั้งหนึ่งว่าการเขียนโปรแกรมเป็นทักษะที่ต้องฝึกฝนลงมือปฏิบัติจริง มากกว่าการอ่านและจำ

ผู้เขียนเลือกใช้ “จาวา” (Java™) เป็นภาษาโปรแกรมในการนำเสนอ ด้วยความทันสมัย และความนิยมทั้งจากภาคการศึกษาและภาคอุตสาหกรรม อีกทั้งเป็นภาษาที่ทำงานอยู่ในสภาพแวดล้อมที่มีคลังคำสั่งให้เรียกใช้ได้มากมาย ทั้งที่เป็นคลังคำสั่งมาตรฐานที่มากับระบบ กับที่เป็นคลังคำสั่งที่มีผู้พัฒนาและเผยแพร่ให้ใช้ได้โดยไม่คิดค่าใช้จ่าย หลักการเขียนโปรแกรมที่เรียกใช้คลังคำสั่งอื่นให้เป็นประโยชน์เป็นแนวทางที่ควรปฏิบัติอย่างยิ่ง เป็นการต่อยอดความสามารถจากของที่มีอยู่ เพื่อนำไปประยุกต์กับงานใหม่ ที่ผู้อ่านจะได้ศึกษากันในตัวอย่าง ดังนั้น ทักษะที่จำเป็นต้องฝึกฝนอีกประการหนึ่ง คือ การอ่านคู่มือการใช้งานของคลังคำสั่งที่สนใจ เพื่อให้ใช้งานได้ตรงตาม

ข้อกำหนดที่วางไว้ จึงขอเน้นว่า หนังสือเล่มนี้นำเสนอหลักการเขียนโปรแกรมโดยใช้จาวาเป็นภาษาในการเขียนโปรแกรมให้ทำงานได้จริง แต่ไม่ใช่หนังสือที่นำเสนอภาษาจาวาในรายละเอียด

ผู้เขียนเริ่มเรียนเขียนโปรแกรมตั้งแต่ปี พ.ศ. 2522 และได้เรียนรู้ภาษาโปรแกรมเพิ่มเติมตลอดมา เนื่องจากภาษาโปรแกรมมีวิวัฒนาการ และมีภาษาใหม่ ๆ เกิดขึ้นอย่างต่อเนื่อง ผู้เขียนเริ่มใช้ภาษาจาวาในการสอนวิชาการเขียนโปรแกรมเบื้องต้นตั้งแต่ปี พ.ศ. 2545 โดยใช้ซอฟต์แวร์ JLab ที่ผู้เขียนได้พัฒนาขึ้นเพื่อเป็นเครื่องมือในห้องปฏิบัติการเสริมทักษะให้ผู้เรียนได้ฝึกเขียนโปรแกรมจริง และได้พัฒนาคลังคำสั่งเสริมจำนวนหนึ่งที่แนบมากับ JLab ที่ช่วยให้เขียนโปรแกรมตัวอย่างที่น่าสนใจได้สะดวกขึ้น

ผู้เขียนขอขอบคุณโครงการสนับสนุนการเขียนตำรา/หนังสือ ของคณาจารย์คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ที่ให้การสนับสนุน (หนังสือ “เริ่มเรียนเขียนโปรแกรม : ฉบับวาจาจาวา” เล่มนี้ เป็นหนังสือเล่มที่ 13 ในโครงการสนับสนุนการเขียนตำรา/หนังสือของคณาจารย์คณะวิศวกรรมศาสตร์) ขอขอบคุณภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย ที่สนับสนุนวัสดุ ครุภัณฑ์ และโอกาส และขอขอบคุณสำนักพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย ที่รับจัดพิมพ์และเผยแพร่ได้อย่างราบรื่น สำหรับผู้สนใจเอกสารอื่นเพิ่มเติม (เช่น แผ่นใส, โปรแกรม, รายการแก้ไขข้อผิดพลาด, เนื้อหาเพิ่มเติมอื่น ๆ เป็นต้น) สามารถหาได้ที่

http://www.cp.eng.chula.ac.th/~somchai/books

สมชาย ประสิทธิ์จูตระกูล
ภาควิชาวิศวกรรมคอมพิวเตอร์
จุฬาลงกรณ์มหาวิทยาลัย
somchaip@chula.ac.th
๕ ธันวาคม ๒๕๕๑

หนังสือเล่มนี้ใช้แบบอักษร Browallia New, Cordia New, Courier New, Segoe UI, Tahoma, Times New Roman วรเสรษฐ (http://worasait.com) เลย์อิจิมหานิยม (http://www.f0nt.com) และ TF-Intanon (http://www.thaiprintingfederation.com/webfont) ผู้เขียนขอขอบคุณผู้ออกแบบ ณ ที่นี้ด้วย

สารบัญ

พร้อมแล้วเริ่ม

เมื่อพร้อมเรียนเขียนโปรแกรม ก็เริ่มกันเลย บทนี้เริ่มด้วยการเร่งให้ผู้อ่านติดตั้งซอฟต์แวร์ช่วยพัฒนาโปรแกรมตั้งแต่ตอนนี้ แล้วเริ่มพิมพ์โปรแกรมเข้าเครื่อง พร้อมสั่งทำงานเลย อย่ากังวลว่าแต่ละบรรทัดที่พิมพ์นั้นมีความหมายอย่างไร จะเรียนเขียนโปรแกรม ก็ต้องฝึกพิมพ์ฝึกเขียนในทางปฏิบัติจริงเพื่อให้คุ้นเคย จากนั้นจึงทำความเข้าใจคร่าว ๆ เกี่ยวกับองค์ประกอบของโปรแกรมที่ประกอบด้วยคลาส คลาสประกอบด้วยเมท็อด และเมท็อดประกอบด้วยคำสั่ง ในช่วงแรกนี้เราจะเขียนแบบง่ายสุด ๆ คือหนึ่งโปรแกรมมีหนึ่งคลาส หนึ่งคลาสมีหนึ่งเมท็อด (แต่ภายในเมท็อดนี้ควรมีหลายคำสั่ง) เขียนโปรแกรมแล้วก็สั่งทำงานทันที ถ้าเขียนผิด ก็ต้องฝึกทักษะตีความข้อผิดพลาด และแก้ไขข้อผิดพลาดนั้น นอกจากนี้บทนี้ยังนำเสนอธรรมเนียมในการเขียนโปรแกรมที่ควรปฏิบัติตามเพื่อให้ได้โปรแกรมที่ทั้งสวยและอ่านง่าย

ซอฟต์แวร์ช่วยพัฒนาโปรแกรม

การลงมือฝึกและปฏิบัติจริงเท่านั้นจึงจะนำไปสู่ความสำเร็จในการเขียนโปรแกรมคอมพิวเตอร์ให้ได้ผล แต่เพื่อให้มือใหม่เขียนโปรแกรมได้สะดวก จำต้องอาศัยเครื่องมือช่วย เพื่อให้เส้นทางการฝึกปฏิบัติเป็นไปอย่างราบรื่นไม่ติดขัด ในปัจจุบันมีซอฟต์แวร์ช่วยเราเขียนโปรแกรมมากมาย แต่ส่วนใหญ่เป็นของนักเขียนโปรแกรมมืออาชีพ มีคุณสมบัติมากมาย มากเสียจนใช้งานลำบาก หนังสือเล่มนี้ขอใช้ซอฟต์แวร์ช่วยพัฒนาโปรแกรมที่ผู้เขียนพัฒนาขึ้นเอง ชื่อว่า JLab มีคุณสมบัติเพียงพอสำหรับมือใหม่ เมื่อเราพร้อมเริ่มเรียนเขียนโปรแกรม ขั้นตอนที่ศูนย์ก็คือ การติดตั้ง JLab ทำได้เพียงแค่เปิดเครื่อง เข้าวินโดวส์ให้เรียบร้อย ใส่แผ่นซีดีรอมที่อยู่ปกหลังหนังสือเล่มนี้เข้าเครื่อง รอสักครู่ ระบบจะแสดงหน้าจอและขั้นตอนการติดตั้ง (รายละเอียดหน้าจอการติด

ตั้งอยู่ในภาคผนวก) เพียงแค่คลิก ๆ ๆ (มักเป็นปุ่ม Next) ทำใจเย็น ๆ เพราะอาจใช้เวลาหลายนาทีในการติดตั้ง เนื่องจากระบบอาจต้องติดตั้งชุดพัฒนาซอฟต์แวร์อื่น ๆ เพิ่มเติม เมื่อติดตั้งเสร็จก็เริ่มได้เลย

โปรแกรมแรก

ขอย้ำตรงนี้ว่า ถ้ายังไม่ได้ติดตั้งซอฟต์แวร์ช่วยพัฒนาโปรแกรม อย่าอ่านต่อ การอ่านอย่างเดียวไม่ได้ทำให้เขียนโปรแกรมเป็น ต้องอ่านและลงมือปฏิบัติตามด้วยจึงจะได้ผล

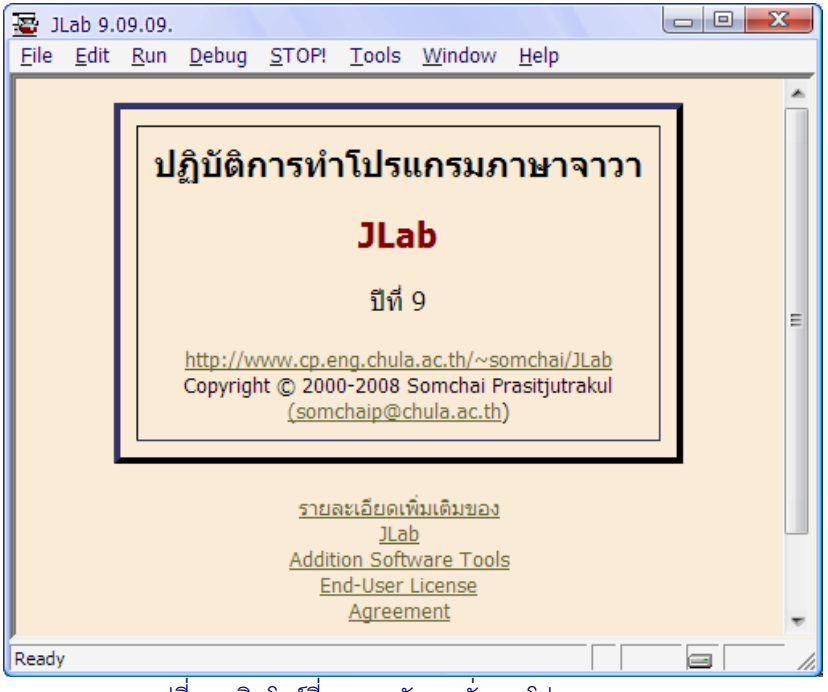
เนื่องจากคอมพิวเตอร์มีหน้าที่ประมวลผลข้อมูลเพื่อให้ได้ผลลัพธ์ ผลที่ได้นี้อาจถูกนำไปเก็บในฐานข้อมูล ส่งไปทางเครือข่าย หรือแสดงออกให้ผู้ใช้รับรู้ในรูปของเสียง สี ภาพนิ่ง ภาพเคลื่อนไหว ข้อความ หรือสัญญาณไฟฟ้าอื่น ๆ ขอเริ่มด้วยแบบง่ายที่สุดคือ การนำผลที่ได้แสดงออกทางจอภาพในรูปของข้อความ รหัสที่ 1-1 แสดงรหัสของโปรแกรมที่เมื่อสั่งให้ทำงานจะแสดงข้อความ `Hello World` ปรากฏทางจอภาพ

```
01 public class Hello {
02   public static void main(String[] args) {
03     System.out.println("Hello World");
04   }
05 }
```

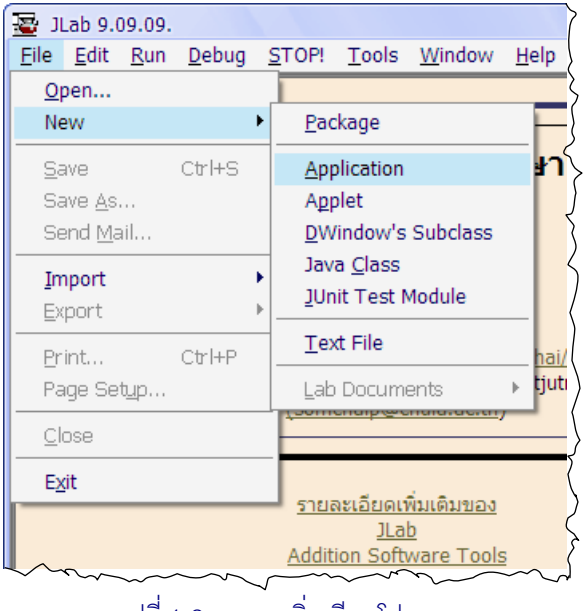
กดปุ่มนี้ เพื่อชมภาพยนตร์สาธิตการเขียนโปรแกรม

รหัสที่ 1-1 โปรแกรมแสดงข้อความ `Hello World` ทางจอภาพ

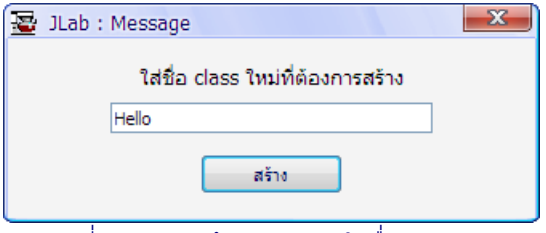
ก่อนจะเข้าใจความหมายของคำสั่งในแต่ละบรรทัดของรหัสที่ 1-1 เรามาเขียนโปรแกรมนี้และลองสั่งทำงานกันจริง ๆ เริ่มจากการเรียกโปรแกรม JLab ด้วยการคลิกสองครั้งที่สัญรูป (icon)  จะปรากฏวินโดว์ดังรูปที่ 1-1 จากนั้นเลือกเมนู File → New → Application ดังรูปที่ 1-2 เพื่อให้ระบบเตรียมเนื้อที่สำหรับเขียนโปรแกรมใหม่ โดยระบบจะแสดงกล่องโต้ตอบดังรูปที่ 1-3 ให้ผู้ใช้กรอกชื่อโปรแกรม (ในรูป ผู้ใช้กรอกคำว่า Hello) แล้วกดปุ่มสร้าง ระบบจะจัดเตรียมเนื้อที่พร้อมทั้งเติมคำสั่งที่มักต้องป้อนจำนวนหนึ่งให้อัตโนมัติดังรูปที่ 1-4 เริ่มพิมพ์คำสั่งต่าง ๆ ที่แสดงในรหัสที่ 1-1 ให้สมบูรณ์ดังรูปที่ 1-5 (ขอเน้นว่าต้องพิมพ์ให้เหมือน อย่าให้พลาดโดยเฉพาะตัวอักษรตัวใหญ่ ตัวเล็ก) เมื่อมั่นใจว่าไม่มีอะไรผิดพลาด ก็เลือกเมนู Run → Run Class "Hello" (หรือจะกดปุ่ม F5 ก็ได้) เป็นการสั่งให้โปรแกรมทำงาน หากพิมพ์ตัวโปรแกรมถูกต้องจริง ๆ จะได้ผลดังแสดงในรูปที่ 1-7 ผลที่ได้นั้นแสดงในส่วนล่างของวินโดว์มีหลายบรรทัด แต่จะมีอยู่บรรทัดหนึ่งที่มีข้อความว่า `Hello World` ซึ่งคือผลของการสั่งโปรแกรมทำงานตามที่ต้องการ



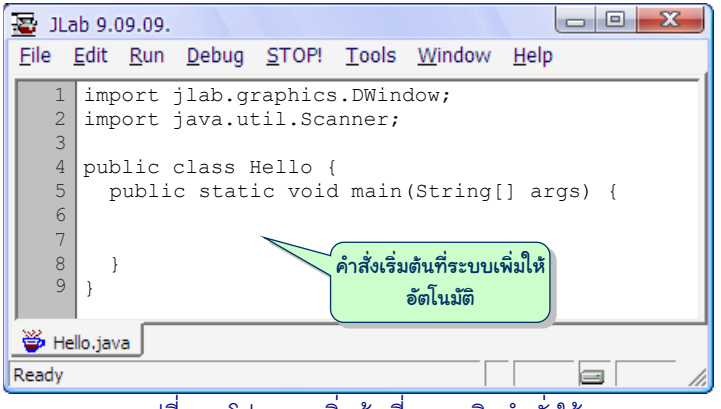
รูปที่ 1-1 วินโดว์ที่แสดงหลังการสั่งงานโปรแกรม JLab



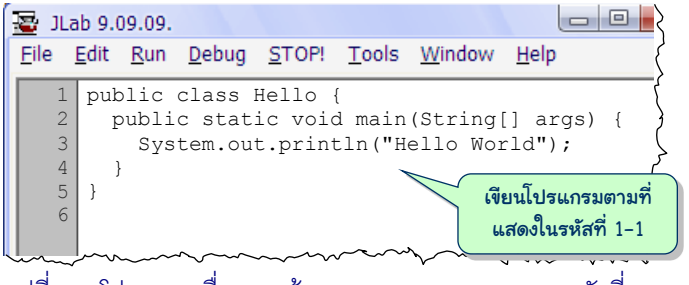
รูปที่ 1-2 เมนูการเริ่มเขียนโปรแกรม



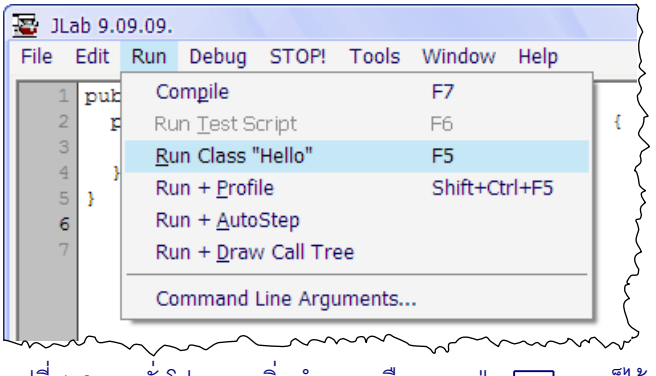
รูปที่ 1-3 กล่องโต้ตอบสำหรับใส่ชื่อโปรแกรม



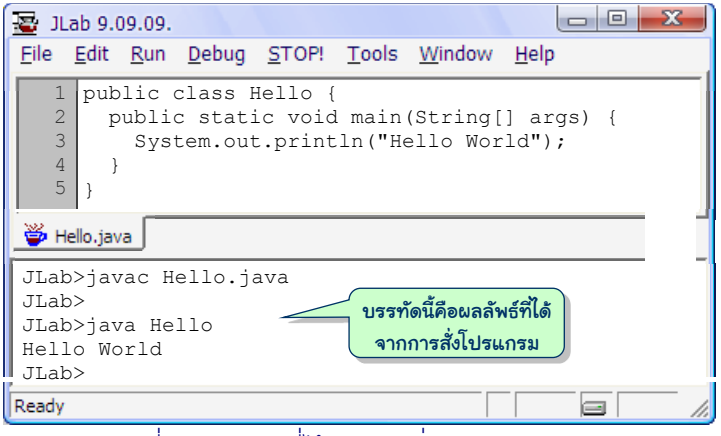
รูปที่ 1-4 โปรแกรมเริ่มต้นที่ระบบเติมคำสั่งให้



รูปที่ 1-5 โปรแกรมเพื่อแสดงข้อความ Hello World ตามรหัสที่ 1-1



รูปที่ 1-6 เมนูสั่งโปรแกรมเริ่มทำงาน หรือจะกดปุ่ม F5 แทนก็ได้



รูปที่ 1-7 ผลลัพธ์ที่ได้จากการสั่งทำงานโปรแกรม

ผู้อ่านอาจใช้โอกาสนี้ทดลองเปลี่ยนข้อความของบรรทัดที่ 3 ในรหัสที่ 1-1 ที่อยู่ภายในเครื่องหมายอัญประกาศคู่ " จาก `Hello World` เป็นข้อความอื่น แล้วกดปุ่ม F5 เพื่อสร้างความคุ้นเคยในการแก้ไขโปรแกรมและสั่งทำงาน คำสั่ง `System.out.println` [1] ในบรรทัดที่ 3 คือ การนำสิ่งที่อยู่ภายในวงเล็บแสดงออกทางจอภาพแล้วขึ้นบรรทัดใหม่ (หากไม่ต้องการขึ้นบรรทัดใหม่หลังจากแสดง ให้ใช้คำสั่ง `System.out.print`) โดยเราเรียกข้อความที่อยู่ภายในเครื่องหมายอัญประกาศคู่นี้ว่า *สตริง* (string) ถ้าต้องการแสดงหลาย ๆ บรรทัด ก็ใช้คำสั่งนี้เพื่อแสดงคำสั่งละบรรทัด ลองแก้ไขโปรแกรมเป็นดังรหัสที่ 1-2 แล้วสั่งทำงานดูว่าจะได้ผลเช่นไร

```
01 public class Hello {
02   public static void main(String[] args) {
03     System.out.print("Hello");
04     System.out.println("World");
05     System.out.println(123);
06     System.out.println(1+2+3);
07   }
08 }
```

บรรทัดที่ 3 แสดง Hello

บรรทัดที่ 4 แสดง World ต่อจากที่ได้แสดงมา ดังนั้น ผลของคำสั่งในบรรทัด 3 และ 4 จึงเป็น HelloWorld

รหัสที่ 1-2 โปรแกรมแสดงคำว่า `HelloWorld`, `123` และ `6` แยกกันคนละบรรทัด

รหัสที่ 1-2 แสดงให้เห็นว่า นอกจากที่ `System.out.println` (และ `print`) สามารถแสดงสตริงทางจอภาพได้แล้ว ยังสามารถแสดงตัวเลข (บรรทัดที่ 5) หรือต้องการให้คำนวณก่อน (บรรทัดที่ 6) แล้วค่อยแสดงก็ได้ (เราจะได้ศึกษาวิธีการคำนวณในรูปแบบต่าง ๆ ในบทถัดไป)

อนึ่งผู้อ่านสามารถใช้เมนู File → Save As... เพื่อสั่งบันทึกโปรแกรมที่เขียน โดยจะบันทึกลงแฟ้มที่มีประเภทเป็น .jlab ซึ่งสามารถใช้เมนู File → Open เปิดกลับมาใช้ได้ในภายหลัง

[1] คำว่า `println` อ่านว่า print line

องค์ประกอบของโปรแกรม

เราเรียกรหัสที่ 1-2 ว่า*คลาส* (class) นิยามของคลาสนั้นมีรายละเอียดที่ไม่ขอกล่าวถึงในตอนนี้ โปรแกรมหนึ่งโปรแกรมประกอบด้วยคลาสตั้งแต่หนึ่งคลาสขึ้นไป แต่ในช่วงแรกเราจะเขียนโปรแกรมเล็ก ๆ ไม่ซับซ้อน หนึ่งโปรแกรมประกอบด้วยคลาสเพียงหนึ่งคลาสเท่านั้น (ดังนั้นอาจใช้คำว่าโปรแกรมและคลาสสลับกันไปมา) รหัสที่ 1-3 แสดงโครงของคลาสหนึ่งคลาส ตัวคลาสเริ่มจากบรรทัดที่เขียนเริ่มต้นด้วยคำว่า `public class` ตามด้วยชื่อคลาส ตามด้วยการบรรยายตัวคลาสซึ่งเขียนไว้ภายในเครื่องหมาย `{` กับ `}`

```
public class ClassName {

}
```

เขียนรายละเอียดของคลาสไว้ภายในเครื่องหมาย { }

รหัสที่ 1-3 โครงของคลาส

ภายในคลาสประกอบด้วย*เมท็อด* (method) เมท็อดคือที่ที่เราเขียนคำสั่งบรรยายขั้นตอนการทำงานของโปรแกรม เมท็อดต่าง ๆ ของคลาสจึงเป็นบริการที่คลาสนั้นมีให้ผู้อื่นเรียกใช้ เราจะได้ศึกษาเรื่องเมท็อดอย่างละเอียดในภายหลัง ในช่วงแรกนี้จะเขียนเฉพาะเมท็อดพิเศษชื่อ `main` รหัสที่ 1-4 แสดงโครงของคลาสที่มีเมท็อด `main` การที่เรียกเมท็อด `main` ว่าเป็นเมท็อดพิเศษเพราะว่า หากเราสั่งให้คลาสใดเริ่มทำงาน (โดยการกดปุ่ม F5) ระบบจะเริ่มทำคำสั่งที่ปรากฏในเมท็อด `main` ของคลาสนั้น

```
public class ClassName {
  public static void main(String[] args) {

  }
}
```

หัวเมท็อด main ต้องเขียนแบบนี้

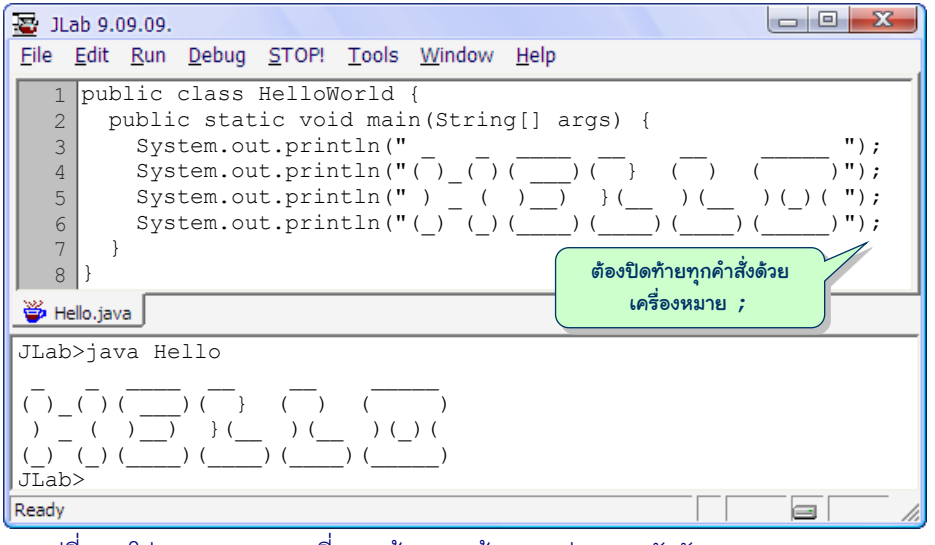
เขียนรายละเอียดของเมท็อดไว้ภายในเครื่องหมาย { } หลังหัวเมท็อด

รหัสที่ 1-4 โครงของคลาสและเมท็อดพิเศษชื่อ `main`

หนึ่งโปรแกรมประกอบด้วยหลายคลาส หนึ่งคลาสประกอบด้วยหลายเมท็อด และสุดท้ายหนึ่งเมท็อดประกอบด้วยหลายคำสั่ง คำสั่งต่าง ๆ ในเมท็อดนี่เองที่ทำให้โปรแกรมทำงานตามที่เราต้องการให้ทำ การทำงานของเมท็อดเริ่มที่คำสั่งแรกของเมท็อด หลังทำคำสั่งแรกเสร็จ ก็ทำต่อที่คำสั่งถัดไป รูปแบบการทำงานของคำสั่งต่าง ๆ ที่เขียนเป็นลำดับเป็นบรรทัด ๆ จากบนลงล่างนี้เรียกว่า *การทำงานแบบลำดับ* ซึ่งเป็นหนึ่งในวิธีการเขียนโปรแกรมเพื่อสั่งคอมพิวเตอร์ทำงาน (ยังมีวิธีอื่น ๆ อีกที่จะได้กล่าวในภายหลัง) รหัสที่ 1-2 มีคำสั่ง `System.out.println` สี่คำสั่งทำงานเป็นลำดับจากบรรทัดที่ 3 ถึง 6 เมื่อทำถึงบรรทัดที่ 7 ซึ่งเป็นเครื่องหมาย `}` ปิดตัวเมท็อด

นั่นหมายความว่า เป็นการสิ้นสุดการทำงานของเมท็อด `main` และหมายความด้วยว่า เป็นการสิ้นสุดการทำงานของโปรแกรม

รูปที่ 1-8 แสดงถึงความพยายามของการใช้ตัวอักขระต่าง ๆ เพื่อประกอบกันเป็นคำว่า Hello ที่แลดูแปลกตา [2] โดยอาศัยการแสดงส่วนต่าง ๆ ด้วย `System.out.println` ให้สังเกตว่า ทุกคำสั่งภายในเมท็อดต้องปิดท้ายด้วยเครื่องหมายอัฒภาค `;` (semicolon)



รูปที่ 1-8 โปรแกรม `Hello` ที่แสดงข้อความด้วยการประกอบตัวอักขระแบบ ascii text

ตัวแปลโปรแกรม

เราเรียกรหัสที่พิมพ์เพื่อบรรยายการทำงานของโปรแกรมว่า *รหัสต้นฉบับ* (source code) การกดปุ่ม F5 เป็นการสั่งให้ระบบแปลรหัสต้นฉบับให้เป็น*รหัสเครื่อง* (machine code) [3] ที่ระบบตีความและทำงานตามได้ แล้วทำไมเราไม่เขียนโปรแกรมด้วยรหัสเครื่องเลย จะได้ไม่ต้องเสียเวลาแปลรหัส นั่นก็เพราะว่า การเขียนโปรแกรมด้วยรหัสเครื่องนั้นยาก มีโอกาสสูงที่จะเขียนผิดต้องการให้เครื่องทำอะไรบางอย่างง่าย ๆ อาจต้องเขียนคำสั่งรหัสเครื่องเป็นจำนวนนับสิบนับร้อยคำสั่ง นอกจากเขียนยาก ยังอ่านทำความเข้าใจยากด้วย จึงเป็นที่มาของการออกแบบภาษาโปรแกรม (ซึ่งมักเรียกว่า *ภาษาระดับสูง* และมองรหัสเครื่องว่าเป็นระดับต่ำ) โดยหวังให้เราเขียนโปรแกรมเพื่อสั่งให้เครื่องทำงานตามที่ต้องการได้สะดวก ผิดพลาดน้อย อ่านเข้าใจง่าย และทำงาน

[2] ผู้อ่านที่สนใจสร้างคำด้วยการประกอบตัวอักขระในลักษณะนี้ สามารถค้นหาบริการในอินเทอร์เน็ตได้ด้วยการใช้ google ค้นคำว่า ascii text

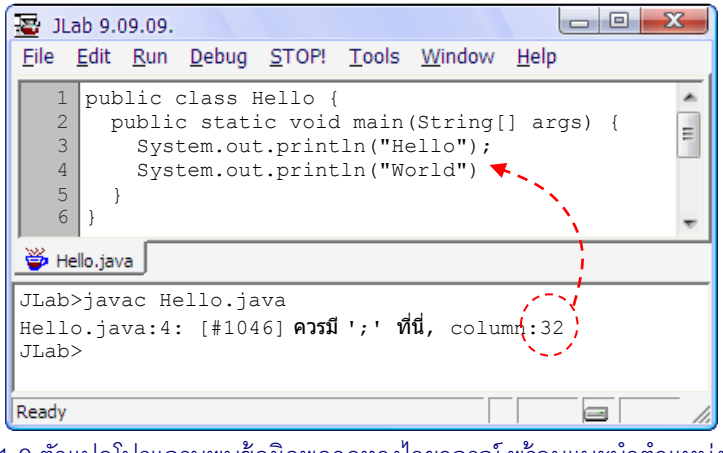
[3] จาวาเรียกรหัสเครื่องของระบบว่า*รหัสไบต์* (byte code)

อย่างมีประสิทธิภาพ ภาษาโปรแกรมทุกระดับทุกภาษามีไวยากรณ์เป็นตัวกำหนดกฎเกณฑ์ในการเขียนโปรแกรม โปรแกรมภาษาจาวาก็ต้องเขียนให้ถูกหลักไวยากรณ์ของภาษาจาวา ต้องถูกต้อง 100% ผิดนิดผิดหน่อยไม่ได้ การตรวจสอบไวยากรณ์ของโปรแกรมเป็นหน้าที่ของ*ตัวแปลโปรแกรม* (compiler) ซึ่งเป็นซอฟต์แวร์ตัวหนึ่งในชุดพัฒนาโปรแกรม ที่นอกจากจะตรวจสอบความถูกต้องทางไวยากรณ์แล้ว ยังมีหน้าที่แปลรหัสต้นฉบับของโปรแกรมให้เป็นรหัสเครื่องด้วย ทุกครั้งที่เราสั่งคลาสให้ทำงาน ระบบจะแปลโปรแกรมให้อัตโนมัติก่อนที่จะเริ่มทำงาน

ข้อผิดพลาดของโปรแกรม

ไม่ว่าจะเป็นมือใหม่หรือมืออาชีพ การเขียนโปรแกรมที่มีข้อผิดพลาดระหว่างการพัฒนาซอฟต์แวร์เป็นเรื่องปกติที่เกิดขึ้นเป็นประจำ เราแบ่งข้อผิดพลาดออกเป็นสองแบบคือข้อผิดพลาดทางไวยากรณ์ของภาษา กับข้อผิดพลาดระหว่างการทำงานของโปรแกรม ข้อผิดพลาดแบบแรกนั้นตรวจสอบได้ง่ายด้วยตัวแปลโปรแกรม ในขณะที่ข้อผิดพลาดแบบหลังนั้นตรวจสอบยากกว่า ต้องใช้ทักษะและเครื่องมืออื่นประกอบในการหาจุดบกพร่องของโปรแกรม ในหัวข้อนี้จะขอกล่าวถึงข้อผิดพลาดทางไวยากรณ์ก่อน

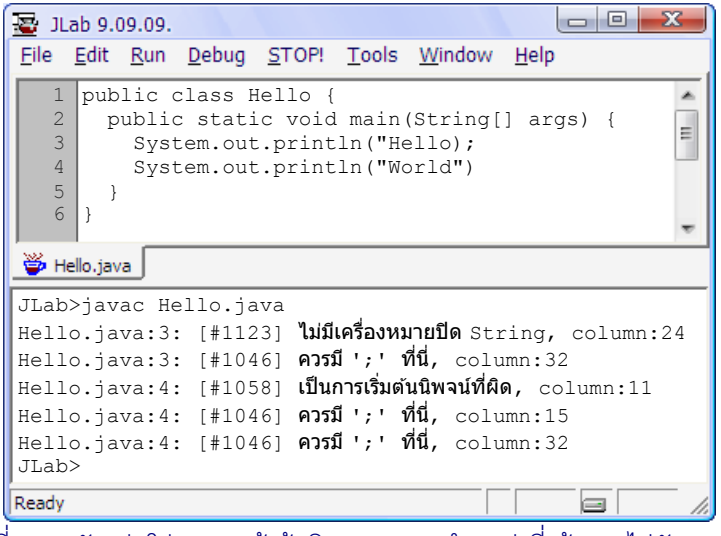
รูปที่ 1-9 แสดงตัวอย่างข้อผิดพลาดทางไวยากรณ์ที่ตัวแปลโปรแกรมพบ เมื่อแปลคลาส `Hello` จะแสดงข้อความ `Hello.java:4: [#1046] ควรมี ';' ที่นี่, column:32` ตีความได้ว่า พบข้อผิดพลาด และแนะนำว่า น่าจะมีเครื่องหมาย ; ที่ตำแหน่ง 32 ของบรรทัดที่ 4 ซึ่งก็เป็นเช่นนั้นจริง ๆ เพราะผู้เขียนโปรแกรมลืมใส่ ; ปิดคำสั่งท้ายบรรทัดที่ 4



รูปที่ 1-9 ตัวแปลโปรแกรมพบข้อผิดพลาดทางไวยากรณ์ พร้อมแนะนำตำแหน่งที่พบ

แต่ขอเตือนไว้ตรงนี้ว่า ข้อความแสดงความผิดพลาดที่ตัวแปลโปรแกรมแจ้งและแนะนำให้ทราบนั้น อาจไม่ตรงกับข้อผิดพลาดที่เกิดขึ้นจริง เพราะในบางครั้งข้อผิดพลาดหนึ่งอาจก่อให้เกิด

ข้อผิดพลาดอื่นตามมา ดังตัวอย่างในรูปที่ 1-10 มีข้อผิดพลาดอยู่สองที่คือ ไม่มีเครื่องหมาย " ปิดสตริงที่บรรทัด 3 และไม่มีเครื่องหมาย ; ที่ท้ายบรรทัด 4 แต่ตัวแปลโปรแกรมพบข้อผิดพลาด 5 ตำแหน่ง มีสองตำแหน่งในนั้นตรงกับข้อผิดพลาดที่เกิดขึ้นจริง (ข้อผิดพลาดแรกกับสุดท้าย) โดยทั่วไปขอแนะนำว่าให้แก้ข้อผิดพลาดแรก ๆ ที่ตัวแปลโปรแกรมแจ้งก่อน จากนั้นลองสั่งทำงานใหม่แล้วแก้ข้อผิดพลาดต่อไปที่ตัวแปลแจ้ง ทำเช่นนี้จนได้โปรแกรมที่ถูกหลักไวยากรณ์

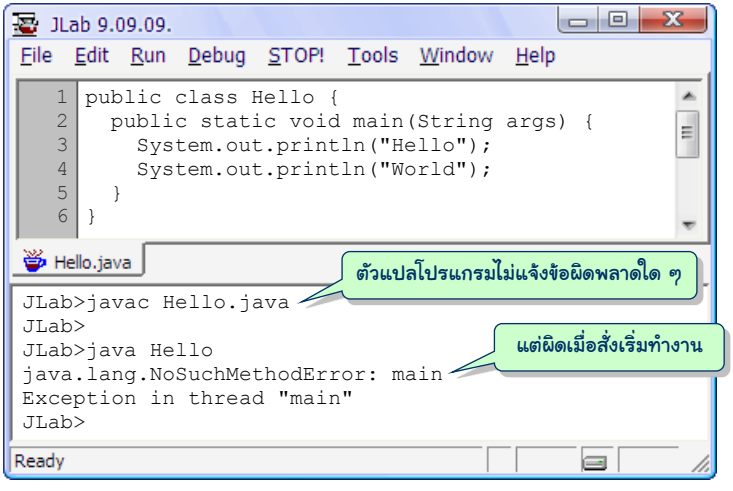


รูปที่ 1-10 ตัวแปลโปรแกรมแจ้งข้อผิดพลาดและตำแหน่งที่แจ้งอาจไม่ชัดเจนนัก

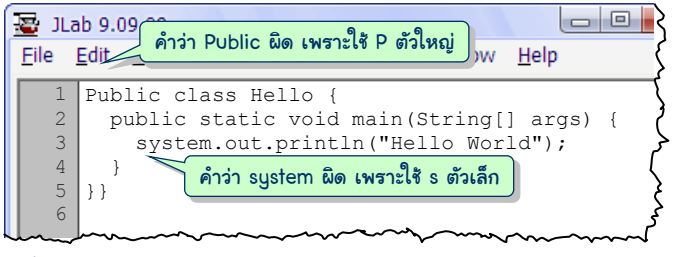
รูปที่ 1-11 แสดงโปรแกรมที่เขียนถูกหลักไวยากรณ์ทุกประการ ตัวแปลโปรแกรมไม่แจ้งข้อผิดพลาดใด ๆ แต่เมื่อระบบสั่งให้ทำงานกลับแจ้งว่า `NoSuchMethodError: main` ซึ่งหมายความว่า ระบบหาเมท็อด `main` ที่เป็นจุดเริ่มของการทำงานไม่พบในคลาส `Hello` นี้ ผู้อ่านสังเกตเห็นไหมว่าผิดตรงไหน (ดูหัวเมท็อดของ `main` จะพบว่าไม่ตรงตามแบบที่เคยเขียนมา)

ข้อผิดพลาดบางอย่างสามารถตรวจพบได้ขณะกำลังเขียนโปรแกรม หากรู้จักสังเกตสีของคำต่าง ๆ ในรหัสต้นฉบับที่ปรากฏบนจอภาพ เนื่องจากซอฟต์แวร์ช่วยพัฒนาโปรแกรมส่วนใหญ่มีคุณสมบัติการให้สีคำ ผู้อ่านที่ได้ลองเขียนโปรแกรม `Hello` ด้วย JLab จะสังเกตได้ว่า แต่ละคำของโปรแกรมมีสีกำกับต่างกัน เพราะแต่ละคำมีความหมายและหน้าที่ต่างกัน เช่น คำสำคัญของภาษาจาวามีสีน้ำเงิน คำที่แทนคลาสมาตรฐานจาวามีสีแดง สตริงมีสีแสด หมายเหตุมีสีเขียว คำที่นักเขียนโปรแกรมเพิ่มเองและเครื่องหมายสัญลักษณ์อื่น ๆ มีสีดำ เป็นต้น ทั้งนี้เพื่อให้นักเขียนโปรแกรมเห็นชัดว่า คำใดคืออะไร หากสีผิด ย่อมแสดงว่าต้องมีอะไรผิดพลาด ซึ่งสามารถตรวจสอบได้ด้วยตาระหว่างการเขียนโปรแกรม เช่นในรูปที่ 1-12 พบคำว่า `Public` ในบรรทัดที่ 1 ซึ่งใช้ `P` ตัวใหญ่ หากดูบนจอภาพใน JLab จะเห็น `Public` มีสีดำ แทนที่จะเป็นสีน้ำเงิน (เพราะ `public` เป็นคำสำคัญในภาษาจาวา) และในบรรทัดที่ 3 คำว่า `system` ซึ่งใช้ `s` ตัวเล็ก จะเห็นคำ

นี้เป็นสีดำ แทนที่จะเป็นสีแดง (เพราะ System เป็นคลาสมาตรฐานของจาวา) ด้วยการสังเกตสีระหว่างการเขียนโปรแกรม ย่อมช่วยหาที่ผิดได้เร็วขึ้น ก่อนที่ตัวแปลโปรแกรมจะตรวจพบ



รูปที่ 1-11 โปรแกรมนี้ไม่มีข้อผิดพลาดทางไวยากรณ์ แต่สั่งทำงานไม่ได้ เพราะเขียน main ผิด



รูปที่ 1-12 คำที่สะกดผิดอาจตรวจสอบได้ด้วยการสังเกตสีของคำระหว่างการเขียนโปรแกรม

โปรแกรมสวยอ่านง่าย

การเขียนโปรแกรมไม่ได้เพียงแค่เขียนให้ถูกหลักไวยากรณ์เพื่อให้ตัวแปลโปรแกรมแปลเป็นรหัสเครื่องที่ทำงานถูกต้องตามที่ตั้งใจไว้เท่านั้น นักเขียนโปรแกรมต้องเขียนรหัสต้นฉบับของโปรแกรมให้สวยและอ่านง่าย เพื่อให้ตนเองหรือผู้อื่นอ่านโปรแกรมแล้วเข้าใจการทำงาน อาจสงสัยว่า เจ้าของคนเขียนเองน่าจะเข้าใจสิ่งที่ตนเขียน แต่ในทางปฏิบัติแล้วนักเขียนโปรแกรมทั่วไปมักลืมแนวคิดการทำงานของโปรแกรมที่ตนเองเขียนหลังเวลาผ่านไป การเขียนโปรแกรมให้สวยและอ่านง่าย กระทำได้ด้วยการจัดรูปแบบรหัสต้นฉบับให้เป็นไปตามหลักปฏิบัติที่นักเขียนโปรแกรมในวงการเขาทำกัน และกระทำได้ด้วยการเติมหมายเหตุแทรกกำกับคำสั่งต่าง ๆ ภายในรหัสต้นฉบับ

ภาษาคอมพิวเตอร์ต่าง ๆ ในปัจจุบันเปิดโอกาสให้นักเขียนโปรแกรมจัดรูปแบบของรหัสต้นฉบับได้อิสระพอสมควร พิจารณารหัสที่ 1-5 กับรหัสที่ 1-6 ซึ่งเป็นโปรแกรมที่ทำงานได้เหมือนกันตามที่ต้องการ แต่รหัสที่ 1-5 อ่านยากกว่ามาก เพราะจัดรูปแบบไม่ดี ควรเขียนบรรทัดละคำสั่ง มีการย่อหน้า ใช้วรรคตอนที่เหมาะสม เพื่อให้เด่นตาว่า ส่วนใดอยู่ภายในส่วนใด อ่านแล้วเข้าใจโครงสร้างของคำสั่งซึ่งประกอบกันเป็นขั้นตอนการทำงานของโปรแกรม รหัสต้นฉบับต่างๆ ที่จะนำเสนอในหนังสือเล่มนี้ถูกจัดรูปแบบคล้ายกับข้อเสนอของบริษัทซันผู้ออกแบบภาษาจาวา [4] จะไม่ขอแจกแจงรายละเอียดของหลักปฏิบัติดังกล่าว แต่จะนำเสนอด้วยตัวอย่างไปเรื่อย ๆ จนผู้อ่านเห็นจนคุ้นตา (อย่างไรก็ตามผู้อ่านต้องเข้าใจด้วยว่า หลักปฏิบัติในการจัดรูปรหัสต้นฉบับนั้นมีหลายรูปแบบ ไม่ได้เป็นข้อบังคับ เป็นเพียงข้อเสนอแนะ)

```
01 public class Hello {                          public static void
02 main( String[]     args) { System.out.println(
03 "Hello World");                                          } }
```

รหัสที่ 1-5 โปรแกรมแสดงคำว่า `Hello World` ที่ทำงานได้ตามต้องการ แต่อ่านเข้าใจยาก

```
01 public class Hello {
02   public static void main(String[] args) {
03     System.out.println("Hello World");
04   }
05 }
```

รหัสที่ 1-6 โปรแกรมแสดงคำว่า `Hello World` ที่ได้รับการจัดรูปแบบที่ดี

วิธีเพิ่มความเข้าใจให้กับผู้อ่านรหัสต้นฉบับของโปรแกรมอีกวิธีที่ได้ผลมาก คือ การแทรก *หมายเหตุ* (comment) กำกับคำสั่งในรหัสต้นฉบับ โดยบรรยายเป็นข้อความภาษาไทยหรืออังกฤษแทรกลงไปในรหัส ณ จุดที่ต้องการอธิบาย ดังตัวอย่างที่แสดงในรหัสที่ 1-7

```
01 /*
02  * โปรแกรมภาษาจาวาเพื่อแสดงข้อความยอดนิยมทางจอภาพ
03  */
04 public class Hello {
05   public static void main(String[] args) {
06     System.out.println("Hello World");  // บรรทัดนี้เองที่แสดงผลทางจอภาพ
07   }
08 }
```

นี่คือการเขียนหมายเหตุแบบหลายบรรทัด

นี่คือการเขียนหมายเหตุแบบบรรทัดเดียวจนจบบรรทัด

รหัสที่ 1-7 โปรแกรมที่มีการเขียนหมายเหตุกำกับ

การเขียนหมายเหตุมีสองแบบ แบบแรกอนุญาตให้เราเขียนหมายเหตุยาว ๆ หลาย ๆ บรรทัดได้ โดยเริ่มหมายเหตุด้วยอักขระ `/*` และจบหมายเหตุด้วยอักขระ `*/` (บรรทัดที่ 1 ถึง 3 ของรหัสที่ 1-7) แบบที่สองเริ่มด้วยอักขระ `//` ตามด้วยหมายเหตุ ไปจนจบบรรทัดนั้น (บรรทัดที่ 6) ตัวแปลโปรแกรมจะไม่สนใจหมายเหตุในรหัสต้นฉบับ ดังนั้น นักเขียนโปรแกรมจึงเขียนรหัส

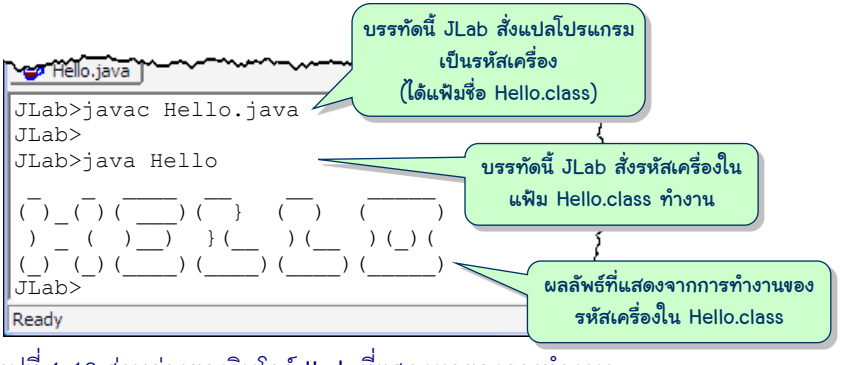
[4] ผู้สนใจสามารถอ่านรายละเอียดเพิ่มเติมได้ที่ http://java.sun.com/docs/codeconv

คำสั่งให้ตัวแปลโปรแกรมอ่าน และเขียนหมายเหตุให้คนอ่าน แต่สิ่งที่ต้องคำนึงถึงประการหนึ่ง คือ หมายเหตุจะต้องอธิบายให้ตรงกับคำสั่งที่เขียนกำกับ (เพราะไม่มีซอฟต์แวร์ตัวใดตรวจสอบได้) จึงขอแนะนำว่า หากต้องการเขียนหมายเหตุกำกับคำสั่งใด ควรเขียน ณ เวลาที่กำลังเขียนคำสั่งนั้น ๆ (จะได้ไม่ลืม) อย่าคิดรอไปเขียนหมายเหตุในภายหลัง (เพราะอาจลืมไปแล้ว)

เพิ่มเติม

รหัสต้นฉบับของแต่ละคลาสที่เราเขียนขึ้นนั้น จะถูกเก็บในแฟ้มที่มีชื่อเดียวกับชื่อคลาส โดยมีประเภทแฟ้มแบบ .java หลังผ่านการแปลเป็นรหัสเครื่องแล้ว จะถูกเก็บในอีกแฟ้มที่มีชื่อเดียวกับชื่อคลาสเช่นกัน แต่มีประเภทแฟ้มแบบ .class เช่น รหัสต้นฉบับของคลาสชื่อ Hello จะถูกเก็บในแฟ้มชื่อ Hello.java ส่วนรหัสเครื่องของคลาสนี้เก็บในแฟ้มชื่อ Hello.class

รูปที่ 1-13 แสดงส่วนล่างของวินโดว์เมื่อสั่งให้โปรแกรม Hello ทำงาน เริ่มด้วยการแปลโปรแกรมให้เป็นรหัสเครื่องก่อน เนื่องจากตัวแปลโปรแกรมของชุดพัฒนาจาวามีชื่อว่า javac.exe (ย่อมาจาก java compiler) จึงเริ่มต้นด้วยคำสั่ง javac Hello.java (บรรทัดแรกในรูปที่ 1-13) เพื่อให้ javac แปลรหัสต้นฉบับซึ่งเก็บในแฟ้ม Hello.java ถ้าไม่มีอะไรผิดพลาดทางไวยากรณ์ javac จะนำรหัสเครื่องที่ได้เก็บใส่แฟ้มชื่อ Hello.class จากนั้นจึงสั่งให้ตัวทำงาน (ในชุดพัฒนาจาวามีชื่อว่า java.exe) นำรหัสเครื่องในแฟ้ม Hello.class มาเริ่มทำงานที่เมท็อด `main` ด้วยคำสั่ง java Hello เพื่ออำนวยความสะดวกให้กับผู้เขียนโปรแกรม JLab จะทำหน้าที่สั่งแปลและสั่งทำงานให้อัตโนมัติเมื่อผู้เขียนโปรแกรมกดปุ่ม F5 เป็นการลดภาระที่ต้องทำเป็นประจำระหว่างการเขียนโปรแกรม



รูปที่ 1-13 ส่วนล่างของวินโดว์ JLab ที่แสดงผลของการทำงาน

แบบฝึกหัด

1. จงเติมคำในช่องว่างของประโยคต่อไปนี้

   1.1. ทุกคำสั่งในจาวาต้องปิดท้ายด้วยเครื่องหมาย_____

   1.2. เมื่อสั่งคลาสหนึ่งทำงาน ระบบจะไปทำที่เมท็อดชื่อ_____ของคลาสนั้น

   1.3. ตัวแปลโปรแกรมจะไม่สนใจส่วนที่เป็น_____ในรหัสต้นฉบับ

   1.4. ตัวแปลโปรแกรมมีหน้าที่เปลี่ยนรหัส_____เป็นรหัส_____

   1.5. เราจัดรูปแบบของรหัสต้นฉบับให้สวยเพื่อให้_____อ่านโปรแกรมได้ง่าย

2. ถ้าเปลี่ยนคำว่า `main` เป็น `Main` ในรหัสที่ 1-1 แล้วสั่งแปลและสั่งทำงาน จะเกิดปัญหาหรือไม่อย่างไร และถ้าลองเปลี่ยน `args` เป็น `abcde` แล้วสั่งทำงานจะเกิดปัญหาหรือไม่

3. จงเขียนโปรแกรมเพื่อแสดงข้อความ **ตนเป็นที่พึ่งแห่งตน** 10 ครั้ง ๆ ละบรรทัด

4. ถ้าต้องการหาค่า 123+456+789 สามารถใช้ `System.out.println(123+456+789)` เพื่อคำนวณและแสดงผลลัพธ์ ถ้าต้องการหาค่า 10! (แฟกทอเรียลของ 10) จะเขียนคำสั่งอย่างไร (การคูณจำนวนในจาวาใช้เครื่องหมาย `*`)

5. ลองเขียนโปรแกรมข้างล่างนี้ แล้วสั่งทำงานดูว่า มีอะไรผิดหรือไม่ ถ้าผิดต้องแก้ไขอย่างไร

```
01 public class Wisawakarma {
02   public static void main(String[] args) {
03     System.out.println(
04         " แต่กรุงไทยศรีวิไลทันเพื่อนบ้าน  จึ่งมีช่างชำนาญวิเลขา
05           ทั้งช่างปั้นช่างเขียนเพียรวิชา    อีกช่างสถาปนาถูกทำนอง
06           ทั้งช่างรูปพรรณสุวรรณกิจ         ช่างประดิษฐ์รัชดาสง่าผ่อง
07           อีกช่างถมลายลักษณะจำลอง          อีกช่ำชองเชิงรัตนะประกร " );
08   }
09 }
```

6. น่าสงสัยว่า เราเขียนหมายเหตุแบบ `/*  */` และ `//` แทรกกลางคำสั่งได้หรือไม่ ลองเขียนโปรแกรมข้างล่างนี้ แล้วสั่งทำงานดูว่า มีอะไรผิดหรือไม่

```
01 public class Comment1 {
02   public static void main(String[] args) {
03     System.out. /*comment*/ println("Hello");
04     System.out. //           println("Hello");
05   }
06 }
```

7. อยากทราบว่า เราเขียนหมายเหตุซ้อนอยู่ในหมายเหตุได้หรือไม่ ลองเขียนโปรแกรมข้างล่างนี้ แล้วสั่งทำงานดูว่า มีอะไรผิดหรือไม่

```
01 public class Comment2 {
02   /*
03      /* --------------- */
04   */
05   public static void main(String[] args) { }
06 }
```

8. โปรแกรมข้างล่างนี้ทำอะไร ช่วยจัดรูปแบบใหม่ให้สวยงาม อ่านง่าย

```
01 public class
02 Ugly {         public   static void main   (      String[]
03 args) { System.
04 out.println                                    (
05 1+2+3+                  4+
06 5+6                        );}                 }
```

9. ที่ผ่านมาเราใช้คำสั่ง `System.out.println` แสดงข้อความทางจอภาพ โปรแกรมข้างล่างนี้เป็นตัวอย่างการแสดงข้อความบนวินโดว์แบบง่าย ให้ลองเพิ่มคำสั่งแสดงชื่อตนเอง และสวัสดีอีกสักสองคำสั่ง แล้วสั่งงานดูว่าได้ผลเช่นไร

```
01 import javax.swing.JOptionPane;
02 public class Hello {
03   public static void main(String[] args) {
04     JOptionPane.showMessageDialog(null, "Hello World");
05   }
06 }
```

10. โปรแกรมข้างล่างนี้ผิดที่ไหน จงแก้ไขให้ถูกต้อง

```
01 public class wrong {
02   public static vOid main(String() a) [
03     System.out.print('Java ')
04     System.out.print('Programming')
05   ]
06 }
```

๒ คำนวณ

คอมพิวเตอร์ซอบคำนวณ ซอฟต์แวร์ทั้งหลายที่เราใช้เล่นเกม ดูหนัง ฟังเพลง ค้นข้อมูล แชต เขียนบล็อก ทำรายงาน หรืออื่น ๆ ล้วนแล้วแต่ใช้การคำนวณเป็นพื้นฐานสำคัญในขั้นตอนการทำงานของโปรแกรมเหล่านี้ จึงขอเริ่มนำเสนอการเขียนโปรแกรมด้วยเรื่องการบวกลบคูณหารจำนวนเต็มและจำนวนจริง การสร้างและการใช้งานตัวแปรเพื่อเก็บข้อมูลชั่วคราวระหว่างการประมวลผล วิธีการอ่านข้อมูลทางแป้นพิมพ์จากผู้ใช้ การเรียกใช้คลังคำสั่งคำนวณฟังก์ชันทางคณิตศาสตร์ที่ใช้บ่อย ๆ และปิดท้ายด้วยการทดสอบโปรแกรมและการแก้จุดบกพร่องของโปรแกรม

บวก ลบ คูณ หาร

หน่วยประมวลผลกลางของคอมพิวเตอร์มีวงจรฮาร์ดแวร์เฉพาะเพื่อการบวก ลบ คูณ และหารจำนวนได้โดยตรง ทำให้การประมวลผลเหล่านี้กระทำได้อย่างรวดเร็วมาก จึงไม่แปลกที่ภาษาคอมพิวเตอร์จะมี*ตัวดำเนินการคำนวณ* (arithmetic operator) เพื่อการบวก ลบ คูณ และหาร โดยใช้เครื่องหมาย `+` `-` `*` และ `/` ตามลำดับ (เนื่องจากแป้นพิมพ์ไม่มีเครื่องหมาย × และ ÷ ครั้นจะใช้ตัวเอ็กซ์แทนคูณก็ไม่ค่อยดีเพราะ x ใช้ในความหมายอื่น) รหัสที่ 2-1 แสดงตัวอย่างโปรแกรมที่คำนวณแล้วนำผลแสดงทางจอภาพ (ผู้อ่านคงคาดการณ์ได้ว่า โปรแกรมนี้จะแสดงผลอะไรทางจอภาพ ถ้าไม่แน่ใจก็ลองเขียนโปรแกรมและสั่งทำงานดู)

เครื่องหมายบวกมีความพิเศษอีกประการหนึ่งเมื่อนำไปใช้กับสตริง การบวกกับสตริงคือการต่อสตริง เช่น `"A"+"B"` จะได้ `"AB"` และถ้านำสตริงบวกกับจำนวน เช่น `"K"+9` ตัวแปลโปรแกรมจะตีความว่า บวกนี้คือการต่อสตริงเช่นกัน จึงเปลี่ยน `9` เป็นสตริงก่อนแล้วค่อยต่อ ดังนั้น `"K"+9` จะได้ `"K9"` ในขณะที่ `9+"K"` จะได้ `"9K"`

```
01 public class Operators {
02   public static void main(String[] args) {
03     System.out.println( 4+2 );
04     System.out.println( 4-2 );
05     System.out.println( 4*2 );
06     System.out.println( 4/2 );
07     System.out.println( 1+2+3+4+5+6+7+8+9+10 );
08   }
09 }
```

รหัสที่ 2-1 ตัวอย่างการใช้ตัวดำเนินการคำนวณ + - * และ /

สิ่งที่ต้องจำอย่างหนึ่งคือ การบวกจะทำจากซ้ายไปขวา ซึ่งไม่แปลกอะไรเลยถ้าเป็นการบวกจำนวน เช่น `1+2+4` เท่ากับ `(1+2)+4` เท่ากับ `3+4` เท่ากับ `7` แต่ถ้าบวกเป็นการต่อสตริงจะต้องระวังเป็นพิเศษ เช่น `"A"+1+2` เท่ากับ `("A"+1)+2` เท่ากับ `"A1"+2` เท่ากับ `"A12"` แต่ถ้าเขียน `1+2+"A"` เท่ากับ `(1+2)+"A"` เท่ากับ `3+"A"` เท่ากับ `"3A"` เพราะบวกตัวซ้ายเป็นการบวกจำนวนกับจำนวน หากต้องการให้ `"x = "+4+2` มีค่าเป็น `"x = 6"` ก็ต้องใส่วงเล็บให้กับการบวกแบบจำนวนเพื่อบังคับให้ทำบวกจำนวนก่อนการต่อสตริง จึงต้องเขียน `"x = "+(4+2)` ดังแสดงด้วยตัวอย่างในรหัสที่ 2-2 ในที่นี้เราต้องการแสดง `4+2 = 6` ทำได้โดยใช้ `print` แสดง `"4+2 = "` ก่อน (บรรทัดที่ 3) แล้วค่อยแสดงผลการคำนวณ `4+2` (บรรทัดที่ 4) หรืออีกวิธี ให้คำนวณ `(4+2)` นำผลไปต่อท้ายสตริง `"4+2 = "` แล้วค่อยนำผลสุดท้ายไปแสดง (บรรทัดที่ 5)

```
01 public class StringConcatenation {
02   public static void main(String[] args) {
03     System.out.print( "4+2 = " );
04     System.out.println( 4+2 );
05     System.out.println( "4+2 = " + (4+2) );
06   }
07 }
```

อย่าลืมใส่วงเล็บเพื่อให้คำนวณตรงนี้ก่อน

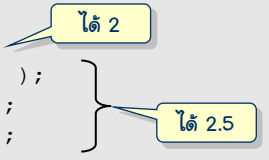
รหัสที่ 2-2 ตัวอย่างการต่อสตริง

จำนวนเต็มกับจำนวนจริง

นอกจากหน่วยประมวลผลกลางจะมีวงจรฮาร์ดแวร์เพื่อการบวกลบคูณหารแล้ว ยังแบ่งเป็นวงจรบวกลบคูณหารสำหรับจำนวนเต็ม และสำหรับจำนวนจริง ภาษาคอมพิวเตอร์ก็มีจำนวนให้ใช้สองแบบนี้เช่นกัน ข้อแตกต่างที่เห็นเด่นชัดคือการหาร การหารแบบจำนวนเต็มจะตัดเศษทิ้ง รหัสที่ 2-3 เปรียบเทียบผลที่ได้ของการหารแบบจำนวนเต็ม กับการหารแบบจำนวนจริง ผลของ `5/2` เป็น `2` เพราะเป็นการหารจำนวนเต็ม ส่วนผลของ `5.0/2.0` เป็น `2.5` ตามคาด สรุปว่า เราเขียนจำนวนเต็มโดยไม่ใส่จุดทศนิยม ถ้าเขียนจำนวนแบบมีจุดทศนิยมจะหมายถึงจำนวนจริง สำหรับกรณีที่เกิดการคำนวณระหว่างจำนวนเต็มกับจำนวนจริง ตัวแปลโปรแกรมจะเปลี่ยนจำนวนเต็มให้

เป็นจำนวนจริงก่อนแล้วจึงคำนวณ ดังตัวอย่างในบรรทัดที่ 5 และ 6 ของรหัสที่ 2-3 ซึ่งจะเปลี่ยน `5.0/2` เป็น `5.0/2.0` และเปลี่ยน `5/2.0` เป็น `5.0/2.0` แล้วจึงคำนวณแบบจำนวนจริง รหัสที่ 2-4 แสดงอีกตัวอย่างซึ่งเป็นโปรแกรมที่คำนวณพื้นที่ของวงกลมที่มีรัศมียาว 4 หน่วย

```
01 public class IntegerAndReal {
02   public static void main(String[] args) {
03     System.out.println( "5/2     = " + (5/2) );
04     System.out.println( "5.0/2.0 = " + (5.0/2.0) );
05     System.out.println( "5.0/2   = " + (5.0/2) );
06     System.out.println( "5/2.0   = " + (5/2.0) );
07   }
08 }
```



รหัสที่ 2-3 ตัวอย่างการคำนวณแบบจำนวนเต็มและจำนวนจริง

```
01 public class CircleArea {
02   public static void main(String[] args) {
03     System.out.println("วงกลมที่มีรัศมี 4 หน่วย มีพื้นที่ " + (3.14159*4*4));
04   }
05 }
```

รหัสที่ 2-4 ตัวอย่างการคำนวณพื้นที่ของวงกลมรัศมี 4 หน่วย

ตัวแปร

นอกจากคอมพิวเตอร์จะถนัดการคำนวณแล้ว ยังถนัดงานเรื่องเก็บข้อมูลไว้ใช้ในภายหน้า การเก็บข้อมูลสามารถเก็บเพื่อใช้ชั่วคราวในหน่วยความจำหลักที่เรียกว่า *แรม* (RAM) กับเก็บไว้ถาวรในหน่วยความจำสำรอง (ข้อมูลไม่สูญหายเมื่อปิดเครื่องคอมพิวเตอร์) เช่น ฮาร์ดดิกส์หรือแฟลชไดรว์ (flash drive หรือ thumb drive) เหตุที่เราเก็บข้อมูลแบบชั่วคราวในแรมก็เพราะเวลาในการเก็บและเรียกใช้ข้อมูลในแรมนั้นเร็วกว่าแบบที่เก็บถาวรเป็นล้านเท่า (และแน่นอนว่าเราไม่ต้องการเก็บข้อมูลนั้นแบบถาวรด้วย) ในที่นี้ขออธิบายเฉพาะถึงการเก็บข้อมูลแบบชั่วคราว

ภาษาคอมพิวเตอร์มีสิ่งที่เรียกว่า *ตัวแปร* (variable) ให้นักเขียนโปรแกรมใช้เก็บข้อมูลชั่วคราว เหตุที่เรียกว่าตัวแปร เพราะเราสามารถเปลี่ยนค่าที่เก็บในตัวแปรได้ เมื่อใดต้องการใช้ตัวแปร ต้องประกาศตัวแปร การประกาศตัวแปรต้องกำหนดชื่อให้กับตัวแปร และต้องกำหนดประเภทข้อมูลที่ตัวแปรนั้นเก็บได้ รหัสที่ 2-5 แสดงตัวอย่างโปรแกรมที่ทำงานเหมือนรหัสที่ 2-4 แต่ใช้ตัวแปรสามตัว บรรทัดที่ 3 ประกาศตัวแปรชื่อ `r` เก็บจำนวนเต็ม บรรทัดที่ 4 ประกาศตัวแปรชื่อ `pi` เก็บจำนวนจริง และบรรทัดที่ 5 ประกาศตัวแปรชื่อ `s` เก็บสตริง (ทั้งสามบรรทัดนี้มีการให้ค่าเริ่มต้นกับตัวแปรทั้งสามด้วย) จากนั้นประกาศตัวแปรชื่อ `area` อีกตัว (บรรทัดที่ 6) เพื่อเป็นที่เก็บผลการคำนวณพื้นที่ แล้วจึงนำตัวแปรนี้ไปแสดงผลในบรรทัดที่ 7

```
01 public class CircleArea {
02   public static void main(String[] args) {
03     int    r  = 4;
04     double pi = 3.14159;
05     String s  = "วงกลมที่มีรัศมี 4 หน่วย มีพื้นที่ ";
06     double area = pi * r * r;
07     System.out.println(s + area);
08   }
09 }
```

การประกาศตัวแปรสามตัว

นำค่าของตัวแปรมาใช้ด้วยชื่อตัวแปร

รหัสที่ 2-5 ตัวอย่างการคำนวณพื้นที่ของวงกลมรัศมี 4 หน่วยโดยใช้ตัวแปร

คำว่า `int` `double` และ `String` เป็นคำที่ระบุประเภทข้อมูลแบบจำนวนเต็ม จำนวนจริง และสตริงตามลำดับ การประกาศตัวแปรเริ่มด้วยประเภทข้อมูล ตามด้วยชื่อตัวแปร และตามด้วยการตั้งค่าเริ่มต้นให้กับตัวแปร (ส่วนหลังนี้จะเขียนหรือไม่ก็ได้) ในกรณีที่ต้องการประกาศตัวแปรหลายตัวที่มีประเภทเดียวกัน ก็สามารถประกาศในบรรทัดเดียวกันก็ได้ เช่น ถ้าต้องการตัวแปรจำนวนเต็มสามตัวชื่อ `a, b, c` อาจประกาศสามครั้งด้วย `int a; int b; int c;` หรือจะประกาศรวมกันเป็น `int a, b, c;` ก็ได้ และต้องอย่าลืมว่า เราต้องประกาศตัวแปรก่อนจึงจะใช้ตัวแปรนั้นได้ การประกาศตัวแปรต่อไปนี้ทางซ้ายถูก แต่ทางขวาผิด

```
int i;
double a, b;
int j=5, k=8;
String s1, s2, s3;
```

```
Int s;      // สะกด Int ผิด ต้อง i ตัวเล็ก
string t;   // สะกด string ผิด ต้อง S ตัวใหญ่
y = 6.5;    // ผิดเพราะใช้ y ก่อนที่จะประกาศ
double y;
int p, double q; // ประกาศคนละประเภท ต้องแยกคำสั่ง
                 // ใช้ , ไม่ได้ ต้องใช้ ;
```

การตั้งชื่อตัวแปร

การประกาศตัวแปรต้องตั้งชื่อตัวแปร หลักการสำคัญของการตั้งชื่อตัวแปรคือ ควรตั้งให้ชื่อสื่อความหมายกับหน้าที่ของตัวแปรนั้น ๆ เพื่อให้อ่านโปรแกรมแล้วเข้าใจง่าย อย่างไรก็ตามจะตั้งชื่อตามอำเภอใจก็ไม่ได้ ต้องเป็นไปตามกฎที่ภาษาคอมพิวเตอร์กำหนดไว้ กฎการตั้งชื่อของจาวาเป็นดังนี้ (กฎนี้ใช้กับทั้งการตั้งชื่อตัวแปร ชื่อคลาส และชื่อเมท็อดด้วย)

1. ประกอบด้วยตัวอักษร ตัวเลข สัญลักษณ์ `$` หรือ `_` ก็ได้
2. ห้ามขึ้นต้นด้วยตัวเลข
3. ชื่อยาวๆ ได้ไม่เป็นไร
4. ตัวอักษรอังกฤษตัวใหญ่ไม่เหมือนตัวเล็ก (เช่น `A` ต่างกับ `a`)
5. ต้องไม่ซ้ำกับคำสงวนของภาษาจาวา ดังแสดงในรูปที่ 2-1
6. ต้องไม่ซ้ำกับชื่ออื่นๆ ของโปรแกรมที่ประกาศมาก่อน

```
abstract    assert      boolean     break       byte
case        catch       char        class       const
continue    default     do          double      else
enum        extends     false       final       finally
float       for         goto        if          implements
import      instanceof  int         interface   long
native      new         null        package     private
protected   public      return      short       static
strictfp    super       switch      synchronized this
throw       throws      transient   true        try
void        volatile    while
```

รูปที่ 2-1 คำสงวนต่าง ๆ ของภาษาจาวา

นักเขียนโปรแกรมมีภาระต้องตั้งทั้งชื่อคลาส ชื่อเมท็อด และชื่อตัวแปร ในวงการนักเขียนโปรแกรมภาษาจาวา มักให้ตัวอักษรแรกของชื่อคลาสเป็นตัวใหญ่ (เช่น `String`, `Robot`, `Window`, `Circle`, ...) แต่ให้ตัวอักษรแรกของชื่อเมท็อดและชื่อตัวแปรเป็นตัวเล็ก (เช่น `area`, `radius`, `main`, ...) หลักปฏิบัติสำคัญในการตั้งชื่อคือ ควรตั้งให้สื่อความหมาย จะเป็นวลีก็ได้ ถ้าอยากตั้งชื่อซึ่งประกอบด้วยคำหลายคำ ให้เขียนทุกคำให้ติดกัน แล้วให้ตัวแรกของแต่ละคำเป็นตัวใหญ่ (ยกเว้นคำแรก ขึ้นกับว่าเป็นคลาส เมท็อด หรือตัวแปร) เช่น ต้องการตั้งชื่อตัวแปรที่เก็บวันของสัปดาห์ ก็ให้ชื่อว่า `dayOfWeek` อย่างไรก็ตามก็อย่าตั้งชื่อยาวเกินไป เอาเป็นว่า ให้สั้นแต่สื่อความหมาย โดยทั่วไปควรหลีกเลี่ยงการใช้สัญลักษณ์ `$` และ `_` (เราสามารถตั้งชื่อ เช่น `$$_$$`, `___`, `$1$2` ได้ไม่ผิดกฎ แต่ดูไม่ดีเลย) [1]

อนึ่งไม่มีข้อห้ามใด ๆ เกี่ยวกับการตั้งชื่อด้วยตัวอักษรในภาษาอื่น เราสามารถตั้งชื่อภาษาไทยก็ได้ ดังแสดงในรหัสที่ 2-6 อย่างไรก็ตามโปรแกรมต่าง ๆ ที่จะนำเสนอในหนังสือเล่มนี้จะตั้งชื่อเป็นภาษาอังกฤษ

```
01 public class พื้นที่วงกลม {
02   public static void main(String[] args) {
03     int    รัศมี  = 4;
04     double ค่าพาย = 3.14159;
05     String ข้อความ  = "วงกลมที่มีรัศมี 4 หน่วย มีพื้นที่ ";
06     double พื้นที่ = ค่าพาย * รัศมี * รัศมี;
07     System.out.println(ข้อความ + พื้นที่);
08   }
09 }
```

รหัสที่ 2-6 โปรแกรมที่ทำงานเหมือนรหัสที่ 2-5 แต่ตั้งชื่อต่าง ๆ ในโปรแกรมเป็นภาษาไทย

[1] นักเขียนโปรแกรมบางคนอาจพยายามตั้งชื่อให้คนอื่นอ่านไม่รู้เรื่อง เพื่อป้องกันไม่ให้ผู้อื่นเข้าใจกลวิธีการเขียนโปรแกรมซึ่งถือว่าเป็นทรัพย์สินทางปัญญาของเขา ถ้าเขามีจุดประสงค์เช่นนั้นจริง ก็ไม่ควรใช้วิธีตั้งชื่อแปลก ๆ เพราะสักวันเจ้าของเองจะไม่เข้าใจโปรแกรมที่ตนเองเขียน ควรหันไปใช้เครื่องมือที่เรียกว่า obfuscator ซึ่งเป็นซอฟต์แวร์ที่อ่านรหัสต้นฉบับมาแปลงให้เป็นรหัสต้นฉบับใหม่ที่ทำงานเหมือนของเดิม แต่อ่านเข้าใจยากมาก ๆ ด้วยเทคนิคของการเปลี่ยนชื่อและเปลี่ยนคำสั่งการทำงานแบบต่าง ๆ

การให้ค่าตัวแปร

เครื่องหมาย = ในภาษาจาวาไม่ได้แสดงความเท่ากันของค่าทั้งสองข้างของ = แต่เป็นตัวดำเนินการให้ค่ากับตัวแปร โดยนำค่าหรือผลของการคำนวณทางขวาของ = ไปใส่ในตัวแปรที่อยู่ทางซ้ายของ = ดังนั้น คำสั่ง `a = b+1` จึงเสมือนกับ a ← (b+1) แทนการนำค่าในตัวแปร `b` มาบวกด้วย `1` ได้ผลไปเก็บในตัวแปร `a` คำสั่ง `a = a+1` แทนการนำค่าในตัวแปร `a` มาบวกด้วย `1` ได้ผลไปเก็บในตัวแปร `a` ตัวเดิม จึงหมายถึงการเพิ่มค่าที่เก็บใน `a` ขึ้นอีก `1` ด้วยความหมายของ = ที่ได้นำเสนอมา คำสั่ง `b+1 = a` จึงผิดหลักไวยากรณ์

สิ่งที่ต้องคำนึงถึงอีกประการหนึ่งเกี่ยวกับการให้ค่าตัวแปรคือ ต้องให้ค่าที่มีประเภทเดียวกับประเภทของตัวแปรที่รับค่า นั่นคือนำจำนวนเต็มเก็บใส่ตัวแปรแบบ `int` นำจำนวนจริงเก็บใส่ตัวแปรแบบ `double` และนำสตริงเก็บใส่ตัวแปร `String` มีข้อยกเว้นให้หนึ่งกรณีคือ สามารถนำจำนวนเต็มเก็บใส่ตัวแปร `double` ได้ (เพราะจำนวนเต็มถือว่าเป็นจำนวนจริง) เช่น สมมติให้มีตัวแปร `int i;double d;` และ `String s;` การให้ค่าต่อไปนี้ทางซ้ายถูก แต่ทางขวาผิด [2]

```
i = 2;
d = 1.2E-3; // 1.2×10^-3
s = "kid";
d = -1 * d;
d = i + 12;
```

```
s = 5;     // ให้ int กับ String ไม่ได้
s = 5.0;   // ให้ double กับ String ไม่ได้
i = "1";   // ให้ String กับ int ไม่ได้
d = "5.1"; // ให้ String กับ double ไม่ได้
i = 1.5;   // ให้ double กับ int ไม่ได้
```

การเปลี่ยนประเภทข้อมูล

ในกรณีที่เราต้องการใส่ข้อมูลให้กับตัวแปรที่เป็นคนละประเภทก็สามารถทำได้ แต่ยุ่งยากเล็กน้อย ภาษาคอมพิวเตอร์บางภาษาใจดี เปลี่ยนประเภทข้อมูลให้อัตโนมัติ บางภาษาไม่ยอมเปลี่ยนให้ (จาวาเป็นหนึ่งในกรณีนี้) ถ้าต้องการเปลี่ยนต้องเขียนเพิ่ม สำหรับจาวาต้องทำดังนี้

1. ***เปลี่ยนจาก `double` เป็น `int`*** : ให้ใส่ `(int)` นำหน้าข้อมูล `double` ที่ต้องการเปลี่ยน ผลที่ได้คือจำนวนเต็มที่ตัดเศษทิ้ง เช่น สมมติว่า `d` เป็นตัวแปร `double` หากเขียน `int i = 100/d;` ถือว่าผิด ถ้าต้องการเปลี่ยนผลของ `100/d` ให้เป็นจำนวนเต็ม ต้องเขียน `int i = (int)(100/d);` ต้องระวังอย่าเขียน `(int)100/d` เพราะคือการเปลี่ยน `100` เป็น `int` ก่อนแล้วหารด้วย `d` จะได้ผลเป็น `double` ซึ่งผิดไวยากรณ์เหมือนเดิม

2. ***เปลี่ยนจาก `int` เป็น `double`*** : ให้ใส่ `(double)` นำหน้าข้อมูล `int` ที่ต้องการเปลี่ยน การเปลี่ยนในลักษณะนี้มักใช้กับการเปลี่ยนจำนวนเต็มให้เป็นจำนวนจริงระหว่างการ

[2] เราสามารถเขียนค่าคงตัวของจำนวนจริงในรูปแบบสัญกรณ์ทางวิทยาศาสตร์ในจาวาได้ เช่น 2.03×10^{-11} เขียนแทนด้วย `2.03e-11` หรือ `2.03E-11` โดยเลขชี้กำลังของ 10 ต้องเป็นจำนวนเต็มเท่านั้น

คำนวณเพื่อให้คำนวณแบบจำนวนจริง เช่น สมมติว่า `i` และ `k` เป็นตัวแปร `int` หากเขียน `double d = i/k;` ก็ไม่ผิด แต่การหารจะเป็นแบบจำนวนเต็ม ถ้าต้องการให้หารแบบจำนวนจริง ก็ต้องเปลี่ยน `i` หรือ `k` ให้เป็นจำนวนจริง ดังนั้น จึงต้องเขียน `d = (double)i/k` หรือไม่ก็ `d = i/(double)k;` แต่ไม่ใช่ `d = (double)(i/k)` เพราะจะหมายความว่าให้คำนวณ `i/k` ก่อนแล้วเปลี่ยนเป็น `double` ซึ่งสายเกินไป

3. ***เปลี่ยนจาก `int` หรือ `double` เป็น `String`*** : แบบนี้เปลี่ยนง่าย โดยนำจำนวนที่ต้องการเปลี่ยนไปบวกกับสตริงว่าง ๆ (เขียน " สองตัวติดกัน "") เนื่องจากการบวกกับสตริงคือการต่อสตริง จึงแปลงจำนวนเป็นสตริงก่อนต่อ เช่น ให้ `i` เป็น `int` ส่วน `d` เป็น `double` คำสั่ง `String s1 = "" + i;` กับ `String s2 = "" + d;` จะเปลี่ยน `i` เป็นสตริงให้กับ `s1` และเปลี่ยน `d` เป็นสตริงให้กับ `s2`

4. ***เปลี่ยนจาก `String` เป็น `int`*** : แบบนี้ยุ่งเล็กน้อย ต้องใช้คำสั่งพิเศษ ถ้า `s` เป็นสตริง คำสั่ง `int i = Integer.parseInt(s);` ทำหน้าที่เปลี่ยนสตริง `s` เป็นจำนวนเต็ม เช่น `s = "12"` ก็จะได้ `i = 12` คำเตือน : ข้อมูลใน `s` ต้องเป็นแบบที่เปลี่ยนได้ เช่น ถ้า `s = "12X"` จะเกิดข้อผิดพลาดเมื่อทำคำสั่งนี้ หรือถ้า `s = "1.0"` ก็ผิดพลาดเหมือนกัน

5. ***เปลี่ยนจาก `String` เป็น `double`*** : แบบนี้คล้ายแบบที่แล้ว ถ้า `s` เป็นสตริง ต้องใช้คำสั่ง `double d = Double.parseDouble(s);` ซึ่งทำหน้าที่เปลี่ยนสตริง `s` เป็นจำนวนจริง เช่น `s = "1.5"` ก็จะได้ `d = 1.5` คำเตือน : ข้อมูลใน `s` ต้องเป็นแบบที่เปลี่ยนได้ เช่น ถ้า `s = "#12"` จะเกิดข้อผิดพลาดเมื่อทำคำสั่งนี้

การอ่านข้อมูลจากแป้นพิมพ์

ที่ผ่านมา เรายังไม่ได้เขียนโปรแกรมที่มีสาระใด ๆ เลย โปรแกรมที่คำนวณพื้นที่วงกลมในรหัสที่ 2-5 ก็คำนวณเฉพาะวงกลมรัศมียาว 4 หากต้องการคำนวณกรณีรัศมีเป็น 5 ก็คงต้องแก้ไขโปรแกรม แล้วสั่งทำงานใหม่ โปรแกรมที่ดีกว่าควรรับความยาวรัศมีจากผู้ใช้ คำนวณพื้นที่วงกลมแล้วแสดงผลให้ผู้ใช้ทราบ เราจึงต้องทราบวิธีการรับข้อมูลจากผู้ใช้ หนทางการรับข้อมูลขาเข้าจากผู้ใช้ที่กระทำได้ง่าย ๆ คือผ่านทางแป้นพิมพ์

รหัสที่ 2-7 แสดงโปรแกรมที่รับรัศมีจากผู้ใช้ทางแป้นพิมพ์ จากนั้นคำนวณพื้นที่ และแสดงผลทางจอภาพ ก่อนจะรับข้อมูลทางแป้นพิมพ์ได้ เราต้องสร้างตัวอ่านแป้นพิมพ์ด้วยคำสั่ง `Scanner kb = new Scanner(System.in);` (บรรทัดที่ 5) คำสั่งนี้แท้จริงคือการประกาศตัวแปรชื่อ `kb` ซึ่งเป็นข้อมูลประเภท `Scanner` พร้อมกับให้ค่าเริ่มต้นที่ได้จากการสร้าง

ตัวอ่าน (คำสั่ง new Scanner) ซึ่งอ่านข้อมูลจากแป้นพิมพ์ (System.in คือแป้นพิมพ์) และถ้าดูบรรทัดที่ 1 จะพบคำสั่งซึ่งบอกตัวแปลโปรแกรมว่า เราจะใช้คลาส Scanner ซึ่งมีชื่อเต็มว่า java.util.Scanner เพื่อให้ตัวแปลรู้ว่าจะไปค้นหาได้ที่ไหนในระบบ ผู้อ่านอย่าเพิ่งสงสัยตกใจเกี่ยวกับความซับซ้อนของคำสั่งเหล่านี้ ขอให้จำไปก่อน แล้วจะค่อย ๆ เข้าใจถึงความหมายที่แท้จริงในบทท้าย ๆ !!! หลังจากมีตัวแปร kb (ตั้งชื่อว่า kb เพราะตั้งใจให้เป็นคำย่อของ keyboard) ก็สามารถเรียกเมท็อดหรือบริการหลากหลายได้ เช่น

- kb.nextDouble() คือ ให้รอรับข้อมูลแบบ double ทางแป้นพิมพ์
- kb.nextInt() คือ ให้รอรับข้อมูลแบบ int ทางแป้นพิมพ์
- kb.nextLine() คือ ให้รอรับข้อมูลทั้งบรรทัดแบบ String ทางแป้นพิมพ์ [3]

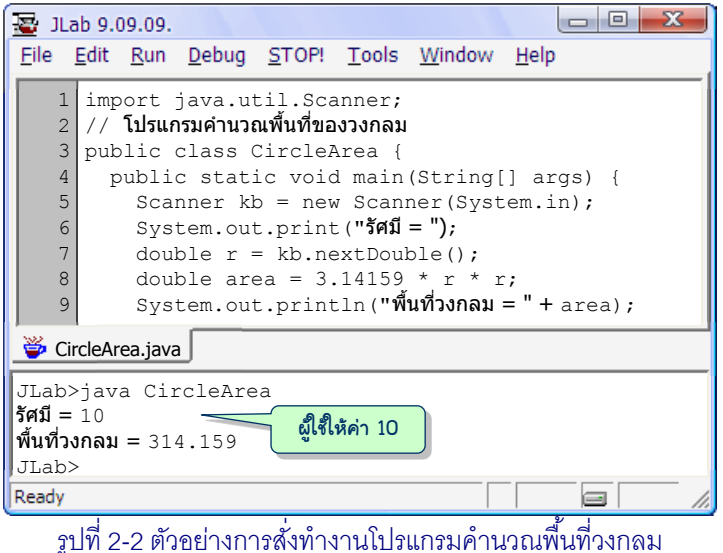
```
01 import java.util.Scanner;
02 // โปรแกรมคำนวณพื้นที่ของวงกลม
03 public class CircleArea {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("รัศมี = ");
07     double r = kb.nextDouble();
08     double area = 3.14159 * r * r;
09     System.out.println("วงกลมมีพื้นที่ = " + area);
10   }
11 }
```

บอกตัวแปลโปรแกรมว่าเราจะใช้คลาสที่มีชื่อเต็มว่า java.util.Scanner

สร้างตัวอ่านแป้นพิมพ์

รออ่านข้อมูลแบบ double

รหัสที่ 2-7 โปรแกรมคำนวณพื้นที่ของวงกลมจากรัศมีที่รับทางแป้นพิมพ์



รูปที่ 2-2 ตัวอย่างการสั่งทำงานโปรแกรมคำนวณพื้นที่วงกลม

[3] ในกรณีที่ต้องการอ่านสตริงในบรรทัดจนถึงช่องว่าง เช่น ผู้ใช้ป้อน AB CD แล้วต้องการอ่านครั้งแรกได้ AB อ่านอีกครั้งได้ CD ให้ใช้ kb.next() แทนการรอรับข้อมูลแบบ String หนึ่ง “ก้อน” ทางแป้นพิมพ์

รูปที่ 2-2 แสดงตัวอย่างการสั่งทำงานโปรแกรมคำนวณพื้นที่วงกลม หลังจากสร้างตัวอ่านในบรรทัดที่ 5 แล้ว บรรทัดที่ 6 แสดงข้อความ "รัศมี = " เพื่อให้ผู้ใช้ทราบว่า ต้องใส่ความยาวของรัศมี บรรทัดที่ 7 รอจนกว่าผู้ใช้จะใส่จำนวนแล้วกดปุ่ม ENTER (ให้สังเกตว่า เราใช้ `print` ในบรรทัดที่ 6 เพื่อให้จำนวนที่ผู้ใช้ป้อนอยู่หลังคำว่า "รัศมี = " เมื่อผู้ใช้ป้อนค่าและกดปุ่ม ENTER ก็เป็นการขึ้นบรรทัดใหม่โดยอัตโนมัติ) จากตัวอย่างในรูปที่ 2-2 ผู้ใช้ให้ค่า 10 ทำให้โปรแกรมทำงานต่อคำนวณ แสดงพื้นที่วงกลม และสิ้นสุดการทำงาน

การใช้ตัวแปร

ถ้าอยากจะรู้ว่า เราอ้วน ผอม หรือสมส่วน ก็เพียงแค่คำนวณดัชนีมวลกายของเราด้วยสูตรข้างล่างนี้ โดยที่น้ำหนัก (*weight*) มีหน่วยเป็นกิโลกรัม ส่วนความสูง (*height*) มีหน่วยเป็นเมตร หากดัชนีมวลกายมีค่าระหว่าง 18.5 ถึง 25 แสดงว่าสมส่วน ถ้าน้อยกว่า 18.5 แสดงว่าผอม และถ้ามากกว่า 25 ก็แสดงว่าอ้วน

$$BMI = \frac{weight(kg.)}{height^2(m^2)}$$

เราจะมาเขียนโปรแกรมเพื่อแสดงดัชนีมวลกาย จากน้ำหนักและความสูงที่ผู้ใช้ป้อนทางแป้นพิมพ์ เพื่ออำนวยความสะดวกให้กับผู้ใช้ จะให้ผู้ใช้ป้อนความสูงซึ่งมีหน่วยเป็นเซนติเมตร (แทนที่จะเป็นเมตรตามที่แสดงในสูตรข้างบนนี้) โปรแกรมที่จะเขียนจึงต้องมีการแปลงหน่วยก่อนเข้าสูตร เขียนเป็นโปรแกรมสมบูรณ์ดังรหัสที่ 2-8 ให้สังเกตว่ามีการใช้ตัวแปรหลายตัว ซึ่งพอจำแนกการใช้ตัวแปรตามหน้าที่ดังนี้

- ใช้ตัวแปรเก็บข้อมูลขาเข้าที่รับจากผู้ใช้ (ตัวแปร `weight` และ `height`)
- ใช้ตัวแปรเก็บข้อมูลเสริมระหว่างการประมวลผล (ตัวแปร `kb` และ `hm`)
- ใช้ตัวแปรเก็บผลลัพธ์ เพื่อการแสดงผล (ตัวแปร `bmi`)

บรรทัดที่ 5 เริ่มด้วยการใช้ตัวแปร `kb` เก็บตัวอ่านแป้นพิมพ์ บรรทัดที่ 6 และ 7 แสดงข้อความเพื่อให้ผู้ใช้ทราบว่า ต้องป้อนน้ำหนัก แล้วรอรับน้ำหนักเก็บใส่ตัวแปร `weight` สองบรรทัดต่อมาทำในลักษณะเดียวกันเพื่อรับความสูงเก็บในตัวแปร `height` (โดยแจ้งให้ผู้ใช้ทราบว่า ให้ป้อนความสูงมีเป็นหน่วยเซนติเมตร) เนื่องจากการคำนวณดัชนีมวลกายต้องใช้ความสูงแบบเมตร จึงใช้ตัวแปรใหม่ `hm` ที่แปลง `height` ให้เป็นเมตรด้วยการหารด้วย 100 ในบรรทัดที่ 10 คำนวณดัชนีมวลกายตามสูตรในบรรทัดที่ 11 และปิดท้ายด้วยการแสดงผลลัพธ์ ต้องขอบอกว่าหลายคนที่เขียนคำสั่งเพื่อทำตามสูตร ไม่ทราบว่าจะคำนวณกำลังสองของความสูงได้อย่างไร โดยลืมไปว่าสามารถใช้การคูณความสูงด้วยความสูง ซึ่งได้ผลเช่นเดียวกัน นี่คือหนึ่งในปัญหาของการนำความรู้เกี่ยวกับคำสั่งและตัวดำเนินการคำนวณต่าง ๆ ที่รู้ ที่มีอยู่ มาประยุกต์ มาปรับใช้กับปัญหาที่กำลังหาทางแก้ไข

```
01 import java.util.Scanner;
02 // โปรแกรมคำนวณดัชนีมวลกาย
03 public class BodyMassIndex {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("น้ำหนัก (kg.) = ");
07     double weight = kb.nextDouble();
08     System.out.print("ความสูง (cm.) = ");
09     double height = kb.nextDouble();
10     double hm = height / 100.0;
11     double bmi = weight / (hm * hm);
12     System.out.println("ดัชนีมวลกาย = " + bmi);
13   }
14 }
```

ตัวแปรเก็บข้อมูลขาเข้า

ตัวแปรเก็บข้อมูลเสริมการประมวลผล

ตัวแปรเก็บผลลัพธ์

รหัสที่ 2-8 โปรแกรมคำนวณดัชนีมวลกาย

บรรทัดที่ 10 และ 11 อาจยุบเป็น `double bmi = weight / (height/100 * height/100)` ก็ได้ (อนึ่งเราเรียกการเขียนบรรยายการคำนวณนี้ว่า *นิพจน์คำนวณ* - arithmetic expression) หรือจะยุบบรรทัดที่ 10 ถึง 12 เป็น `System.out.println("ดัชนีมวลกาย"+(weight / (height/100 * height/100)))` ก็คงไม่ผิด แต่ขอให้เข้าใจด้วยว่าการใช้ตัวแปรเปลืองไม่ได้มีผลเสีย เมื่อแลกกับโปรแกรมที่อ่านง่ายกว่า ย่อมดีกว่า (ผู้อ่านควรใช้โอกาสนี้ลองเขียนโปรแกรมนี้ และสั่งทำงานจริง เพื่อเสริมความมั่นใจในการเขียนโปรแกรม)

คลาส Math และลำดับการดำเนินการ

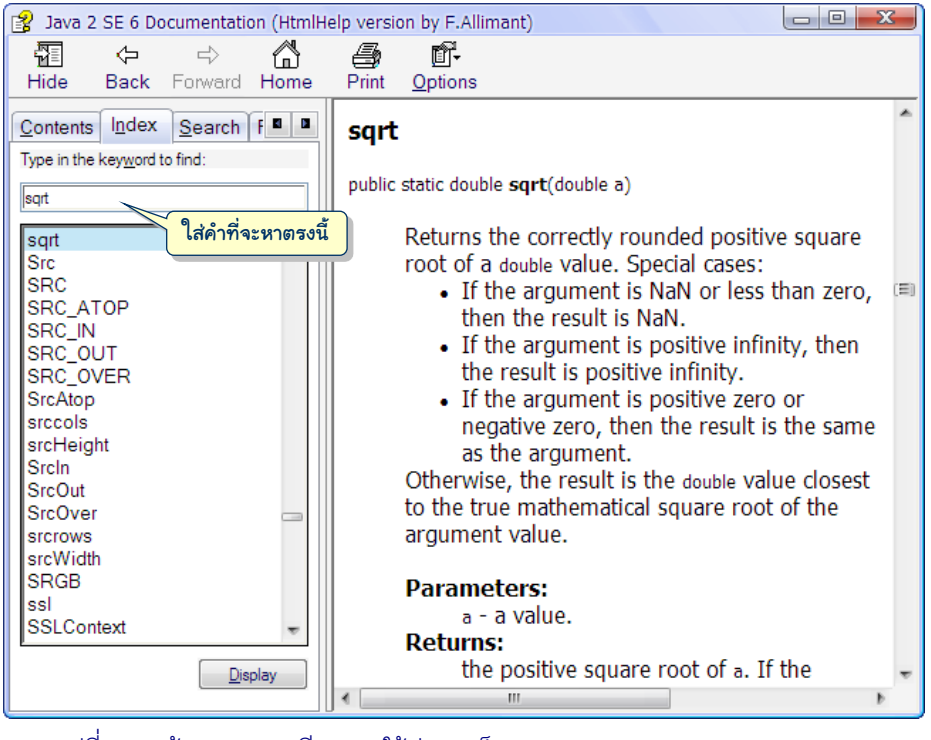
รากของสมการ $ax^2 + bx + c = 0$ หาได้ไม่ยากด้วยสูตร $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ จงเขียนโปรแกรมที่รับค่าของ a, b, และ c เพื่อแสดงรากทั้งสองค่าของสมการกำลังสอง

หลังจากพิจารณาสูตรข้างบนนี้ ก็ต้องถามตัวเองว่า เรารู้จักตัวดำเนินการที่ใช้ในสูตรหรือไม่ เครื่องหมาย ± บอกว่ารากมีสองค่า ค่าหนึ่งใช้ + อีกค่าใช้ – พจน์ b^2 ก็ใช้ b คูณ b ที่ยังไม่รู้ตอนนี้ก็คือการหารากที่สอง $\sqrt{\ }$ จาวามีคลาสมาตรฐานชื่อ `Math` ภายในบรรจุเมท็อดมากมายที่ให้บริการคำนวณฟังก์ชันทางคณิตศาสตร์ต่าง ๆ ตารางที่ 2-1 แสดงตัวอย่างบางเมท็อด พบว่ามีเมท็อดชื่อ `Math.sqrt` ซึ่งให้บริการหารากที่สองตามที่เราต้องการ (`sqrt` ย่อมาจาก square root) หากสนใจรายละเอียดของเมท็อดใดในคลาสมาตรฐานของจาวา ก็สามารถอ่านเพิ่มเติมได้ด้วยการกดปุ่ม F1 แล้วกรอกชื่อเมท็อดในช่องทางซ้าย [4] ดังตัวอย่างในรูปที่ 2-3 แสดงรายละเอียดของเมท็อด `Math.sqrt`

[4] อย่าลืมดาวน์โหลดและติดตั้ง Java API Help File ก่อนกดปุ่ม F1 (อ่านรายละเอียดการติดตั้งได้ในภาคผนวก ก)

ตารางที่ 2-1 ตัวอย่างเมท็อดให้บริการคำนวณฟังก์ชันทางคณิตศาสตร์ในคลาส `Math`

การเรียกใช้	ความหมาย
`Math.abs(x)`	คืนค่าสัมบูรณ์ของ x
`Math.sin(r), Math.cos(r)`	คืนค่า sine และ cosine ของมุม r (หน่วยเรเดียน)
`Math.log(x), Math.log10(x)`	คืนค่า log ฐาน e และฐาน 10 ของ x
`Math.max(x,y), Math.min(x,y)`	คืนค่ามากกว่า และน้อยกว่า ระหว่าง x กับ y
`Math.random()`	คืนค่าจำนวนจริงสุ่มในช่วง [0, 1)
`Math.sqrt(x)`	คืนค่ารากที่สองของ x
`Math.toRadians(d)`	คืนค่ามุมแบบเรเดียนของมุมแบบองศา d
`Math.toDegrees(r)`	คืนค่ามุมแบบองศาของมุมแบบเรเดียน r



รูปที่ 2-3 หน้าจอรายละเอียดการใช้อ่านเมท็อด `Math.sqrt` ของ Java help file

คราวนี้ก็พร้อมเขียนโปรแกรมกัน ดังแสดงในรหัสที่ 2-9 เริ่มด้วยการสร้างตัวอ่านแป้นพิมพ์ตามด้วยการรอรับค่า `a`, `b`, และ `c` จากผู้ใช้ (บรรทัดที่ 5 ถึง 11) เก็บใส่ตัวแปรแบบ `double` จากนั้นใช้สูตรคำนวณรากทั้งสองเก็บใส่ตัวแปร `r1` และ `r2` (ซึ่งก็ต้องเป็น `double` เช่นกัน) ปิดท้ายด้วยการแสดงผล ความยากของตัวอย่างนี้อยู่ที่การเปลี่ยนสูตรหารากไปเป็นนิพจน์คำนวณ

ด้วยคำสั่งและตัวดำเนินการของจาวา หลายคนอาจรีบร้อนเปลี่ยนสูตร $\frac{-b+\sqrt{b^2-4ac}}{2a}$ เป็น `-b + Math.sqrt(b*b - 4*a*c) / 2*a` ซึ่งแลดูคล้ายสูตร แต่กลับทำงานผิด เพราะแบบนี้คือ $-b+\frac{\sqrt{b^2-4ac}}{2}a$ จะเขียนให้ถูก ต้องรู้ลำดับการดำเนินการว่า ตัวดำเนินการตัวใดทำก่อน ตัวใดทำหลัง หากไม่รู้ ไม่แน่ใจ หรือจำไม่ได้ (ตอนนี้ก็ไม่น่าจะรู้ เพราะยังไม่ได้นำเสนอ) ให้ใช้วงเล็บเพื่อกำหนดลำดับการดำเนินการ ดังนี้

$$\frac{-b+\sqrt{b^2-4ac}}{2a} = (-b+\sqrt{b^2-4ac}\ ) \ / \ (2a)$$
$$= ((-b)+\sqrt{(b^2)-(4ac)}\ ) \ / \ (2a)$$

จากนั้นค่อยเปลี่ยนเป็น `((-b) + Math.sqrt((b*b)-(4*a*c))) / (2*a)`

```
01 import java.util.Scanner;
02 // โปรแกรมหารากของสมการ ax^2 + bx + c = 0
03 public class QuadraticRoot {
04  public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("a = ");
07     double a = kb.nextDouble();
08     System.out.print("b = ");
09     double b = kb.nextDouble();
10     System.out.print("c = ");
11     double c = kb.nextDouble();
12     double r1 = ((-b) + Math.sqrt((b*b) - (4*a*c))) / (2*a);
13     double r2 = ((-b) - Math.sqrt((b*b) - (4*a*c))) / (2*a);
14     System.out.print("รากของสมการคือ ");
15     System.out.println(r1 + " และ " + r2);
16   }
17 }
```

คำนวณตามสูตร $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$

รหัสที่ 2-9 โปรแกรมหารากของสมการกำลังสอง $ax^2 + bx + c = 0$

ลำดับการคำนวณ

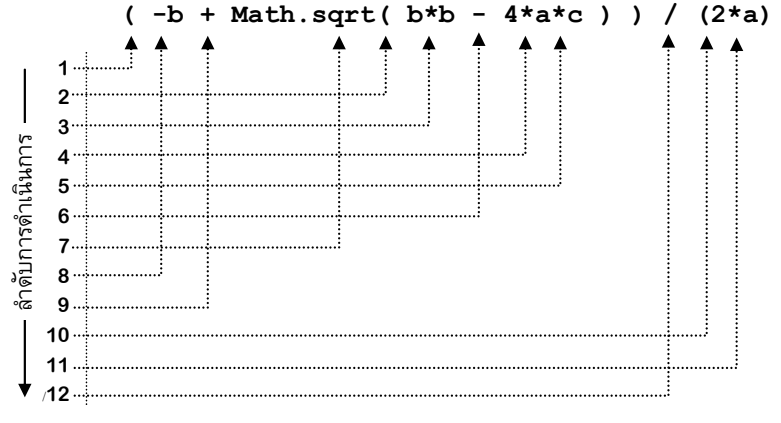
หากผู้อ่านใช้เครื่องคิดเลขคำนวณค่าของ 2+3×4 โดยกดปุ่มเครื่องคิดเลขตามลำดับที่แสดง จะได้คำตอบเป็น 20 หรือ 14 ? หากได้ 20 แสดงว่าเครื่องที่ใช้ทำบวกก่อนแล้วค่อยคูณ แต่ถ้าได้ 14 แสดงว่าทำคูณแล้วค่อยบวก จะได้ผลเท่าไร ขึ้นกับยี่ห้อเครื่องคิดเลข [5] ในกรณีที่เขียนเป็นคำสั่งในโปรแกรม คงหวังให้คำตอบขึ้นกับยี่ห้อเครื่องไม่ได้ ภาษาคอมพิวเตอร์จึงกำหนดลำดับการ

[5] ผู้เขียนใช้เครื่องคิดเลขยี่ห้อ Citizen รุ่น CT-600 คำนวณ 2+3×4 ได้ 20 แต่ถ้าใช้ Sharp รุ่น EL-515s ได้ 14

ดำเนินการก่อนหลังของตัวดำเนินการอย่างชัดเจนว่า `2+3*4` จะได้ผลเท่าไร ภาษาจาวากำหนดลำดับการดำเนินการจากความสำคัญมากไปน้อยไว้ดังนี้ (สำคัญมากจะทำก่อน)

1. คำนวณในวงเล็บ
2. เรียกใช้เมท็อด
3. ติดลบ
4. คูณ หาร
5. บวก ลบ
6. การให้ค่า (ด้วยตัวดำเนินการ =)

ในกรณีที่ตัวดำเนินการมีความสำคัญเท่ากันอยู่หลายตัว จะเลือกทำตัวทางซ้ายก่อนตัวทางขวา (ยกเว้นการติดลบและการให้ค่า) ดังนั้น นิพจน์คำนวณ `(-b + Math.sqrt(b*b - 4*a*c)) / (2*a)` มีลำดับการทำงานดังแสดงในรูปที่ 2-4 (ถึงตรงนี้รู้ไหมว่า จาวาคำนวณ `2+3*4` ได้ค่าเท่าไร)



รูปที่ 2-4 ตัวอย่างแสดงลำดับการทำงานของตัวดำเนินการ (ตัวเลขทางซ้ายแสดงลำดับการทำงาน)

เมื่อรู้ความสำคัญของตัวดำเนินการ ผนวกกับการใช้ตัวแปรเสริมเก็บการคำนวณที่ต้องทำซ้ำซ้อน เราสามารถปรับปรุงบรรทัดที่ 12 และ 13 ของรหัสที่ 2-9 ด้วยรหัสที่ 2-10 โดยสร้างตัวแปร `tmp` เก็บค่า $\sqrt{b^2-4ac}$ แล้วใช้ซ้ำในการคำนวณรากทั้งสอง

```
    double tmp = Math.sqrt(b*b - 4*a*c);
    double r1 = (-b + tmp) / (2*a);
    double r2 = (-b - tmp) / (2*a);
```

รหัสที่ 2-10 การใช้ตัวแปรช่วยเก็บค่าชั่วคราวที่จะใช้ซ้ำระหว่างการคำนวณ

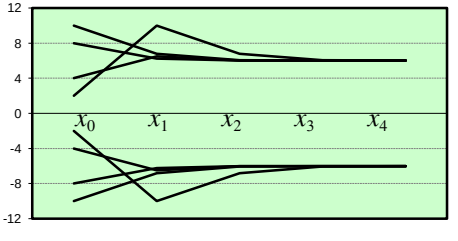
การลดจำนวนตัวแปร

เมื่อไรที่ต้องการหารากที่สองของ a ก็ใช้เมท็อด `Math.sqrt(a)` แต่หัวข้อนี้ขอนำเสนอวิธีการหารากที่สองที่ใช้กันมานานนับพันปีของชาวบาบิโลน (Babylonian method) ถึงแม้จะไม่ใช่วิธีที่ดีสุด แต่ขอใช้เป็นตัวอย่างเพื่อแสดงรูปแบบการใช้ตัวแปรในการเขียนโปรแกรม

กำหนดให้ $x_0 \approx \sqrt{a}$ และ $x_k = \frac{1}{2}\left(x_{k-1} + \frac{a}{x_{k-1}} \right)$ จะได้ว่า $\sqrt{a} = \lim_{k\to\infty} x_k$ หมายความว่า ถ้า x คือค่าประมาณของ $\sqrt{a}$ จะได้ $(x + a / x) / 2$ เป็นค่าประมาณของ $\sqrt{a}$ ที่แม่นยำกว่า x ดังนั้นหากต้องการหา $\sqrt{a}$ ก็ให้เริ่มต้นด้วยการเดาค่าของ $\sqrt{a}$ แล้วใช้สูตรข้างต้นคำนวณค่าประมาณที่แม่นยำขึ้นเรื่อย ๆ เช่น ต้องการหารากที่สองของ 36

- $a = 36$, ให้ $x_0 = 1$
- $x_1 = (x_0 + a/x_0) / 2 = (1 + 36/1)/2 = 18.5$
- $x_2 = (x_1 + a/x_1) / 2 = (18.5 + 36/18.5)/2 = 10.22$
- $x_3 = (x_2 + a/x_2) / 2 = (10.22 + 36/10.22)/2 = 6.87$
- $x_4 = (x_3 + a/x_3) / 2 = (6.87 + 36/6.87)/2 = 6.06$

เราจะต้องคำนวณซ้ำกี่ครั้งก็ขึ้นกับค่าเริ่มต้น x_0 รูปที่ 2-5 แสดงการเปลี่ยนแปลงของ x_k ระหว่างการหาค่า $\sqrt{36}$ เมื่อตั้งค่าเริ่มต้น x_0 ต่างกัน (10, 8, 4, 2, -2, -4, -8, และ -10) ซึ่งใช้จำนวนรอบในการคำนวณต่างกันกว่าจะได้คำตอบเป็น 6 (หรือ –6)



รูปที่ 2-5 กราฟเส้นแสดงการเปลี่ยนแปลงของ x_k ระหว่างการหา $\sqrt{36}$ ด้วยค่า x_0 ที่ต่างกัน

รหัสที่ 2-11 แสดงโปรแกรมหารากที่สองของจำนวนที่ผู้ใช้ป้อนทางแป้นพิมพ์ โปรแกรมนี้ตั้งค่าเริ่มต้น `x0` เป็น 1 (บรรทัดที่ 8) จากนั้นคำนวณหา `x1` จาก `x0` (บรรทัดที่ 9) คำนวณหา `x2` จาก `x1` (บรรทัดที่ 10) คำนวณซ้ำในลักษณะเดียวกันได้ `x3` และ `x4` แล้วแสดงผลทางจอภาพ เราตัดสินใจคำนวณซ้ำ 4 ครั้ง ซึ่งหวังว่าจะพอ ถ้าคิดว่าไม่พอ ก็เพิ่มคำสั่งเพื่อคำนวณซ้ำให้แม่นยำมากขึ้น ๆ [6]

[6] การเพิ่มคำสั่งเพื่อการทำซ้ำในลักษณะนี้ ไม่ใช่วิธีที่ดีนัก ในบทถัด ๆ ไป จะนำเสนอวิธีที่ดีกว่า

เราจะพยายามลดจำนวนการใช้ตัวแปร x0, x1, ..., x4 ในรหัสที่ 2-11 ถ้าสังเกตให้ดีจะพบว่า หลังได้ค่า x1 แล้ว x0 ก็ไม่ได้ถูกใช้อีก หลังได้ค่า x2 แล้ว x1 ก็ไม่ได้ถูกใช้อีก เป็นเช่นนี้กับ x อื่น ๆ จะขอปรับปรุงให้ใช้แค่สองตัวแปรคือ x0 กับ x1 โดยให้ x0 มีความหมายเป็นค่าประมาณของ $\sqrt{a}$ ตัวเดิม และ x1 เป็นค่าประมาณของ $\sqrt{a}$ ตัวใหม่ หลังจากได้ x1 แล้วต้องการคำนวณตัวใหม่อีก ก็ให้ตั้ง x0 = x1 ก่อน แล้วค่อยคำนวณ x1 = (x0 + a/x0)/2 กระทำเช่นนี้ไปได้เรื่อย ๆ โดยใช้ตัวแปร x0 และ x1 เพียงสองตัว ดังแสดงในรหัสที่ 2-12

```
01 import java.util.Scanner;
02 // โปรแกรมหารากที่สองของจำนวนจริงด้วยวิธีของชาวบาบิโลน
03 public class SquareRoot {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("a = ");
07     double a = kb.nextDouble();
08     double x0 = 1; // เริ่มที่ค่าใดก็ได้ที่ไม่ใช่ 0
09     double x1 = (x0 + a/x0) / 2.0;
10     double x2 = (x1 + a/x1) / 2.0;
11     double x3 = (x2 + a/x2) / 2.0;
12     double x4 = (x3 + a/x3) / 2.0;
13     System.out.println("รากที่สองของ " + a + " = " + x4);
14   }
15 }
```

คำนวณตาม Babylonian method

$$x_k = \frac{1}{2}\left(x_{k-1} + \frac{a}{x_{k-1}}\right)$$

รหัสที่ 2-11 โปรแกรมหารากที่สองของจำนวนจริง

```
01 import java.util.Scanner;
02 // โปรแกรมหารากที่สองของจำนวนจริงด้วยวิธีของชาวบาบิโลน
03 public class SquareRoot {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("a = ");
07     double a = kb.nextDouble();
08     double x0 = 1, x1;
09     x1 = (x0 + a/x0) / 2.0;
10     x0 = x1;
11     x1 = (x0 + a/x0) / 2.0;
12     x0 = x1;
13     x1 = (x0 + a/x0) / 2.0;
14     x0 = x1;
15     x1 = (x0 + a/x0) / 2.0;
16     System.out.println("รากที่สองของ " + a + " = " + x1);
17   }
18 }
```

ปรับรหัสที่ 2-11 ให้ใช้ตัวแปรแค่ x0 กับ x1
x0 คือค่าประมาณ $\sqrt{a}$ ตัวเดิม
x1 คือค่าประมาณ $\sqrt{a}$ ตัวใหม่

รหัสที่ 2-12 โปรแกรมหารากที่สองของจำนวนจริง (ใช้ตัวแปร x0 กับ x1)

และถ้าสังเกตต่อจะพบว่า เราไม่จำเป็นต้องตั้งค่า x0 = x1 แล้วตามด้วยการคำนวณ x1 หรอก ทำไมเราไม่สลับบทบาทของ x0 และ x1 รอบแรกให้คำนวณ x1 จาก x0 รอบต่อมาก็ให้คำนวณ x0 จาก x1 รอบถัดไปก็กลับมาคำนวณ x1 จาก x0 ได้ดังรหัสที่ 2-13

ยัง ยังไม่หยุดเพียงเท่านี้ เราสามารถปรับปรุงต่อไปได้อีก คำสั่ง x1=(x0+a/x0)/2 บอกว่า ใช้ค่า x0 ไปคำนวณหาค่าให้ x1 นั่นคือ x0 คือค่าเดิม ส่วน x1 คือค่าใหม่ เมื่อเป็นเช่นนี้ ก็สามารถเขียนเป็น x = (x+a/x)/2 คือใช้ตัวแปรชื่อ x เพียงตัวเดียว เพราะ x ทางขวาของ = ก็คือ x ค่าเดิม เมื่อคำนวณ (x+a/x)/2 แล้ว ก็ตั้งค่าใหม่ให้กับ x ได้เลย (เพราะเราไม่สนใจค่าเก่าของ x อีกแล้ว) ได้ดังรหัสที่ 2-14

```
01 import java.util.Scanner;
02 // โปรแกรมหารากที่สองของจำนวนจริงด้วยวิธีของชาวบาบิโลน
03 public class SquareRoot {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("a = ");
07     double a = kb.nextDouble();
08     double x0 = 1, x1;
09     x1 = (x0 + a/x0) / 2.0;
10     x0 = (x1 + a/x1) / 2.0;
11     x1 = (x0 + a/x0) / 2.0;
12     x0 = (x1 + a/x1) / 2.0;
13     System.out.println("รากที่สองของ " + a + " = " + x0);
14   }
15 }
```

รหัสที่ 2-13 โปรแกรมหารากที่สองของจำนวนจริง (ใช้ตัวแปร x0 กับ x1) รุ่นปรับปรุง

```
01 import java.util.Scanner;
02 // โปรแกรมหารากที่สองของจำนวนจริงด้วยวิธีของชาวบาบิโลน
03 public class SquareRoot {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("a = ");
07     double a = kb.nextDouble();
08     double x = 1;
09     x = (x + a/x) / 2.0;
10     x = (x + a/x) / 2.0;
11     x = (x + a/x) / 2.0;
12     x = (x + a/x) / 2.0;
13     System.out.println("รากที่สองของ " + a + " = " + x);
14   }
15 }
```

x ทางขวาของ = คือค่าประมาณ $\sqrt{a}$ ตัวเดิม
x ทางซ้ายของ = คือค่าประมาณ $\sqrt{a}$ ตัวใหม่

รหัสที่ 2-14 โปรแกรมหารากที่สองของจำนวนจริง ปรับปรุงให้ใช้แค่ตัวแปร x ตัวเดียว

การทดสอบและการแก้จุดบกพร่อง

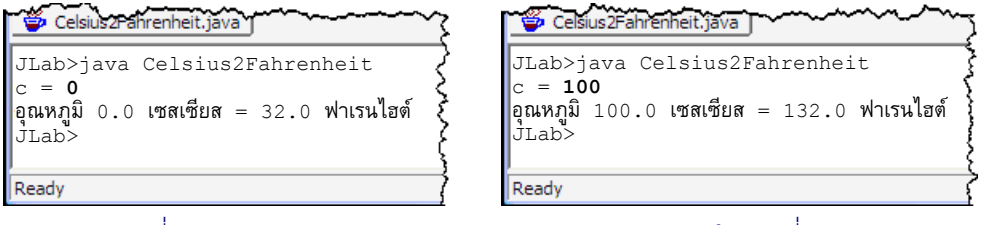
ขอปิดท้ายบทนี้อีกสักตัวอย่าง เป็นโปรแกรมง่าย ๆ แปลงอุณหภูมิที่รับทางแป้นพิมพ์จากเซลเซียสให้เป็นฟาเรนไฮต์ ด้วยสูตร $f = (9/5)c + 32$ เขียนได้ดังรหัสที่ 2-15

```
01 import java.util.Scanner;
02 // โปรแกรมแปลงอุณหภูมิจากเซลเซียสให้เป็นฟาเรนไฮต์
03 public class Celsius2Fahrenheit {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("c = ");
07     double c = kb.nextDouble();
08     double f = (9/5)*c + 32;
09     System.out.println("อุณหภูมิ " + c + " เซลเซียส = " + f + " ฟาเรนไฮต์");
10   }
11 }
```

แปลงองศาเซลเซียสเป็นฟาเรนไฮต์

รหัสที่ 2-15 โปรแกรมอุณหภูมิจากเซลเซียสให้เป็นฟาเรนไฮต์ (ที่ทำงานผิด)

เมื่อใดที่เขียนโปรแกรมเสร็จ ก็ต้องสั่งทำงานเพื่อทดสอบความถูกต้อง รหัสที่ 2-15 ผ่านด่านการแปลโปรแกรม ไม่พบสิ่งใดผิดพลาดทางไวยากรณ์ เมื่อทำงานแล้วลองใส่ค่า `c = 0` ได้ผลดังรูปที่ 2-6 ซ้าย ถ้าใส่ค่า `c = 100` จะได้ผลดังรูปที่ 2-6 ขวา ผู้อ่านพบสิ่งผิดปกติหรือไม่ อุณหภูมิ 0 °C เท่ากับ 32 °F นั้นถูกต้อง แต่อุณหภูมิ 100 °C ต้องเท่ากับ 212 °F ไม่ใช่ 132 °F ดังที่แสดงในรูปที่ 2-6 ขวา



รูปที่ 2-6 ตัวอย่างผลการทำงานของโปรแกรมแปลงอุณหภูมิในรหัสที่ 2-15

ข้อผิดพลาดแบบนี้ถือว่ารุนแรง รหัสต้นฉบับนั้นถูกต้องตามไวยากรณ์ของภาษา แต่ทำงานผิด ไม่ตรงตามวัตถุประสงค์ หากเราไม่ทดสอบอย่างถี่ถ้วน แล้วนำโปรแกรมไปใช้งานโดยไม่ทราบว่า โปรแกรมนั้นทำงานถูกบ้างผิดบ้าง ย่อมเกิดผลเสียต่อระบบงาน การทดสอบโปรแกรมจึงเป็นภาระสำคัญที่นักเขียนโปรแกรมจะต้องสร้างข้อมูลทดสอบ ป้อนให้โปรแกรมลองทำงาน และสำรวจผลลัพธ์ว่า เป็นไปดังที่คาดหวังหรือไม่ หากพบข้อผิดพลาด ก็ต้องหาจุดบกพร่องในรหัสต้นฉบับเพื่อแก้ไขข้อผิดพลาดนั้น

การหาและแก้จุดบกพร่องนั้นกระทำโดยการทำตัวเองเสมือนเป็นเครื่องคอมพิวเตอร์ ที่รับข้อมูลทดสอบซึ่งก่อให้เกิดข้อผิดพลาด แล้วลองติดตามการทำงานของโปรแกรมทีละคำสั่ง ๆ

สังเกตว่า ตัวแปรต่าง ๆ มีการเปลี่ยนแปลงค่าเป็นไปตามที่คาดหวังหรือไม่ ซึ่งต้องใช้ทักษะ สมาธิ และความอดทนเพื่อหาและแก้จุดบกพร่องของโปรแกรม ที่อาจไม่ได้มีเพียงจุดเดียว การเขียนโปรแกรม การสั่งทำงานด้วยข้อมูลทดสอบ และการแก้จุดบกพร่อง จึงเป็นวงจรที่ต้องทำซ้ำ ๆ ระหว่างการพัฒนาโปรแกรมจนกว่าจะมั่นใจว่า โปรแกรมไร้ข้อผิดพลาด พร้อมนำไปใช้งานจริง [7]

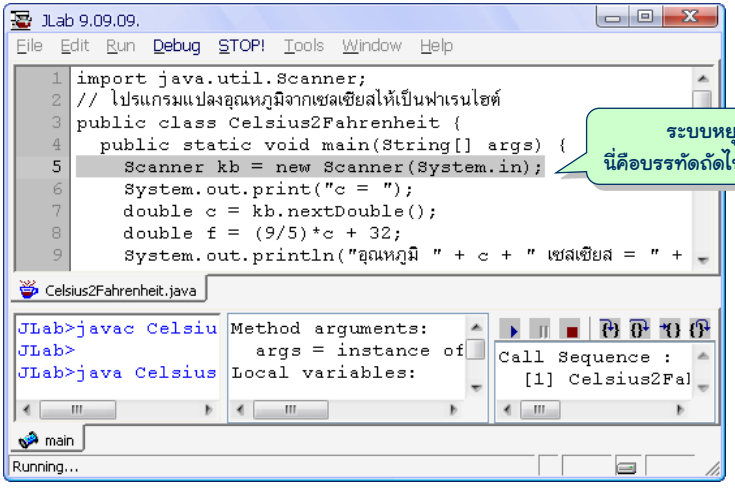
การแก้จุดบกพร่องที่เกิดในโปรแกรมเล็ก ๆ ไม่ซับซ้อนนัก อาจอาศัยการมอง การจำ และการนึกคิดของนักเขียนโปรแกรม เพื่อติดตามการทำงานของโปรแกรมได้ แต่หากโปรแกรมใหญ่และซับซ้อนขึ้น การใช้ซอฟต์แวร์ที่เรียกว่า debugger ช่วยติดตามการทำงาน จะช่วยหาจุดบกพร่องได้สะดวกขึ้น สำหรับข้อผิดพลาดที่เกิดขึ้นกับรหัสที่ 2-15 นั้นสามารถดูได้ด้วยตา โดยสังเกตทีละบรรทัดว่า ผิดตอนรับข้อมูลในบรรทัดที่ 7 หรือผิดตอนคำนวณค่า `f` ในบรรทัดที่ 8 หรือผิดตอนแสดงผลในบรรทัดที่ 9 ถึงแม้โปรแกรมนี้จะสั้น จะขอลองใช้ debugger สั่งโปรแกรมทำงานทีละบรรทัดกันดูเป็นตัวอย่าง

ใน JLab เราใช้เมนู Debug → Start Debugging หรือกดปุ่ม Ctrl+F5 [8] เพื่อเรียก debugger ทำให้วินโดว์เปลี่ยนเป็นดังรูปที่ 2-7 (ก) สังเกตส่วนล่างของวินโดว์แบ่งเป็นสามช่องย่อย ช่องซ้ายคือส่วนแสดงผลจากโปรแกรม ช่องกลางแสดงค่าของตัวแปรระหว่างการทำงาน และช่องขวาแสดงรายการของเมท็อดที่ถูกเรียก สำหรับส่วนบนนั้นเหมือนเดิม คือแสดงตัวโปรแกรม โดยการทำงานของโปรแกรมจะหยุดรออยู่ที่บรรทัดที่มีสีพื้นเข้ม (ในรูป (ก) คือบรรทัดที่ 5 ซึ่งเป็นบรรทัดเริ่มต้นของเมท็อด `main`) เมื่อต้องการสั่งให้บรรทัดที่รออยู่นี้ทำงานต่อ ก็กดปุ่ม Shift+F8 บรรทัดที่มีสีพื้นเข้มจะเลื่อนลงมาอีกหนึ่งบรรทัด (รูป (ข)) เรียกการทำงานเช่นนี้ว่า single step, รูป (ค) แสดงผลเมื่อกดให้โปรแกรมทำอีกบรรทัด บรรทัดที่ 6 สั่งแสดง `"c = "` จึงปรากฏข้อความ `c =` ทางซ้ายของส่วนล่างของวินโดว์ สั่งทำอีกบรรทัด โปรแกรมจะรอรับจำนวนทางแป้นพิมพ์ รูป (ง) แสดงผลหลังจากผู้ใช้ป้อนค่า 100 และกดปุ่ม ENTER ให้สังเกตช่องกลางซึ่งแสดงค่า `c = 100` ตรงช่องนี้เองที่เราต้องคอยสังเกตค่าของตัวแปรระหว่างการทำงานว่าเป็นไปตามที่คาดหรือไม่ (สำหรับกรณีนี้ `c` มีค่าถูกต้อง ตรงตามที่เราป้อนคือ 100) สั่งทำงานต่ออีกหนึ่งบรรทัด ซึ่งคือคำสั่งแปลง °C เป็น °F ได้ดังรูป (จ) และเราก็เห็นได้ชัดว่า `f = 132` ไม่ตรงกับค่า 212 ที่ควรจะเป็น ก็น่าสงสัยว่า บรรทัดที่ 8 คือบรรทัดที่คงต้องมีจุดบกพร่องแน่ (ถึงตรงนี้ หากต้องการเลิกตัว debugger ก็ให้เลือกเมนู Debug → End Debugging หรือจะใช้เมาส์กดปุ่มสี่เหลี่ยมที่แสดงในรูป (จ) ก็ได้)

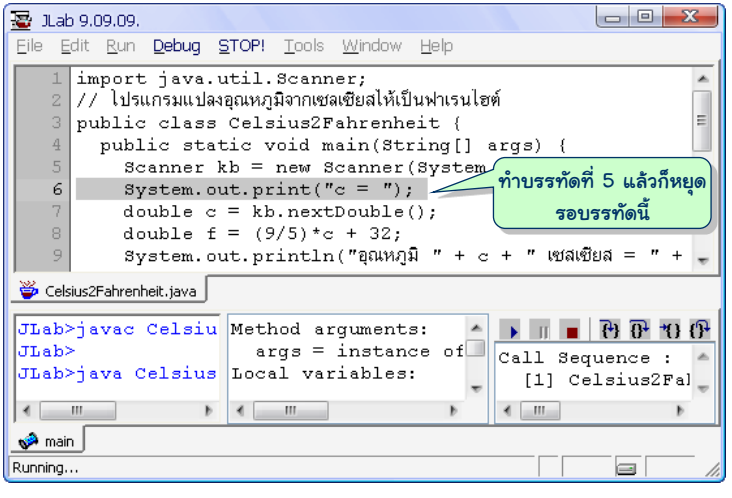
---

[7] ในวงการคอมพิวเตอร์เรียกจุด*บกพร่อง*ของโปรแกรมว่า “bug” (แปลว่าแมลง) และเรียกการแก้จุดบกพร่องว่า debugging โดยนักคอมพิวเตอร์หญิงชื่อ Grace Hopper เป็นผู้ใช้คำว่า bug ครั้งแรกในวงการคอมพิวเตอร์ เมื่อปี ค.ศ. 1946 หลังจากที่พบแมลงตัวหนึ่งตายในเครื่องคอมพิวเตอร์ ทำให้การทำงานของเครื่องผิดพลาด (อนึ่ง การใช้คำว่า bug เพื่อเรียกจุดบกพร่อง มีมาตั้งแต่คริสต์ศตวรรษที่ 19 ก่อนที่จะมีคอมพิวเตอร์)

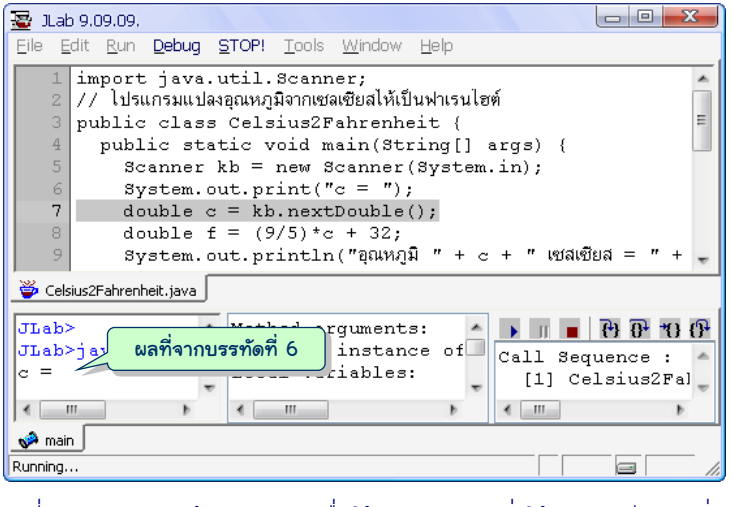
[8] เครื่องหมายที่แสดงนี้ คือให้กดปุ่ม Ctrl ค้างไว้ แล้วกดปุ่ม F5 ตาม



(ก)

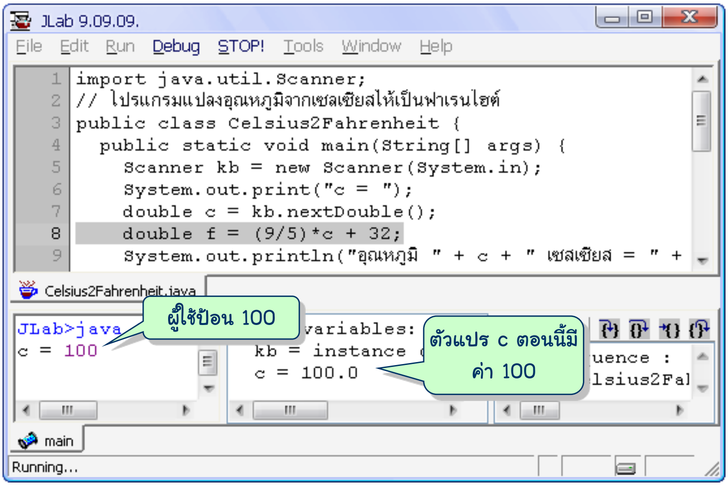


(ข)

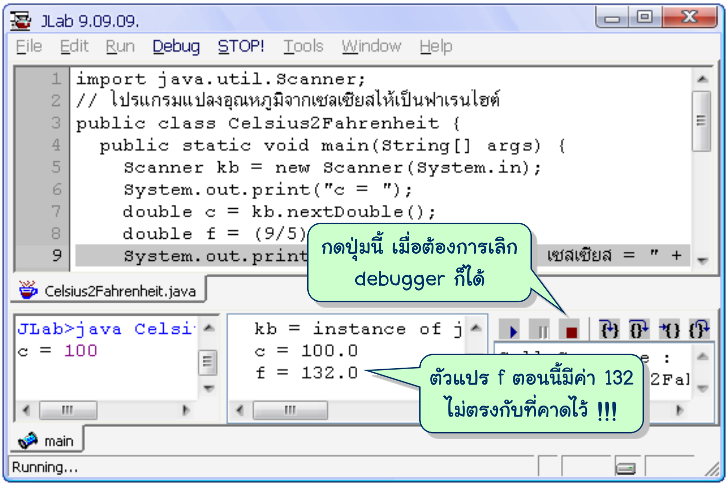


(ค)

รูปที่ 2-7 ตัวอย่างหน้าจอ JLab เมื่อใช้ debugger สั่งให้ทำงานทีละคำสั่ง



(ง)



(จ)

รูปที่ 2-7 ตัวอย่างหน้าจอ JLab เมื่อใช้ debugger สั่งให้ทำงานทีละคำสั่ง (ต่อ)

เมื่อค้นพบบรรทัดที่น่าจะมีจุดบกพร่อง ก็ต้องมาพิจารณาในรายละเอียดของบรรทัดนั้นว่าอะไรคือสาเหตุที่แท้จริง บรรทัดที่ 8 คือคำสั่ง `double f = (9/5)*c+32;` ซึ่งก็ดูตรงกับสูตรที่ควรจะเป็น หาก `c=100` ก็น่าจะได้ `f=(9/5)*100+32 = 1.8*100+32 = 180+32 = 212` แล้วทำไมถึงผิด ? ถ้ายังไม่รู้ ก็ควรต้องลองเปลี่ยนการเขียนคำสั่งดู เช่น ลองเปลี่ยนเป็น `f = 9/5*c+32` หรือ `f = 9*c/5+32` หรือ `f = 32+9/5*c` หรือ `f = 32+9*c/5` เป็นต้น ผู้อ่านอาจจะรู้สึกแปลก นิพจน์ที่เขียนมานี้น่าจะเหมือนกันทั้งหมด ลองไปก็เท่านั้น อยากให้ผู้อ่านลองเปลี่ยน แล้วสั่งทำงานดู ก็จะรู้ว่าได้ผลเหมือนกันหรือไม่ และสาเหตุที่แท้จริงคืออะไร [9] [10]

---

[9] (9/5)*c+32 [illegible] (9/5) [illegible]

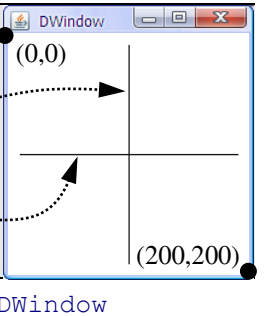
[10] วิธีที่อ่านเฉลยข้างบนนี้ทำได้โดยใช้กระจกเงา (หรือแผ่นซีดี) วางตรงเส้น ก็สามารถอ่านเฉลยได้ในกระจก

DWindow

JLab มีคลาสพิเศษชื่อเต็มว่า `jlab.graphics.DWindow` เพื่อช่วยให้เขียนโปรแกรมวาดรูป แสดงภาพกราฟิกเคลื่อนไหว หรือประมวลผลรูปภาพได้ง่ายขึ้น เราจะเขียนโปรแกรมที่ใช้ `DWindow` นี้เป็นตัวช่วย หัวข้อนี้นำเสนอตัวอย่างง่ายๆ กันก่อน แล้วค่อย ๆ เพิ่มความซับซ้อนขึ้นในบทถัด ๆ ไป

เริ่มด้วยตัวอย่างในรหัสที่ 2-16 บรรทัดที่ 5 ประกาศตัวแปร `w` ซึ่งมีไว้เก็บข้อมูลประเภท `DWindow` (ซึ่งเรา `import` ชื่อเต็มในบรรทัดที่ 1) คำสั่ง `new DWindow(200,200)` คือการสร้างวินโดว์ขนาดกว้าง 200 สูง 200 จุดภาพ (ขนาด 200×200 นี้เป็นขนาดของพื้นที่ภายในวินโดว์ไม่รวมกรอบและหัววินโดว์ด้านบน) เมื่อคำสั่งนี้ทำงาน จะปรากฏวินโดว์บนจอภาพทันที บรรทัดที่ 6 และ 7 ลากเส้นแนวตั้งและแนวนอน ด้วยคำสั่ง `w.drawLine(`x_0`,`y_0`,`x_1`,`y_1`)` เพื่อลากเส้นในวินโดว์ `w` เริ่มจากจุด (x_0, y_0) ไปยังจุด (x_1, y_1) ระบบพิกัดของวินโดว์จะแปลกเล็กน้อย มุมซ้ายบนของพื้นที่ในวินโดว์มีพิกัด $(x, y) = (0,0)$ ค่า x เพิ่มขึ้นเมื่อไปทางขวา และค่า y เพิ่มขึ้นเมื่อลงล่าง คำสั่งของบรรทัดที่ 6 จึงลากเส้นจากจุด (100,10) ไปยังจุด (100,190) ซึ่งเป็นเส้นแนวตั้ง ส่วนบรรทัดที่ 7 ลากเส้นจากจุด (10,100) ไปยังจุด (190,100) ซึ่งเป็นเส้นแนวนอน

```
01 import jlab.graphics.DWindow;
02
03 public class LineDrawing {
04   public static void main(String[] args) {
05     DWindow w = new DWindow(200, 200);
06     w.drawLine(100, 10, 100, 190);
07     w.drawLine(10, 100, 190, 100);
08   }
09 }
```



รหัสที่ 2-16 โปรแกรมลากเส้นโดยเมท็อด `drawLine` ของ `DWindow`

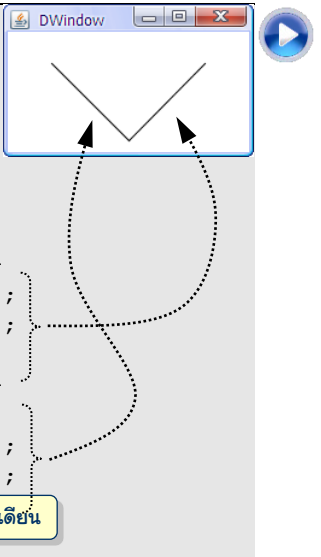
ขอปรับปรุงรหัสที่ 2-16 ให้เขียนแบบสามารถเปลี่ยนแปลงได้ง่ายขึ้น รหัสที่ 2-16 วาดเส้นที่เว้นช่องว่างจากขอบไว้ 10 หากต้องการเปลี่ยนเป็น 20 ก็ต้องแก้ไขค่าคงตัวหลายที่ในโปรแกรม จึงขอปรับเปลี่ยนดังรหัสที่ 2-17 โดยประกาศตัวแปร 3 ตัวคือ `width` กับ `height` ไว้เก็บขนาดของวินโดว์ และ `gap` ไว้เก็บช่องว่างที่เว้นจากขอบก่อนลากเส้น จากนั้นปรับคำสั่งในบรรทัดที่ 6 ถึง 8 ให้ใช้ตัวแปรทั้งสาม ด้วยการเขียนโปรแกรมในลักษณะนี้ การเปลี่ยนขนาดของวินโดว์และช่องว่างที่ขอบ สามารถกระทำที่บรรทัดที่ 5 เท่านั้นก็พอ

คราวนี้ขอเขียนใหม่อีกแบบ คือให้ลากเส้นโดยกำหนดจุดเริ่ม `(`x_0`,`y_0`)` ความยาวเส้น `length` และมุมของเส้น `angle` (กับเส้นแนวนอน) จึงต้องคำนวณจุดปลาย `(`x_1`,`y_1`)` ดังแสดงในรูปที่ 2-8 นำแนวคิดนี้เขียนโปรแกรมลากสองเส้นเอียง 45° และ 135° ได้ดังรหัสที่ 2-18

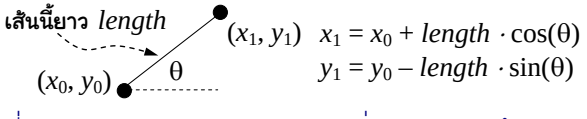
```
01 import jlab.graphics.DWindow;
02
03 public class LineDrawing {
04   public static void main(String[] args) {
05     double width = 200, height = 200, gap = 10;
06     DWindow w = new DWindow(width, height);
07     w.drawLine(width / 2, gap, width / 2, height - gap);
08     w.drawLine(gap, height / 2, width - gap, height / 2);
09   }
10 }
```

รหัสที่ 2-17 โปรแกรมลากเส้นที่ปรับปรุงจากรหัสที่ 2-16 ที่ใช้ตัวแปรเสริม

```
01 import jlab.graphics.DWindow;
02
03 public class LineDrawing {
04   public static void main(String[] args) {
05     int width = 200, height = 100, len = 90;
06     DWindow w = new DWindow(width, height);
07     double x0 = width / 2, y0 = height - 10;
08     double x1, y1, angle = 45, delta = 90;
09     //-------------------------------------------
10     x1 = x0 + len * Math.cos(Math.toRadians(angle));
11     y1 = y0 - len * Math.sin(Math.toRadians(angle));
12     w.drawLine(x0, y0, x1, y1);
13     //-------------------------------------------
14     angle = angle + delta;
15     x1 = x0 + len * Math.cos(Math.toRadians(angle));
16     y1 = y0 - len * Math.sin(Math.toRadians(angle));
17     w.drawLine(x0, y0, x1, y1);
18   }
19 }
```

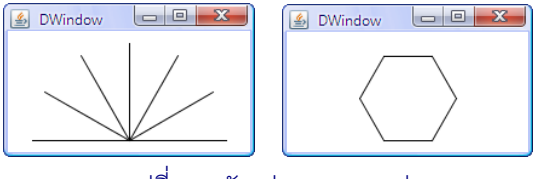


รหัสที่ 2-18 โปรแกรมลากเส้นที่กำหนดด้วยจุดเริ่ม ความยาวเส้น และมุม



รูปที่ 2-8 การคำนวณจุดปลายจากจุดเริ่ม ความยาวเส้น และมุม

ขอผู้อ่านลองใช้แนวคิดการลากเส้นในรหัสที่ 2-18 เพื่อปรับปรุงโปรแกรมให้ได้รูปที่ 2-9



รูปที่ 2-9 ตัวอย่างการวาดรูป

ตัวดำเนินการเสริม

ตัวดำเนินการที่ได้นำเสนอมาเป็นตัวดำเนินการพื้นฐานที่ตรงกับความสามารถของฮาร์ดแวร์หน่วยประมวลผล ทำให้ทำงานได้รวดเร็ว ในกรณีของการประมวลผลที่ซับซ้อนขึ้น จาวามีคลาสมาตรฐาน `Math` ที่มีบริการเสริมให้เรียกใช้ นอกจากนี้จาวายังมีตัวดำเนินการเพิ่มเติมให้เขียนคำสั่งได้กะทัดรัด เหมาะกับรูปแบบคำสั่งที่พบบ่อยมากในการเขียนโปรแกรม หัวข้อย่อยต่อไปนี้นำเสนอตัวดำเนินการเสริมเหล่านี้ สำหรับตัวอย่างการใช้งานจะได้นำเสนอในบทถัด ๆ ไป

เศษจากการหาร

`(int)(a/b)` คือผลของ `a/b` แล้วปัดเศษทิ้ง ถ้าต้องการหาเศษของ `a/b` สามารถใช้นิพจน์ `a - b*(int)(a/b)` หรือจะใช้ตัวดำเนินการ*มอดุโล* (modulo) `%` โดยที่ `a % b` มีค่าเท่ากับเศษของการหาร `a` ด้วย `b` ซึ่งใช้ได้กับทั้งจำนวนเต็มและจำนวนจริง เช่น `(5%2)` ได้ `1`, `(5.5%2.0)` ได้ `1.5`, `(5.5%1.0)` ได้ `0.5` เป็นต้น

บวกบวก ลบลบ

จาวามีตัวดำเนินการพิเศษใช้เพื่อเพิ่มค่าของตัวแปรอีกหนึ่ง (`++`) และลดค่าของตัวแปรลงหนึ่ง (`--`) ให้ `j` เป็นตัวแปรประเภทจำนวน `j++` หรือ `++j` จะทำให้ค่าของ `j` เพิ่มขึ้นหนึ่ง และ `j--` หรือ `--j` จะทำให้ค่าของ `j` ลดลงหนึ่ง เราเขียนตัวดำเนินการนี้กับตัวแปรเท่านั้น จะเขียนกับนิพจน์คำนวณไม่ได้ เช่น `++(x+1)` หรือ `(2*x)--` ไม่ได้ แต่สามารถเขียนเป็นส่วนหนึ่งของนิพจน์คำนวณได้ เช่น `2 + x++` หรือ `a + --k` ได้ โดย `++` `--` มีความสำคัญเช่นเดียวกับการติดลบ จึงทำก่อนการบวกลบคูณหาร สำหรับการนำตัวดำเนินการ `++` `--` วางไว้ข้างหน้าหรือข้างหลังตัวแปรนั้นมีความหมายต่างกันที่ควรจดจำดังนี้ (สมมติให้ `j = 4`)

- `++j` หมายถึงการเพิ่มค่าของ `j` อีกหนึ่ง ก่อนนำไปใช้ เช่น `2 + ++j` มีค่า `7`
- `j++` หมายถึงการนำค่าของ `j` ไปใช่ก่อนแล้วค่อยเพิ่มค่า `j` เช่น `2 + j++` มีค่า `6`
- `--j` หมายถึงการลดค่าของ `j` ลงหนึ่ง ก่อนนำไปใช้ เช่น `2 + --j` มีค่า `5`
- `j--` หมายถึงการนำค่าของ `j` ไปใช่ก่อนแล้วค่อยลดค่า `j` เช่น `2 + j--` มีค่า `6`

ตัวดำเนินการให้ค่า

สมมติเรามีตัวอ่านแป้นพิมพ์ `kb` และต้องการอ่านจำนวนจริงจากผู้ใช้เพิ่มเข้าตัวแปร ก็ใช้คำสั่ง `sum = sum+kb.nextDouble()` หรือเขียนสั้นๆ `sum += kb.nextDouble()` ก็ได้ จาวาให้เราเขียนแบบลัดในลักษณะนี้ได้กับตัวดำเนินการคำนวณได้ทุกตัว ดังตัวอย่างเช่น

- `x = x + (a + b)` เขียนได้เป็น `x += (a + b)`

- `x = x - (a + b)` เขียนได้เป็น `x -= (a + b)`
- `x = x * (a + b)` เขียนได้เป็น `x *= (a + b)`
- `x = x / (a + b)` เขียนได้เป็น `x /= (a + b)`
- `x = x % (a + b)` เขียนได้เป็น `x %= (a + b)`

เพิ่มเติม

byte, short, long, float

นอกจากจาวาจะมีข้อมูลแบบ `int` และ `double` เพื่อแทนจำนวนเต็มและจำนวนจริงแล้ว จาวายังมีประเภทข้อมูลแบบ `byte` `short` และ `long` สำหรับจำนวนเต็ม และแบบ `float` สำหรับจำนวนจริงอีกด้วย ตารางที่ 2-2 แสดงข้อมูลประเภทจำนวนในจาวา แต่ละแบบใช้เนื้อที่เก็บต่างกัน สามารถเก็บค่าในช่วงที่ต่างกัน จำนวนเต็มที่มีขนาดเล็กก็เก็บได้ช่วงแคบ ส่วนจำนวนจริงแบบ `float` เก็บช่วงแคบกว่า และเก็บค่าได้ละเอียดแม่นยำน้อยกว่าแบบ `double`

ตารางที่ 2-2 ข้อมูลประเภทจำนวนในภาษาจาวา

ประเภทข้อมูล	จำนวนไบต์ที่ใช้	ช่วงของค่าที่เก็บได้	
`byte`	1	-2^7 ถึง $2^7 - 1$	−128 ถึง 127
`short`	2	-2^{15} ถึง $2^{15} - 1$	≈ บวกลบสามหมื่น
`int`	4	-2^{31} ถึง $2^{31} - 1$	≈ บวกลบสองพันล้าน
`long`	8	-2^{63} ถึง $2^{63} - 1$	≈ บวกลบเก้าล้านล้านล้าน (เลข 19 หลัก)
`float`	4	$\pm 3.4 \times 10^{38}$	แม่นยำได้ประมาณ 6 ถึง 9 หลัก
`double`	8	$\pm 1.8 \times 10^{308}$	แม่นยำได้ประมาณ 15 ถึง 17 หลัก

```
01 public class NumericDataTypes {
02   public static void main(String[] args) {
03     byte b = 3;
04     short h = 7;
05     int i = 100;
06     long g = 1234567891234L;  // ค่าแบบ long ต้องปิดท้ายด้วยตัวแอล
07     float f = 1.5F;           // ค่าแบบ float ต้องปิดท้ายด้วยตัวเอฟ
08     double d = 2.1;
09     b = (byte)(b + i);  // byte+int เป็น int เก็บใส่ byte ต้องเปลี่ยน
10     h = (short)(h + i); // short+int เป็น int เก็บใส่ short ต้องเปลี่ยน
11     g = g + 1;          // long+int เป็น long เก็บใส่ long ได้เลย
12     f = (float)(f + d); // float+double เป็น double เก็บใส่ float ต้องเปลี่ยน
13   }
14 }
```

รหัสที่ 2-19 การใช้ `byte, short, long, float` มีเรื่องจุกจิก

โดยทั่วไปแนะนำให้ใช้ `int` เมื่อต้องการใช้จำนวนเต็ม และใช้ `double` เมื่อต้องการใช้จำนวนจริง เพราะว่าการใช้ประเภทอื่นนั้นมีเรื่องจุกจิก รหัสที่ 2-19 แสดงความจุกจิกเหล่านั้น เช่น หากต้องการเขียนค่าคงตัวแบบ `long` ต้องปิดท้ายด้วยตัวแอล เขียนค่าคงตัวแบบ `float` ต้องปิดท้ายด้วยตัวเอฟ เนื่องจาก `byte` และ `short` เก็บค่าในช่วงแคบกว่า `int` เมื่อผลการคำนวณได้ค่า `int` แล้วไปเก็บในตัวแปรที่เก็บค่าได้แคบกว่าก็ต้องบังคับให้เปลี่ยนประเภทข้อมูลก่อน (เหมือนตอนที่เราต้องเปลี่ยน `double` เป็น `int` ด้วยการใส่ `(int)` นำหน้าผลการคำนวณ) จึงขอย้ำอีกครั้งว่า ถ้าต้องใช้จำนวนเต็ม ก็ให้ใช้ `int` และถ้าต้องใช้จำนวนจริง ก็ให้ใช้ `double` จะสะดวกที่สุด จะเลือกใช้ `byte`, `short`, และ `float` ก็เมื่อต้องการประหยัดเนื้อที่หน่วยความจำ และเลือกใช้ `long` เมื่อต้องการเก็บจำนวนเต็มที่มีค่ามากจริง ๆ

จำนวนเต็มในระบบเลขฐานสอง

เครื่องคอมพิวเตอร์เก็บและประมวลผลจำนวนต่าง ๆ ในระบบเลขฐานสอง หน่วยที่เล็กสุดของข้อมูลในเครื่องคอมพิวเตอร์คือ*บิต* (bit – ย่อมาจาก binary digit) หนึ่งบิตเก็บข้อมูลได้สองค่า คือ 0 กับ 1 ขอทบทวนเรื่องระบบเลขฐานสักเล็กน้อย เพื่อความชัดเจนขอเขียน $(123)_{10}$ แทน 123 ที่เราเขียนกันในระบบเลขฐานสิบ มีค่าเท่ากับ $100 + 20 + 3 = 1\times10^2 + 2\times10^1 + 3\times10^0$ ถ้าเขียน $(101)_2$ ก็จะหมายถึงจำนวน 101 ในระบบเลขฐานสอง มีค่าเท่ากับ $1\times2^2 + 0\times2^1 + 1\times2^0 = 4 + 0 + 1 = (5)_{10}$ คำถามที่ตามมาคือ $(–5)_{10}$ จะแทนอย่างไร ?

จำนวนฐานสอง 1 บิตเก็บได้ 2 ค่าคือ 0 กับ 1, ถ้า 2 บิตเก็บได้ 4 ค่าคือ 00, 01, 10, และ 11, กล่าวในกรณีทั่วไปได้ว่า เลข n บิตเก็บได้ 2^n ค่า ถ้าเราแบ่งการเก็บจำนวนบวกให้พอดี ๆ กับจำนวนลบ ก็เก็บจำนวนบวกได้ 2^{n-1} จำนวน และจำนวนลบอีก 2^{n-1} จำนวน แต่เนื่องจากเราต้องเก็บ 0 ด้วย ก็ต้องสละไม่จำนวนบวกก็จำนวนลบสักตัวมาแทน 0 มีวิธีหนึ่งที่ได้รับความนิยมอย่างสูงในการแทนจำนวนฐานสองเรียกว่า แบบ*ส่วนเติมเต็มของสอง* (two's complement) กำหนดให้ $(b)_2 = (d)_{10}$ การแทน $(–d)_{10}$ ทำได้ด้วยการกลับบิตใน b จาก 1 เป็น 0, จาก 0 เป็น 1 แล้วบวกด้วย $(1)_2$ มาดูตัวอย่างกัน สมมติเราต้องการแทนจำนวนเต็มด้วยเลข 4 บิต จะได้ $(5)_{10} = (0101)_2$ ดังนั้น $(–5)_{10}$ จึงแทนได้ด้วยการกลับบิต $(0101)_2$ กลายเป็น $(1010)_2$ แล้วบวกด้วย $(1)_2$ ได้ $(1011)_2$ ตารางที่ 2-3 แสดงจำนวนฐานสองขนาด 4 บิต ที่แทนจำนวนเต็มตั้งแต่ –8 ถึง 7 หรือเขียนว่าตั้งแต่ $–2^3$ ถึง $2^3 – 1$ สรุปว่า หากใช้ขนาด n บิต ก็ย่อมแทนจำนวนเต็มตั้งแต่ $–2^{n-1}$ ถึง $2^{n-1} – 1$ ถ้ากลับไปดูตารางที่ 2-2 ก็คงจะทราบแล้วว่า ทำไมช่วงของจำนวนเต็มที่เก็บได้ของ `int`, `byte`, `short`, และ `long` จึงเป็นดังที่เขียนในตาราง

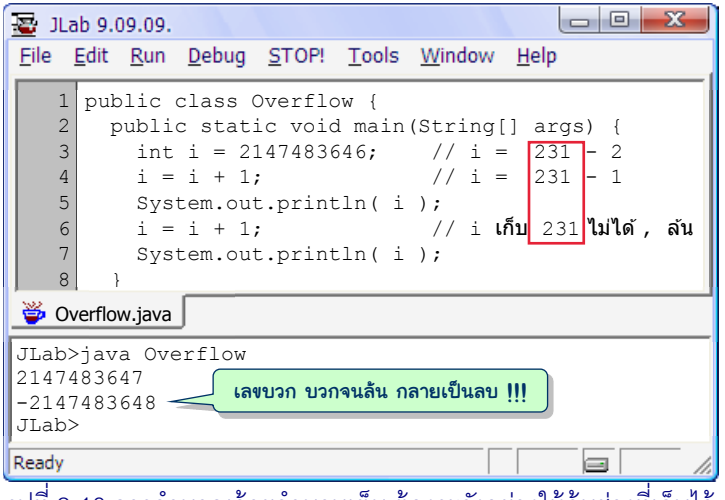
เพื่อให้การประมวลผลจำนวนเต็มเป็นไปอย่างมีประสิทธิภาพ การประมวลผลจำนวนเต็มจึงละเลยการตรวจสอบเมื่อข้อมูลอยู่นอกช่วงที่เก็บได้ รูปที่ 2-10 แสดงตัวอย่างกรณีที่ตัวแปร `i` มีค่ามากสุดๆ หลังเพิ่มค่าในบรรทัดที่ 4 เมื่อไปเพิ่มอีกครั้งในบรรทัดที่ 6 กลับได้ค่าลบ หากกลับไปดูใน

ตารางที่ 2-3 จะเห็นว่ากรณี 4 บิต $(7)_{10} = (0111)_2$ ถ้าให้เพิ่มอีก 1 ถ้าไม่ตรวจสอบอะไร ปล่อยให้บวกแบบเลขฐานสองตามปกติ จะได้ $(0111)_2 + (1)_2 = (1000)_2$ พบว่ามีค่าเป็น $(-8)_{10}$ ซึ่งเป็นเหตุการณ์เดียวกับที่เกิดขึ้นในรูปที่ 2-10 (เพียงแต่ในรูปที่ 2-10 นั้นเป็นจำนวนแบบ `int` ขนาด 32 บิต)

ตารางที่ 2-3 การแทนจำนวนเต็มตั้งแต่ -8 ถึง 7 ด้วยจำนวนฐานสองขนาด 4 บิต

จำนวนฐานสิบ	จำนวนฐานสอง
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

จำนวนฐานสิบ	จำนวนฐานสอง
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111



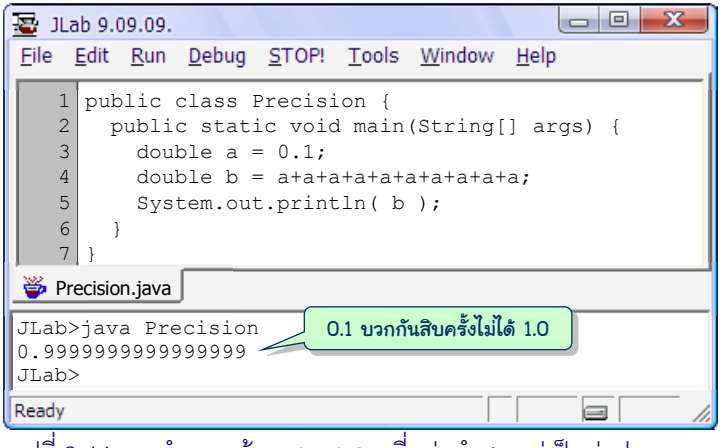
รูปที่ 2-10 การคำนวณด้วยจำนวนเต็ม ต้องระวังอย่างให้ล้นช่วงที่เก็บได้

จำนวนจริงในระบบเลขฐานสอง

คราวนี้มาดูการแทนจำนวนจริง ในกรณีของเลขฐานสิบ จำนวน $(123.25)_{10} = 100 + 20 + 3 + 0.2 + 0.05 = 1\times10^2 + 2\times10^1 + 3\times10^0 + 2\times10^{-1} + 5\times10^{-2}$ ในทำนองเดียวกัน $(101.11)_2 = 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} = 4+0+1+0.5+0.25 = (5.75)_{10}$ คำว่า `double` มาจาก double precision คือแม่นยำเป็นสองเท่าของแบบ `float` ส่วนคำว่า `float` มาจาก floating point ซึ่งคือรูปแบบการแทนจำนวนจริงรูปแบบหนึ่ง แทนในลักษณะให้ “จุดลอย” มาอยู่ด้านซ้าย เช่น

$(123.45)_{10}$ ให้เลื่อนจุดทศนิยมไปเรื่อยๆ เลื่อนซ้ายหนึ่งตำแหน่งก็คูณด้วย 10 หนึ่งครั้ง เลื่อนจุดมาจนอยู่ซ้ายสุด กลายเป็น 0.12345×10^3 ในกรณีที่เป็นฐานสอง $(101.11)_2$ เมื่อเขียนแบบจุดลอย จะเป็น $(0.10111)_2 \times 2^3$ การเก็บจำนวนจริงในเครื่องจะเก็บสามส่วนด้วยกัน คือเก็บเครื่องหมายบวกลบ เก็บเลขยกกำลัง และเก็บจำนวนหลังจุดทศนิยม

ผู้อ่านคงทราบว่ามีจำนวนจริงมากมายที่เขียนแทนด้วยจำนวนทศนิยมไม่ได้ เช่น $^1/_3$ = $(0.333333...)_{10}$ ในทำนองเดียวกันก็มีจำนวนจริงมากมายที่เขียนแทนด้วยจำนวนที่มีจุดในระบบเลขฐานสองไม่ได้เช่นกัน เช่น $(0.1)_{10}$ = $(0.\ 0001100110011001100110011...)_2$ จาวาเก็บจำนวนจริงด้วยปริมาณเนื้อที่จำกัดอย่าง `double` ซึ่งใช้เนื้อที่ 8 ไบต์ (1 ไบต์มี 8 บิต) เท่ากับ 64 บิต ประกอบด้วย 1 บิตไว้เก็บเครื่องหมายบวกหรือลบ, 11 บิตเก็บเลขยกกำลัง, และที่เหลืออีก 52 บิต เก็บจำนวนหลังจุด ย่อมเก็บจำนวนจริงบางจำนวนได้แค่เพียงค่าประมาณเท่านั้น ดังตัวอย่างในรูปที่ 2-11 แสดงให้เห็นว่าการนำ $(0.1)_{10}$ บวกกัน 10 ครั้งได้ค่าไม่เท่ากับ $(1.0)_{10}$ นักเขียนโปรแกรมจึงต้องเข้าใจพฤติกรรมการคำนวณด้วยจำนวน floating point ของเครื่องคอมพิวเตอร์ที่ต้องถือว่าแม่นยำสูง แต่ก็ยังเป็นเพียงค่าประมาณเท่านั้น



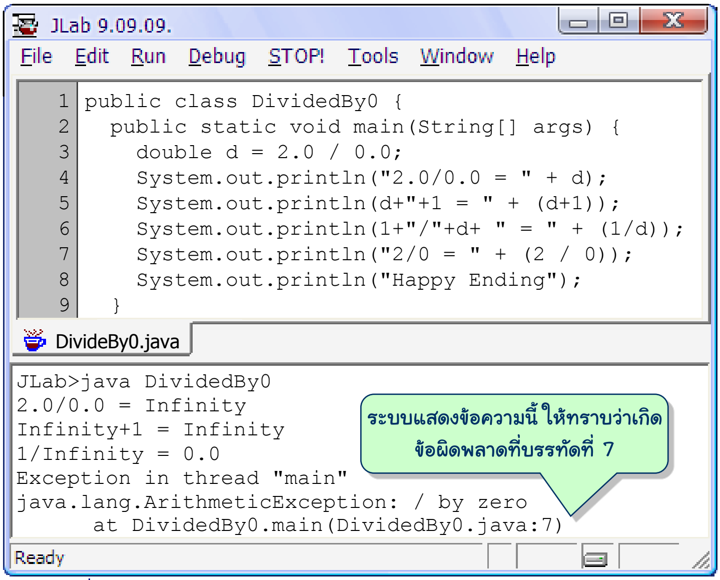
รูปที่ 2-11 การคำนวณด้วย `double` ที่แม่นยำสูง แต่เป็นค่าประมาณ

การหารด้วยศูนย์

มีเรื่องต้องรู้เล็กน้อยเกี่ยวกับการหารจำนวนด้วยศูนย์ ในกรณีของจำนวนจริง การหารด้วยศูนย์จะไม่มีข้อผิดพลาดใด ได้ค่าอนันต์ (หรืออลบอนันต์) ที่สามารถนำไปประมวลผลต่อได้ตามนิยามทางคณิตศาสตร์ทั่วไป ดังตัวอย่างในรูปที่ 2-12 อนันต์บวกอีกหนึ่งได้อนันต์ (บรรทัดที่ 5), หนึ่งหารด้วยอนันต์ได้ศูนย์ (บรรทัดที่ 6) การที่ `println` แสดงค่าอนันต์ออกมาเป็น `Infinity` นั้นไม่ได้หมายความว่าค่าอนันต์เก็บเป็นสตริง `"Infinity"` แต่ที่แสดงเช่นนั้น เพราะการทำงานภายใน `println` มีการเปลี่ยนจำนวนจริงที่แทนค่าอนันต์ไปเป็นสตริง `"Infinity"` เพื่อให้

ผู้ใช้เข้าใจความหมาย ขอให้ผู้อ่านลองแสดงค่าของอนันต์หารอนันต์, อนันต์ลบอนันต์, ศูนย์คูณอนันต์ ดูว่าจะได้ผลอะไร [11]

สำหรับการหารจำนวนเต็มด้วยศูนย์ จะเกิดสิ่งผิดปกติขึ้นในระบบ โปรแกรมจะเลิกทำงานทันที บรรทัดที่ 7 ในรูปที่ 2-12 เกิดการหารด้วยศูนย์แบบจำนวนเต็ม ระบบจะไม่ทำบรรทัดที่ 8 ต่อ แต่จะแสดงข้อความให้ทราบ กรณีนี้ระบบแจ้งให้ทราบว่าเกิด `ArithmeticException: / by zero` ที่บรรทัดที่ 7 (สิ่งผิดปกติที่เกิดขึ้นนี้เรียกว่า exception ซึ่งจะได้นำเสนอกันอย่างละเอียดในบทหลังๆ)



รูปที่ 2-12 การหารด้วยศูนย์แบบจำนวนเต็มกับแบบจำนวนจริง

เรื่องจุกจิกของ Scanner

เราได้นำเสนอการใช้ `Scanner` เพื่อสร้างตัวอ่านข้อมูล ซึ่งสามารถใช้บริการ `nextInt`, `nextDouble`, และ `nextLine` เพื่ออ่านจำนวนเต็ม จำนวนจริง และข้อความทางแป้นพิมพ์ ถ้าผู้อ่านลองสั่งโปรแกรมที่รอรับค่าจากแป้นพิมพ์อาจพบปัญหากันมาบ้างแล้วว่า ถ้าโปรแกรมรอรับจำนวนเต็ม แต่ผู้ใช้ใส่จำนวนจริงหรือข้อความ หรือรอรับจำนวนจริง แต่ใส่ข้อความ จะเกิดปัญหาคือ โปรแกรมจะแจ้งความผิดพลาด แล้วหยุดการทำงานทันที ปัญหานี้เป็นลักษณะการทำงานของตัวอ่านแบบ `Scanner` ซึ่งก็ต้องยอมรับสภาพ (ในบทหลัง ๆ จะได้นำเสนอการจัดการกับเหตุ-

---

[11] จะแสดงค่าเป็น NaN ย่อมาจาก Not a Number ซึ่งแทนค่าอะไรบางอย่างที่ไม่ใช่จำนวนจริง ค่านี้มักใช้แทนผลการคำนวณของจำนวนจริงที่ได้ผลที่ไม่อาจนิยามได้ เช่น ∞ ÷ ∞ หรือไม่อาจแทนค่าได้ เช่นค่าที่ได้จาก Math.sqrt(-1)

การณ์ที่มีความผิดปกติในทำนองนี้ โดยสามารถปรับโปรแกรมให้รอรับข้อมูลใหม่ ถ้าผู้ใช้ใส่ข้อมูลผิดประเภท โดยจะไม่ขอนำเสนอหรือลงในรายละเอียดตอนนี้)

มีเรื่องจุกจิกอีกเรื่องหนึ่งที่ต้องคำนึงถึงคือ การเรียกใช้ `nextLine` ตามหลัง `nextInt` หรือ `nextDouble` รหัสที่ 2-20 แสดงโปรแกรมที่รอรับจำนวนเต็มที่บรรทัดที่ 6 ตามด้วยการรอรับสตริงที่บรรทัดที่ 8 เมื่อสั่งทำงาน ผู้ใช้ป้อนจำนวนเต็ม 12 พอกดปุ่ม ENTER จะแสดงข้อความ **กรุณาใส่ข้อความ > 12, <** ซึ่งคือผลการทำงานของบรรทัดที่ 7 ถึง 9 เสมือนกับว่า การรอรับข้อความจากแป้นพิมพ์ในบรรทัดที่ 8 ได้ข้อความที่ไม่มีค่าอะไรเลยกลับคืนมาใส่ในสตริง `s` !! ทำไมจึงเป็นเช่นนั้น ?

```
04      Scanner kb = new Scanner(System.in);
05      System.out.print("กรุณาใส่จำนวนเต็ม > ");
06      int n = kb.nextInt();
07      System.out.print("กรุณาใส่ข้อความ > ");
08      String s = kb.nextLine();
09      System.out.println(n + ", " + s + "<");
```

```
TestScanner.java
JLab>java TestScanner
กรุณาใส่จำนวนเต็ม > 12
กรุณาใส่ข้อความ > 12, <
JLab>
Ready
```

รหัสที่ 2-20 การเรียก `nextLine` ตามหลัง `nextInt, nextDouble` หรือ `next`

อาการเช่นนี้มาจากพฤติกรรมการทำงานของ `nextLine` ที่จะอ่านข้อมูลจากจุดที่เพิ่งอ่านไปจนถึงสุดบรรทัด การป้อนข้อมูลของผู้ใช้ครั้งแรกคือ 12 แล้วกดปุ่ม ENTER ภายในหน่วยความจำจะเก็บอักขระเรียงกันไปคือ 1|2|↵ (เครื่องหมาย ↵ ทางขวาแทนรหัสที่ได้จากการกดปุ่ม ENTER) การเรียก `nextInt` ในบรรทัดที่ 6 ทำให้ระบบอ่าน 1 และ 2 ต่อกันเข้ามาพร้อมกับแปลงเป็นจำนวนเต็ม 12 เก็บใส่ตัวแปร `n` โดยตำแหน่งที่จะพร้อมให้อ่านต่อไปคือตัว ↵ พอมาถึงบรรทัดที่ 8 `nextLine` จึงอ่านจนสุดบรรทัด (ถึงตัว ↵) นั่นคือได้สตริงที่ไม่มีอะไรสักตัว ทำให้ผลการทำงานเป็นดังที่แสดงข้างต้น

ดังนั้น หากต้องการอ่านสตริงทั้งบรรทัดหลังจากที่ใช้ `nextInt` หรือ `nextDouble` อ่านข้อมูลก่อนหน้า จึงต้องให้ทำ `nextLine` หนึ่งครั้งเพื่ออ่าน ↵ ให้หมดไป ก่อนที่จะเริ่มอ่านสตริงของบรรทัดใหม่ทั้งบรรทัดด้วย `nextLine` อีกหนึ่งครั้ง ดังแสดงในรหัสที่ 2-21

```
04      Scanner kb = new Scanner(System.in);
05      System.out.print("กรุณาใส่จำนวนเต็ม > ");
06      int n = kb.nextInt();
07      kb.nextLine(); // อ่านที่เหลือของบรรทัดทิ้งไป
08      System.out.print("กรุณาใส่ข้อความ > ");
09      String s = kb.nextLine();
10      System.out.println(n + ", " + s + "<");
```

```
TestScanner.java
JLab>java TestScanner
กรุณาใส่จำนวนเต็ม > 12
กรุณาใส่ข้อความ > ABC
12, ABC<
JLab>
Ready
```

รหัสที่ 2-21 การเรียก `nextLine` เพื่ออ่านส่วนที่เหลือของบรรทัดทิ้งไป

แบบฝึกหัด

1. จงหาว่า นิพจน์คณิตศาสตร์ต่อไปนี้มีค่าเท่าไร ถ้าหาค่าไม่ได้ ให้อธิบายเหตุผลด้วย

```
7/3             7%3             7%1             8/0
8/0.0           0.0/8           1/2+1/2         0.0+1/2
(0.0+1)/2       7-3*7/3         (int)1.0/2.0    (int)1.0/2
5++             (5/9)*(212-32)  (double)1/2+1/2
```

2. ชื่อตัวแปรต่อไปนี้ ชื่อใดเขียนถูกต้องตามกฎของจาวา ชื่อใดผิด

```
hello           Hello           hEllo           7zean
pantip          $money          $12             $US12.00
_$_$_           $_$_$           :)              java.101
public          Public          publics         main
string          String          void            Void
krungthepmahanakorn_bowornratanakosin           bkk10330
กรุงเทพมหานครบวรรัตนโกสินทร์   กท๕๕๕๕         ๑๒๓๔๕
```

3. จงเขียนนิพจน์คณิตศาสตร์เพื่อคำนวณสูตรต่าง ๆ ต่อไปนี้

$(a+b^2c)^2$ $(1+x)^{-2}$ $(x_1-x_2)(x_1-x_3)$ $\sqrt{(a-b)(b-c)(c-a)}$

$\frac{c}{a^{-1}+b^{-1}}$ $\frac{1}{\sqrt{5}}(a+b)$ $\frac{(x_1-x_2)^3}{x_1^{-2}+x_2^{-2}}$ $1+x+x^2+x^3+x^4+x^5$

4. จงเขียนโปรแกรมรับจำนวนเต็มทางแป้นพิมพ์ (กำหนดให้ผู้ใช้ป้อนค่าได้ตั้งแต่ 0 ถึง 9999) เพื่อนำมาแยกแสดงว่า มีเลขโดดหลักพัน หลักร้อย หลักสิบ และหลักหน่วยอะไรบ้าง (เช่น ถ้าผู้ใช้ป้อนจำนวน `469` จะได้ผลลัพธ์เป็น `0, 4, 6, 9` เป็นต้น)

5. จงเขียนชุดคำสั่งเพื่อย้ายจำนวนเต็มที่เก็บในตัวแปร `a1, a2, a3, a4,` และ `a5` ในลักษณะแบบวง เช่น เดิม `a1, a2, a3, a4, a5` เก็บจำนวนเต็ม 2,3,4,5,6 ตามลำดับ หลังการย้ายข้อมูลในตัวแปรเหล่านี้แบบวงแล้ว จะได้ `a1, a2, a3, a4, a5` เก็บ 3,4,5,6,2 ตามลำดับ

6. เครื่องวัดความเร็วที่ตำรวจทางหลวงใช้อาศัยการยิงคลื่นความถี่ f_0 ไปกระทบตัวรถที่เคลื่อนเข้าหา แล้วรอรับคลื่นที่สะท้อนกลับมา f_1 เพื่อนำไปคำนวณความเร็วรถด้วยสูตร

$$v = (10.7585\times10^8)\frac{(f_1-f_0)}{(f_1+f_0)} \text{ กิโลเมตรต่อชั่วโมง, โดยที่ } f_0 = 2\times10^{10} \text{ วินาที}^{-1}$$

จงเขียนโปรแกรมรับความถี่ f_1 เพื่อแสดงความเร็วรถ (ลองให้ $f_1 = 2.000004\times10^{10}$ วินาที$^{-1}$ ดู)
หมายเหตุ : เราสามารถเขียนค่าคงตัวของจำนวนจริงในรูปแบบสัญกรณ์ทางวิทยาศาสตร์ได้ในจาวา เช่น -2.03×10^{11} เขียนแทนด้วย `-2.03e11` โดยสามารถป้อนการเขียนอย่างสั้นนี้ผ่านแป้นพิมพ์ก็ได้ เมื่อใช้ `nextDouble` ในการอ่านจำนวนจริงทางแป้นพิมพ์

7. จงเขียนโปรแกรมเพื่อหาค่าของ x และ y ที่ทำให้สมการ $a_1x + b_1y = c_1$ และ $a_2x + b_2y = c_2$ เป็นจริง จากค่า a_1, b_1, c_1, a_2, b_2, และ c_2 ที่รับทางแป้นพิมพ์

8. จงหาว่า เมื่อทุกคำสั่งข้างล่างนี้ทำงานเสร็จแล้ว ตัวแปรต่างๆ เก็บค่าอะไรบ้าง

```
int a = 0, b = 1, c = 2;
double x = 0, y = 1, z = 2;
x = a-- + ++b + c++ + ++y / z++;
```

9. จำนวนฮาร์มอนิก (Harmonic number) H_n มีค่าเท่ากับ $\sum_{k=1}^{n} \frac{1}{k}$ เช่น $H_3 = {}^1/_1 + {}^1/_2 + {}^1/_3$

จงเขียนโปรแกรมแสดงค่าของจำนวนฮาร์มอนิก H_1, H_2, H_3, ..., H_{10}

10. เราสามารถประมาณค่าของ $n!$ ได้ด้วยสูตรดังนี้ (เรียกว่า Stirling's approximation)

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

เช่น $100! \approx 9.32484762526942\times10^{157}$ จงเขียนโปรแกรมรับค่า n เพื่อคำนวณค่าประมาณของ $n!$ ให้ผู้อ่านศึกษาการใช้ค่า π (`Math.PI`) ค่า e (`Math.E`) ฟังก์ชันหารากที่สอง (`Math.sqrt`) และ ฟังก์ชันการยกกำลัง (`Math.pow`) จากคลาส `Math`

11. พื้นที่ของร่างกายเป็นตัวชี้วัดตัวหนึ่งของร่างกายทางการแพทย์ สามารถประมาณได้จากความสูงและน้ำหนัก ด้วยสูตรต่างๆ ข้างล่างนี้ (น้ำหนัก w มีหน่วยเป็นกิโลกรัม และความสูง h มีหน่วยเป็นเซนติเมตร ผลที่ได้มีหน่วยเป็นตารางเมตร)

สูตรของ Mosteller	$S = \sqrt{\frac{w\times h}{3600}}$
สูตรของ Du Bois	$S = \frac{71.84\times w^{0.425}\times h^{0.725}}{10000}$
สูตรของ Boyd	$S = 0.0003207\times h^{0.3}\times(1000\times w)^{(0.7285-0.0188(3+\log_{10} w))}$

จงเขียนโปรแกรมที่รับน้ำหนักและความสูง เพื่อแสดงพื้นที่ของผิวกายด้วยสูตรทั้งสาม (ให้ผู้อ่านศึกษาการใช้ฟังก์ชันคณิตศาสตร์เพิ่มเติมจากคลาส `Math`)

12. จงเขียนโปรแกรมเปลี่ยนจำนวนวินาทีเป็นจำนวนปี เดือน วัน ชั่วโมง นาที และวินาที เช่น

1000000 วินาที = 0 ปี 0 เดือน 11 วัน 13 ชั่วโมง 46 นาที 40 วินาที

เพื่อความง่ายให้ประมาณว่า 1 เดือนมี 30 วัน

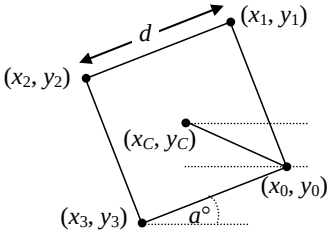
13. จงเขียนโปรแกรมคำนวณเกรดเฉลี่ยของทั้งวิชา ซึ่งคำนวณจาก

4×(จำนวนนักเรียนที่ได้ A) + 3×(จำนวนนักเรียนที่ได้ B) + 2×(จำนวนนักเรียนที่ได้ C) +
1×(จำนวนนักเรียนที่ได้ D) + 0×(จำนวนนักเรียนที่ได้ F)

แล้วนำผลรวมที่ได้มาหารด้วยจำนวนนักเรียนทั้งหมด

14. จงเขียนโปรแกรมวาดรูปที่แสดงในรูปที่ 2-9 ด้วย `DWindow`

15. จงเขียนโปรแกรมที่รับความยาวด้าน d, จุดศูนย์กลาง (x_C, y_C), และมุมเอียง $a°$ ของสี่เหลี่ยมจัตุรัส ดังรูปทางขวานี้ เพื่อคำนวณและแสดงพิกัดของมุมทั้งสี่ของสี่เหลี่ยม พร้อมทั้งวาดสี่เหลี่ยมดังกล่าวด้วยการลากเส้นตรงสี่เส้นจากพิกัดทั้งสี่ที่คำนวณได้โดยใช้ `DWindow` เพื่อยืนยันความถูกต้อง



16. แทนที่จะใช้ `Scanner` ในการอ่านข้อมูลทางแป้นพิมพ์จากผู้ใช้ เราสามารถให้ผู้ใช้กรอกข้อมูลทางกล่องโต้ตอบที่แสดงเป็นวินโดว์ได้ด้วยคำสั่งของคลาส `JOptionPane` (ที่เคยนำเสนอในแบบฝึกหัดของบทที่ 1) โปรแกรมข้างล่างนี้แสดงตัวอย่างการรับจำนวนจากผู้ใช้สองจำนวนเพื่อหาผลรวมแล้วแสดงผลลัพธ์

```
01 import javax.swing.JOptionPane;
02 public class InputDialogDemo {
03   public static void main(String[] args) {
04     String a = JOptionPane.showInputDialog("Enter a number");
05     String b = JOptionPane.showInputDialog("Enter a number");
06     double a1 = Double.parseDouble(a);
07     double b1 = Double.parseDouble(b);
08     String sum = "" + (a1 + b1);
09     JOptionPane.showMessageDialog(null, sum);
10   }
11 }
```

ให้สังเกตว่า ข้อมูลที่ได้มาจากผู้ใช้ (บรรทัดที่ 4 และ 5) เป็นสตริง ซึ่งถ้าต้องการนำไปคำนวณต้องแปลงเป็นจำนวนก่อน ในที่นี้เราใช้คำสั่ง `Double.parseDouble` ในการแปลงสตริงเป็นจำนวนจริง และเมื่อหาผลรวมเป็นจำนวนจริงแล้ว ก็ต้องแปลงกลับเป็นสตริงเพื่อให้สามารถนำไปแสดงผลในบรรทัดที่ 9 (ผู้เขียนคลาส `JOptionPane` กำหนดไว้เช่นนี้ จึงต้องปฏิบัติตาม) จงดัดแปลงโปรแกรมข้างต้นนี้เพื่อหาค่าเฉลี่ยจากข้อมูล 5 จำนวนที่รอรับจากผู้ใช้

07 ทำซ้ำ ๆ

คอมพิวเตอร์จำได้มาก คำนวณได้รวดเร็ว และไม่บ่นเมื่อสั่งให้ทำงานซ้ำๆ การทำซ้ำทำได้ง่ายด้วยการเขียนคำสั่งซ้ำ หรือใช้วงวนซึ่งทำกลุ่มคำสั่งในวงวนซ้ำ ๆ บทนี้นำเสนอคำสั่งสร้างวงวนในบางงานอาจใช้วงวนที่หมุนทำกลุ่มคำสั่งแบบไม่รู้จบ แต่บางงานก็อาจต้องการเพิ่มเงื่อนไขภายในให้ทดสอบว่า เมื่อใดจะกระโดดออกจากวงวน เพื่อทำคำสั่งอื่นต่อ ด้วยคำสั่งวงวนและคำสั่งทดสอบการออกจากวงวน ทำให้สามารถเขียนโปรแกรมเพื่อแก้ไขปัญหาที่สลับซับซ้อนขึ้น นอกจากนี้ขอเสริมด้วยการนำเสนอเครื่องมือที่ช่วยร่าง ช่วยออกแบบ และช่วยบรรยายขั้นตอนการทำงานของโปรแกรมเพื่อสื่อสารแนวคิดให้นักเขียนโปรแกรมด้วยกัน เข้าใจการทำงานของโปรแกรมได้ดีขึ้น

วงวนไม่รู้จบ

โปรแกรมที่เราได้เขียนกันมาในบทที่แล้ว มีรูปแบบการทำงานคร่าว ๆ คือ รอรับข้อมูลขาเข้าจากผู้ใช้ทางแป้นพิมพ์ ประมวลผลข้อมูลขาเข้านั้นให้ได้ผลลัพธ์ที่ต้องการ ปิดท้ายด้วยการแสดงผลที่หาได้ทางจอภาพ แล้วโปรแกรมก็เลิกทำงาน ยังมีลักษณะการทำงานของโปรแกรมอีกรูปแบบหนึ่ง ที่รับข้อมูลจากผู้ใช้ ประมวลผล แสดงผล แล้วก็วนกลับไปรับข้อมูลจากผู้ใช้ ทำซ้ำเช่นนี้ไปเรื่อย ๆ จนกว่าผู้ใช้จะเลิกการทำงานของโปรแกรมเอง วิธีสร้างวงวน (loop) เพื่อให้กลุ่มคำสั่งหนึ่งทำซ้ำไปเรื่อย ๆ ไม่รู้จบ กระทำได้ด้วยคำสั่ง `while(true)` ดังแสดงในรูปที่ 3-1

```
while (true) {
   // กลุ่มคำสั่งที่ต้องการทำซ้ำ

} // เมื่อถึงบรรทัดนี้จะวนกลับไปทำใหม่
```

รูปที่ 3-1 การใช้ `while(true)` เพื่อสร้างวงวนไม่รู้จบ

รหัสที่ 3-1 แสดงตัวอย่างโปรแกรมซึ่งปรับปรุงจากโปรแกรมคำนวณดัชนีมวลกายที่ได้เขียนในบทที่แล้ว นำส่วนการทำงานหลักมาครอบด้วยคำสั่ง `while(true)` ให้สังเกตว่าเราจัดรูปแบบให้บรรทัดที่ 6 ถึง 13 เยื้องไปทางขวาเล็กน้อย เพื่อแสดงให้เห็นเด่นชัดว่า อยู่ภายในวงเล็บของ `while(true){...}` นอกจากนี้เราสามารถปรับปรุง โดยย้ายส่วนที่ไม่จำเป็นต้องทำทุกรอบออกไปอยู่นอกวงวน ให้ทำก่อนเข้าวงวน สำหรับโปรแกรมนี้คือ ให้สร้างตัวอ่านทางแป้นพิมพ์เพียงครั้งเดียว ก่อนเข้าทำงานในวงวน ดังแสดงในรหัสที่ 3-2

```
01 import java.util.Scanner;
02 // โปรแกรมคำนวณดัชนีมวลกาย แบบวนทำไม่รู้จบ
03 public class BodyMassIndex {
04   public static void main(String[] args) {
05     while (true) {
06       Scanner kb = new Scanner(System.in);
07       System.out.print("น้ำหนัก (kg.) = ");
08       double weight = kb.nextDouble();
09       System.out.print("ความสูง (cm.) = ");
10       double height = kb.nextDouble();
11       double hm = height / 100.0;
12       double bmi = weight / (hm * hm);
13       System.out.println("ดัชนีมวลกาย = " + bmi);
14     }
15   }
16 }
```

นำส่วนที่ต้องการทำซ้ำมาครอบด้วยคำสั่ง while(true) {...}

รหัสที่ 3-1 โปรแกรมคำนวณดัชนีมวลกาย (วนทำไม่รู้จบ)

```
01 import java.util.Scanner;
02 // โปรแกรมคำนวณดัชนีมวลกาย แบบวนทำไม่รู้จบ
03 public class BodyMassIndex {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     while (true) {
07       System.out.print("น้ำหนัก (kg.) = ");
08       double weight = kb.nextDouble();
09       System.out.print("ความสูง (cm.) = ");
10       double height = kb.nextDouble();
11       double hm = height / 100.0;
12       double bmi = weight / (hm * hm);
13       System.out.println("ดัชนีมวลกาย = " + bmi);
14     }
15   }
16 }
```

เราสร้างตัวอ่านเพียงครั้งเดียวก็พอ ก่อนเข้าวงวน ไม่จำเป็นต้องสร้างใหม่ทุกรอบ

รหัสที่ 3-2 โปรแกรมคำนวณดัชนีมวลกาย (วนทำไม่รู้จบ) รุ่นปรับปรุง

ถ้าสั่งโปรแกรมในรหัสที่ 3-2 ทำงาน พอทำจนถึงบรรทัดที่ 13 ก็จะวนกลับไปทำบรรทัดที่ 7 ต่อ ผู้อ่านอาจสงสัยว่า แล้วโปรแกรมจะเลิกทำงานเมื่อไร ก็ต้องขอบอกว่า ตัวโปรแกรมจะทำงาน

ไม่เลิก ถ้าผู้ใช้ต้องการให้เลิก ต้องหยุดการทำงานเอง ใน JLab ผู้ใช้สามารถยกเลิกการทำงานของโปรแกรมได้โดยกดปุ่ม Ctrl + Pause Break หรือเลือกเมนู STOP!

DWindow

หัวข้อนี้กลับมานำเสนอตัวอย่างการใช้ DWindow กันอีกครั้ง คราวนี้จะใช้ร่วมกับวงวนไม่รู้จบ เพื่อเขียนโปรแกรมแสดงภาพกราฟิกเคลื่อนไหวกัน (ผู้อ่านควรลองเขียนโปรแกรม และสั่งทำงาน จะได้เห็นภาพเคลื่อนไหวจริงๆ)

นาฬิกาเข็มเดียว

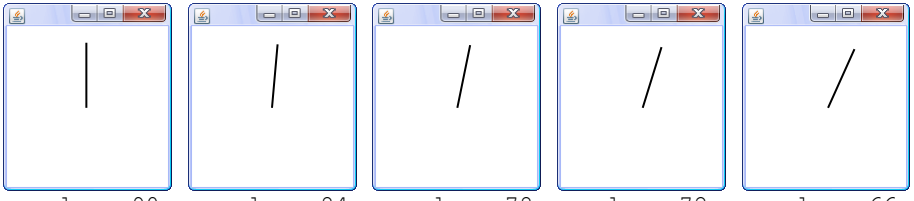
ขอนำโปรแกรมวาดเส้นตรงในบทที่แล้ว ซึ่งอาศัยการกำหนดจุดเริ่มต้น ความยาว และมุมของเส้นมาทำเป็นเข็มนาฬิกา และใช้วงวนไม่รู้จบสั่งให้วาดเส้นตรงไปเรื่อย ๆ โดยต้องการให้เข็มวนหนึ่งรอบ 360 องศาในเวลา 1 นาที ดังนั้น ทุก ๆ วินาทีต้องวาดเส้นใหม่ที่มีมุมของเข็มลดลงครั้งละ 360/60 = 6 องศา (ที่องศาของมุมลดลง เพราะต้องการให้มุมตามเข็มนาฬิกา) นำแนวคิดนี้มาเขียนโปรแกรมได้ดังรหัสที่ 3-3 ถ้าลองสั่งให้ทำงาน ใน 4 วินาทีแรกจะมีการเปลี่ยนแปลงของวินโดว์เป็นดังรูปที่ 3-2

```
01 import jlab.graphics.DWindow;
02 // โปรแกรมนาฬิกาเข็มเดียว
03 public class Clock {
04   public static void main(String[] args) {
05     int width = 160, height = width, len = width*4/10;
06     DWindow w = new DWindow(width, height);
07     double x0 = width/2, y0 = height/2;
08     double x1, y1, angle = 90, delta = 6; // เริ่มที่ตำแหน่งเลข 12
09     while (true) {
10       x1 = x0 + len * Math.cos(Math.toRadians(angle)); //เหมือนใน
11       y1 = y0 - len * Math.sin(Math.toRadians(angle)); //รหัส 2-18
12       w.clearBackground();          // ล้างวินโดว์
13       w.drawLine(x0, y0, x1, y1);   // วาดเส้นใหม่
14       w.sleep(1000);                // หยุดการทำงาน 1000 มิลลิวินาที
15       angle -= delta;               // ปรับมุม เพื่อวาดเส้นในรอบต่อไป
16     }
17   }
18 }
```

รหัสที่ 3-3 โปรแกรมนาฬิกาที่มีแต่เข็มวินาที

รหัสที่ 3-3 ใช้เมท็อดของ DWindow ที่ยังไม่เคยอธิบายมาก่อนคือ clearBackground และ sleep เมท็อดแรกล้างพื้นที่ในวินโดว์ ซึ่งเราต้องเรียกก่อนวาดเส้นใหม่ จะได้ให้ความรู้สึกเสมือนว่าเข็มเคลื่อนไหวจากตำแหน่งที่แล้ว มายังตำแหน่งใหม่ เมท็อดที่สองทำให้การทำงานของ

โปรแกรมหยุดรอเป็นระยะเวลาตามที่กำหนดไว้ภายในวงเล็บ (ระยะเวลามีหน่วยเป็นมิลลิวินาที) ดังนั้น เมื่อต้องการให้หยุด 1 วินาที ก็ใช้ `w.sleep(1000)`

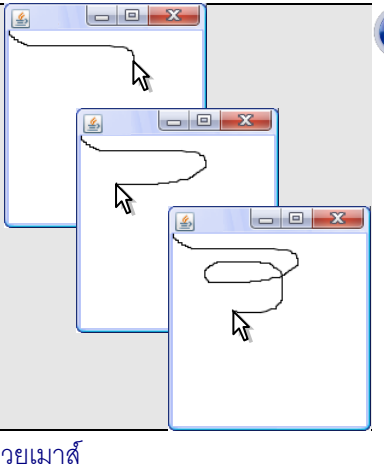


รูปที่ 3-2 การเปลี่ยนแปลงของวินโดว์ใน 4 วินาทีแรก

วาดลายเส้นด้วยเมาส์

กลับมาเขียนโปรแกรมในลักษณะที่รับข้อมูลทางผู้ใช้ด้วย คราวนี้เป็นการรับตำแหน่งของตัวชี้เมาส์ที่ผู้ใช้กำลังเลื่อนไปมาบนวินโดว์ จะขอเขียนโปรแกรมให้ผู้ใช้วาดลายเส้นบนวินโดว์ด้วยเมาส์ดังรหัสที่ 3-4 โปรแกรมอ่านพิกัดของเมาส์บนวินโดว์ `w` ด้วย `w.getMouse().getX()` และ `w.getMouse().getY()` (ขอให้จำวิธีอ่านตำแหน่งเมาส์ไปก่อน แล้วจะค่อยเข้าใจรูปแบบการเรียกใช้ในบทหลัง ๆ) จากนั้นวาดเส้นตรงจากตำแหน่งที่แล้วของเมาส์ (ซึ่งเก็บในตัวแปร `x0` และ `y0`) มายังตำแหน่งของเมาส์ตอนนี้ (เก็บในตัวแปร `x1` และ `y1`) และก่อนจะวนกลับไปอ่านตำแหน่งเมาส์ตำแหน่งใหม่ ก็ให้ย้ายค่าของ `x1` และ `y1` ไปเก็บใน `x0` และ `y0` (บรรทัดที่ 11) เพื่อเตรียมวาดเส้นใหม่ต่อจากเส้นที่เพิ่งวาดในรอบถัดไป

```
01 import jlab.graphics.DWindow;
02 // โปรแกรมวาดลายเส้นด้วยเมาส์
03 public class Scribble {
04   public static void main(String[] args) {
05     DWindow w = new DWindow(160, 160);
06     double x0 = 0, y0 = 0, x1, y1;
07     while (true) {
08       x1 = w.getMouse().getX();
09       y1 = w.getMouse().getY();
10       w.drawLine(x0, y0, x1, y1);
11       x0 = x1; y0 = y1;
12     }
13   }
14 }
```



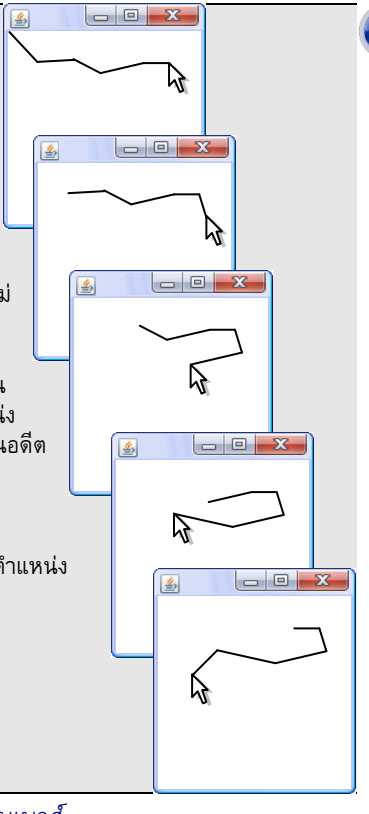
รหัสที่ 3-4 โปรแกรมวาดลายเส้นด้วยเมาส์

งูเลื้อยตามเมาส์

ตัวอย่างนี้ต้องการผลคล้ายกับตัวอย่างที่แล้ว ต่างกันตรงที่ต้องการแสดงเส้น 5 เส้นล่าสุด ดังนั้น หากเลื่อนเมาส์ไปมาบนวินโดว์ จึงเสมือนกับงูเลื้อยไล่ตามเมาส์ สิ่งที่ต้องจำคือ 5 เส้นล่าสุด

จะลาก 5 เส้นที่ต่อเนื่องกัน ก็ต้องเก็บจุดปลาย 6 จุด จุดละ 2 ตัวแปร คอยจำตำแหน่ง จึงตั้งชื่อตัวแปรเหล่านี้ว่า `x0,x1,...,x5` และ `y0,y1,...,y5` โดยให้ `(x5,y5)` เก็บตำแหน่งล่าสุดของเมาส์ ดังนั้น ก่อนจะขึ้นรอบต่อไปเพื่ออ่านตำแหน่งเมาส์ไปเก็บใส่ `(x5,y5)` ต้องย้ายค่าของจุดที่ 5, 4, 3, 2, 1 ไปเก็บใส่ตัวแปรของจุดที่ 4, 3, 2, 1, 0 ตามลำดับ และต้องหยุดการทำงานชั่วขณะ เพื่อให้ผู้ใช้มองเห็นการเปลี่ยนแปลงด้วย เขียนแนวคิดนี้เป็นโปรแกรมดังรหัสที่ 3-5

```
01 import jlab.graphics.DWindow;
02 // โปรแกรมงูเลื้อยตามเมาส์
03 public class Snake {
04   public static void main(String[] args) {
05     DWindow w = new DWindow(160, 160);
06     double x0=0, y0=0, x1=0, y1=0;
07     double x2=0, y2=0, x3=0, y3=0;
08     double x4=0, y4=0, x5=0, y5=0;
09     while (true) {
10       x5 = w.getMouse().getX(); //อ่านตำแหน่งใหม่
11       y5 = w.getMouse().getY();
12       w.clearBackground();       //ล้างภาพ
13       w.drawLine(x0, y0, x1, y1); //วาดงู 5 เส้น
14       w.drawLine(x1, y1, x2, y2); // จากตำแหน่ง
15       w.drawLine(x2, y2, x3, y3); // ของเมาส์ในอดีต
16       w.drawLine(x3, y3, x4, y4);
17       w.drawLine(x4, y4, x5, y5);
18       w.sleep(100);       //หยุด 100ms.
19       x0 = x1; y0 = y1;   //เลื่อนตำแหน่งของงูไปหนึ่งตำแหน่ง
20       x1 = x2; y1 = y2;
21       x2 = x3; y2 = y3;
22       x3 = x4; y3 = y4;
23       x4 = x5; y4 = y5;
24     }
25   }
26 }
```



รหัสที่ 3-5 โปรแกรมแสดงงูเลื้อยตามเมาส์

งูเลื้อยตามเมาส์ (อีกแบบ)

ขอเขียนโปรแกรมงูเลื้อยตามเมาส์โดยใช้กลวิธีการทำภาพให้จางลง ๆ ในแต่ละรอบที่วน ส่งผลให้สีของเส้นแปรตามอายุของเส้น เส้นใดถูกวาดไว้นาน ก็ถูกทำให้จางหลายรอบ เลยจางมากกว่าเส้นที่เพิ่งวาด เส้นที่หางงูในอดีตจึงค่อย ๆ จางหายไป แต่มีเส้นใหม่ที่หัวงูเพิ่มขึ้นมาอย่างต่อเนื่อง

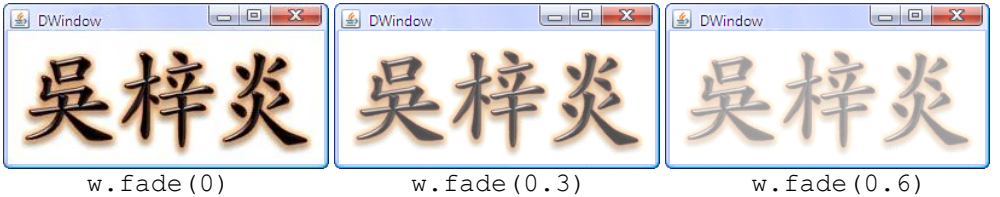
การทำให้ภาพในวินโดว์จางนั้นใช้เมท็อด `fade` ถ้าเรียก `w.fade(0.3)` ก็หมายความว่าให้ทำภาพในวินโดว์ `w` จางลงประมาณ 30% ค่าที่ส่งให้ `fade` นั้นมีค่าตั้งแต่ 0 ถึง 1 ค่า 0 คือการไม่ทำภาพจางแต่อย่างใด ให้ขณะที่ 1 คือทำภาพจางลงจนหมด ซึ่งคล้ายการลบภาพทิ้งไปนั่นเอง

รหัสที่ 3-6 แสดงตัวอย่างการอ่านแฟ้มภาพมาแสดงในวินโดว์ แล้วทำให้ภาพนั้นจางลง สั่งโปรแกรมทำงานสามครั้งด้วยค่าความจางคือ 0, 0.3, และ 0.6 กับภาพตัวอย่าง ได้ผลลัพธ์ดังรูปที่ 3-3

```
01 import jlab.graphics.DWindow;
02 // โปรแกรมแสดงภาพจากแฟ้มข้อมูลและทำภาพจาง
03 public class FadingWindow {
04   public static void main(String[] args) {
05     DWindow w = new DWindow(250,100);
06     w.loadImage("c:/java101/name.jpg");
07     w.fade(0.3);  // ตั้งค่าได้ตั้งแต่ 0 ถึง 1 : 0 = ไม่จางเลย, 1 = ภาพจางจนไม่เห็น
08   }
09 }
```

อ่านแฟ้มรูปภาพมาแสดงในวินโดว์ พร้อมกับปรับขนาดวินโดว์ให้พอดีกับรูปภาพ

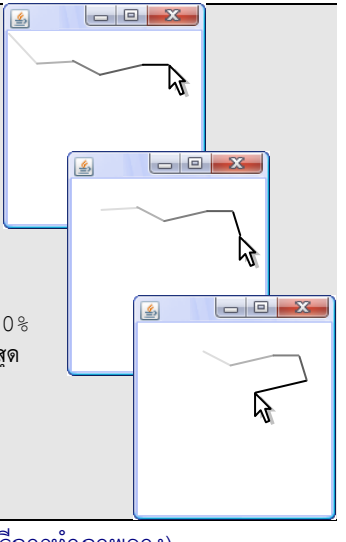
รหัสที่ 3-6 โปรแกรมแสดงภาพจากแฟ้มข้อมูลและทำภาพจางด้วยเมท็อด fade



รูปที่ 3-3 ผลของการทำภาพจางด้วยเมท็อด fade

เมื่อนำโปรแกรมวาดลายเส้นด้วยเมาส์ในรหัสที่ 3-4 มาปรับปรุงเล็กน้อย โดยทำภาพในวินโดว์ให้จางลงเล็กน้อย ก่อนวาดเส้นใหม่ และให้หยุดชั่วขณะหลังการวาดเส้นใหม่ ได้ดังรหัสที่ 3-7 จะเป็นโปรแกรมงูเลื้อยตามเมาส์ในอีกลักษณะหนึ่ง

```
01 import jlab.graphics.DWindow;
02 // โปรแกรมงูเลื้อยตามเมาส์
03 public class Snake {
04   public static void main(String[] args) {
05     int width = 160, height = width;
06     DWindow w = new DWindow(width, height);
07     double x0 = 0, y0 = 0, x1, y1;
08     while (true) {
09       x1 = w.getMouse().getX();
10       y1 = w.getMouse().getY();
11       w.fade(0.3);      // ทำภาพให้จางลงนิดหน่อยสัก 30%
12       w.drawLine(x0, y0, x1, y1); //เส้นล่าสุด เข้มสุด
13       w.sleep(100);     // หยุดพักสักครู่ให้ผู้ใช้เห็นภาพ
14       x0 = x1; y0 = y1;
15     }
16   }
17 }
```

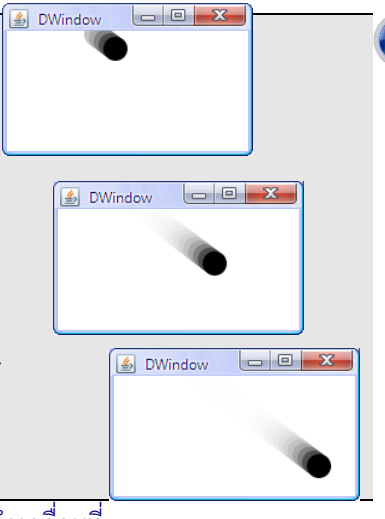


รหัสที่ 3-7 โปรแกรมแสดงงูเลื้อยตามเมาส์ (ใช้กลวิธีการทำภาพจาง)

ลูกบอลเคลื่อนที่

ขอเปลี่ยนจากการวาดเส้นตรง มาวาดวงรี ถ้าต้องการวาดวงรีสีแดงในวินโดว์ w ที่มีจุดศูนย์กลางที่พิกัด (x,y) กว้าง a สูง b ก็ใช้คำสั่ง w.fillEllipse(w.RED,x,y,a,b) ถ้า a มีค่าเท่า b จะได้วงกลม (ในกรณีที่ต้องการเฉพาะเส้นรอบวง ให้ใช้คำสั่ง drawEllipse แทน fillEllipse) รหัสที่ 3-8 คือโปรแกรมวนวาดวงกลมไม่รู้จบ แต่ละรอบจะเลื่อนตำแหน่งของวงกลมไปเล็กน้อยด้วยระยะทาง dx และ dy และจะทำภาพในวินโดว์ให้จางก่อนวาด เพื่อให้เสมือนมีการทิ้งร่องรอยการเคลื่อนที่ของลูกบอล และมีการหยุดทำงานชั่วขณะเพื่อควบคุมความเร็วของการเคลื่อนที่ด้วย เราสามารถคำนวณความเร็วของลูกบอลได้จากค่า dx, dy, และเวลาที่หยุดชั่วขณะ จากรหัสที่ 3-8 สรุปได้ว่าทุก 50ms ลูกบอลเคลื่อนได้ระยะ $\sqrt{dx^2 + dy^2} = \sqrt{4^2 + 3^2} = 5$ ลูกบอลจึงเคลื่อนด้วยความเร็ว 5/50ms = 100 จุดภาพต่อวินาที

```
01 import jlab.graphics.DWindow;
02 // โปรแกรมแสดงลูกบอลสีดำเคลื่อนที่
03 public class MovingBall {
04   public static void main(String[] args) {
05     DWindow w = new DWindow(200, 100);
06     double x = 70, y = 0;
07     double dx = 4, dy = 3;
08     while (true) {
09       x = x + dx;
10       y = y + dy;
11       w.fade(0.3);
12       w.fillEllipse(w.BLACK, x, y, 20, 20);
13       w.sleep(50);
14     }
15   }
16 }
```



รหัสที่ 3-8 โปรแกรมแสดงลูกบอลสีดำเคลื่อนที่

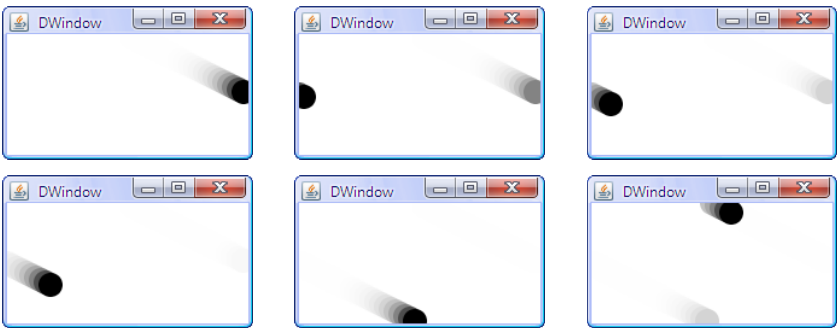
หากผู้อ่านลองสั่งโปรแกรมนี้ทำงาน จะเห็นว่าลูกบอลเคลื่อนที่ให้เห็นได้ชั่วขณะ แล้วก็เคลื่อนออกนอกวินโดว์ไป บทถัดไปจะบอกวิธีทำให้ลูกบอลเด้งผนังวินโดว์ได้ แต่ตอนนี้ขอปรับปรุงโปรแกรมแค่ให้ลูกบอลที่ออกนอกวินโดว์ฝั่งหนึ่ง เคลื่อนวนกลับมาอีกฝั่งหนึ่ง ซึ่งทำได้โดยแก้ไขการคำนวณในบรรทัดที่ 9 และ 10 ขอแสดงเฉพาะกรณีของ x สิ่งที่ต้องการคือถ้า x มีค่ามากกว่าความกว้างของวินโดว์ ก็ให้ลบด้วยค่าความกว้างของวินโดว์ จะได้ตำแหน่ง x วนกลับมาอยู่ภายในวินโดว์ เช่น วินโดว์กว้าง 200, x = 198, dx = 4, ค่า x ถัดไปคือ 198 + 4 = 202 ซึ่งมากกว่า 200 ก็ให้ลบออก 200 จะได้ x = 2 (เสมือน x วนกลับมาทางซ้าย) แล้วจะเขียนคำสั่งตามที่อธิบายได้อย่างไร เราสามารถใช้วิธีหาเศษจากการหารค่า (x + dx) ด้วยความกว้าง ให้เป็นค่าใหม่ของ x จากตัวอย่าง (198+4) ÷ 200 = (202 ÷ 200) ได้เศษ 2 จึงสามารถใช้ตัวดำเนินการ % เพื่อการนี้ได้ ดังนั้น สิ่งที่ต้องปรับปรุงคือ เปลี่ยนบรรทัดที่ 9 และ 10 เป็น x = (x+dx)%200 และ y =

`(y+dy)%100` แต่เราไม่ควรกำหนดความกว้างและความสูงของวินโดว์ให้เป็นค่าคงตัว 200 และ 100 แบบนี้ เพราะผู้ใช้อาจขยายหรือหดขนาดของวินโดว์ระหว่างการทำงานก็ได้ เราสามารถขอความกว้างและความสูงของวินโดว์ `w` ได้ด้วยการเรียกเมท็อด `w.getWidth()` และ `w.getHeight()` ตามลำดับ เมื่อรู้เช่นนี้แล้ว ก็จะได้โปรแกรมใหม่เขียนได้ดังรหัสที่ 3-9

```
01 import jlab.graphics.DWindow;
02 // โปรแกรมแสดงลูกบอลสีดำเคลื่อนที่ ถ้าตกขอบ จะวนกลับมาอีกด้านของวินโดว์
03 public class MovingBall {
04   public static void main(String[] args) {
05     DWindow w = new DWindow(200, 100);
06     double x = 100, y = 0;
07     double dx = 4, dy = 2;
08     while (true) {
09       x = (x + dx) % w.getWidth();
10       y = (y + dy) % w.getHeight();
11       w.fade(0.3);
12       w.fillEllipse(w.BLACK, x, y, 20, 20);
13       w.sleep(50);
14     }
15   }
16 }
```

w.getWidth() ได้ความกว้างของวินโดว์ w
ส่วน getHeight() ได้ความสูง

รหัสที่ 3-9 โปรแกรมแสดงลูกบอลสีดำเคลื่อนที่ ซึ่งวนกลับซ้าย และวนขึ้นบนได้



รูปที่ 3-4 ภาพถ่ายแสดงลูกบอลกำลังวนกลับทางซ้าย และวนกลับด้านบน ทำงานตามรหัสที่ 3-9

ถ้าเปลี่ยนให้ลูกบอลเคลื่อนที่ไปทางซ้าย โดยให้ค่าเริ่มของ `dx` เป็นลบ เช่น เปลี่ยนบรรทัดที่ 7 จาก `dx = 4` เป็น `dx = -4` แล้วลองสั่งโปรแกรมทำงานดู ผู้อ่านคิดว่าลูกบอลจะเคลื่อนที่อย่างไร และจะมีวิธีแก้ไขอย่างไร ? [1]

1 [illegible]

การกระโดดออกจากวงวน

การทำงานของวงวนเพื่อทำกลุ่มคำสั่งซ้ำ ๆ นั้น จะน่าสนใจมากขึ้น ถ้าเราสามารถหยุดการทำซ้ำ แล้วกระโดดออกจากวงวนไปทำอย่างอื่นต่อ การหารากที่สองด้วยวิธีของชาวบาบิโลนที่ได้นำเสนอในบทที่แล้ว เป็นตัวอย่างหนึ่งของการทำซ้ำ ๆ ในรหัสที่ 2-14 เราเขียนคำสั่ง `x = (x + a/x)/2.0` สี่บรรทัดต่อ ๆ กัน อันเป็นวิธีเดียวที่เรารู้ตอนนั้นเพื่อทำคำสั่งซ้ำ โดยทำซ้ำเป็นจำนวนครั้งคงตัว ตามจำนวนบรรทัดที่เขียน แต่การหารากที่สองด้วยวิธีนี้ ความแม่นยำของคำตอบขึ้นกับจำนวนครั้งที่ทำคำสั่งดังกล่าว ยิ่งซ้ำยิ่งแม่นยำถูกต้อง จึงควรเขียนเป็นโปรแกรมที่อาศัยวงวนทำคำสั่งนี้ซ้ำ ๆ จนกระทั่งได้คำตอบที่แม่นยำผิดพลาดน้อยตามเกณฑ์ที่ยอมรับได้ ก็ให้กระโดดออกจากวงวน และแสดงผลลัพธ์

การกระโดดออกจากวงวนใช้คำสั่ง `if ( `*เงื่อนไข*` ) break;` หมายความว่า ถ้าเงื่อนไขภายในวงเล็บเป็นจริง ให้กระโดดออกจากวงวนที่คำสั่งนี้อยู่ รหัสที่ 3-10 แสดงโปรแกรมที่ปรับการหารากที่สองในรหัสที่ 2-14 มาใช้วงวน เพื่อคำนวณ x ซึ่งเป็นค่าประมาณของ $\sqrt{a}$ ภายในวงวนมีคำสั่งที่บรรทัดที่ 11 ตรวจสอบว่า ถ้า $(x^2 - a)$ มีค่าน้อยกว่า 10^{-5} แสดงว่าแม่นยำพอ [2] ก็หลุดจากวงวน ไปทำงานต่อที่บรรทัดที่ 13 เพื่อแสดงผลลัพธ์ที่หาได้ อนึ่งเราไม่สามารถใช้คำสั่ง `break` โดดๆ โดยไม่มี `if` เพื่อกระโดดออกจากวงวน เพราะถ้าเขียนได้ คำสั่งที่เขียนตามหลัง `break` จะเขียนไปทำไม (จริงไหม ?)

เงื่อนไขในวงเล็บหลังคำสั่ง `if` คือ*นิพจน์ตรรกะ* (logical expression) ที่ให้ผลเป็นจริงกับเท็จ (ในจาวามีค่าคงตัวสองค่า `true` และ `false` แทนค่าจริงและเท็จตามลำดับ) เราเปรียบเทียบจำนวนได้ด้วย*ตัวดำเนินการสัมพันธ์* (relational operator) ดังนี้ `<` (น้อยกว่า) `>` (มากกว่า) `==` (เท่ากับ) `!=` (ไม่เท่ากับ) `>=` (มากกว่าหรือเท่ากับ) `<=` (น้อยกว่าหรือเท่ากับ) [3] โดยทั่วไปเราสามารถเขียนบรรยายการเปรียบเทียบได้หลายแบบที่ทำงานเหมือนกัน เช่น ต้องการเปรียบเทียบว่าค่าในตัวแปร `a` น้อยกว่า `b` หรือไม่ สามารถเขียน `a<b` หรือ `b>a` หรือ `(a-b)<0` หรือ `(b-a)>0` ซึ่งได้ผลเดียวกันหมด อย่างไรก็ตามควรเขียนแบบที่อ่านแล้วเข้าใจง่ายสุด

---

[2] ถ้าจะเขียนเงื่อนไขนี้ให้ชัดเจนขึ้น อาจต้องระบุว่า ถ้า $|x^2 - a| < 10^{-5}$ จึงกระโดดจากวงวน ซึ่งเขียนเป็นคำสั่งจาวาว่า `if ( Math.abs(x*x - a) < 1e-5 ) break;` แต่เราไม่ต้องหาค่าสัมบูรณ์ก็ได้ เพราะสามารถพิสูจน์ได้ว่า ด้วยวิธีของบาบิโลน x^2 มีค่ามากกว่า a เสมอ (ผู้อ่านลองพิสูจน์ดู)

[3] ไม่มีตัวดำเนินการ `=<` `=>` `!<` `!>` และต้องเน้นว่า เท่ากันต้องเขียนเครื่องหมาย = สองตัวติดกัน == ตัวดำเนินการ `< > <= >= == !=` เหล่านี้มีไว้เปรียบเทียบจำนวนเท่านั้น นำไปเปรียบเทียบสตริงไม่ได้ (สำหรับการเปรียบเทียบสตริงจะกล่าวในบทถัดไป)

```
01 import java.util.Scanner;
02 // โปรแกรมหารากที่สองของจำนวนจริงด้วยวิธีของชาวบาบิโลน (ผิดพลาดไม่เกิน 10^-5)
03 public class SquareRoot {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("a = ");
07     double a = kb.nextDouble();
08     double x = 1;
09     while (true) {
10       x = (x + a/x) / 2.0;
11       if ((x*x - a) < 1e-5) break;
12     }
13     System.out.println("รากที่สองของ " + a + " = " + x);
14   }
15 }
```

ถ้า $x^2 - a$ น้อยกว่า 1e-5 ให้ออกจากวงวน
1e-5 คือ $1\times10^{-5} = 0.00001$

รหัสที่ 3-10 โปรแกรมหารากที่สอง ใช้วงวนจนได้คำตอบที่ผิดพลาดไม่เกิน 10^{-5}

วงวนรู้จบ

การใส่คำสั่ง `if(...) break;` ไว้ภายในวงวนไม่รู้จบที่เขียนด้วย `while (true) {...}` ทำให้การทำงานของวงวนนั้นรู้จบ จบในที่นี้ไม่ได้หมายความว่า โปรแกรมทำงานจบ แต่หมายความว่า ไม่วนต่อ แต่ทำคำสั่งที่เขียนต่อจากเครื่องหมาย `}` ของวงวน วงวนรู้จบที่เขียนกันมากคือ วงวนที่วนเป็นจำนวนครั้งที่กำหนดไว้คงตัว หรือกำหนดจำนวนรอบตามที่เก็บในตัวแปร ตัวอย่างเช่น ต้องการเขียนส่วนของโปรแกรมที่ทำกลุ่มคำสั่งหนึ่งเป็นจำนวน 100 รอบ ก็เขียนได้ดังรูปที่ 3-5 โดยมีตัวแปร `k` ทำหน้าที่เป็นตัวนับจำนวนรอบที่ทำไปแล้ว `k` มีค่าเพิ่มขึ้นหนึ่งในแต่ละรอบ ดังนั้น จึงออกจากวงวนเมื่อ `k == 100` นั่นคือ เมื่อครบ 100 รอบแล้ว

```
int k = 0;
while (true) {
  // กลุ่มคำสั่งที่ต้องการทำซ้ำ 100 ครั้ง
  ...
  k++;
  if (k == 100) break;
}
```

รูปที่ 3-5 รูปแบบของวงวน ทำซ้ำเป็นจำนวนที่กำหนดไว้คงตัว

ในกรณีที่ต้องการให้วงวนทำซ้ำเป็นจำนวนตามค่าของตัวแปร `n` ก็เพียงแค่เปลี่ยนเลข 100 ในรูปที่ 3-5 เป็น `n` ได้ดังรูปที่ 3-6 แต่ถ้า `n` มีค่าเป็น 0 จะไม่ได้ทำศูนย์รอบ แต่ทำซ้ำมากมาย จึงควรย้ายคำสั่ง `if` ขึ้นมาไว้เป็นบรรทัดแรกในวงวน ดังรูปที่ 3-7 แต่ก็ยังมีปัญหาอีก ถ้า `n` มีค่าเป็นจำนวนลบ ซึ่งไม่ควรทำสักรอบ แต่จะทำเป็นจำนวนรอบมากมาย จึงต้องเปลี่ยนเงื่อนไขใน `if` เป็น `k >= n` ดังรูปที่ 3-8 จะทำงานถูกต้องสมบูรณ์ ต้องเข้าใจด้วยว่า การเขียนโปรแกรมต้องจู้จี้

จุกจิก รอบคอบ เมื่อใดเขียนโปรแกรมเสร็จ ต้องทำตนเองเป็นนักทดสอบโปรแกรมที่คอยจับผิดโปรแกรมที่ตนเองเขียน ☺

```
int k = 0;
while (true) {
  // กลุ่มคำสั่งที่ต้องการทำซ้ำ n ครั้ง
  ...
  k++;
  if (k == n) break;
}
```

รูปที่ 3-6 รูปแบบของวงวนทำซ้ำเป็นจำนวน n รอบ (ทำงานผิด เมื่อ n≤0)

```
int k = 0;
while (true) {
  if (k == n) break;
  // กลุ่มคำสั่งที่ต้องการทำซ้ำ n ครั้ง
  ...
  k++;
}
```

รูปที่ 3-7 รูปแบบของวงวนทำซ้ำเป็นจำนวน n รอบ (ทำงานผิดเมื่อ n <0)

```
int k = 0;
while (true) {
  if (k >= n) break;
  // กลุ่มคำสั่งที่ต้องการทำซ้ำ n ครั้ง
  ...
  k++;
}
```

รูปที่ 3-8 รูปแบบของวงวนทำซ้ำเป็นจำนวน n รอบ

โปรแกรมหาค่าเฉลี่ย (3 แบบ)

ขอนำรูปแบบการทำงานของวงวนรู้จบ มาเขียนโปรแกรมหาค่าเฉลี่ยของชุดข้อมูล ที่ผู้ใช้ป้อนทางแป้นพิมพ์ ทบทวนเล็กน้อย ค่าเฉลี่ยของชุดข้อมูลหนึ่ง หาได้จากการหารผลรวมของชุดข้อมูลนั้นด้วยจำนวนข้อมูล เริ่มด้วยแบบง่าย กำหนดให้หาค่าเฉลี่ยของข้อมูล 4 จำนวน ดังนั้นสามารถนำรูปแบบของวงวนในรูปที่ 3-8 มาปรับให้วนทำ 4 รอบ (จะนำรูปแบบของรูปที่ 3-5 มาใช้ก็ได้ แต่ขอใช้รูปที่ 3-8 แบบเดียว จะได้จำแบบเดียว) ได้ดังรหัสที่ 3-11 ก่อนเข้าวงวนให้เตรียมตัวอ่านแป้นพิมพ์ kb, ตัวแปร k ไว้นับจำนวนรอบ, และตัวแปร sum เป็นตัวแปรเสริมเพื่อเก็บผลรวมของข้อมูล ภายในวงวนรอรับข้อมูลตัวใหม่ทางแป้นพิมพ์ บวกเพิ่มเข้าใน sum (บรรทัดที่ 11) เมื่อออกจากวงวน ก็หารผลบวกนี้ด้วย 4 ได้ค่าเฉลี่ย แสดงทางจอภาพ

คราวนี้เขียนอีกแบบ โดยไม่กำหนดจำนวนข้อมูลไว้คงตัว แต่จะถามผู้ใช้ก่อนว่า ข้อมูลมีกี่ตัว แล้วจะวนรับข้อมูลเป็นจำนวนรอบตามนั้น เราใช้รูปแบบของวงวนในรูปที่ 3-8 มาเขียนได้ดัง

รหัสที่ 3-12 ใช้ตัวแปร n รับจำนวนข้อมูลที่ถามจากผู้ใช้ (บรรทัดที่ 7) แล้วเข้าวงวนรับข้อมูลบวกเพิ่มเข้าตัวแปร sum เมื่อออกจากวงวน ได้ค่าเฉลี่ยที่ต้องการเท่ากับ sum/n

```
01 import java.util.Scanner;
02 // โปรแกรมหาค่าเฉลี่ยจากข้อมูล 4 จำนวน
03 public class Average1 {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     double sum = 0; // sum เก็บผลบวกชุดข้อมูล
07     int k = 0;      // k ไว้นับจำนวนรอบ
08     while (true) {
09       if (k >= 4) break;
10       System.out.print("จำนวนที่ " + (k+1) + " = ");
11       sum += kb.nextDouble(); // อ่านข้อมูล แล้วเพิ่มเข้าผลรวม
12       k++;
13     }
14     System.out.println("ค่าเฉลี่ย = " + (sum / 4));
15   }
16 }
```

```
Average1.java
JLab>java Average1
จำนวนที่ 1 = 2
จำนวนที่ 2 = 3
จำนวนที่ 3 = 4
จำนวนที่ 4 = 5
ค่าเฉลี่ย = 3.5
JLab>
Ready
```

รหัสที่ 3-11 โปรแกรมหาค่าเฉลี่ยจากข้อมูล 4 จำนวน

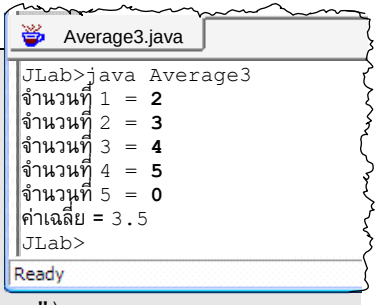
```
01 import java.util.Scanner;
02 // โปรแกรมหาค่าเฉลี่ยจากข้อมูล n จำนวนจากผู้ใช้
03 public class Average2 {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("จำนวนข้อมูล = ");
07     int n = kb.nextInt();
08     double sum = 0;
09     int k = 0;
10     while (true) {
11       if (k >= n) break;
12       System.out.print("จำนวนที่ " + (k+1) + " = ");
13       sum += kb.nextDouble();
14       k++;
15     }
16     System.out.println("ค่าเฉลี่ย = " + (sum / n));
17   }
18 }
```

```
Average2.java
JLab>java Average2
จำนวนข้อมูล = 4
จำนวนที่ 1 = 2
จำนวนที่ 2 = 3
จำนวนที่ 3 = 4
จำนวนที่ 4 = 5
ค่าเฉลี่ย = 3.5
JLab>
Ready
```

รหัสที่ 3-12 โปรแกรมหาค่าเฉลี่ยจากข้อมูล n จำนวน (ผู้ใช้ต้องป้อนค่าของ n ก่อน)

บางครั้งผู้ใช้ไม่รู้จำนวนข้อมูลที่จะป้อน (หรืออาจเป็นกรณีที่ผู้ใช้ไม่อยากนับจำนวนข้อมูลที่จะป้อน) เราจะอำนวยความสะดวกให้ โดยรอรับข้อมูลจากผู้ใช้ไปเรื่อย ๆ เพียงแต่บอกผู้ใช้ว่าเมื่อใดต้องการหยุดป้อนข้อมูล ให้ใส่ข้อมูลที่มีค่าพิเศษที่ได้ตกลงกันก่อน (ถ้าป้อนข้อมูลนี้ ถือว่าป้อนข้อมูลหมดแล้ว) ในกรณีของโปรแกรมหาค่าเฉลี่ย ถ้าเรารู้ล่วงหน้าว่า ชุดข้อมูลที่จะหาค่าเฉลี่ยมีแต่จำนวนบวกเท่านั้น ก็สามารถกำหนดค่า 0 (หรือจำนวนลบ) ให้เป็นค่าพิเศษเพื่อบอกว่า

ข้อมูลหมดแล้ว เขียนเป็นโปรแกรมได้ดังรหัสที่ 3-13 มีตัวแปร k ไว้นับจำนวนข้อมูล ภายในวงวนใช้ตัวแปร v เก็บข้อมูลที่อ่านจากผู้ใช้ไว้ก่อน ยังไม่รวมเข้าตัวแปร sum เพราะต้องตรวจสอบก่อนว่า ถ้ามีค่าน้อยกว่าหรือเท่ากับศูนย์ จะได้ออกจากวงวน




```
01 import java.util.Scanner;
02 // โปรแกรมหาค่าเฉลี่ย (ผู้ใช้ให้ค่า 0 หรือน้อยกว่า ถือว่าหมด)
03 public class Average3 {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     double sum = 0;
07     int k = 0; // k ไว้นับจำนวนข้อมูล
08     while (true) {
09       System.out.print("จำนวนที่ " + (k+1) + " = ");
10       double v = kb.nextDouble();
11       if (v <= 0) break; // หลุดจากวงวน เมื่อข้อมูลหมดแล้ว
12       sum += v;
13       k++;
14     }
15     System.out.println("ค่าเฉลี่ย = " + (sum / k));
16   }
17 }
```

รหัสที่ 3-13 โปรแกรมหาค่าเฉลี่ยของชุดข้อมูลที่เป็นจำนวนบวก (ให้ค่า ≤ 0 ถือว่าข้อมูลหมดแล้ว)

การทดสอบจำนวนเฉพาะ

จำนวนเฉพาะ (prime number) คือจำนวนเต็มมากกว่า 1 ที่หารด้วย 1 และตัวเองลงตัวเท่านั้น ดังนั้น วิธีที่ง่ายสุดในการทดสอบว่า จำนวนเต็ม n เป็นจำนวนเฉพาะหรือไม่ ก็เพียงแค่ลองนำ 2, 3, 4, 5, ..., จนถึง $n-1$ ไปหาร n ถ้าไม่มีตัวใดหาร n ลงตัวเลย ย่อมแสดงว่า n เป็นจำนวนเฉพาะ แนวคิดนี้นำไปสู่การใช้วงวนเพื่อทดสอบการหารลงตัวของ $n \div k$ เริ่มค่า k ที่ 2 และเพิ่มค่าขึ้นรอบละ 1 โดยจะกระโดดออกจากวงวนเมื่อ k มีค่าเท่ากับ n ที่แสดงว่า k ทุกค่าหาร n ไม่ลงตัวเลย (นั่นคือ n เป็นจำนวนเฉพาะ) หรืออีกกรณีที่จะกระโดดจากวงวน เมื่อพบ k ที่หาร n ลงตัว (นั่นคือพบว่า k เป็นตัวประกอบของ n) เขียนเป็นโปรแกรมได้ดังรหัสที่ 3-14

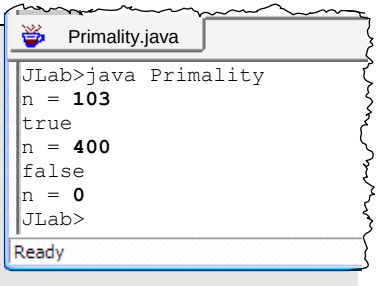
โปรแกรมนี้ใช้รูปแบบวงวนคล้ายกับรูปที่ 3-8 ต่างกันตรงที่เราเริ่มค่า k = 2 จบที่ n และมีเงื่อนไขของการกระโดดออกจากวงวนสองเงื่อนไขที่บรรทัดที่ 10 และ 11 เงื่อนไขแรกเมื่อ k>=n และเงื่อนไขที่สอง เมื่อพบว่า k หาร n ลงตัว การทดสอบว่า k หาร n ลงตัวหรือไม่ ก็คือการถามว่า n หารด้วย k เหลือเศษเป็น 0 หรือไม่ (บรรทัดที่ 11) เนื่องจากสามารถหลุดจากวงวนได้สองกรณี เมื่อมาถึงบรรทัดที่ 14 ถ้า k == n แสดงว่า n เป็นจำนวนเฉพาะ ถ้า k < n แสดงว่า n เป็นจำนวนประกอบ บรรทัดที่ 14 จึงนำผลการเปรียบเทียบ k == n ไปแสดงทางจอภาพ ถ้ามีค่าจริง println จะแสดงคำว่า true ถ้าเป็นเท็จ จะแสดงคำว่า false

```
01 import java.util.Scanner;
02 // โปรแกรมทดสอบว่าจำนวนเต็มที่ได้รับเป็นจำนวนเฉพาะหรือไม่
03 public class Primality {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("n = ");
07     int n = kb.nextInt();
08     int k = 2;                    // ลองหารด้วย k = 2, 3, ..., n-1
09     while (true) {
10       if (k >= n) break;          // หลุดจากวงวนเมื่อลองหารครบทุกตัวแล้ว
11       if ((n % k) == 0) break;    // หลุดจากวงวนเมื่อพบ k ที่หาร n ลงตัว
12       k++;
13     }
14     System.out.println(k == n); // ถ้า k เท่ากับ n แสดงว่า n เป็นจำนวนเฉพาะ
15   }
16 }
```

รหัสที่ 3-14 โปรแกรมทดสอบว่าจำนวนเต็มที่ได้รับเป็นจำนวนเฉพาะหรือไม่

เรานำรหัสที่ 3-14 มาปรับปรุงให้วนรับจำนวนเพื่อทดสอบความเป็นจำนวนเฉพาะได้หลายตัว จนกว่าผู้ใช้จะใส่ `n <= 1` ก็จะเลิกทำงาน เขียนเป็นโปรแกรมดังรหัสที่ 3-15 แสดงให้เห็นว่าเราสามารถใส่วนวงหนึ่งซ้อนไว้ภายในอีกวงวนหนึ่งได้ และขอให้สังเกตว่า การกระโดด (ด้วย `if...break`) จะออกจากวงวนที่คำสั่งนี้อยู่เท่านั้น ดังนั้น `break` ที่บรรทัดที่ 9 จะออกมาทำต่อที่บรรทัดที่ 18 ส่วน `break` ที่บรรทัดที่ 12 และ 13 จะออกมาทำต่อที่บรรทัดที่ 16

```
01 import java.util.Scanner;
02 // โปรแกรมทดสอบว่าจำนวนเต็มที่ได้รับเป็นจำนวนเฉพาะหรือไม่
03 public class Primality {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     while (true) { // วงวนรับจำนวนจากผู้ใช้
07       System.out.print("n = ");
08       int n = kb.nextInt();
09       if (n <= 1) break;
10       int k = 2;
11       while (true) { // วงวนทดสอบจำนวนเฉพาะ
12         if (k >= n) break;
13         if ((n % k) == 0) break;
14         k++;
15       }
16       System.out.println(k == n);
17     }
18   }
19 }
```



รหัสที่ 3-15 โปรแกรมทดสอบความเป็นจำนวนเฉพาะ แบบวนรับจากผู้ใช้ได้หลายตัว

การบรรยายแนวคิดของโปรแกรม

โปรแกรมคอมพิวเตอร์บรรยายขั้นตอนวิธีการทำงานที่นักเขียนโปรแกรมต้องการ เพื่อให้ตัวแปลโปรแกรมสร้างรหัสเครื่องให้คอมพิวเตอร์ทำงานตาม ในกรณีที่ขั้นตอนวิธีการทำงานมีความซับซ้อน นักเขียนโปรแกรมอาจใช้วิธีการบรรยายขั้นตอนวิธีนั้นคร่าว ๆ ก่อนลงมือเขียนโปรแกรม โดยทั่วไปการบรรยายแนวคิดนี้มี 3 วิธีคือ บรรยายด้วยข้อความ บรรยายด้วยรหัสเทียม และบรรยายด้วยผังงาน

การบรรยายด้วยข้อความ

ในกรณีที่ขั้นตอนวิธีไม่ซับซ้อน สามารถใช้ข้อความบรรยายแนวคิดการทำงาน เราได้ใช้วิธีนี้มาตลอดตั้งแต่โปรแกรมแรกที่เขียนในบทที่ 2 เช่น เราบรรยายการทำงานของโปรแกรมนาฬิกาเข็มวินาทีด้วยข้อความที่แสดงในรูปที่ 3-9

> **โปรแกรมนี้ใช้การวาดเส้นตรงซึ่งอาศัยการกำหนดจุดเริ่มต้น ความยาว และมุมของเส้นมาทำเป็นเข็มนาฬิกา ใช้วงวนไม่รู้จบสั่งให้วาดเส้นตรงไปเรื่อย ๆ โดยต้องการให้เข็มวนหนึ่งรอบ 360 องศาในเวลา 1 นาที ดังนั้น ทุก ๆ วินาทีต้องวาดเส้นใหม่ที่มีมุมของเข็มลดลงครั้งละ 360/60 = 6 องศา**

รูปที่ 3-9 คำบรรยายการทำงานของโปรแกรมนาฬิกาเข็มเดียว

การบรรยายแบบนี้ไร้รูปแบบ สามารถใช้รูปภาพหรือใช้สูตรคณิตศาสตร์ ประกอบได้ตามความเหมาะสม แต่ควรบรรยายให้ตรงจุดตรงประเด็น รัดกุม อธิบายให้เข้าใจแนวคิดหลัก โดยไม่ต้องลงรายละเอียดของการเขียนรหัสต้นฉบับ นักเขียนโปรแกรมที่อ่านแล้วอาจต้องคิดต่อว่า จะเปลี่ยนแนวคิดที่อ่านมาเป็นรหัสต้นฉบับได้อย่างไร

การบรรยายด้วยรหัสเทียม

รหัสเทียม (pseudo code) มีลักษณะคล้ายกับภาษาคอมพิวเตอร์ แต่ไม่มีหลักไวยากรณ์ที่เคร่งครัด นักเขียนโปรแกรมทั่วไป อ่านแล้วเข้าใจ และสามารถเปลี่ยนไปเป็นรหัสต้นฉบับของภาษาคอมพิวเตอร์ที่ตนเองถนัดได้ เราเขียนรหัสเทียมไม่ได้เพื่อนำไปให้ตัวแปลโปรแกรมสร้างเป็นรหัสเครื่อง แต่เขียนให้คนอ่าน ให้นักเขียนโปรแกรมทั่วๆ ไป อ่านแล้วน่าจะเข้าใจก็พอ ดังนั้นไม่ต้องประกาศตัวแปร ไม่ต้องบอกตรง ๆ ว่าตัวแปรเป็นประเภทใด เพราะอ่านแล้วก็รู้ เช่น เขียน $x \leftarrow 15$ ก็น่าจะรู้ว่าต้องการให้ค่า 15 กับตัวแปร x ที่คงต้องเป็นตัวแปรที่เก็บจำนวน ไม่ต้องระบุด้วยว่าจำนวนเต็มหรือจำนวนจริง เพราะถ้าต้องการให้หารแบบจำนวนเต็ม ก็ควรเขียนให้ชัดเจน อาจใช้สัญลักษณ์ทางคณิตศาสตร์มาใช้ก็ได้ เช่น $x \leftarrow \sqrt{y}$ หรือจะเขียนว่า $x \leftarrow$ รากที่สองของ y ก็ไม่มีอะไรผิด เพราะอ่านแล้วสามารถเปลี่ยนเป็นคำสั่งในภาษาคอมพิวเตอร์ได้ง่าย ๆ รหัสที่ 3-16 แสดง

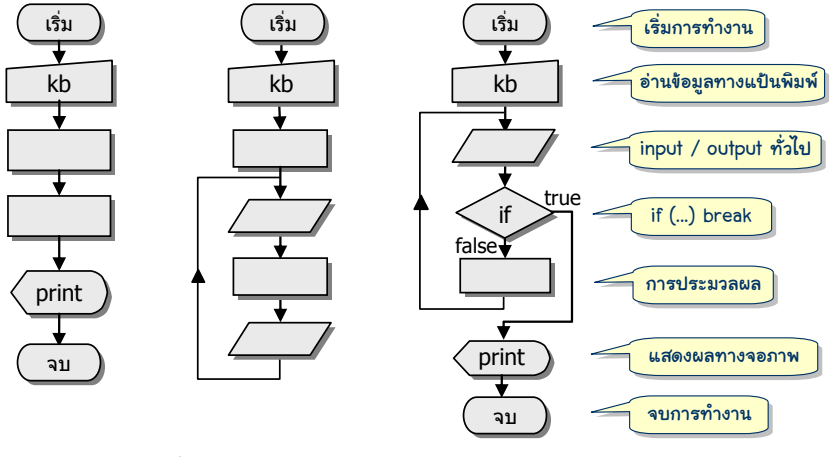
รหัสเทียมบรรยายการทำงานของนาฬิกาเข็มเดียว ให้สังเกตว่า เรามีอิสระที่จะใช้สัญลักษณ์ใดก็ได้มาแทนตัวแปรที่คิดว่าอ่านแล้วเข้าใจมากขึ้น อีกทั้งสามารถใช้ข้อความสั้นๆ บรรยายการทำงานของคำสั่งที่ต้องการให้ทำ แล้วให้นักเขียนโปรแกรมที่ลงรหัสต้นฉบับเลือกใช้คำสั่งจริงเองของภาษาหรือคลังคำสั่งที่ตนเองเลือกใช้

```
program SingleHandClock( {
  w ← create a window of size 200 and 200
  (x0, y0) ← center of w
  θ ← 90°, L ← 40% of window's width
  repeat forever {
    (x1, y1) ← (x0+L*cos(θ), y0+L*sin(θ))
    clear background of w
    draw a line on w from (x0, y0) to (x1, y1)
    pause for 1 second
    θ ← θ - 6°
  }
}
```

รหัสที่ 3-16 รหัสเทียมของโปรแกรมนาฬิกาเข็มเดียว

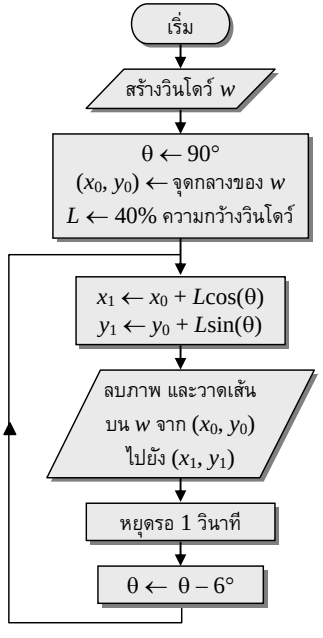
การบรรยายด้วยผังงาน

ผังงาน (flowchart) เป็นแผนภาพที่ใช้บรรยายกระบวนการ กรรมวิธี หรือขั้นตอนการทำงาน ผังงานประกอบด้วยสัญลักษณ์ที่แทนการทำงาน มีเส้นเชื่อมระหว่างสัญลักษณ์ เพื่อแสดงลำดับการทำงานของขั้นตอนต่าง ๆ จึงสามารถนำผังงานมาใช้บรรยายขั้นตอนการทำงานของโปรแกรมได้ รูปที่ 3-10 แสดงตัวอย่างผังงานสามรูปแบบ ผังงานทางซ้ายแสดงการทำงานเริ่มจากอ่านข้อมูลทางแป้นพิมพ์ ประมวลผล แล้วก็แสดงผล ผังงานตรงกลางแสดงการทำงานแบบวงวนไม่รู้จบ ในขณะที่ผังงานทางขวามีการเพิ่มเงื่อนไขให้หลุดจากวงวน



รูปที่ 3-10 ตัวอย่างผังงานแสดงการทำงานของโปรแกรม

รูปที่ 3-11 แสดงผังงานของโปรแกรมนาฬิกาเข็มเดียว กล่องสี่เหลี่ยมผืนผ้าแทนการประมวลผลทั่วไป ภายในมีรายละเอียดการทำงาน ซึ่งเป็นคำบรรยายที่ไม่มีกฎเกณฑ์ใด ๆ ควรเขียนให้สั้น รัดกุม เข้าใจง่าย การทำงานใดที่เกี่ยวกับการรับหรือส่งข้อมูลไปยังอุปกรณ์ขาเข้าขาออก (รวมทั้งแป้นพิมพ์และจอภาพ) สามารถเขียนไว้ภายในกล่องสี่เหลี่ยมด้านขนาน หรือจะใช้สัญลักษณ์เฉพาะของแป้นพิมพ์กับจอภาพดังแสดงในรูปที่ 3-10 ก็ได้ จุดเด่นของผังงานคือการแสดงให้เห็นอย่างชัดเจนถึงลำดับการทำงานของขั้นตอนต่าง ๆ ด้วยการไล่ตามเส้น รูปที่ 3-11 แสดงอย่างชัดเจนว่า การทำงานเป็นวงวนไม่รู้จบ



รูปที่ 3-11 ผังงานของโปรแกรมนาฬิกาเข็มเดียว

เพิ่มเติม

เราใช้ `if...break` เพื่อกระโดดออกจากวงวนที่คำสั่งนี้อยู่ แต่ในบางครั้งระหว่างการทำงานภายในวงวน ก็อาจต้องการกระโดดกลับไปทำบรรทัดแรกของวงวน `if ... continue` คือคำสั่งที่เมื่อเงื่อนไขหลัง `if` เป็นจริง `continue` จะทำให้วนกลับไปทำบรรทัดแรกของวงวนที่คำสั่งนี้อยู่ทันที มาลองดูตัวอย่างการใช้งานคำสั่งนี้กัน

ผู้เขียนเคยอ่านหนังสือ “The Magic of Mathematics” ของคุณ Theoni Pappas ซึ่งรวบรวมเรื่องราวทางคณิตศาสตร์ ขอนำเรื่องหนึ่งในหนังสือเล่มนี้มาเป็นตัวอย่าง กำหนดให้ a เป็นจำนวนเต็มสามหลักที่มีหลักร้อยไม่เหมือนหลักหน่วย ให้ b คือจำนวนเต็มที่ได้จากการสลับหลักร้อยกับ

หลักหน่วยของ a ให้ c คือผลต่างของ a กับ b และให้ d คือจำนวนเต็มที่ได้จากการสลับหลักร้อยกับหลักหน่วยของ c จะพบว่า $c + d$ มีค่าเท่ากับ 1089 เสมอ ตัวอย่างเช่น ให้ $a = 123$ จะได้ $b = 321$, $c = 321 – 123 = 198$, $d = 891$, และสุดท้ายได้ $c + d = 198+981 = 1089$ ตามที่บอก

ด้วยความสงสัยว่า คำกล่าวข้างต้นเป็นจริงหรือไม่ จึงขอใช้พลังของคอมพิวเตอร์ช่วยลุยพิสูจน์ แทนที่จะพิสูจน์ด้วยวิธีทางคณิตศาสตร์ เราจะเขียนโปรแกรมเพื่อตรวจสอบว่า ด้วยการนำจำนวนเต็มทุกตัวที่อยู่ในเกณฑ์ มาทำตามขั้นตอน และตรวจสอบว่าจะได้ 1089 ทุกๆ กรณีหรือไม่ โดยใช้วงวนที่เราใช้กันมาหลายครั้งแล้ว แจงค่า a ที่เราสนใจตั้งแต่ 1 (ซึ่งก็คือ 001) เพิ่มทีละหนึ่งไปจนถึง 999 เมื่อ a มีค่า 1000 ก็จะเลิกจากวงวน เนื่องจากเราต้องหยิบหลักหน่วย หลักสิบ และหลักร้อยของ a มาใช้ หลักหน่วยของ a ก็คือเศษของการหาร a ด้วย 10 นั่นคือ a % 10 ได้หลักหน่วย ส่วนหลักร้อยของ a ก็เริ่มด้วยการขจัดหลักหน่วยของ a ออกก่อนด้วยการหาร a ด้วย 10 แล้วตัดเศษทิ้ง จากนั้นก็นำผลที่ได้ไปหยิบหลักหน่วย นั่นคือ (a / 10) % 10 จะได้หลักสิบ ผู้อ่านลองคิดต่อเองว่า หลักร้อยของ a หาได้อย่างไร เนื่องจากความมหัศจรรย์ 1089 ใช้ได้เฉพาะกับจำนวนเต็ม 3 หลักที่หลักร้อยต้องต่างกับหลักหน่วย ดังนั้น ถ้าพบว่าหลักร้อยเหมือนหลักหน่วยต้องข้ามไปไม่ตรวจ เราจึงนำ `if...continue` มาใช้เพื่อการนี้ ดังรหัสที่ 3-17

```
01 public class Magic1089 {
02   public static void main(String[] args) {
03     int a = 0;
04     while (true) {
05       a++;
06       if (a > 999) break;
07       int a0 = a % 10;             // หลักหน่วยของ a
08       int a1 = (a / 10) % 10;      // หลักสิบของ a
09       int a2 = a / 100;            // หลักร้อยของ a
10       if (a0 == a2) continue;      // หลักหน่วยเหมือนหลักสิบ ข้ามตัวนี้ ทำตัวต่อไป
11       int b = a0 * 100 + a1 * 10 + a2; // สลับหลักหน่วยกับหลักร้อย
12       int c = Math.abs(b - a);
13       int c0 = c % 10;
14       int c1 = (c / 10) % 10;
15       int c2 = c / 100;
16       int d = c0 * 100 + c1 * 10 + c2;
17       if ((c + d) != 1089) break; // ถ้าพบตัวที่ไม่เท่า 1089 รีบเลิกได้แล้ว
18     }
19     System.out.println("มหัศจรรย์ 1089 เป็น " + (a==1000) );
20   }
21 }
```

รหัสที่ 3-17 โปรแกรมทดสอบมหัศจรรย์ 1089

ก่อนเข้าวงวนตัวแปร `a = 0` บรรทัดแรกของวงวนเพิ่มค่า `a` แล้วตรวจสอบว่า ถ้าเกิน 999 ก็เลิกจากวงวนได้ (บรรทัดที่ 6) สามบรรทัดต่อมาดึงเลขโดดทั้งสามหลักของ `a` ออกมา แล้วก็ตรวจสอบ (ด้วย `if...continue`) ในบรรทัดที่ 10 ว่า ถ้าหลักหน่วยเท่ากับหลักร้อย ให้กลับไป

ทำต่อบรรทัดที่ 5 เป็นการข้ามการตรวจสอบความมหัศจรรย์เมื่อพบจำนวนเต็มที่หลักร้อยเหมือนหลักหน่วย แต่ถ้าไม่เท่า เงื่อนไขในบรรทัดที่ 10 เป็นเท็จ ก็ทำต่อบรรทัดที่ 11 สร้างจำนวนเต็มที่ได้จากการสลับหลักหน่วยและหลักร้อยของ a เก็บในตัวแปร b ทำต่อตามขั้นตอนจนถึงบรรทัดที่ 16 จะได้ค่าของ c และ d นำผลรวมมาตรวจสอบว่า มีค่า 1089 หรือไม่ในบรรทัดที่ 17 ถ้าเกิดพบว่าไม่เท่า ก็ให้รีบเลิกจากวงวนทันที ถ้าเท่าก็วนกลับไปทำรอบถัดไปที่บรรทัดที่ 5 ดังนั้น ถ้าออกจากวงวนที่บรรทัดที่ 19 แล้ว a มีค่า 1000 แสดงว่า การตรวจสอบผ่านทุกด่าน ความมหัศจรรย์ 1089 ก็เป็นจริง ถ้า a ไม่เท่ากับ 1000 ก็แสดงว่า คำกล่าวอ้างไม่จริง จึงสามารถใช้ผลของการเปรียบเทียบ (a==1000) เป็นผลลัพธ์ของการทำงาน

แบบฝึกหัด

1. อยากทราบว่า วงวนแต่ละวงข้างล่างนี้ พิมพ์ * กี่ตัว

```
int k = 0;
while (true) {
  if (k >= n) break;
  System.out.print("*");
  k++;
}
```

```
int k = 0;
while (true) {
  System.out.print("*");
  k++;
  if (k > n) break;
}
```

```
int k = n;
while (true) {
  System.out.print("*");
  k--;
  if (k != 0) break;
}
```

```
int k = n;
while (true) {
  System.out.print("*");
  k--;
  if (k == 0) break;
}
```

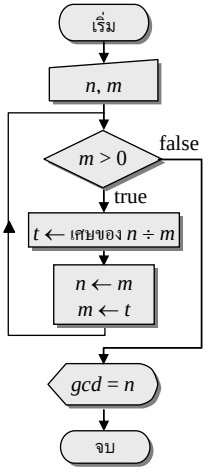
2. จงเขียนผังงานและโปรแกรมพิมพ์ตารางสูตรคูณ ตั้งแต่แม่ 2 ถึงแม่ 12

3. จงเขียนผังงานและโปรแกรมรับจำนวนเต็มจากผู้ใช้หนึ่งจำนวน แล้วแสดงจำนวนเต็มอีกจำนวนหนึ่งที่มีเลขโดดที่กลับลำดับกับจำนวนที่รับเข้ามา เช่น รับ 123456 จะแสดง 654321, รับ 915 แสดง 519 เป็นต้น

4. จงเขียนผังงานและโปรแกรมเพื่อหาจำนวนเฉพาะตัวที่มีค่ามากสุดที่น้อยกว่า 100,000,000

5. จำนวนจริงแบบ double ในจาวานั้น แทนค่าได้ไม่แม่นยำ 100% จงเขียนโปรแกรมเพื่อหาจำนวนเต็มบวก x ที่มีค่าน้อยสุด ที่ทำให้การคำนวณ 1.0 / x * x มีค่าไม่เท่ากับ $\frac{x}{1}$

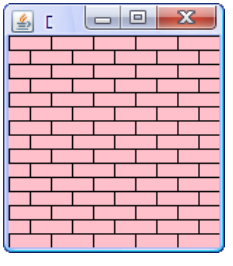
6. ผังงานข้างขวานี้แสดงขั้นตอนการทำงานของโปรแกรมที่รับจำนวนเต็ม n กับ m มาหาค่าหารร่วมมาก (หรม.) ของ n และ m จงเขียนโปรแกรมที่ทำงานตามขั้นตอนที่แสดงในผังงานนี้



7. จงเขียนโปรแกรมหาค่า π จากสูตร $\pi = 4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\frac{1}{11}+\cdots\right)$ โดยสิ่งที่อยากรู้คือ ต้องบวกลบบวกลบเช่นนี้ไปสักกี่พจน์ จึงจะได้ค่าที่ถูกต้องถึงทศนิยมตำแหน่งที่หก นั่นคือ ได้ค่าเป็น 3.141592xxx

8. จงเขียนโปรแกรมหาค่า π จากสูตร $\pi = 4\left(\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdot\frac{6}{7}\cdot\frac{8}{7}\cdot\frac{8}{9}\cdots\right)$ โดยสิ่งที่อยากรู้คือ ต้องคูณเช่นนี้ไปสักกี่พจน์ จึงจะได้ค่าที่ถูกต้องถึงทศนิยมตำแหน่งที่หก นั่นคือ ได้ค่าเป็น 3.141592xxx และให้ลองเปรียบเทียบว่า การหาค่า π ของข้อนี้กับข้อที่แล้ว วิธีใดได้ผลที่เร็วกว่ากัน

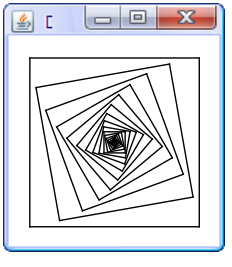
9. จงเขียนโปรแกรมเพื่อแสดงภาพผนังอิฐ ดังตัวอย่างที่แสดงในรูปขวานี้ ด้วยการวาดรูปสี่เหลี่ยมผืนผ้าโดยใช้คำสั่ง `drawRect` ของ `DWindow` ซึ่งมีรูปแบบการใช้งานดังนี้



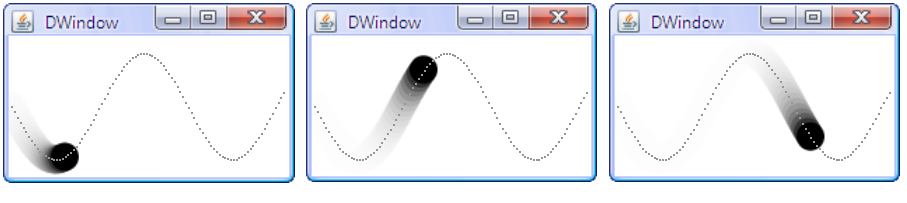
```
w.drawRect(x, y, width, height)
```

แทนการวาดสี่เหลี่ยมผืนผ้ากว้าง `width` สูง `height` บนวินโดว์ `w` ที่มีมุมซ้ายของสี่เหลี่ยมอยู่ที่พิกัด `(x,y)`

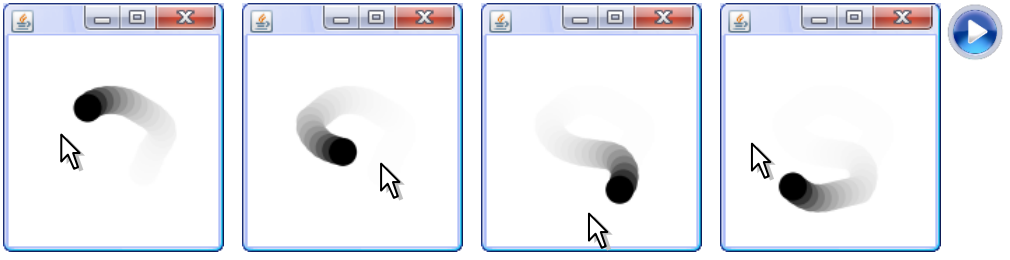
10. จงเขียนโปรแกรมแสดงรูปสี่เหลี่ยมจัตุรัสหลายรูปที่หมุนและซ้อนกัน ดังตัวอย่างที่แสดงในรูปขวานี้ โดยอาศัยส่วนของโปรแกรมที่เคยเขียนในแบบฝึกหัดท้ายบทที่ 2 ที่สามารถวาดสี่เหลี่ยมจัตุรัสที่เอียงได้ (รูปทางขวานี้วาดสี่เหลี่ยม 20 รูป แต่ละรูปในมีความยาวด้านเป็น 80% ของความยาวด้านของรูปนอก และมีมุมเอียงจากรูปนอกไปอีก 10°



11. จงเขียนโปรแกรมแสดงลูกบอลเคลื่อนที่แบบคลื่นขึ้นลง ดังตัวอย่างที่แสดงในรูปข้างล่างนี้ (เส้นประที่แสดงในรูปถูกวาดเพิ่มเข้าไปเองเพื่อให้เห็นเส้นทางการเคลื่อนที่ของลูกบอล โปรแกรมที่เขียนไม่ต้องแสดงเส้นประ)



12. จงเขียนโปรแกรมแสดงลูกบอลเคลื่อนที่ในวินโดว์ โดยจะเคลื่อนที่ไล่ตามตัวชี้เมาส์ ดังตัวอย่างที่แสดงในรูปข้างล่างนี้

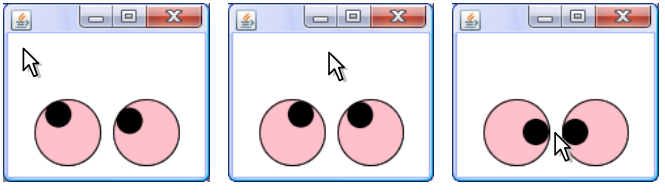


ข้อแนะนำ : การบังคับในลูกบอลวิ่งไล่ตัวชี้เมาส์ อาศัยการปรับตำแหน่งของลูกบอลไปในทิศที่ตัวชี้เมาส์อยู่ นั่นคือ ถ้า x_m กับ y_m เป็นตำแหน่งของตัวชี้เมาส์ และ x กับ y คือตำแหน่งปัจจุบันของลูกบอล จะได้

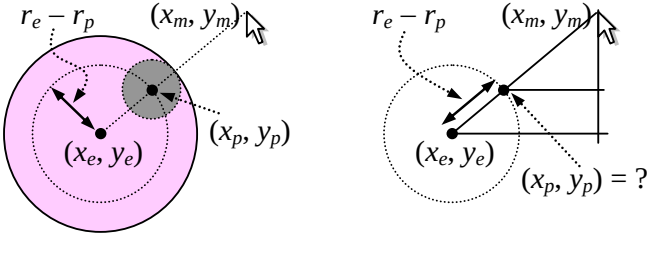
$$x + V(x_m - x) / \sqrt{(x_m - x)^2 + (y_m - y)^2} \quad \text{และ} \quad y + V(y_m - y) / \sqrt{(x_m - x)^2 + (y_m - y)^2}$$

คือตำแหน่งใหม่ของลูกบอล โดยที่ V คือความเร็วในการเคลื่อนที่

13. จงเขียนโปรแกรมแสดงตาสองดวงที่มองตามตัวชี้เมาส์ ดังตัวอย่างที่แสดงในรูปข้างล่างนี้



ข้อแนะนำ : สิ่งที่ต้องคำนวณหา คือ จุดศูนย์กลางของตาดำ กำหนดให้ x_m และ y_m เป็นตำแหน่งของตัวชี้เมาส์, x_e และ y_e เป็นจุดศูนย์กลางของลูกตา, r_e และ r_p คือรัศมีของลูกตาและตาดำตามลำดับ และให้วงกลม P คือวงกลมที่มีจุดศูนย์กลางที่ (x_e, y_e) และมีรัศมี $(r_e - r_p)$ จากรูปข้างล่างนี้ P คือวงกลมเส้นไข่ปลา จะได้ว่าจุดศูนย์กลางของตาดำ x_p และ y_p หาได้จากการลากเส้นตรงจาก (x_e, y_e) ไปยัง (x_m, y_m) แล้วดูว่า ตัดเส้นรอบวงของวงกลม P ที่ใด

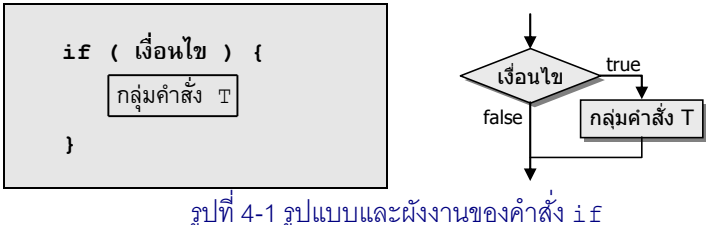


เลือกปฏิบัติ

การโปรแกรมให้เครื่องคอมพิวเตอร์รู้จักเลือกปฏิบัติ ทำอะไรบางอย่างขึ้นกับเงื่อนไขที่กำหนดไว้ ทำให้ได้โปรแกรมที่ “ฉลาด” บทนี้ว่าด้วยคำสั่ง `if` และ `if-else` เพื่อทดสอบเงื่อนไขที่เขียนในรูปของนิพจน์ตรรกะซึ่งให้ผลเป็นค่าจริงหรือเท็จ คำสั่งอาจจะอยู่ในรูปของการให้ทำเมื่อเงื่อนไขเป็นจริงเท่านั้น หรือในรูปของการเลือกทำแบบจริงทำอย่าง เท็จทำอีกอย่าง นิพจน์ตรรกะบรรยายเงื่อนไขที่ได้จากการเปรียบเทียบข้อมูล ผสมกับการเชื่อมด้วยตัวดำเนินการตรรกะ “และ” “หรือ” “ไม่” เพื่อบรรยายเงื่อนไขให้สื่อความหมาย การผสมการทำงานแบบวงวน ที่ภายในมีการเลือกปฏิบัติและการคำนวณ ส่งผลให้เขียนโปรแกรมคอมพิวเตอร์ที่แก้ไขปัญหาได้หลากหลายขึ้น

จริงถึงจะทำ

บทที่แล้วเราได้พบคำสั่ง `if...break` ซึ่งระบุเงื่อนไขให้กระโดดออกจากวงวนเมื่อเงื่อนไขนั้นเป็นจริง หากแทนคำสั่ง `break` ด้วยคำสั่งอื่น จะได้ลักษณะการทำงานที่แตกต่างออกไป รูปที่ 4-1 แสดงรูปแบบคำสั่งและผังงานของคำสั่ง `if` ซึ่งสั่งให้กลุ่มคำสั่งหนึ่งทำงานก็เฉพาะเมื่อเงื่อนไขที่เขียนไว้เป็นจริง เช่น คำสั่ง `if (a < 0) {a = a + 1;}` หมายความว่า ถ้า `a` มีค่าน้อยกว่า 0 จะเพิ่มค่าใน `a` อีกหนึ่ง ถ้าเงื่อนไขไม่เป็นจริงก็ไม่ทำ



รูปที่ 4-1 รูปแบบและผังงานของคำสั่ง `if`

การทดสอบความถูกต้องของข้อมูลขาเข้า

โปรแกรมที่มีการรับข้อมูลขาเข้าจากผู้ใช้ จะทำงานถูกต้องก็เมื่อข้อมูลนั้นต้องเป็นไปตามข้อกำหนด เช่น รหัสที่ 4-1 แสดงส่วนของโปรแกรมหารากที่สอง (จากรหัส 3-10) ซึ่งจะทำงานถูกต้องก็เมื่อค่า `a` ที่รับเข้ามา มีค่าไม่ติดลบ ผู้อ่านมองออกไหมว่า ถ้า `a<0` โปรแกรมจะทำงานอย่างไร [1] โปรแกรมที่ดีควรตรวจสอบความถูกต้องของข้อมูลขาเข้าด้วย เพื่อเตือนผู้ใช้ให้ทราบว่าโปรแกรมเราไม่ยอมรับข้อมูลที่ไม่ตรงตามข้อกำหนด รหัสที่ 4-2 แสดงส่วนของโปรแกรมที่หลังจากรับข้อมูลเข้ามาแล้ว ก็ทดสอบว่า ถ้า `a` มีค่าน้อยกว่า 0 จะทำคำสั่งภายใน `{ }` ซึ่งคือการแสดงข้อความแจ้งให้ผู้ใช้ทราบ ตามด้วยบังคับให้โปรแกรมเลิกทำงานด้วยคำสั่ง `System.exit(-1)`

```
double a = kb.nextDouble();
double x = 1;
while (true) {
  x = (x + a/x) / 2.0;
  if ((x*x - a) < 1e-5) break;
}
System.out.println("รากที่สองของ " + a + " = " + x);
```

รหัสที่ 4-1 ส่วนของโปรแกรมหา $\sqrt{a}$ ด้วยวิธีของชาวบาบิโลน

```
double a = kb.nextDouble();
if (a < 0) {
  System.out.println("กรุณาใส่ a เป็นจำนวนบวก");
  System.exit(-1);
}
```

รหัสที่ 4-2 ส่วนของโปรแกรมเพื่อตรวจสอบข้อมูลขาเข้า

ลูกบอลเด้งได้

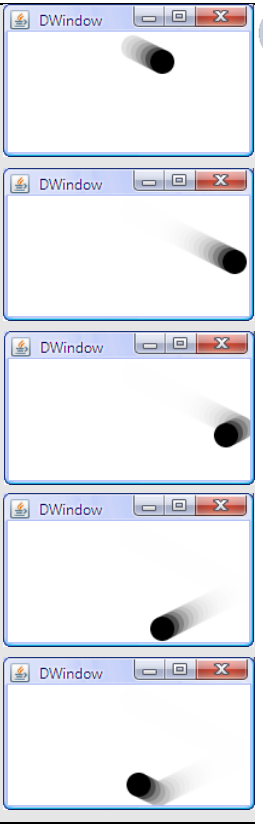
บทที่แล้วได้นำเสนอโปรแกรมเล็ก ๆ (รหัสที่ 3-8) ที่แสดงลูกบอลเคลื่อนที่ในวินโดว์ เราจะมาปรับเพื่อทำให้ลูกเด้งผนังได้ โดยใช้คำสั่ง `if` ที่ทดสอบว่า ถ้าลูกบอลชนผนังวินโดว์ ให้เด้งกลับ ปัญหาคือ รู้ได้อย่างไรว่าชนผนัง และจะทำให้เด้งกลับอย่างไร เรามีตัวแปร `x` และ `y` เก็บจุดศูนย์กลางของลูกบอล ให้ `r` คือรัศมีของลูกบอล จะได้ว่า `(x-r)` คือขอบซ้ายของลูกบอล ซึ่งหากน้อยกว่าหรือเท่ากับ 0 แสดงว่าลูกบอลชนหรือตกขอบซ้ายของวินโดว์ และในทำนองเดียวกัน `(x+r)` คือขอบขวาของลูกบอล ซึ่งหากมากกว่าหรือเท่ากับความกว้างวินโดว์ แสดงว่าลูกบอลชนหรือตกขอบขวาของวินโดว์ (ผู้อ่านคงทราบว่า เงื่อนไขการชนผนังด้านบนและด้านล่างควรเป็นเช่นไร) ส่วนการให้ลูกบอลเด้งกลับนั้น ก็ต้องมาพิจารณาอีกสองตัวแปรคือ `dx` กับ `dy` ที่มีไว้เปลี่ยนแปลงค่า `x` และ `y` ของลูกบอลในแต่ละรอบ การที่ `dx` มีค่าบวก แสดงว่าลูกบอลกำลังเคลื่อนไปทางขวา ดังนั้น ถ้าต้องการให้ลูกบอลกลับทิศการเคลื่อนที่ คือเคลื่อนกลับไปทางซ้าย ก็ทำให้ `dx`

---

1 [illegible]

เป็นลบ และในทำนองเดียวกัน หากลูกบอลกำลังเคลื่อนไปทางซ้าย (dx เป็นค่าลบ) แล้วต้องการให้เปลี่ยนทิศมาเคลื่อนไปทางขวา ก็ทำให้ dx เป็นค่าบวก ดังนั้น เมื่อพบว่าลูกบอลชนผนังขวาหรือซ้าย ต้องทำให้ค่าของ dx เปลี่ยนจากบวกเป็นลบ หรือเปลี่ยนจากลบเป็นบวก ซึ่งทำได้ด้วยคำสั่ง dx = 0-dx หรือ dx = -1*dx หรือ dx = -dx (แล้วถ้าลูกบอลชนผนังล่างหรือบนของวินโดว์ จะทำอย่างไร) นำแนวคิดนี้มาเขียนได้ดังรหัสที่ 4-3

```
01 import jlab.graphics.DWindow;
02 // โปรแกรมแสดงลูกบอลสีดำเคลื่อนที่ และเด้งผนังได้
03 public class BouncingBall {
04   public static void main(String[] args) {
05     DWindow w = new DWindow(200, 100);
06     double r = 10, x = 100, y = r+1;
07     double dx = 4, dy = 2;
08     while (true) {
09       x = x + dx;  y = y + dy;
10       if ((x-r) <= 0) {                // ชนด้านซ้าย
11         dx = -dx;  x = r;
12       }
13       if ((y-r) <= 0) {                // ชนด้านบน
14         dy = -dy;  y = r;
15       }
16       if ((x+r) >= w.getWidth() ) {  // ชนด้านขวา
17         dx = -dx;  x = w.getWidth() - r;
18       }
19       if ((y+r) >= w.getHeight()) {  // ชนด้านล่าง
20         dy = -dy;  y = w.getHeight() - r;
21       }
22       w.fade(0.3);
23       w.fillEllipse(w.BLACK, x, y, 2*r, 2*r);
24       w.sleep(50);
25     }
26   }
27 }
```



รหัสที่ 4-3 โปรแกรมแสดงลูกบอลสีดำเคลื่อนที่ และเด้งผนังวินโดว์ได้

วันใดของสัปดาห์

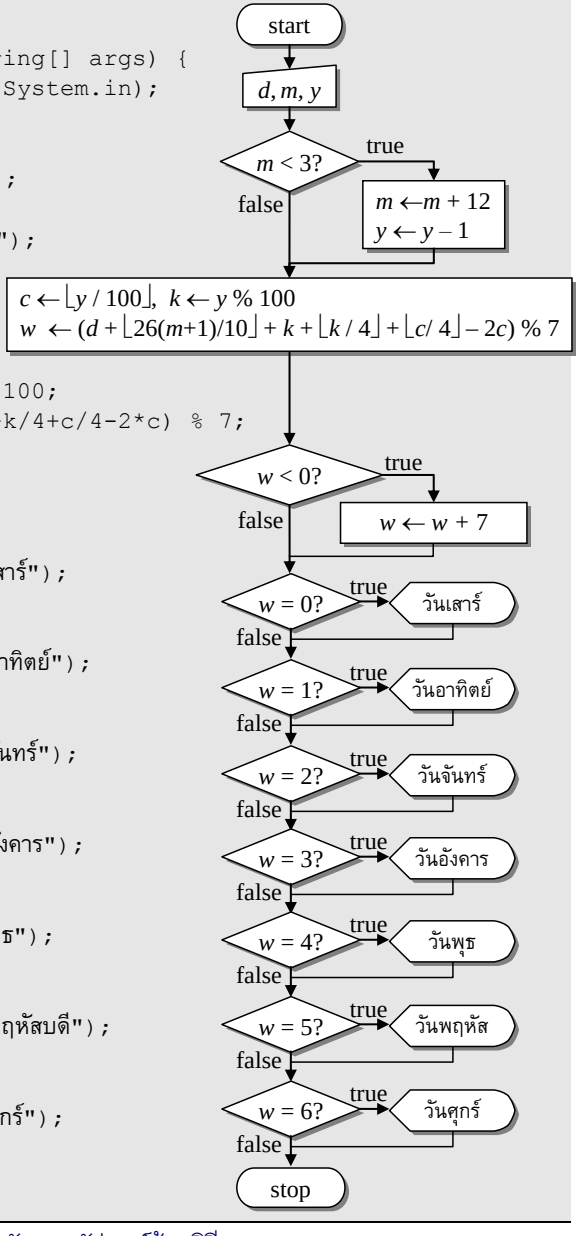
คุณ Christian Zeller นักคณิตศาสตร์ชาวเยอรมัน ได้ออกแบบขั้นตอนวิธีที่เรียกว่า Zeller's congruence เพื่อคำนวณว่า วันเดือนปี (d / m / y) ที่ได้รับคือวันใดของสัปดาห์ จากสูตรข้างล่างนี้

$$w = \left(d + \left\lfloor \frac{26(m+1)}{10} \right\rfloor + k + \left\lfloor \frac{k}{4} \right\rfloor + \left\lfloor \frac{c}{4} \right\rfloor - 2c \right) \bmod 7 \quad \text{โดยที่ } k = y \bmod 100,\ c = \left\lfloor \frac{y}{100} \right\rfloor$$

y คือเลขปี ค.ศ ส่วน m = 3, 4, ..., 13, 14 แปลกเล็กน้อย m = 13 แทนมกราคม และ 14 แทนกุมภาพันธ์ ถ้าเป็นสองเดือนพิเศษนี้ให้ลดค่าปีลง 1 ด้วย ผลของสูตร ถ้า w = 0 คือวันเสาร์, 1 คือวันอาทิตย์, ..., 6 คือวันศุกร์ สัญลักษณ์ $\lfloor x \rfloor$ คือจำนวนเต็มมากสุดที่ไม่เกิน x (ถ้า $x \geq 0$ ก็คือการตัด

เศษของ x ทิ้งนั่นเอง) และ $a \bmod b$ คือ $a - b\lfloor a/b \rfloor$ ซึ่งให้ผลเท่ากับ a % b เมื่อ a และ b เป็นบวกหรือเป็นลบทั้งคู่ แต่จากสูตรข้างต้น a อาจเป็นลบ ดังนั้น $(-1 \bmod 7 = 6) \neq$ (-1%7 = -1) ดังนั้น ถ้าใช้ % ทำ mod แล้วได้จำนวนลบ ก็ให้บวกอีก 7 ก็จะได้ค่าของ w ตามความหมายข้างต้น (หรือจะแทน $-2c$ ด้วย $+5c$ ในสูตรหา w ก็ได้) เราสามารถเขียนผังงานแสดงวิธีของคุณ Zeller และเขียนเป็นโปรแกรมได้ดังรหัสที่ 4-4

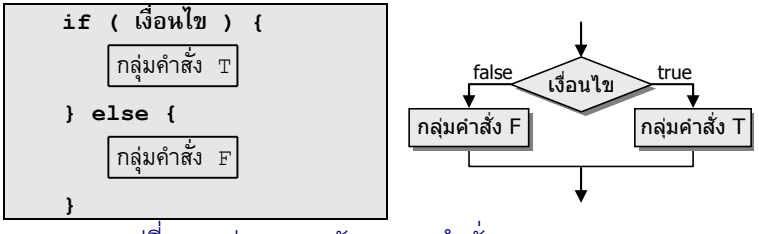
```
01 import java.util.Scanner;
02 public class DayOfWeek {
03   public static void main(String[] args) {
04     Scanner kb = new Scanner(System.in);
05     System.out.print("วัน = ");
06     int d = kb.nextInt();
07     System.out.print("เดือน = ");
08     int m = kb.nextInt();
09     System.out.print("ปี(ค.ศ.) = ");
10     int y = kb.nextInt();
11     if (m < 3) {
12       m = m + 12; y = y - 1;
13     }
14     int c = y / 100, k = y % 100;
15     int w = (d+26*(m+1)/10+k+k/4+c/4-2*c) % 7;
16     if (w < 0) {
17       w = w + 7;
18     }
19     if (w == 0) {
20       System.out.println("วันเสาร์");
21     }
22     if (w == 1) {
23       System.out.println("วันอาทิตย์");
24     }
25     if (w == 2) {
26       System.out.println("วันจันทร์");
27     }
28     if (w == 3) {
29       System.out.println("วันอังคาร");
30     }
31     if (w == 4) {
32       System.out.println("วันพุธ");
33     }
34     if (w == 5) {
35       System.out.println("วันพฤหัสบดี");
36     }
37     if (w == 6) {
38       System.out.println("วันศุกร์");
39     }
40   }
41 }
```



รหัสที่ 4-4 โปรแกรมหาวันของสัปดาห์ด้วยวิธีของคุณ Zeller

จริงทำอย่าง เท็จทำอย่าง

หากเราต้องการให้โปรแกรมเลือกทำ โดยถ้าเงื่อนไขที่ทดสอบเป็นจริงให้ทำกลุ่มคำสั่งหนึ่ง แต่ถ้าเป็นเท็จให้ทำอีกกลุ่มคำสั่ง ก็ใช้คำสั่ง `if-else` ซึ่งมีรูปแบบและผังงานดังรูปที่ 4-2 เช่น คำสั่ง `if ((n%2)==0) {s="even";} else {s="odd";}` จะให้ `s` มีค่า `"even"` เมื่อ `n` เป็นจำนวนคู่ (เพราะหารด้วย 2 ลงตัว) ถ้าไม่ใช่จะให้ `s` มีค่า `"odd"`



รูปที่ 4-2 รูปแบบและผังงานของคำสั่ง `if ... else`

ตัวน้อยตัวมาก

การหาตัวน้อยกว่าตัวมากกว่าของค่าในตัวแปร 2 ตัว เกิดขึ้นบ่อยมากในการเขียนโปรแกรม ให้ `a` และ `b` เป็นตัวแปร เราต้องการนำค่าของตัวน้อยกว่าเก็บใส่ตัวแปร `min` และนำตัวมากกว่าเก็บในตัวแปร `max` เขียนเป็นรหัสได้หลายแบบ ขอนำเสนอ 3 แบบดังแสดงในรหัสที่ 4-5

```
int min, max;
if (a < b) {
  min = a;
  max = b;
} else {
  min = b;
  max = a;
}
```

```
int min = b;
int max = a;
if (a < b) {
  min = a;
  max = b;
}
```

```
int min, max;
if (a < b) {
  min = a;
  max = b;
}
if (a >= b) {
  min = b;
  max = a;
}
```

รหัสที่ 4-5 ส่วนของโปรแกรมหาตัวน้อยและตัวมากของตัวแปรสองตัว

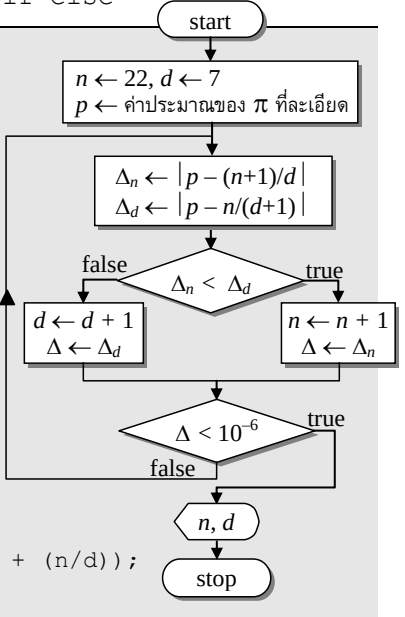
รหัสทางซ้ายใช้ `if-else` ถ้า `a<b` ก็ให้ `a` เป็นตัวน้อย `b` เป็นตัวมาก ไม่เช่นนั้น `b` ก็เป็นตัวน้อย และ `a` เป็นตัวมาก รหัสตรงกลางเริ่มให้ `min` ด้วย `b` และ `max` ด้วย `a` เลย แล้วค่อยตัดสินใจว่า ควรเปลี่ยนใจไปอีกแบบหรือไม่ ส่วนรหัสทางขวาใช้ `if` สองตัว ตัวแรกสำหรับกรณี `a<b` ส่วนตัวหลังสำหรับกรณี `a>=b` แต่รหัสทางขวานี้มีปัญหา ถ้าคำสั่งถัดไปมีการใช้ `min` หรือ `max` ตัวแปลโปรแกรมจะตีความว่า `min` และ `max` มีโอกาสไม่ได้รับการตั้งค่า ถือเป็นข้อผิดพลาด แม้ว่าเราจะเห็นชัดว่า เงื่อนไขของสอง `if` นี้ต้องมีตัวใดก็ตัวหนึ่งเป็นจริงแน่ๆ `min` และ `max` ต้องได้รับค่าแน่ๆ แต่ตัวแปลโปรแกรมไม่รู้ ในขณะที่รหัสทางซ้ายนั้นตัวแปลโปรแกรมมั่นใจว่าเงื่อนไข `a<b` ไม่จริงก็ต้องเท็จ ตัวแปร `min` และ `max` ต้องได้รับการตั้งค่าแน่ ๆ

ค่าประมาณของ π

π คือสัดส่วนของความยาวเส้นรอบวงกับความยาวเส้นผ่านศูนย์กลางของวงกลม π เป็นจำนวนอตรรกยะ ไม่สามารถเขียนในรูปเศษส่วนของจำนวนเต็ม n/d ถ้าเขียน π ในแบบที่มีจุดทศนิยมจะมีตัวเลขหลังจุดทศนิยมไม่ซ้ำและไม่รู้จบ π มีค่าประมาณ 3.14159265358979323... เราจะเขียนโปรแกรมในหัวข้อนี้เพื่อหาจำนวนตรรกยะที่ประมาณค่าของ π ได้ดีกว่า $^{22}/_{7}$ ที่เรารู้กันมาตั้งแต่เด็ก [2]

ขอตั้งความคาดหวังไว้ก่อนว่า จะหาค่า n/d ที่มี $|\pi - n/d| < 10^{-6}$ หากค่าของ n/d ยังไม่ใช่ค่าที่ต้องการ จำนวนตรรกยะที่ต้องพิจารณาตัวถัดไปคือ $(n+1)/d$ หรือไม่ก็ $n/(d+1)$ จะเลือกตัวไหนดี ? ก็ลองคำนวณ $|\pi - (n+1)/d|$ กับ $|\pi - n/(d+1)|$ เลือกตัวที่ทำให้ได้ค่าน้อย แนวคิดแบบนี้นำไปสู่การใช้วงวน ในแต่ละรอบ ไม่เพิ่มเศษ ก็เพิ่มส่วน ขึ้นกับการทดสอบว่า เพิ่มแบบใดได้ค่าที่ดีกว่า วนเช่นนี้ไปจนกว่าจะถึงเป้าหมายของความแม่นยำที่ตั้งไว้ การเลือกเพิ่มเศษหรือเพิ่มส่วนขึ้นกับผลการทดสอบว่าจริงหรือเท็จ จึงเหมาะกับการใช้คำสั่ง `if-else`

```
01 public class ApproxPi {
02   public static void main(String[] args) {
03     double n = 22, d = 7;
04     double pi = 3.141592653589793;
05     double diff, diff1, diff2;
06     while (true) {
07       diff1 = Math.abs(pi - (n + 1) / d);
08       diff2 = Math.abs(pi - n / (d + 1));
09       if (diff1 < diff2) {
10         n = n + 1;
11         diff = diff1;
12       } else {
13         d = d + 1;
14         diff = diff2;
15       }
16       if (diff < 1e-6) break;
17     }
18     System.out.println(n + "/" + d + " = " + (n/d));
19   }
20 }
```



รหัสที่ 4-6 โปรแกรมหาจำนวนตรรกยะ n/d ที่มีค่าต่างจาก π น้อยกว่า 10^{-6}

รหัสที่ 4-6 แสดงโปรแกรมหาค่าของ `n/d` ที่ต้องการ มีตัวแปร `pi` ซึ่งเก็บค่าประมาณของ π ที่มีความละเอียดถึง 16 ตำแหน่ง มีตัวแปร `n` และ `d` เริ่มประมาณค่า π ที่ `n/d` = $^{22}/_{7}$ เราใช้วงวน `while(true)` ประกอบกับเงื่อนไขการออกจากวงวนในบรรทัดที่ 16 ซึ่งถ้าเป็นจริงแสดงว่าค่าของ `n/d` กับ `pi` ต่างกันน้อยกว่าเกณฑ์ที่ตั้งไว้ บรรทัดที่ 7 และ 8 เป็นการลองเพิ่มเศษ กับ

[2] ปี ค.ศ. 480 祖沖之 (Zu Chongzhi) ประมาณ π ด้วยจำนวนตรรกยะ $^{355}/_{113}$ ซึ่งมีความถูกต้องถึง 7 ตำแหน่ง

ลองเพิ่มส่วนตามลำดับ เพื่อคำนวณว่า ต่างกับ pi อยู่เท่าใด บรรทัด 9 ถึง 15 ใช้ผลต่างทั้งสองในการตัดสินด้วย if-else ว่าจะเพิ่มเศษหรือเพิ่มส่วนดี นอกจากนี้ยังใช้ if-else เดียวกันในการตั้งค่าตัวน้อย (ระหว่าง diff1 กับ diff2) ให้กับตัวแปร diff ซึ่งเมื่อถึงบรรทัดที่ 16 ก็คือผลต่างของ n/d กับ pi ที่ใช้ทดสอบว่า พบคำตอบที่ต้องการหรือยัง ผู้อ่านควรลองสั่งโปรแกรมนี้ทำงานดูว่า จะได้คำตอบอะไร ?

อนึ่ง เมื่อต้องการใช้ค่า π ในภาษาจาวา ให้ใช้คำว่า Math.PI เพื่ออ้างอิงค่าประมาณของ π ที่มีความถูกต้องถึง 16 ตำแหน่ง (Math.PI มีค่า 3.141592653589793 ซึ่งเป็นค่าคงตัวค่าหนึ่งในคลาส Math)

คำสั่งกับกลุ่มคำสั่ง

ตามข้อกำหนดของคำสั่ง if และ if-else ที่ได้นำเสนอมา เราสามารถเขียนคำสั่งให้ทำหลัง if และหลัง else ได้หนึ่งกลุ่มคำสั่ง (หนึ่งกลุ่มคำสั่งคือคำสั่งต่าง ๆ ที่จับกลุ่มและถูกครอบด้วยเครื่องหมาย { }) ในกรณีที่เรามีเพียงคำสั่งเดียวเท่านั้นในกลุ่ม ก็สามารถเขียนคำสั่งนั้นหลัง if หรือ else ได้เลย โดยไม่ต้องมีเครื่องหมาย { } หากย้อนกลับไปดูบรรทัด 16 ถึง 39 ของรหัสที่ 4-4 ล้วนเป็นคำสั่ง if ที่มีเพียงคำสั่งเดียวในกลุ่มให้ทำเมื่อเงื่อนไขหลัง if เป็นจริง จึงสามารถเขียนให้กระชับจาก 24 บรรทัด เหลือเพียง 8 บรรทัดในอีกรูปแบบดังแสดงในรหัสที่ 4-7

```
if (w < 0) w = w + 7;
if (w == 0) System.out.println("วันเสาร์");
if (w == 1) System.out.println("วันอาทิตย์");
if (w == 2) System.out.println("วันจันทร์");
if (w == 3) System.out.println("วันอังคาร");
if (w == 4) System.out.println("วันพุธ");
if (w == 5) System.out.println("วันพฤหัสบดี");
if (w == 6) System.out.println("วันศุกร์");
```

รหัสที่ 4-7 ตัวอย่างการเขียนคำสั่ง if ให้สั้น ในกรณีที่มีเพียงคำสั่งเดียวหลังเงื่อนไข if

อย่างไรก็ตาม ต้องระมัดระวังการเขียนแบบสั้นนี้ อย่าเผลอไปใส่เกินหนึ่งคำสั่งหลัง if หรือ else เพราะอาจหลอกตาคิดว่าเป็นชุดคำสั่งที่ทำหลัง if เท่านั้น แต่กลับไม่ใช่ เช่น การหาตัวน้อยตัวมากในรหัสที่ 4-8 (ซ้าย) ทำงานถูกต้อง อยากเขียนให้สั้นลง ก็เลยลบเครื่องหมาย { } ออกได้ รหัสที่ 4-8 (กลาง) มีการเขียนคำสั่งให้เยื้องเข้า ทำให้ดูเผิน ๆ คิดว่าเป็นชุดคำสั่งที่ทำหลัง if และ else แต่กลับไม่ใช่ เพราะจะเหมือนกับรหัสทางขวา ที่มีเพียงหนึ่งคำสั่งเท่านั้นที่ทำหลัง if และ else (ในกรณีเช่นนี้ ยังถือว่าโชคดี ที่ตัวแปลโปรแกรมแจ้งว่ามีข้อผิดพลาด ตรงที่ else ไม่มี if จับคู่ด้วย ดูรหัสทางขวาจะเห็นชัดว่า คำสั่ง if ที่เขียนถือว่าจบคำสั่งไปแล้วตรง min=a; คำสั่ง max=b; เป็นคำสั่งโดดๆ ของตัวเอง ตามด้วย else ซึ่งผิด)

```
int min, max;
if (a < b) {
  min = a;
  max = b;
} else {
  min = b;
  max = a;
}
```
ถูก

```
int min, max;
 if (a < b)
   min = a; max = b;
 else
   min = b; max = a;
```
ผิด

```
int min, max;
if (a < b)
  min = a;
max = b;
else
  min = b;
max = a;
```
ผิด

รหัสที่ 4-8 ตัวอย่างการเขียน if-else (ซ้าย) เป็นแบบสั้น (กลาง) ซึ่งผิด

และ หรือ ไม่

นอกจากเรามีตัวดำเนินการคำนวณ `+ - * /` และ `%` ที่ใช้ทำกับนิพจน์คำนวณได้ผลเป็นนิพจน์คำนวณ มีตัวดำเนินการสัมพันธ์ `< <= > >= == !=` ที่ใช้ทำกับนิพจน์คำนวณได้ผลเป็นนิพจน์ตรรกะแล้ว เรายังมี*ตัวดำเนินการตรรกะ* (logical operator) “และ”, “หรือ”, “ไม่” ที่ใช้ทำกับนิพจน์ตรรกะได้ผลเป็นนิพจน์ตรรกะใหม่ ที่สื่อความหมาย และลดความซ้ำซ้อนในการเขียนคำสั่งอีกด้วย จาวาใช้เครื่องหมาย `&&` แทน “และ” , `||` แทน “หรือ” , และ `!` แทน “ไม่”

ให้ `P` และ `Q` แทนนิพจน์ตรรกะ จะได้ว่า `P && Q` เป็นจริง ก็ต่อเมื่อ ทั้ง `P` และ `Q` เป็นจริง ส่วน `P || Q` เป็นจริง ก็ต่อเมื่อ `P` หรือ `Q` (ตัวใดตัวหนึ่งหรือทั้งคู่) เป็นจริง ส่วน `!P` เป็นจริงเมื่อ `P` เป็นเท็จ สรุปตัวดำเนินการทั้งสามดังตารางที่ 4-1

ตารางที่ 4-1 ตารางความจริงของตัวดำเนินการ `&&`, `||`, `!`

P	Q	P && Q	P || Q
true	true	true	true
true	false	false	true
false	true	false	true
fasle	false	false	false

P	!P
true	false
false	true

ตัวอย่างเช่น รหัสที่ 4-9 (ซ้าย) แสดงชุดคำสั่งเพื่อทดสอบว่าลูกบอลชนผนังวินโดว์หรือไม่ โดยใช้ `if` หนึ่งคำสั่งต่อการทดสอบการชนผนังหนึ่งด้าน ซึ่งเราสามารถยุบรวมสอง `if` ของการชนผนังซ้ายและขวาด้วยการทดสอบว่า ขอบซ้ายลูกบอลชนผนังด้านซ้าย หรือ ขอบขวาลูกบอลชนผนังด้านขวา และสามารถรวมการทดสอบการชนผนังด้านบนกับด้านล่างได้เช่นกัน ดังแสดงในรหัสที่ 4-9 (ขวา)

กุมภาพันธ์มีกี่วัน

เป็นที่รู้กันว่า ถ้าเลขปี (ค.ศ.) หารด้วย 4 ลงตัว เดือนกุมภาพันธ์จะมี 29 วัน แต่มีข้อยกเว้นว่า เลขปีที่หารด้วย 4 ลงตัวนี้ ต้องหารด้วย 100 ไม่ลงตัว ยกเว้นกรณีที่หารด้วย 400 ลงตัว เช่น เดือนกุมภาพันธ์ของปี ค.ศ. 2000 มี 29 วัน (เพราะหารด้วย 400 ลงตัว), ของปี 1900 มี 28 วัน

(เพราะหารด้วย 100 ลงตัว แต่หารด้วย 400 ไม่ลงตัว), ของปี 2008 มี 29 วัน (เพราะหารด้วย 4 ลงตัวและหารด้วย 100 ไม่ลงตัว) เป็นต้น

```
if ((x-r) <= 0) {
  dx = -dx;
}
if ((x+r)>=w.getWidth()) {
  dx = -dx;
}
if ((y-r) <= 0) {
  dy = -dy;
}
if ((y+r)>=w.getHeight()){
  dy = -dy;
}
```

```
if ((x-r)<=0 || (x+r)>=w.getWidth()) {
  dx = -dx;
}



if ((y-r)<=0 || (y+r)>=w.getHeight()) {
  dy = -dy;
}
```

รหัสที่ 4-9 ตัวอย่างการใช้ตัวดำเนินการตรรกะในการทดสอบการชนผนังวินโดว์

รหัสที่ 4-10 แสดงส่วนของโปรแกรม 4 แบบเพื่อหาจำนวนวันในเดือนกุมภาพันธ์ของปี ค.ศ. `y` เก็บผลในตัวแปร `d2` แบบที่หนึ่งมี `if` แรกทดสอบกรณีเลขปีหารด้วย 400 ลงตัว ตามด้วยสอง `if` ซ้อนกัน ซึ่งทดสอบว่าหารด้วย 4 ลงตัวหรือไม่ ถ้าใช่ก็ทดสอบอีกด่านว่าต้องหารด้วย 100 ไม่ลงตัว แบบที่สองยุบรวมสอง `if` ที่ซ้อนกันด้วยตัวดำเนินการ `&&` เพราะกรณีนี้เป็นจริงเมื่อต้องหารด้วย 4 ลงตัว “และ” หารด้วย 100 ไม่ลงตัว สังเกตคำสั่งภายในสอง `if` ของแบบที่สองนี้ พบว่าเหมือนกัน (คือ `d2 = 29`) จึงสามารถยุบรวมสอง `if` นี้ด้วย `||` ได้แบบที่สาม ส่วนแบบสุดท้ายเปลี่ยนมาใช้ `if-else` ซึ่งได้ผลเหมือนแบบที่สาม

```
int d2 = 28;
if ((y%400) == 0) {
  d2 = 29;
}
if ((y%4) == 0) {
  if ((y%100) != 0) {
    d2 = 29;
  }
}
```

1

```
int d2 = 28;
if ((y%400) == 0) {
  d2 = 29;
}
if ( ((y%4) == 0) && ((y%100) != 0) ) {
  d2 = 29;
}
```

2

```
int d2 = 28;
if ( ((y%400)==0) || (((y%4)==0) && ((y%100)!=0))) {
  d2 = 29;       // สังเกตเงื่อนไขบน ใส่วงเล็บให้ทำ && ก่อน ||
}
```

3

```
int d2;
if ( ((y%400)==0) || (((y%4)==0) && ((y%100)!=0))) {
  d2 = 29;
} else {
  d2 = 28;
}
```

4

รหัสที่ 4-10 ส่วนของโปรแกรมหาว่าเดือนกุมภาพันธ์มี 28 หรือ 29 วันจากเลขปี ค.ศ.

ตัดเกรด

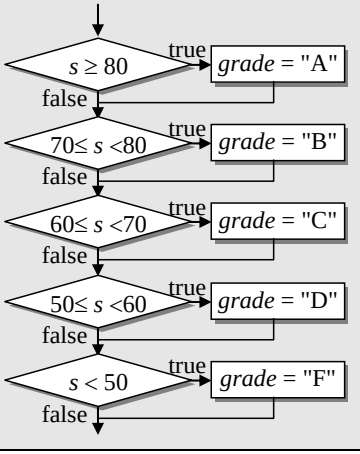
การให้เกรดจากคะแนน (กำหนดให้คะแนนเต็ม เป็น 100) มีเกณฑ์ที่ใช้กันทั่วไปคือ คะแนนตั้งแต่ 80 ขึ้นไปได้ A, ต่ำกว่า 50 ได้ F ส่วนเกรดที่เหลือ ใช้การแบ่งเป็นช่วง ๆ เท่า ๆ กัน ตั้งแต่ 50 ถึง 80 หากเขียนเป็นโปรแกรม ก็ต้องนึกถึง `if` เป็นเครื่องมือเพื่อตัดสินใจว่า จะให้เกรดใดเพื่อให้โปรแกรมสั้น ขอมีแค่ 5 เกรดคือ A, B, C, D, และ F เขียนเป็นรหัสเทียมได้รหัสที่ 4-11

```
if ( คะแนน ≥ 80 ) ได้เกรด A
if ( 70 ≤ คะแนน < 80 ) ได้เกรด B
if ( 60 ≤ คะแนน < 70 ) ได้เกรด C
if ( 50 ≤ คะแนน < 60 ) ได้เกรด D
if ( คะแนน < 50) ได้เกรด F
```

รหัสที่ 4-11 รหัสเทียมแสดงขั้นตอนการตัดเกรด

การทดสอบ $a \le x < b$ เขียนเป็นคำสั่งจาวา `a <= x < b` แบบนี้ไม่ได้ เราต้องแบ่งการทดสอบออกเป็นการเปรียบเทียบสองครั้งแล้วเชื่อมด้วย “และ” เป็น `(a<=x) && (x<b)` เขียนการตัดเกรดทุกเกรดได้ดังรหัสที่ 4-12

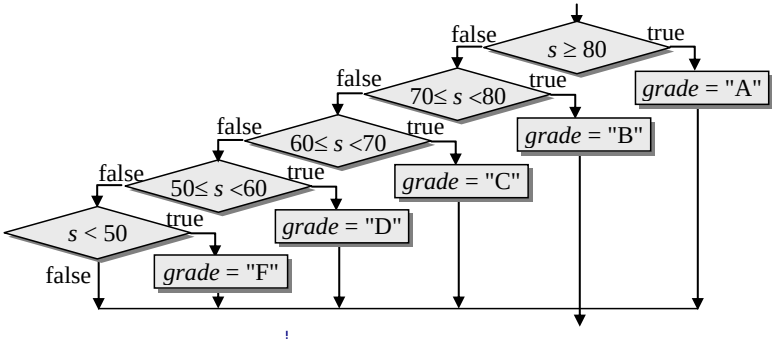
```
if ( score >= 80 ) {
  grade = "A";
}
if ((70 <= score) && (score < 80)) {
  grade = "B";
}
if ((60 <= score) && (score < 70)) {
  grade = "C";
}
if ((50 <= score) && (score < 60)) {
  grade = "D";
}
if ( score < 50 ) {
  grade = "F";
}
```



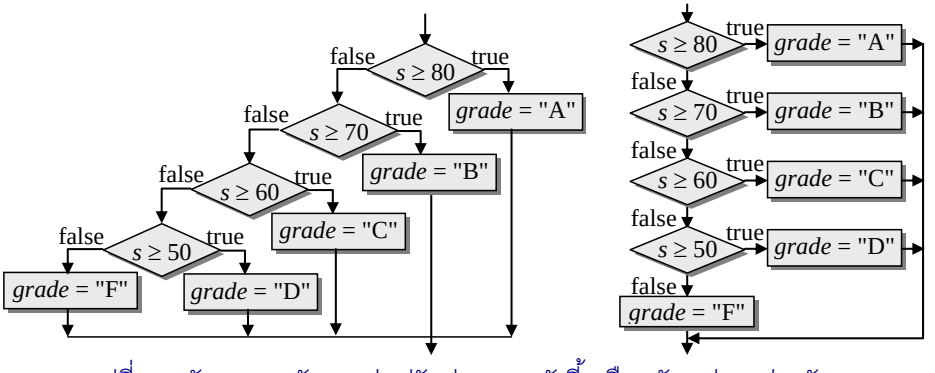
รหัสที่ 4-12 ส่วนของโปรแกรมแสดงขั้นตอนการตัดเกรด

ให้สังเกตว่าเราสามารถสลับลำดับการทดสอบใดๆ ในรหัสข้างต้นนี้ได้โดยยังคงตัดเกรดได้ เพราะแต่ละเงื่อนไขที่ทดสอบนั้น จะเป็นจริงแค่กรณีเดียวเท่านั้น ถ้าสังเกตต่อ จะพบว่า การทดสอบใดที่เป็นจริงแล้ว ก็ไม่น่าต้องทดสอบต่อ เพราะรู้เกรดแล้ว จึงควรนำการทดสอบที่ตามมาไว้หลัง `else` ของการทดสอบตัวก่อน วาดเป็นผังงานได้ดังรูปที่ 4-3 พิจาณาผังงานนี้ พบว่า หากการทดสอบแรก ($s \ge 80$) เป็นเท็จ ย่อมแสดงว่า s ต้องการน้อยกว่า 80 ดังนั้น การทดสอบครั้งต่อไป ไม่ต้องทดสอบ $70 \le s < 80$ ทดสอบแค่ $s \ge 70$ ก็พอ การทดสอบที่เหลือ ก็ทำได้ในทำนองเดียวกัน จึงสามารถปรับปรุงผังงานได้ดังรูปที่ 4-4 ซึ่งแสดงผังงานสองผังที่เหมือนกัน แต่วาดต่างกันเท่านั้น โดยผังงานทางขวาแลดูสะอาดกว่า เขียนเป็นส่วนของโปรแกรมได้ดังรหัสที่ 4-13 (รหัสทั้งสอง

ทำงานเหมือนกัน แต่รหัสทางขวาเขียนแบบสั้น เนื่องจากมีเพียงหนึ่งคำสั่งเท่านั้นหลังทุก `if` และ `else`)



รูปที่ 4-3 ผังงานการตัดเกรด



รูปที่ 4-4 ผังงานการตัดเกรดรุ่นปรับปรุง (สองผังนี้เหมือนกัน แต่วาดต่างกัน)

```
if ( score >= 80 ) {
  grade = "A";
} else {
  if ( score >= 70 ) {
    grade = "B";
  } else {
    if ( score >= 60 ) {
      grade = "C";
    } else {
      if ( score >= 50 ){
        grade = "D";
      } else {
        grade = "F";
      }
    }
  }
}
```

```
if ( score >= 80 ) grade = "A";
else if ( score >= 70 ) grade = "B";
else if ( score >= 60 ) grade = "C";
else if ( score >= 50 ) grade = "D";
else grade = "F";
```

if-else ทั้งก้อนนี้ถือเป็นเพียงหนึ่งคำสั่ง หลัง else ของเงื่อนไข score >= 80

รหัสที่ 4-13 ส่วนของโปรแกรมแสดงขั้นตอนการตัดเกรด (ตามผังงานในรูปที่ 4-4)

ลำดับการทำงานของตัวดำเนินการ

เรารู้ลำดับการทำงานของตัวดำเนินการคำนวณต่างๆ เช่น + - * / และอื่น ๆ กันแล้วในบทที่ 2 เมื่อรวมตัวดำเนินการสัมพันธ์ และตัวดำเนินการตรรกะด้วย จะมีลำดับการทำงานจากสำคัญมากไปน้อยดังนี้ (สำคัญมากจะทำก่อน)

1. วงเล็บ
2. การเรียกใช้เมท็อด
3. การดำเนินการ `!, ++, --, -`
4. การคำนวณ `* / %`
5. การคำนวณ `+ -`
6. การเปรียบเทียบ `< > <= >=`
7. การเปรียบเทียบ `== !=`
8. ตรรกะ `&&`
9. ตรรกะ `||`
10. การให้ค่า (ด้วยตัวดำเนินการ =)

ดังนั้น เงื่อนไข `((y%400)==0) || (((y%4)==0) && ((y%100)!=0))` ที่ได้เขียนในหัวข้อที่แล้วเพื่อหาว่ากุมภาพันธ์มี 29 วันหรือไม่ จึงสามารถถอดวงเล็บออกได้หมด กลายเป็น `y%400 == 0 || y%4 == 0 && y%100 != 0` ก็ได้ เพราะลำดับความสำคัญจากมากไปน้อยของตัวดำเนินการต่างๆ ในนิพจน์ข้างบนนี้คือ `% == && ` และ `||`

การจัดการนิพจน์ตรรกะ

เราสามารถลดรูปหรือเปลี่ยนรูปการเปรียบเทียบและนิพจน์ตรรกะที่บางครั้งดูซับซ้อน ให้ง่ายขึ้น สะอาดตา อ่านเข้าใจได้ง่าย ด้วยกฎการจัดการดังนี้

- `!(a < b)` ⇔ `(a >= b)`
- `!(a <= b)` ⇔ `(a > b)`
- `!(a > b)` ⇔ `(a <= b)`
- `!(a >= b)` ⇔ `(a < b)`
- `!(a == b)` ⇔ `(a != b)`
- `!!P` ⇔ `P`
- `(P&&Q) || (P&&S)` ⇔ `P && (Q || S)`
- `(P||Q) && (P||S)` ⇔ `P || (Q && S)`
- `!(P && Q)` ⇔ `(!P) || (!Q)`
- `!(P || Q)` ⇔ `(!P) && (!Q)`

เรียกสองกฎนี้ว่า กฎเดอมอร์แกน (De Morgan's Law)

โดยที่ a และ b แทนนิพจน์คำนวณ ส่วน P, Q, และ S แทนนิพจน์ตรรกะ เช่น เราต้องการทดสอบว่า ตัวแปร m มีค่าอยู่นอกช่วง 1 ถึง 12 หรือไม่ อาจบรรยายด้วย !(1<=m && m<=12) หากใช้กฎเดอมอร์แกนจะได้ !(1<=m) || !(m<=12) เขียนให้ง่ายขึ้นเป็น m<1 || m>12

หรือถ้าเราเริ่มที่ m<1 || m>12 นิเสธไปสองครั้งได้ !!(m<1 || m>12) ใช้กฎเดอมอร์แกนได้ !(!(m<1) && !(m>12)) เปลี่ยนภายในวงเล็บได้ !(1<=m && m<=12)

ข้อผิดพลาดประการหนึ่งที่พบบ่อยในการเขียนนิพจน์ตรรกะคือ ได้นิพจน์ที่ให้ค่าเป็นจริงหรือเป็นเท็จตลอด ทั้งนี้มักมีสาเหตุจากการตีความของ “และ” กับ “หรือ” ที่ไม่ถูกต้อง ดังตัวอย่างที่แสดงข้างล่างนี้

- (x!=0 || x!=5) ได้ผลเป็นจริงเสมอ เพราะถ้า x เป็น 0 ย่อมไม่เป็น 5 ทำให้เป็นจริงและในทางกลับกัน ถ้า x เป็น 5 ย่อมไม่เป็น 0 จึงเข้าใจว่าน่าจะเขียนผิด ถ้าเปลี่ยนจาก || เป็น && จะอ่านได้ความหมายว่า ต้องการทดสอบว่า x ต้องไม่เป็น 0 และ ไม่เป็น 5 หรือไม่
- (x<3 && x>9) ได้ผลเป็นเท็จเสมอ นิพจน์นี้เป็นจริงได้ก็เมื่อมี x ที่ทั้งน้อยกว่า 3 และมากกว่า 9 ซึ่งไม่มีจำนวนที่มีคุณสมบัติเช่นนี้ จึงเข้าใจว่าน่าจะเขียนผิด ถ้าเปลี่ยน && เป็น || จะอ่านได้ว่า ต้องการทดสอบว่า x น้อยกว่า 3 หรือ มากกว่า 9 หรือไม่
- (x==0 && x!=0) ได้ผลเป็นเท็จเสมอ กับ (x==0 || x!=0) ได้ผลเป็นจริงเสมอ (ผู้อ่านลองอ่านตีความดูว่าทำไม ?)

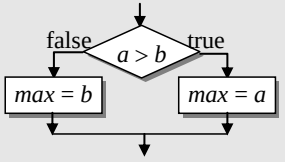
if หลายชั้น

โปรแกรมการตัดเกรดเป็นตัวอย่างการใช้ if ซ้อนอยู่ใน if ซ้อนอยู่หลายชั้น ซึ่งหากเขียนซ้อนกันมาก ก็จะซับซ้อนมาก หัวข้อนี้ขอนำเสนอตัวอย่างของการเขียน if หลายชั้นเพิ่มเติม

ตัวมากสุด

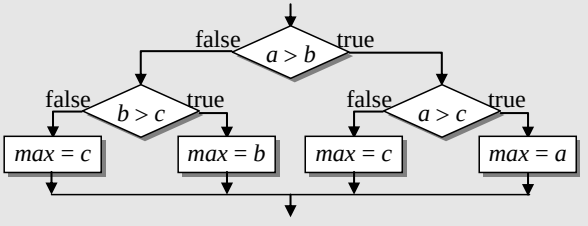
หากต้องการหาค่ามากสุดของตัวแปร a กับ b ก็สามารถใช้ if-else เพียงหนึ่งคำสั่ง ดังรหัสที่ 4-14 ในกรณีที่มีตัวแปร c เพิ่มอีกตัว ต้องเพิ่ม if เพื่อนำตัวที่มากกว่า (ระหว่าง a กับ b) มาเปรียบเทียบกับ c ซ้อนไว้ภายในว่าใครมากกว่า ตัวมากครั้งที่สองย่อมเป็นตัวมากสุดของทั้งสามตัว เขียนได้ดังรหัสที่ 4-15 ผู้อ่านลองคิดต่อดูว่าถ้าเพิ่มตัวแปรอีกตัว จะปรับผังงานและรหัสอย่างไร

```
int max;
if (a > b) {
    max = a;
} else {
    max = b;
}
```



รหัสที่ 4-14 ส่วนของโปรแกรมหาค่ามากสุดของ a และ b

```
int max;
if (a > b) {
  if (a > c)
    max = a;
  else
    max = c;
} else {
  if (b > c)
    max = b;
  else
    max = c;
}
```



รหัสที่ 4-15 ส่วนของโปรแกรมหาค่ามากสุดของ a, b และ c (วิธีที่หนึ่ง)

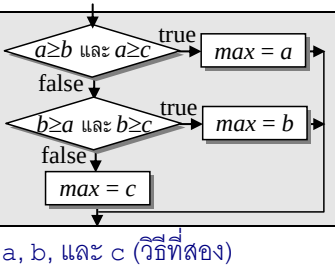
ขอเสนออีกวิธี ตรงไปตรงมา ถ้าตัวแปรใดมีค่ามากกว่าตัวแปรอื่นทุกตัว ค่ามากสุดย่อมเท่ากับตัวแปรตัวนั้น ดังรหัสที่ 4-16

```
int max;
if ((a>b) && (a>c)) max = a;       // a มากสุด
else if ((b>a) && (b>c)) max = b; // b มากสุด
else max = c;     // a และ b ไม่มากสุด, c ก็ต้องมากสุด
```

รหัสที่ 4-16 ส่วนของโปรแกรมหาค่ามากสุดของ a, b, และ c (ทำงานผิด)

ผู้อ่านควรทดสอบส่วนของโปรแกรมนี้ดู (โดยเขียนคำสั่งเพิ่มให้มีการรับ a, b, c หาค่ามากสุด และแสดงผล จากนั้นสั่งทำงาน) ไม่ว่าจะให้ a=3, b=2, c=1 หรือให้ a=2, b=3, c=1 หรือให้ a=1, b=2, c=3 หรือให้ a=1, b=1, c=3 จะได้ max=3 ถูกต้องทุกกรณี แต่ถ้าให้ a=3, b=3, c=1 กลับได้ max=1 !!! แสดงว่าเมื่อข้อมูลมีค่าเท่ากัน อาจให้ผลลัพธ์ที่ผิด ก็ต้องแก้ไขให้ครอบคลุมกรณีดังกล่าวได้ดังรหัสที่ 4-17 เพื่อเปรียบเทียบกรณีเท่ากันด้วย ผู้อ่านลองคิดต่อดูว่า ถ้าเพิ่มตัวแปรอีกตัว จะปรับผังงานและรหัสอย่างไร

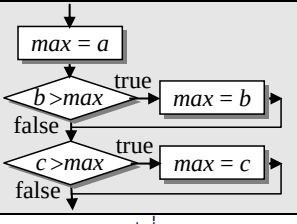
```
int max;
if ((a>=b) && (a>=c)) max = a;
else if ((b>=a) && (b>=c)) max = b;
else max = c;
```



รหัสที่ 4-17 ส่วนของโปรแกรมหาค่ามากสุดของ a, b, และ c (วิธีที่สอง)

ขอนำเสนอวิธีสุดท้ายในการหาค่ามากสุดของตัวแปร 3 ตัว โดยค่อยๆ ดูทีละตัว ดูว่าตัวใหม่มีค่ามากกว่าค่ามากสุดที่ได้ดูมาหรือไม่ ถ้ามากกว่า ก็เปลี่ยนค่ามากสุดเป็นค่าของตัวใหม่ ดังแสดงในรหัสที่ 4-18 จะเห็นได้ว่า ถ้าจะปรับให้หาค่ามากสุดของตัวแปรที่มีจำนวนมากขึ้น จะทำได้ง่ายกว่าอีกสองวิธีที่นำเสนอมา

```
int max = a;
if (b > max) max = b;
if (c > max) max = c;
```



รหัสที่ 4-18 ส่วนของโปรแกรมหาค่ามากสุดของ a, b, และ c (วิธีที่สาม)

มัธยฐาน

ถ้าเราเปลี่ยนจากการหาค่ามากสุด เป็นการหาค่ามัธยฐานของตัวแปร 3 ตัว วิธีการหาจะซับซ้อนขึ้น มัธยฐานของ *a, b* และ *c* คือค่าที่มีอย่างน้อยหนึ่งตัวไม่มากกว่า และอย่างน้อยหนึ่งตัวไม่น้อยกว่า เช่น มัธยฐานของ 1,2,3 คือ 2, ของ 3,3,5 คือ 3, ของ 1,2,2 คือ 2 เป็นต้น เราสามารถนำวิธีที่สองของการหาค่ามากสุดในหัวข้อที่แล้ว มาปรับใช้ได้ คือการลุยเปรียบเทียบตามนิยามตรงไปตรงมา (แต่ใช้จำนวนการเปรียบเทียบมากหน่อย) ดังรหัสที่ 4-19

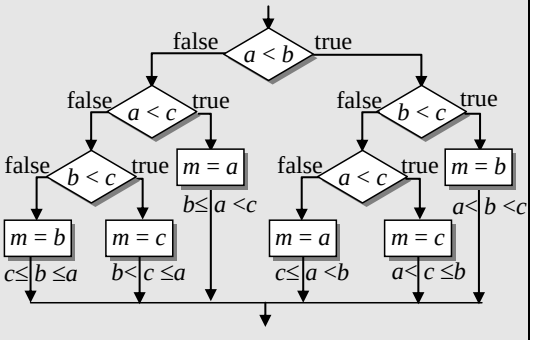
```
int median;
if ((b<=a) && (a<=c)) median = a;        // b ≤ a ≤ c
else if ((c<=a) && (a<=b)) median = a;   // c ≤ a ≤ a  b
else if ((a<=b) && (b<=c)) median = b;   // a ≤ b ≤ c
else if ((c<=b) && (b<=a)) median = b;   // c ≤ a ≤ b  a
else median = c;           // a ≤ c ≤ b หรือ  b ≤ c ≤ a
```

รหัสที่ 4-19 ส่วนของโปรแกรมหาค่ามัธยฐานของ a, b, และ c

หรือจะใช้แนวคิดเดียวกับวิธีที่หนึ่งของการหาค่ามากสุดในหัวข้อที่แล้ว โดยเปรียบเทียบข้อมูลทีละคู่ การเปรียบเทียบหนึ่งครั้งทำให้เราค่อย ๆ จัดเรียงข้อมูลไปเรื่อย ๆ จนสรุปได้ว่าตัวใดอยู่ตรงกลางเมื่อจัดเรียงจากน้อยไปมากสำเร็จ จากผังงานที่แสดงประกอบรหัสที่ 4-20 มีการเปรียบเทียบ $a<b$ ก่อน ถ้าเป็นจริง สรุปได้ว่าการจัดเรียงจากน้อยไปมากมีโอกาสเป็น *a, b, c* หรือ *a, c, b* หรือ *c, a, b* พอเปรียบเทียบ $b<c$ ต่อ ถ้าเป็นจริง ย่อมสรุปได้ชัดว่าการจัดเรียงคือ *a, b, c* มัธยฐานจึงเป็น *b* แต่ถ้า $b<c$ เป็นเท็จ การจัดเรียงที่เป็นไปได้จะเหลือกรณี *a, c, b* หรือ *c, a, b* จึงต้องใช้การเปรียบเทียบ $a<c$ อีกครั้ง จึงจะสรุปได้ว่า มัธยฐานเป็น *a* หรือ *c* (ผู้อ่านลองพิจารณาการเปรียบเทียบที่เหลือในผังงานดู)

ถึงแม้ว่าวิธีนี้ได้โปรแกรมที่ซับซ้อนกว่าวิธีที่แล้ว แต่ถ้ามานับจำนวนการเปรียบเทียบกันจะพบว่าวิธีหลังนี้เร็วกว่า เปรียบเทียบอย่างมากเพียงสามครั้ง ก็พบมัธยฐานแล้ว

```
int median;
if (a < b) {
  if (b < c)
     median = b;    // a < b < c
  else
     if (a < c)
       median = c; // a < c ≤ b
     else
       median = a; // c ≤ a < b
} else {
  if (a < c)
     median = a;    // b ≤ a < c
  else
     if (b < c)
       median = c; // b < c ≤ a
     else
       median = b; // c ≤ b ≤ a
}
```



รหัสที่ 4-20 ส่วนของโปรแกรมหาค่ามัธยฐานของ a, b, และ c

ตัวแปรแบบ boolean

นอกจากตัวแปรแบบจำนวนและสตริงที่ได้เคยนำเสนอมาในบทก่อน ๆ จาวายังมีตัวแปรประเภท `boolean` อันเป็นตัวแปรที่เก็บค่าได้สองค่าเท่านั้น คือไม่เก็บค่าจริง (`true`) ก็เก็บค่าเท็จ (`false`) คำว่า `true` และ `false` เป็นค่าคงตัวในจาวาที่แทนค่าจริงและเท็จตามลำดับ (ให้สังเกตว่าไม่มีเครื่องหมาย " ใดๆ ครอบค่าคงตัวสองค่านี้ `true` และ `false` คือสองคำสำคัญของจาวา ต้องเขียนตัวเล็กหมด) ขอนำเสนอการใช้งานด้วยตัวอย่างในหัวข้อย่อยต่อไปนี้

เกมทายตัวเลข

ผู้อ่านคงเคยเล่นเกมทายตัวเลข มีผู้เล่นสองคน คนหนึ่งนึกจำนวนเต็มหนึ่งจำนวน (ตั้งแต่ 1 ถึง 100) เก็บไว้ในใจ อีกคนเดาว่าเลขนั้นคืออะไร ทุกครั้งที่เดา ถ้าใช่ก็จบ ถ้าไม่ใช่ คนแรกต้องบอกว่า เลขที่ตนนึกนั้นน้อยกว่าหรือมากกว่าจำนวนที่อีกคนเดา ถ้าเดาเกิน 7 ครั้ง แล้วยังผิดก็แพ้ เราจะมาเขียนโปรแกรมให้คอมพิวเตอร์เป็นผู้เล่นคนแรก สุ่มเลขเก็บไว้ให้ผู้ใช้โปรแกรมเดา

รหัสที่ 4-21 แสดงโปรแกรมเกมทายตัวเลข มีการสร้างตัวอ่านแป้นพิมพ์ (บรรทัดที่ 7) ตามด้วยการสุ่มจำนวนเต็มระหว่าง 1 ถึง 100 (บรรทัดที่ 8) โดยใช้ฟังก์ชัน `Math.random()` ซึ่งคืนจำนวนจริงสุ่มตั้งแต่ 0 ถึง 1 (ไม่รวม 1) นำผลมาคูณด้วย 100 แล้วตัดเศษทิ้ง จะได้จำนวนเต็มสุ่มตั้งแต่ 0 ถึง 99 บวกเพิ่มอีก 1 ก็ได้จำนวนสุ่มที่ต้องการ มีตัวแปร `k` ไว้นับจำนวนรอบประกอบกับวงวน `while(true)` เพื่อคุมจำนวนรอบไว้แค่ 7 รอบ มีตัวแปรแบบ `boolean` ชื่อ `correct` ไว้เก็บผลการเดาของผู้ใช้ว่าถูกต้องหรือยัง ดังนั้น จะหลุดจากวงวน (บรรทัดที่ 12) เมื่อเดาถูกต้อง

แล้ว หรือไม่ก็หมุนครบ 7 รอบแล้ว ถ้ายังให้เดาต่อ ก็รับจำนวนจากผู้ใช้ แล้วนำผลการทดสอบ (yourGuess == myNumber) เก็บใส่ตัวแปร correct ซึ่งถ้าเดาตรงก็ได้ค่าจริง ไม่ตรงก็ได้ค่าเท็จ จากนั้นเลือกแสดงข้อความ "น้อยกว่า" หรือ "มากกว่า" ขึ้นกับผลการเปรียบเทียบในบรรทัดที่ 16 และ 17 ถ้าเดาตรงแล้วก็ไม่แสดง เพราะเดี๋ยววนกลับไปที่บรรทัดที่ 12 ก็จะหลุดออกจากวงวน เมื่อออกจากวงวนก็ตรวจสอบตัวแปร correct อีกครั้งเพื่อแจ้งผลการเล่นให้ทราบ ให้สังเกตว่าเราเขียน if (correct) ในบรรทัดที่ 20 ได้เลย เพราะ correct ให้ค่าจริงหรือเท็จไม่จำเป็นต้องเขียน if (correct == true) เพราะเขียนยาวกว่าและได้ผลเหมือนกัน และยิ่งน่ากลัวมาก ถ้าเขียน if(correct = true) เพราะเขียนพลาดใช้เครื่องหมาย = เพียงตัวเดียวกลายเป็นการให้ค่าจริงกับ correct และทำให้เงื่อนไขนี้เป็นจริงเสมอ โดยตัวแปลโปรแกรมไม่เห็นข้อผิดพลาดดังกล่าวด้วย (ผู้อ่านอาจลองปรับโปรแกรมนี้โดยหลีกเลี่ยงไม่ใช้ตัวแปรแบบ boolean เพื่อเก็บผล)

```
01 import java.util.Scanner;
02 // โปรแกรมเกมทายตัวเลข
03 public class WhatsThatNumber {
04   public static void main(String[] args) {
05     System.out.println("เกมทายตัวเลข : คุณมีโอกาส 7 ครั้ง");
06     System.out.println("ทายว่าผมนึกถึงจำนวนใดอยู่ (1-100)");
07     Scanner kb = new Scanner(System.in);
08     int myNumber = 1 + (int)(100 * Math.random());
09     int k = 1;
10     boolean correct = false;
11     while (true) {
12       if (correct || k > 7) break;
13       System.out.print("การทายครั้งที่ " + k + " : ");
14       int yourGuess = kb.nextInt();
15       correct = (yourGuess == myNumber);
16       if (myNumber < yourGuess) System.out.println("น้อยกว่า");
17       if (myNumber > yourGuess) System.out.println("มากกว่า");
18       k++;
19     }
20     if (correct) {
21       System.out.println("ถูกต้องแล้วครับ");
22     } else {
23       System.out.println("เสียใจครับ");
24       System.out.println("ผมกำลังนึกถึง " + myNumber + " ครับ");
25     }
26   }
27 }
```

รหัสที่ 4-21 โปรแกรมเกมทายตัวเลข

รากของสมการกำลังสอง

เราเคยเขียนโปรแกรมหารากของสมการกำลังสอง $ax^2 + bx + c = 0$ ในบทที่ 2 แต่โปรแกรม

นั้นมีข้อจำกัดมากมาย เช่น ถ้า a=0 จะได้ผลผิด อาจตอบเป็น Infinity ยังมีผิดกรณีอื่น ๆ อีก เราจะมาใช้ if เขียนให้สมบูรณ์มากขึ้น จะขอปรับปรุงให้ครอบคลุมกรณีต่าง ๆ ดังนี้

- $a = 0, b = 0, c = 0$: สมการคือ $0 = 0$, ทุกจำนวนคือราก
- $a = 0, b = 0, c \neq 0$: สมการคือ $c = 0$, ไม่มีราก
- $a = 0, b \neq 0$: เป็นสมการเชิงเส้น $bx + c = 0$; รากคือ $-c/b$
- $a \neq 0, b^2 < 4ac$: รากเป็นจำนวนเชิงซ้อน เพราะ $\sqrt{b^2 - 4ac}$ เป็นจำนวนเชิงซ้อน
- ไม่ตรงกรณีใดข้างบน หารากได้จากสูตร $(-b \pm \sqrt{b^2 - 4ac}) / 2a$

เขียนเป็นโปรแกรมได้ดังรหัสที่ 4-22 ซึ่งรับค่า a, b, และ c จากนั้นทดสอบกรณีต่าง ๆ ข้างต้นเก็บใส่ตัวแปรแบบ boolean เพื่อนำไปใช้ในพิจารณากรณีต่าง ๆ ในบรรทัดที่ 16 ถึง 31

```
01 import java.util.Scanner;
02 // โปรแกรมหารากของสมการ ax^2 + bx + c = 0
03 public class QuadraticRoots {
04  public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("a = ");
07     double a = kb.nextDouble();
08     System.out.print("b = ");
09     double b = kb.nextDouble();
10     System.out.print("c = ");
11     double c = kb.nextDouble();
12     boolean isTrivial  = (a == 0 && b == 0 && c == 0);
13     boolean noSolution = (a == 0 && b == 0 && c != 0);
14     boolean isLinear   = (a == 0 && b != 0);
15     boolean isComplex  = (b*b < 4*a*c);
16     if (isTrivial) {
17       System.out.println("รากคือจำนวนใด ๆ");
18     } else if (noSolution) {
19       System.out.println("สมการนี้ไม่มีราก");
20     } else if (isLinear) {
21       System.out.println("รากคือ " + (-c / b));
22     } else if (isComplex) {
23       System.out.println("รากเป็นจำนวนเชิงซ้อน (โปรแกรมนี้หาให้ไม่ได้)");
24     } else {
25       double t = Math.sqrt(b*b - 4*a*c);
26       double r1 = (-b + t) / (2*a);
27       double r2 = (-b - t) / (2*a);
28       System.out.print("รากคือ " + r1);
29       if (r1 != r2) System.out.print(" และ " + r2);
30       System.out.println();
31     }
32   }
33 }
```

รหัสที่ 4-22 โปรแกรมหารากของสมการกำลังสอง $ax^2 + bx + c$

เพิ่มเติม

ตัวดำเนินการเงื่อนไข

รูปแบบการใช้คำสั่ง `if-else` ในลักษณะของการให้ค่าตัวแปรหนึ่งตามเงื่อนไขว่า จริงให้ค่าหนึ่ง เท็จให้อีกค่าหนึ่ง เช่น `if (a > b) max = a; else max = b;` เป็นรูปแบบที่พบบ่อยมาก จาวามีตัวดำเนินการ `?:` เรียกว่า *ตัวดำเนินการเงื่อนไข* (conditional operator) ซึ่งสามารถเขียนคำสั่ง `if-else` ข้างต้นได้กะทัดรัดเป็น `max = (a > b) ? a : b;` โดยมีรูปแบบการเขียนดังนี้

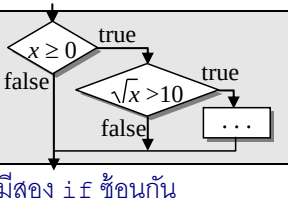
เงื่อนไข **?** *ค่าที่ได้เมื่อเป็นจริง* **:** *ค่าที่ได้เมื่อเป็นเท็จ*

ค่าที่เขียนหลัง `?` และหลัง `:` ต้องเป็นนิพจน์ (นั่นคือจะใส่คำสั่ง เช่น `if`, `if-else` หรือวงวนไม่ได้) ต้องเขียนทั้งสองกรณี ละกรณีใดกรณีหนึ่งไม่ได้ และเนื่องจากผลของ `?:` ถือว่าเป็นนิพจน์ จึงเขียนซ้อนกันได้ เช่น `max = (a>b) ? (a>c ? a : c) : (b>c ? b : c);` คือการหาค่ามากสุดของ `a,b, c` นอกจากนี้เราสามารถใช้ผลของ `?:` เป็นส่วนหนึ่งของการคำนวณในนิพจน์ได้ เช่น `m = 1 + (a<0 ? -a : a);` เป็นต้น

วงจรลัด

ผู้อ่านอาจสงสัยว่า เรากำลังศึกษาการพัฒนาซอฟต์แวร์ แล้ววงจรลัดให้เกิดไฟช็อตมาเกี่ยวอะไรด้วย คำนี้ใช้กับตัวดำเนินการ `&&` กับ `||` ซึ่งมีพฤติกรรมพิเศษดังนี้ สมมติให้ `P` และ `Q` เป็นนิพจน์ตรรกะ หากเราเขียน `P&&Q` จาวาจะคิดค่าของ `P` ก่อน ถ้าพบว่าเท็จ จะสรุปทันทีว่า `P&&Q` เป็นเท็จ (โดยไม่ต้องคิดค่าของ `Q`) ถือว่าเป็นการคิดลัด เพราะอะไรก็ตามเมื่อ `&&` กับเท็จย่อมได้เท็จ เพื่อให้ชัดเจนมากขึ้น หากเราเขียน `P&&Q&&R&&S` จะเริ่มคิดค่าของ `P` ถ้าเป็นจริง ก็คิดค่าของ `Q` ต่อ ถ้าเป็นจริงอีก ก็คิดค่าของ `R` ถ้าคราวนี้เป็นเท็จ ก็หยุด ไม่ต้องคิด `S` สรุปได้ว่านิพจน์นี้เป็นเท็จ แล้ววงจรลัดมีประโยชน์อย่างไร สมมติเราต้องการเปรียบเทียบว่า `Math.sqrt(x)>10` หรือไม่ โดยจะทำเช่นนี้ก็เมื่อ `x` ต้องไม่เป็นจำนวนลบ เพราะ `Math.sqrt` หาค่ารากของจำนวนลบไม่ได้ โดยทั่วไปจะเขียน `if` สองตัวซ้อนกัน ดังรหัสที่ 4-23

```
if (x >= 0) {
  if (Math.sqrt(x) > 10) {
    ...
  }
}
```



รหัสที่ 4-23 ส่วนของโปรแกรมมีสอง `if` ซ้อนกัน

ด้วยการทำงานแบบวงจรลัด ทำให้เราสามารถยุบสอง `if` นี้ แล้วเชื่อมด้วย `&&` ได้เป็นคำสั่งที่กะทัดรัด เข้าใจง่าย

```
if ((x>=0) && (Math.sqrt(x)>10))
```

เพราะถ้า `(x>=0)` เป็นเท็จ จะได้ไม่คิดค่าของการเปรียบเทียบในนิพจน์ทางขวา (แต่อย่ายุบแล้วเขียนเป็น `if ((Math.sqrt(x)>10) && (x>=0))` เพราะถ้า `x` เป็นลบ ก็สายไปแล้ว)

วงจรลัดใช้กับตัวดำเนินการ `||` ด้วย สมมติให้ `P` และ `Q` เป็นนิพจน์ตรรกะ หากเราเขียน `P||Q` จาวาจะคิดค่าของ `P` ก่อน ถ้าพบว่าจริง จะสรุปทันที ว่า `P||Q` เป็นจริง (โดยไม่ต้องคิดค่าของ `Q`) เพราะอะไรก็ตามเมื่อ `||` กับจริงย่อมได้จริง

switch-case

การเขียนคำสั่งที่ประกอบด้วย `if-else` ต่อกันเป็นทอด ๆ เป็นรูปแบบที่เขียนกันบ่อยมาก โดยเฉพาะอย่างยิ่งเมื่อเงื่อนไขในแต่ละวงเล็บหลัง `if` เป็นการทดสอบค่าของจำนวนเต็มว่าเป็นค่าตามที่กำหนดหรือไม่ ตัวอย่างเช่น รหัสที่ 4-24 ทางซ้ายแสดงการตั้งชื่อวันให้กับตัวแปร `d` ตามค่าของ `w` ว่าจะมีค่าอะไร หากพบรูปแบบการเขียนคำสั่งในลักษณะนี้ เราสามารถแทนได้ด้วยคำสั่ง `switch-case` ดังแสดงในรหัสที่ 4-24 ทางขวา

```
if (w == 0)
  t = "วันเสาร์";
else if (w == 1)
  t = "วันอาทิตย์";
else if (w == 2)
  t = "วันจันทร์";
else if (w == 3)
  t = "วันอังคาร";
else if (w == 4)
  t = "วันพุธ";
else if (w == 5)
  t = "วันพฤหัสบดี";
else if (w == 6)
  t = "วันศุกร์";
else
  t = "???";
```

```
switch( w ) {
case 0:
  t = "วันเสาร์"; break;
case 1:
  t = "วันอาทิตย์"; break;
case 2:
  t = "วันจันทร์"; break;
case 3:
  t = "วันอังคาร"; break;
case 4:
  t = "วันพุธ"; break;
case 5:
  t = "วันพฤหัสบดี"; break;
case 6:
  t = "วันศุกร์"; break;
default:
  t = "???"; break;
}
```

รหัสที่ 4-24 ส่วนของโปรแกรมการแทนลำดับ `if-else` ด้วย `switch-case`

สิ่งที่ปรากฏในวงเล็บหลังคำว่า `switch` ต้องเป็นนิพจน์ที่ให้ค่าเป็น `int` ซึ่งหากมีค่าตรงตามกรณีใดที่เขียนด้วยคำว่า `case` ก็จะกระโดดไปทำคำสั่งของกรณีนั้น จากตัวอย่าง หาก `w` มีค่า 3 ก็จะไปทำคำสั่งหลัง `case 3 :` หากค่า `w` มีค่าที่ไม่ตรงค่าใดเลย จะกระโดดไปทำหลัง `default :` ให้สังเกตว่ามีคำว่า `break` เป็นคำสั่งสุดท้ายของทุกๆ กรณีในตัวอย่าง คำสั่ง

break นี้บอกว่าให้กระโดดออกจากคำสั่ง switch หากไม่มี break การทำงานจะลงไปทำต่อในกรณีถัดไปโดยอัตโนมัติ ดังตัวอย่างในรหัสที่ 4-25 เป็นการใช้ switch-case หาจำนวนวันในเดือนที่กำหนด ถ้า m มีค่า 1,3,5,7,8,10 หรือ 12 ก็จะมี 31 วัน จะเห็นว่าถ้าหลัง case ไม่มีคำสั่งอะไร การทำงานก็จะไหลลงมา case ต่อไป ทำงานจนกว่าจะพบ break จึงจะหลุดจาก switch

```
switch ( m ) {
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12:
    days = 31;
    break;
  case 4:
  case 6:
  case 9:
    days = 30;
    break;
  case 2:
    days = (y%400== 0 || y%4 == 0 && y%100 != 0) ? 29 : 28;
    break;
  default:
    System.out.println("หมายเลขเดือนไม่ถูกต้อง");
    break;
}
```

รหัสที่ 4-25 ตัวอย่างการใช้ switch-case หาจำนวนวันของเดือน

0.1 + 0.1 + 0.1 ≠ 0.3

หัวข้อเพิ่มเติมในบทที่ 2 ได้นำเสนอการแทนจำนวนจริงในระบบเลขฐานสองที่ถึงแม้ว่ามีความแม่นยำสูง แต่ก็ยังคงเป็นค่าประมาณ มีความคลาดเคลื่อนบ้าง เช่น (0.1 + 0.1 + 0.1) – 0.3 จะไม่ได้ 0 แต่จะได้ 5.55×10^{-17} ดังนั้น การเปรียบเทียบความเท่ากันของจำนวนจริงจึงต้องระวังเป็นพิเศษ เพราะถึงแม้ว่าในทางทฤษฎีจะมีค่าเท่ากัน แต่ในทางปฏิบัติไม่เท่า จึงเสี่ยงมากถ้าเขียน if (x==y) เมื่อ x และ y เป็นจำนวนจริง บางคนใช้ if (Math.abs(x-y) <= 1e-14) เพื่อทดสอบว่า ค่า x และ y เกือบเท่ากัน ก็ยังไม่ถูกต้อง (เช่น การทดสอบ $x = 10^{-50}$ และ $y = 10^{-56}$ ซึ่งมีค่าต่างกันเป็นล้านเท่า แต่กลับบอกว่าเกือบเท่า) จึงควรใช้ความแตกต่างสัมพัทธ์เปรียบเทียบ $\frac{|x-y|}{\max(|x|, |y|)} \leq \varepsilon$ โดย ε มีค่าน้อย ๆ เช่น 10^{-14} เขียนเป็นคำสั่งจาวาได้ดังนี้

```
Math.abs(x-y) <= 1e-14 * Math.max(Math.abs(x),Math.abs(y))
```

(ข้อสังเกต : เราย้ายตัวหารทางซ้ายมาคูณทางขวา เพราะการหารในสูตรจะมีปัญหากรณี x และ y เป็น 0 ทั้งคู่ และการทดสอบนี้ถือว่า ค่าใด ๆ ที่ไม่ใช่ 0 มีค่าแตกต่างสัมพัทธ์มากเมื่อเทียบกับ 0)

แบบฝึกหัด

1. จงเขียนข้อความต่อไปนี้ด้วยคำสั่ง `if` หรือ `if-else`
   - ถ้า *i* หาร *j* ลงตัว ให้ *k* มีค่าเป็นสองเท่าของ *i*
   - ถ้า *i* เป็นสี่เท่าของ *j* ให้ *k* มีค่าเป็นครึ่งหนึ่งของ *i*
   - ถ้า *i* หรือ *j* ตัวใดตัวหนึ่ง (ไม่ใช่ทั้งสองตัว) เป็นศูนย์ ให้ตัวที่เป็นศูนย์มีค่าเท่ากับตัวที่ไม่ใช่ศูนย์
   - ถ้า *i* เป็นเลขคี่ที่ไม่น้อยกว่า *j* ให้เพิ่มทั้ง *i* และ *j* อีก 1
   - ถ้า *i* เป็นจำนวนเต็มบวกที่เป็นเลขคู่ที่มีค่าไม่เกิน 20 แต่ต้องไม่ใช่ 8 และ 10 ให้ *k* มีค่าเพิ่มจากเดิมอีกสองเท่า ถ้า *i* น้อยกว่า 10 ไม่เช่นนั้นให้ *k* ลดลงจากเดิมครึ่งหนึ่ง

2. กำหนดให้ `x` และ `y` เป็นตัวแปร `int` ส่วน `p` และ `q` เป็นตัวแปรแบบ `boolean` จงเติมวงเล็บให้กับนิพจน์ตรรกะในแต่ละข้อต่อไปนี้เพื่อให้ถูกต้องตามหลักไวยากรณ์ สำหรับข้อใดที่ไม่มีทางเติมวงเล็บให้ถูกต้องได้ ให้อธิบายเหตุผลประกอบ
   - `p == x == y`
   - `x < y != p`
   - `p = x == y`
   - `x != y == q`
   - `! x == ! y`
   - `! ! x >= y`
   - `p && ! x == y`
   - `x == p && q`

3. จงใส่เครื่องหมาย `{ }` ให้ครบและจัดรูปแบบย่อหน้าของกลุ่มคำสั่งข้างล่างนี้ให้สวยงาม โดยยังคงให้รหัสใหม่ทำงานเหมือนเดิม

```
if (i == 0) if (j < 0)
a = 0; else
if (k == 1) a = 1;
else a = 2;
```

```
if (i == 0)
a = 0; if (j == 0)
a = 1; else if (k == 0)
a = 2; if (k < 0) a = 3;
```

4. นิพจน์ตรรกะใดข้างล่างนี้ที่เป็นจริงเสมอ หรือเป็นเท็จเสมอ หรือไม่แน่ขึ้นกับค่าของ `x`
   - `!(x > 0) && (x > 0)`
   - `(x != 0) || (x == 0)`
   - `(x != 1) == !(x == 1)`
   - `(x > 0) || (x < 0)`
   - `(x >= 0) || (x < 0)`
   - `(x <= 0) && (x >= 0)`

5. จงเขียนโปรแกรมรับความยาวด้านแต่ละด้านของสามเหลี่ยม จากนั้นตรวจสอบพร้อมกับแสดงผลการตรวจสอบว่า เป็นความยาวด้านที่ประกอบกันเป็นสามเหลี่ยมได้หรือไม่ (ข้อแนะนำ : ผลรวมของความยาวด้านสองด้านของสามเหลี่ยมใด ๆ ต้องยาวกว่าด้านที่สาม)

6. จงเขียนโปรแกรมรับจำนวนเต็ม 3 จำนวน จากนั้นตรวจสอบพร้อมกับแสดงผลการตรวจสอบว่า มีสองจำนวนใด ๆ ในข้อมูลที่รับเข้ามาที่มีค่าเท่ากันหรือไม่

7. เรามักพบโฆษณาส่งเสริมการขายประเภท “ซื้อ 3 แถม 1” ซึ่งโดยทั่วไปมักหมายความว่า ซื้อสินค้า 4 ชิ้น แล้วทางร้านจะไม่คิดราคาชิ้นที่มีราคาน้อยสุด จงเขียนโปรแกรมที่รับราคาสินค้า 4 จำนวน, แสดงผลรวมราคาของสินค้าทั้งสี่ และแสดงจำนวนเงินที่ต้องชำระ

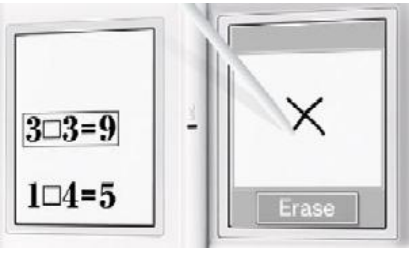
8. จงเขียนโปรแกรมที่รอรับจำนวนเต็มทางแป้นพิมพ์ เพื่อแสดงจำนวนนี้ในรูปแบบที่มีเครื่องหมายจุลภาค , คั่นทุกสามหลัก เช่น รับ –19283445 มา ก็แสดง –19,283,445

9. จงเขียนผังงานและโปรแกรมหาค่ามากสุดของจำนวน 10 จำนวนที่รับทางแป้นพิมพ์

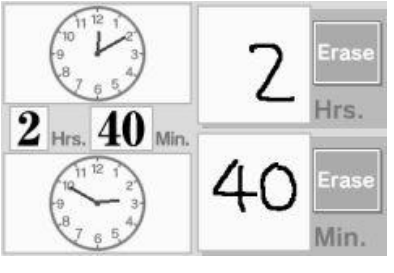
10. จงเขียนผังงานและโปรแกรมหาค่าที่มากสุดเป็นอันดับที่สองของจำนวน 10 จำนวนที่รับเข้ามาทางแป้นพิมพ์

11. จงเขียนผังงานและโปรแกรมแสดงจำนวนเต็ม a, b และ c ทุกจำนวนที่น้อยกว่า 500 ที่ค่าของ $a^2 + b^2$ เท่ากับ c^2 (เช่น $3^2 + 4^2 = 5^2$) โดยไม่แสดงค่าซ้ำ เช่น เคยแสดง 3, 4, 5 แล้ว จะไม่แสดง 4, 3, 5 (ข้อแนะนำ : คงต้องใช้วงวนซ้อนกันถึงสามชั้นเพื่อแปรค่าในตัวแปร a, b และ c)

12. ใครเคยเล่นเกม BrainAge™ บนเครื่องเล่นเกม NintendoDS ก็คงเคยทดสอบความไวในการเติมเครื่องหมาย + − × หรือ / ให้กับ สมการง่าย ๆ ที่แสดงบนจอดังตัวอย่างในรูปทางขวาที่ถามว่า 3 ทำอะไรกับ 3 ถึงจะเท่ากับ 9 คำตอบก็คือ × จงเขียนโปรแกรมให้รับตัวเลขจำนวนเต็มจากแป้นพิมพ์ 3 จำนวน จากนั้นแสดงเครื่องหมายที่เหมาะสมที่ทำให้สองจำนวนแรกดำเนินการกันแล้วได้จำนวนที่สาม (เครื่องหมายมีแสดงมีแค่ + − × หรือ /)



13. อีกสักเกมของ BrainAge เกมนี้มีไว้ทดสอบการคำนวณผลต่างของเวลาสองเวลาว่าห่างกันเท่าไร ดังตัวอย่างในรูปทางขวา ถามว่าเวลา 2:50 (นาฬิการูปล่าง) ห่างจากเวลา 12:10 (นาฬิการูปบน) เท่าไร (คำตอบก็คือ 2 ชั่วโมง 40 นาที หมายเหตุ : ไม่ต้องสนใจว่าเวลาที่ให้คำนวณจะข้ามเกิน 12 ชั่วโมง อย่างในกรณีตัวอย่าง ถ้า 2:50 คือตีสองห้าสิบ 12:10 ก็คือเที่ยงคืนสิบ) จงเขียนโปรแกรมรับเวลาเริ่มต้นซึ่งประกอบด้วยตัวเลข 2 จำนวน (ชั่วโมงและนาที เลขชั่วโมงอยู่ในช่วง 1 ถึง 12) และรับเวลาสิ้นสุดซึ่งประกอบด้วยตัวเลขอีก 2 จำนวน (ในรูปแบบเดียวกัน) จากนั้นแสดงผลต่างของเวลาทั้งสองเป็นจำนวนชั่วโมงตามด้วยจำนวนนาที



14. จงเขียนโปรแกรมรับจำนวนเต็มที่แทนวัน เดือน ปี (พ.ศ.) เพื่อแสดงผลการคำนวณว่า วัน เดือนปีที่ป้อนเข้ามานั้นเป็นวันที่เท่าใดของปี เช่น วันที่ 1 มกราคม 2550 เป็นวันที่ 1 ของปี, วันที่ 31 ธันวาคม 2550 เป็นวันที่ 365 ของปี ในขณะที่วันที่ 31 ธันวาคม 2551 เป็นวันที่ 366 ของปี (เพราะเดือนกุมภาพันธ์ของปี พ.ศ. 2551 มี 29 วัน)

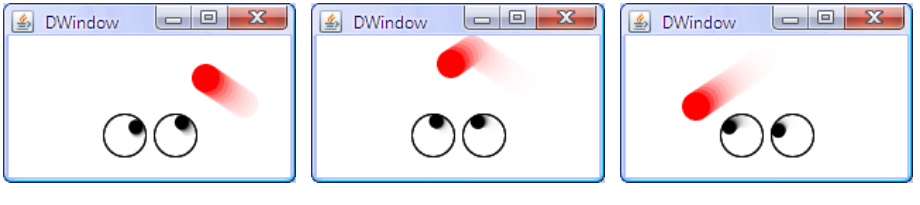
15. *ปัญหา* $3x + 1$: ให้ k คือจำนวนเต็ม เราสามารถเขียนลำดับของจำนวนเต็ม x_n เริ่มจาก $x_0 = k$ จากนั้นคำนวณให้ $x_n = x_{n-1} / 2$ ถ้า x_{n-1} เป็นจำนวนคู่ หรือให้ $x_n = 3x_{n-1} + 1$ ถ้า x_{n-1} เป็นจำนวนคี่ ตัวอย่างเช่น ให้ $k = 10$ จะได้ ลำดับ $x_0, x_1, x_2, x_3, \ldots$ ที่ผลิตได้เป็น 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ... (ซ้ำ 4, 2, 1 ไปเรื่อย ๆ) หรือให้ $k = 15$ จะได้ลำดับเป็น 15, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1, ... (ซ้ำ 4, 2, 1 ไปเรื่อย ๆ) เท่าที่นักคณิตศาสตร์พยายามศึกษากันมา ยังไม่พบจำนวนเต็มบวก k ใด ที่เมื่อผ่านการผลิตลำดับเช่นนี้แล้ว จะไม่ลงท้ายด้วย 4, 2, 1 ซ้ำไปเรื่อย ๆ (แต่ก็ไม่มีใครพิสูจน์ได้ว่าเป็นเช่นนี้กับทุกจำนวนเต็มบวก) จงเขียนโปรแกรมหาว่า ค่า k ใดในช่วง 1 ถึง 1000 ที่ให้ลำดับการผลิตจำนวนด้วยวิธีดังกล่าวที่ยาวที่สุดก่อนจะจบลงที่ 1 เป็นครั้งแรก

16. เราสามารถหา $\sqrt{a}$ ได้ด้วย *วิธีแบ่งครึ่ง* (bisection) ซึ่งอาศัยการกำหนดช่วงของจำนวนจริง $[x_L, x_R]$ ที่เรามั่นใจว่า $\sqrt{a}$ ต้องอยู่ในช่วงนี้แน่ โดยเริ่มที่ช่วง $[x_L, x_R] = [0, a]$ จากนั้นให้ x มีค่าที่ตำแหน่งครึ่งหนึ่งของช่วง $[x_L, x_R]$ นั่นคือให้ $x = (x_L + x_R) / 2$
    - ถ้าพบว่า x^2 มีค่าใกล้กับ a มากพอ ก็ถือว่าได้ x เป็นคำตอบ
    - ถ้า x^2 มีค่ามากกว่า a ให้เปลี่ยน $[x_L, x_R]$ เป็น $[x_L, (x_L + x_R) / 2]$
    - ถ้า x^2 มีค่าน้อยกว่า a ให้เปลี่ยน $[x_L, x_R]$ เป็น $[(x_L + x_R) / 2, x_R]$

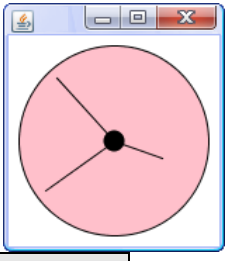
    จงเขียนโปรแกรมรับค่า a ทางแป้นพิมพ์ แล้วใช้วิธี bisection เพื่อหาและแสดงค่าประมาณของ $\sqrt{a}$ ทางจอภาพ

17. จงปรับปรุงโปรแกรมในรหัสที่ 4-3 ซึ่งแสดงลูกบอลเด้งขอบไปมาในวินโดว์ ให้เด้งตัวชี้เมาส์ด้วย นั่นคือให้เคลื่อนที่กลับทิศทุกครั้งที่วิ่งชนตัวชี้เมาส์ (ใช้ `w.getMouse().getX()` และ `w.getMouse().getY()` เพื่ออ่านตำแหน่งของตัวชี้เมาส์ในวินโดว์ `w`)

18. จงนำส่วนของโปรแกรมที่แสดงลูกตามองตามตัวชี้เมาส์ของแบบฝึกหัดในบทที่แล้ว มารวมกับส่วนของโปรแกรมแสดงลูกบอลเด้งผนังในบทนี้ ให้ได้โปรแกรมลูกตามองตามลูกบอลเด้งผนังดังตัวอย่างข้างล่างนี้



19. จงเขียนโปรแกรมแสดงนาฬิกาเข็ม (สามเข็ม) เดินได้ดังแสดงข้างขวานี้ ให้เข็มวินาทีขยับครั้งละ 1 วินาที เข็มนาทีขยับครั้งละนาที ส่วนเข็มชั่วโมงขยับทีละนิดตามเข็มนาทีด้วย (อย่าขยับทีละชั่วโมงตอนเปลี่ยนชั่วโมง จะดูไม่สวย) รหัสข้างล่างนี้แสดงส่วนต้นของโปรแกรมที่ขอเวลาปัจจุบันจากระบบ จะได้แสดงนาฬิกาของเวลาขณะทำงาน



```
import java.util.Date;
import jlab.graphic.DWindow;

public class Clock {
  public static void main(String[] args) {
    Date now = new Date();
    int h = now.getHours();   // h คือชั่วโมงตอนนี้
    int m = now.getMinutes(); // m คือนาทีตอนนี้
    int s = now.getSeconds(); // s คือวินาทีตอนนี้
    ...
```

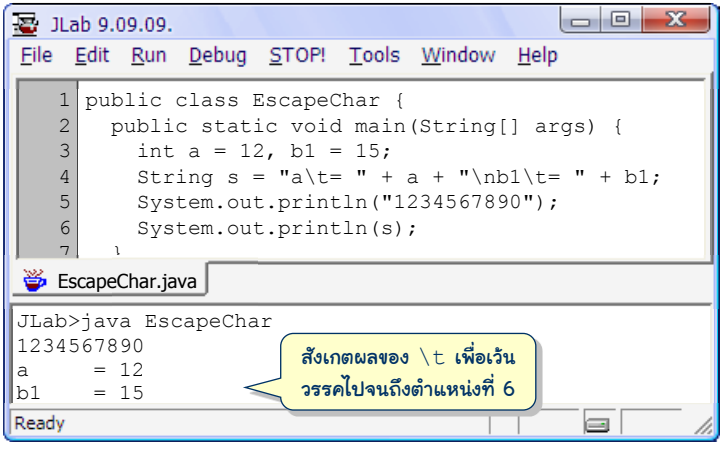
ข้อความ

คอมพิวเตอร์ชอบคำนวณ ชอบประมวลผลจำนวน แต่มนุษย์เราเข้าใจข้อความ ภาพ และเสียงได้ดีกว่าจำนวน บทนี้นำเสนอการเขียนโปรแกรมเพื่อประมวลผลข้อความ จาวามีประเภทข้อมูลชื่อ `String` ที่ใช้ในการเก็บอักขระ ตัวอักษร ตัวเลข และเครื่องหมายของภาษาต่างๆ ที่ใช้สื่อสารกันในโลก นักเขียนโปรแกรมจาวาจึงต้องเข้าใจลักษณะ ความสามารถ และข้อจำกัดต่าง ๆ ของสตริง (หรือที่เรียกกันว่าสายอักขระ) นอกจากนี้จะนำเสนอคำสั่งวงวนเพิ่มเติมเพื่อให้เขียนวงวนได้สั้นกะทัดรัด รวมทั้งนำเสนอการอ่านเขียนแฟ้มข้อมูลที่เป็นข้อความ เพราะเป็นแหล่งที่มาของข้อความเพื่อการประมวลผล พร้อมทั้งการประยุกต์บริการต่าง ๆ ของสตริงในการประมวลผลข้อความ

สตริง

เราได้ใช้สตริงกันมาตั้งแต่โปรแกรมแรกที่เขียนในหนังสือเล่มนี้ ทั้งนี้เพราะสตริงคือข้อมูลที่เก็บข้อความอันประกอบด้วยตัวอักษร อักขระ สัญลักษณ์เครื่องหมายต่างๆ ที่สื่อสารกับผู้ใช้งานได้ดี เราเขียนสตริงด้วยการเขียนข้อความไว้ภายในเครื่องหมายอัญประกาศคู่ เช่น `"สวัสดี Hello 你好"` เนื่องจากเครื่องหมาย " ถูกนำไปใช้เป็นตัวเปิดปิดสตริง แล้วถ้าต้องการสตริงที่ภายในมีเครื่องหมาย " จะทำอย่างไร จาวากำหนดว่าให้เขียน \" แล้วตัวแปลโปรแกรมจะแปลงให้เป็นเครื่องหมาย " ตัวเดียวในสตริง เช่น หากต้องการแสดงข้อความ **ผมพูดว่า "สวัสดี"** ก็เขียนว่า `System.out.println("ผมพูดว่า \"สวัสดี\"");` แล้วถ้าต้องการสตริงที่มีเครื่องหมาย \ ไว้ภายใน จะทำอย่างไร จาวากำหนดว่าให้เขียน \\ เช่น ต้องเก็บชื่อแฟ้ม `c:\temp\a.txt` ในสตริง ก็ต้องเขียน `"c:\\temp\\a.txt"` นอกจาก \" และ \\ ยังมี `\n` `\t` และอื่น ๆ โดยที่ `\n` แทนการขึ้นบรรทัดใหม่ ส่วน `\t` แทนรหัส tab เพื่อให้ตัวแสดงเลื่อนไปที่ตำแหน่งตั้ง

ระยะที่ระบบกำหนดไว้ (ส่วนใหญ่ตั้งระยะไว้ห่างกัน 6 ตัวอักษร) ดังตัวอย่างในรูปที่ 5-1 จาวาเรียกเครื่องหมาย \ นี้ว่า *อักขระหลีก* (escape character) ซึ่งเป็นอักขระที่บอกให้ตัวแปลโปรแกรมรู้ว่าอักขระอีกตัวที่ตามมานั้นมีความหมายไม่เหมือนกับที่ควรเป็น ให้ตัวแปลหลีกออกจากการตีความแบบปกติไปตีความในลักษณะอื่นตามที่ภาษากำหนดไว้ เช่น `\n` ก็บอกว่า `n` ที่ตามหลัง \ นั้นมีความหมายพิเศษคือขึ้นบรรทัดใหม่ เป็นต้น



รูปที่ 5-1 ตัวอย่างการใช้อักขระหลีกในสตริง

สตริงจัดเก็บอักขระแต่ละตัวเรียงติดกันไปในหน่วยความจำ แต่ละตัวจึงมีหมายเลขตำแหน่งเรียกว่า*ดัชนี* (index) กำกับ สตริงที่มีอักขระ n ตัว ตำแหน่งซ้ายสุดคือดัชนี 0 ตำแหน่งขวาสุดคือดัชนี $n-1$ ต้องอย่าลืมว่า การใช้อักขระหลีกในสตริงนั้น ถึงแม้จะใช้อักขระสองตัว แต่สองตัวนี้จะถูกเปลี่ยนเป็นอักขระตัวเดียว เก็บในหน่วยความจำ จึงกินเนื้อที่เพียงหนึ่งตัวเท่านั้น ตัวอย่างเช่น สตริง `"123\n456"` มีอักขระ 7 ตัว (นับ `\n` เป็นหนึ่งตัว) เราเรียกสตริงที่มีอักขระ n ตัวว่า มีความยาวเป็น n ดังนั้น สตริง `""` จึงยาวศูนย์

บริการที่น่าสนใจของสตริง

สตริงไม่ใช่ข้อมูลพื้นฐานในจาวา แต่ถือว่าเป็นข้อมูลแบบอ็อบเจกต์ (จะอธิบายในบทหลังๆ) มีวิธีการจัดเก็บข้อมูลที่ถูกออกแบบไว้ในคลาสมาตรฐานชื่อ `String` ของจาวา ดังนั้น การจัดการกับสตริงจึงอาศัยการเรียกเมท็อดที่สตริงมีให้ใช้ (`String` มีบริการอยู่ 65 เมท็อดในจาวารุ่นที่ 6) นอกจากนี้ตัวดำเนินการ + ซึ่งเมื่อใช้กับสตริงยังหมายถึงการต่อสตริงด้วย ตารางที่ 5-1 แสดงบางเมท็อดของสตริง ขอให้สังเกตเมท็อด `toUpperCase`, `toLowerCase`, `substring`, และ `trim` เป็นเมท็อดที่นักเขียนโปรแกรมหลายคนตีความว่าจะเกิดการเปลี่ยนแปลงกับสตริง เช่น `s = "  ABC  "` เมื่อเรียก `s.trim()` อาจคิดว่าจะทำให้ `s` กลายเป็น `"ABC"` แต่ผู้ออกแบบ

สตริงได้กำหนดไว้ว่า s ไม่เปลี่ยน แต่ s.trim() จะคืนสตริงใหม่ที่ได้ผลจากการนำ s ไปลบช่องว่างทางซ้ายและขวาออก ขอเน้นว่า ไม่มีเมท็อดใดที่ใช้กับสตริงหนึ่ง แล้วจะเปลี่ยนแปลงสตริงนั้น (ตรงนี้เป็นสิ่งที่ผู้ออกแบบ String ตั้งใจไว้เช่นนั้น)

ตารางที่ 5-1 ตัวอย่างเมท็อดให้บริการของสตริง

การเรียกใช้	ความหมาย
s.length()	คืนความยาวของ s (ซึ่งคือจำนวนอักขระใน s)
s.trim()	คืนสตริงใหม่ที่ตัดอักขระว่างทางซ้ายและขวาของ s
s.toUpperCase()	คืนสตริงใหม่ที่เหมือน s แต่ตัวอักษรอังกฤษทุกตัวเป็นตัวใหญ่หมด
s.toLowerCase()	คืนสตริงใหม่ที่เหมือน s แต่ตัวอักษรอังกฤษทุกตัวเป็นตัวเล็กหมด
s.substring(i, j)	คืนสตริงใหม่ที่ได้จากตัวที่ i ถึง j-1 ของ s
s.indexOf(t, i)	คืนตำแหน่งใน s ที่พบ t เริ่มจากดัชนี i ใน s ถ้าไม่พบคืน -1
s.equals(t)	คืนค่า true ถ้า s เหมือนกับ t ทุกอักขระ ถ้าไม่เหมือนคืน false
s.equalsIgnoreCase(t)	เหมือน s.equals(t) แต่ถือว่าตัวอังกฤษใหญ่หรือเล็กเหมือนกัน

```
//..........0123456789012345678901..............
String a = "   Java Programming   ";
String s = a.trim();                 // s = "Java Programming"
int i = a.length() - s.length();     // i = 22 - 16
String b = s.toUpperCase();          // b = "JAVA PROGRAMMING"
String c = s.toLowerCase();          // c = "java programming"
String d = a.substring(3, 7);        // d = "Java"
int j = a.indexOf("Prog", 0);        // j = 8
boolean e1 = "Java".equals(d);       // e1 เป็นจริง
boolean e2 = d.equals("Java");       // e2 เป็นจริง
boolean e3 = "java".equals(d);       // e3 เป็นเท็จ
```

รหัสที่ 5-1 ตัวอย่างการใช้เมท็อดของสตริง

รหัสที่ 5-1 แสดงตัวอย่างการเรียกใช้เมท็อดของสตริง โดยทั่วไปเรานำเมท็อดเหล่านี้ไปประยุกต์กับเหตุการณ์ดังต่อไปนี้

- trim : ใช้ในกรณีที่ข้อมูลที่ได้รับมาอาจมีช่องว่างทางซ้ายหรือขวา ซึ่งไม่มีความสำคัญกับตัวข้อมูล เช่น อ่านสตริงจากแป้นพิมพ์ จากแฟ้มข้อมูล หากลบช่องว่างทางซ้ายและขวาออก จะทำให้การประมวลผลกระทำได้ง่ายขึ้น

- indexOf : มักใช้ประกอบกับ substring ในการค้นและสกัดข้อความย่อยในสตริงออกมาใช้งาน เช่น เว็บเพจ (web page) ในอินเทอร์เน็ตมักมีบรรทัดที่เก็บที่อยู่ของไปรษณีย์อิเล็กทรอนิกส์ (email address) ในรูปแบบ mailto:*username*@server ซึ่งเราสามารถสกัดชื่อผู้ใช้จากสตริงที่ปรากฏระหว่างสตริงย่อย "mailto:" กับ "@" ได้ด้วยการใช้ indexOf ค้น "mailto:" หนึ่งครั้ง และค้น "@" อีกครั้ง เมื่อได้ตำแหน่งของสองตัวนี้ก็ใช้ substring ดึงสตริงย่อยออกไปใช้งานต่อ

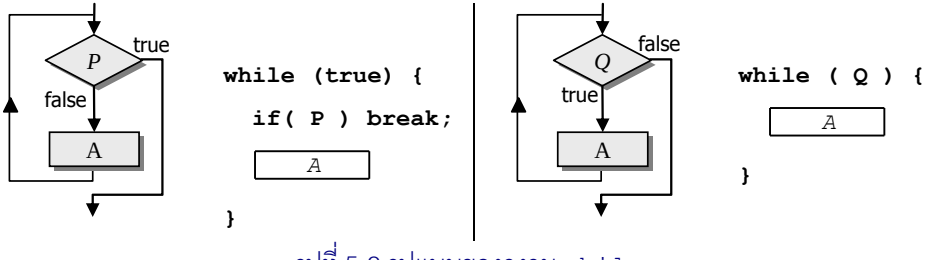
- `toLowerCase, toUpperCase` : ในกรณีที่ต้องการค้นหรือเปรียบเทียบตัวอักษรภาษาอังกฤษ เราอาจไม่สนใจว่าเป็นตัวใหญ่หรือตัวเล็ก เช่น การค้น `"mailto:"` นั้นอาจเป็น `"Mailto:"` หรือ `"MailTo:"` หรือ `"MAILTO:"` ก็ได้
- `equals` : การเปรียบเทียบความเท่ากันของสตริงมีการใช้งานมากมาย แต่มีข้อพึงระวังประการหนึ่งคือ อย่าใช้ `s==t` เปรียบเทียบสตริงในจาวา ให้ใช้ `s.equals(t)` เสมอ ถึงแม้ว่าการเขียน `s==t` จะไม่ผิดหลักไวยากรณ์ แต่บางครั้งก็เปรียบเทียบความเท่ากันได้ บางครั้งก็บอกว่าไม่เท่าทั้ง ๆ ที่เท่า เช่น `"A".toLowerCase() == "a"` จะได้ค่าเท็จ จะได้กล่าวถึงประเด็นนี้ในบทที่เกี่ยวกับอ็อบเจกต์ในภายหลัง

วงวน (อีกครั้ง)

การประมวลผลข้อความมีการใช้วงวนมากมาย เราได้ทราบลักษณะการเขียนโปรแกรมที่ใช้วงวนทั้งแบบรู้จบและไม่รู้จบกันมาแล้วด้วยการใช้ `while(true)` กับ `if... break` แต่ด้วยการที่วงวนเป็นกลไกการประมวลผลที่ใช้บ่อยมากๆ ภาษาคอมพิวเตอร์ทั่วไปจึงมักมีคำสั่งเพิ่มเติมเกี่ยวกับวงวน เพื่อให้เขียนโปรแกรมได้สั้น กะทัดรัด และถ่ายทอดแนวคิดการทำงานได้ดีขึ้น สำหรับภาษาจาวา มีคำสั่งวงวนอยู่สามแบบที่จะนำเสนอต่อไปนี้คือ `while`, `do-while` และ `for`

วงวน while

รูปแบบการเขียนโปรแกรมที่ใช้ `if...break` ที่ต้นวงวน เพื่อตรวจสอบว่าจะหลุดจากวงวนหรือไม่ มีให้เห็นกันมากพอสมควร ดังแสดงในรูปที่ 5-2 ทางซ้าย หากเรากลับเงื่อนไขที่ต้นวงวนให้หลุดจากวงวนเมื่อเป็นเท็จ และทำต่อในวงวนเมื่อเป็นจริง จะได้ผังงานและเขียนเป็นคำสั่งได้ดังรูปที่ 5-2 ทางขวา จากคำสั่งที่แสดง อ่านได้ว่า “ตราบเท่าที่ Q เป็นจริง ให้ทำ A”



รูปที่ 5-2 รูปแบบของวงวน `while`

รหัสที่ 5-2 แสดงส่วนของโปรแกรมเพื่อทดสอบว่า `n` คือจำนวนเฉพาะหรือไม่ (ที่ได้เขียนในรหัส 3-14) จากที่เคยใช้วงวน `while(true)` ในรหัสที่ 5-2 ทางซ้าย ซึ่งมีการตรวจสอบสองกรณีที่ทำให้ออกจากวงวนคือ `(k>=n)` หรือ `(n%k == 0)` ก็ให้กลับการตรวจสอบเป็น จะทำ

ต่อในวงวนเมื่อ (k<n) และ (n%k !=0) เพียงเท่านี้ ก็ได้วงวนที่สั้นดังรหัสที่ 5-2 ทางขวา อ่านเข้าใจได้ง่าย

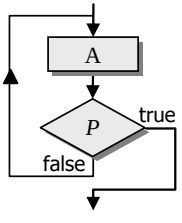
```
int k = 2;
while (true) {
  if (k >= n) break;
  if ((n % k) == 0) break;
  k++;
}
```

```
int k = 2;
while (k < n && (n%k) != 0) {
  k++;
}
```

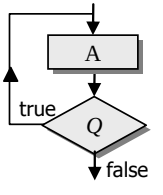
รหัสที่ 5-2 การใช้วงวน while เพื่อการทดสอบจำนวนเฉพาะของ n

วงวน do-while

สำหรับกรณีที่ใช้ if...break ที่คำสั่งสุดท้ายของวงวน while(true) เพื่อตรวจสอบว่าจะหลุดจากวงวนหรือไม่ ดังแสดงในรูปที่ 5-3 ทางซ้าย หากเรากลับเงื่อนไขให้หลุดจากวงวนเมื่อเป็นเท็จ และขึ้นไปทำต่อในวงวนเมื่อเป็นจริง จะได้ผังงานและเขียนเป็นคำสั่งได้ดังรูปที่ 5-3 ทางขวา จากคำสั่งที่แสดง อ่านได้ว่า “ทำ A ตราบเท่าที่ Q เป็นจริง”




```
while (true) {
  A
  if (P) break;
}
```




```
do {
  A
} while (Q);
```

รูปที่ 5-3 รูปแบบของวงวน do-while

รหัสที่ 5-3 แสดงส่วนของโปรแกรมเพื่อหารากที่สองของ x (ที่ได้เขียนในรหัส 3-10) จากเดิมใช้วงวน while(true) ในรหัสที่ 5-3 ทางซ้าย ซึ่งวนจนได้ค่า x ที่ใกล้เคียงกับที่ต้องการ (นั่นคือเมื่อ $(x^2 - a) < 10^{-5}$) ก็ให้กลับการตรวจสอบว่า จะทำต่อในวงวนเมื่อยังได้ค่าที่ห่างจากคำตอบคือ $(x^2 - a) \geq 10^{-5}$ เพียงเท่านี้ ก็ได้วงวนที่สั้นดังรหัสที่ 5-3 ทางซ้าย อ่านเข้าใจง่ายในอีกลักษณะหนึ่ง

```
double x = 1;
while (true) {
  x = (x + a/x) / 2.0;
  if ((x*x-a) < 1e-5) break;
}
```

```
double x = 1;
do {
  x = (x + a/x) / 2.0;
} while ((x*x-a) >= 1e-5);
```

รหัสที่ 5-3 การใช้วงวน do-while เพื่อหารากที่สองของ x

วงวน for

วงวนที่กำหนดว่าให้ทำซ้ำ n รอบก็เป็นวงวนอีกรูปแบบหนึ่งที่ใช้กันมาก ที่ผ่านมาได้เสนอให้เขียนคำสั่งดังรหัสที่ 5-4 ทางซ้าย มีตัวแปร k เริ่มที่ 0 เพิ่มรอบละ 1 วนทำจนกระทั่ง k มีค่า

มากกว่าหรือเท่ากับ n ซึ่งสามารถเขียนให้สั้น เข้าใจง่ายด้วยวงวน for ดังรหัสที่ 5-4 ทางขวา อ่านเข้าใจได้ถึงจุดประสงค์ของวงวนที่บรรทัด for เลยว่า ให้ k เริ่มที่ 0 วงวนนี้จะทำตราบเท่าที่ k<n และหลังทุกรอบที่ทำเสร็จให้ทำ k++ ซึ่งขั้นตอนการทำงานเหมือนรหัสทางซ้ายทุกประการ

```
int k = 0;
while (true) {
  if (k >= n) break;
  ...
  k++;
}
```

```
int k;
for (k=0; k<n; k++) {
  ...
}
```

รหัสที่ 5-4 การใช้วงวน for เพื่อสร้างวงวนทำซ้ำจำนวน n รอบ

รหัสที่ 5-5 ทางซ้ายแสดงการใช้ for ซึ่งเริ่ม k ที่ค่าเท่ากับ n ลดลงรอบละ 1 จนเมื่อ k เท่ากับ 0 (k>=1 เป็นเท็จ) ก็ออกจากวงวน ส่วนรหัสที่ 5-5 ทางขวาเริ่มค่า k ที่ 1 เพิ่มรอบละ 2 (จึงเป็นจำนวนคี่) เมื่อใดที่ k>n ก็ออกจากวงวน

```
int k;
for (k=n; k>=1; --k) {
  ...
}
```

```
int k;
for (k=1; k<=n; k+=2) {
  ...
}
```

รหัสที่ 5-5 ตัวอย่างการใช้วงวน for ในลักษณะอื่น ๆ

กล่าวโดยสรุป ภายในวงเล็บของ for นั้นแบ่งเป็นสามส่วน แต่ละส่วนแยกออกจากกันด้วยเครื่องหมาย ; ส่วนแรกเป็นการตั้งค่าเริ่มต้นให้กับตัวแปร ส่วนที่สองเป็นเงื่อนไขที่ทดสอบว่าถ้าเป็นจริงจะทำต่อในวงวน และส่วนสุดท้ายจะทำหลังทำงานในวงวนเสร็จในแต่ละรอบ จึงสามารถเขียนส่วนของโปรแกรมทดสอบจำนวนเฉพาะได้ดังรหัสที่ 5-6 ซึ่งทำงานเหมือนกันทั้ง 4 แบบ แต่เขียนต่างกันเล็กน้อย รหัสซ้ายบน ไม่มีส่วนที่สามของ for เพราะย้ายมาเขียนในวงวน ส่วนรหัสขวาบนนำ k++ ย้ายมาไว้ส่วนที่สามของ for ตามปกติ แต่ในวงวนไม่ได้เขียนอะไร ในกรณีที่ภายใน {} มีเพียงคำสั่งเดียว สามารถละไม่เขียน {} ได้ (แบบเดียวกับกรณีกลุ่มคำสั่งกับกรณีคำสั่งเดียวที่เขียนหลัง if กับ else) จึงสามารถเขียนได้ดังรหัสที่ 5-6 ด้านล่าง

```
int k;
for(k=2; k<n && (n%k)!=0; ) {
  k++;
}
```

```
int k;
for(k=2; k<n && (n%k)!=0; k++) {
}
```

```
int k;
for(k=2; k<n && (n%k)!=0; )
  k++;
```

```
int k;
for(k=2; k<n && (n%k)!=0; k++) ;
```

รหัสที่ 5-6 ตัวอย่างการใช้ for เพื่อทดสอบจำนวนเฉพาะ

การละส่วนที่หนึ่งหรือส่วนที่สองก็ทำได้ แต่ต้องมี ; คงไว้ให้รู้ว่าส่วนใดไม่มี ในกรณีที่ละส่วนที่สอง จะเหมือนกับมีคำว่า true อยู่ส่วนที่สอง หมายความว่าให้หมุนในวงวนตลอด (นอกเสียจากว่าในวงวนจะมี if...break) ดังนั้น การเขียน for(;;) จึงเหมือนกับ while(true)

เรื่องสุดท้ายที่ควรรู้เกี่ยวกับวงวน `for` คือ เราสามารถประกาศตัวแปรที่ใช้เฉพาะในวงวนตรงส่วนที่หนึ่งของ `for` ได้เลย เช่น รหัสที่ 5-7 แสดงการประกาศตัวแปร `k` เพื่อใช้งานในวงวนเท่านั้น พอออกนอกวงวนก็ใช้ไม่ได้ การเขียนในลักษณะนี้ชี้ให้เห็นว่า เป็นความประสงค์ของนักเขียนโปรแกรมที่ต้องการใช้ตัวแปรที่ประกาศใน `for` เฉพาะในวงวนเท่านั้น หากเผลอไปใช้นอกวงวน ตัวแปลโปรแกรมจะแจ้งข้อผิดพลาดให้รู้

```
for (int k=0; k<n; k++) {
  ...
}
```

รหัสที่ 5-7 ตัวอย่างการประกาศตัวแปรที่ใช้ใน `for`

ตัวอย่างการใช้งาน

หัวข้อนี้นำเสนอตัวอย่างการใช้เมท็อดต่าง ๆ ของสตริงที่ได้นำเสนอมา อันได้แก่ โปรแกรมหาเลขโดดตรวจสอบของรหัสแท่งตามมาตรฐาน EAN-13 ซึ่งใช้ `substring` ในการหยิบตัวอักษรภายในออกมาใช้งาน โปรแกรมตรวจพาลินโดรมที่ใช้ `equalsIgnoreCase` ในการเปรียบเทียบสตริง เพราะปัญหานี้ไม่สนใจตัวใหญ่ตัวเล็ก ปิดท้ายด้วยโปรแกรมเข้ารหัสแบบ ROT-13 ที่มีการใช้ `indexOf` ในการค้นข้อมูล

มาตรฐานรหัสแท่ง EAN-13

ผู้อ่านคงเคยเห็นรหัสแท่ง (bar code) ดังตัวอย่างในรูปที่ 5-4 ที่พิมพ์ไว้ข้างสินค้าที่จำหน่ายกันทั่วไป โดยพนักงานขายจะใช้เครื่องอ่านรหัสแท่งที่ฉายแสงไปกระทบและอ่านรหัสแท่งของสินค้ากลับเข้าเครื่องคอมพิวเตอร์เพื่อนำข้อมูลเกี่ยวกับสินค้านั้นออกมาประมวลผล ข้อมูลที่อ่านเข้ามาเป็นตัวเลข 13 ตัว สองตัวซ้ายระบุประเทศหรือภูมิภาค ห้าตัวถัดมาระบุหมายเลขผู้ผลิต ตามด้วยอีกห้าตัวระบุหมายเลขสินค้า และตัวขวาสุดหนึ่งตัวสุดท้าย เป็นตัวเลขที่คำนวณได้มาจาก 12 ตัวทางซ้าย เรียกว่า*เลขโดดตรวจสอบ* (check digit) เมื่ออ่านรหัสแท่งเข้าเครื่อง ระบบจะตรวจสอบว่าตัวขวาสุดมีค่าเป็นไปตามสูตรที่คำนวณได้จาก 12 หลักทางซ้ายที่อ่านมาหรือไม่ ถ้าไม่ตรง แสดงว่าเกิดข้อผิดพลาดในการอ่าน ระบบจะไม่ส่งเสียงให้พนักงานขายได้ยิน พนักงานผู้นั้นก็ต้องทำใหม่ ถ้ารหัสตรงตามสูตรก็ส่งเสียงให้ทราบว่าถูกต้อง

รูปที่ 5-4 ตัวอย่างรหัสแท่งที่มีหมายเลขจำนวน 13 หลัก

กำหนดให้ตำแหน่งของตัวเลขซ้ายสุดคือ 0 ดังนั้น ตัวเลข 12 ตัวซ้ายอยู่ที่ตำแหน่ง 0 ถึง 11 ให้ d_k คือตัวเลขตัวที่ตำแหน่ง k เราสามารถคำนวณเลขโดดตรวจสอบ c ตามสูตรต่อไปนี้

$$c = \left(10 - \left(\sum_{\substack{0 \le k \le 11 \\ \text{k is even}}} d_k + 3 \sum_{\substack{0 \le k \le 11 \\ \text{k is odd}}} d_k \right) \bmod 10 \right) \bmod 10$$

จากตัวอย่างในรูปที่ 5-4 รหัสแท่งคือ 9781932698183 สามารถคำนวณเลขโดดตรวจสอบได้เท่ากับ (10 – ((9+8+9+2+9+1) + 3(7+1+3+6+8+8)) mod 10) mod 10 = (10 – (38 + 3×33) mod 10) mod 10 = (10 – 7) mod 10 = 3 ซึ่งตรงกับตัวขวาสุดในรูป

รหัสที่ 5-8 แสดงโปรแกรมที่รับตัวเลข 12 หลักมาคำนวณเลขโดดตรวจสอบ เนื่องจากเรารับตัวเลข 12 หลักเข้ามาทางแป้นพิมพ์ด้วย `nextInt` ไม่ได้ เพราะ `int` เก็บจำนวนเต็มได้แค่ 10 หลัก ดังนั้น จึงเลือกอ่านเข้ามาเป็นสตริงด้วย `nextLine` ใช้ `substring` หยิบออกมาทีละหลัก แล้วใช้ `Integer.parseInt` (ที่ได้อธิบายในบทที่ 2) เปลี่ยนเป็น `int` ออกมาคำนวณ

```
01 import java.util.Scanner;
02 // โปรแกรมคำนวณเลขโดดตรวจสอบตามมาตรฐาน EAN-13
03 public class EAN13 {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("ตัวเลข 12 หลัก = ");
07     String d = kb.nextLine();
08     int s = 0;
09     s += Integer.parseInt(d.substring(0, 1));      // ตำแหน่งคู่
10     s += Integer.parseInt(d.substring(2, 3));
11     s += Integer.parseInt(d.substring(4, 5));
12     s += Integer.parseInt(d.substring(6, 7));
13     s += Integer.parseInt(d.substring(8, 9));
14     s += Integer.parseInt(d.substring(10, 11));
15     s += 3 * Integer.parseInt(d.substring(1, 2)); // ตำแหน่งคี่
16     s += 3 * Integer.parseInt(d.substring(3, 4));
17     s += 3 * Integer.parseInt(d.substring(5, 6));
18     s += 3 * Integer.parseInt(d.substring(7, 8));
19     s += 3 * Integer.parseInt(d.substring(9, 10));
20     s += 3 * Integer.parseInt(d.substring(11, 12));
21     System.out.println("Check Digit คือ " + ((10-(s%10))%10));
22   }
23 }
```

รหัสที่ 5-8 โปรแกรมคำนวณเลขโดดตรวจสอบตามมาตรฐาน EAN-13

รหัสที่ 5-8 แสดงคำสั่งที่มีลักษณะซ้ำๆ กัน จึงน่าจะใช้วงวนหยิบออกมาทีละตัว (ด้วยการใช้ตัวแปรระบุดัชนีของอักขระในสตริง) เพื่อแปลงเป็นจำนวนเต็มแล้วคำนวณ ได้ดังรหัสที่ 5-9 ใช้วงวน `for` ให้ค่า `k` = 0,1, ..., 11 โดยมีการตรวจสอบ `k` ซึ่งเป็นดัชนีที่สนใจภายในวงวนว่า มีค่าเป็นจำนวนคี่หรือคู่ จะได้ตัดสินใจว่าจะต้องคูณด้วย 3 หรือไม่ตามสูตร หรือจะเขียนแยกเป็นสองวงวน

วงวนหนึ่งหยิบเฉพาะตำแหน่งคู่ และอีกวงวนหยิบตัวเลขที่ตำแหน่งคี่ จะได้การทำงานในอีกลักษณะที่คงให้ผลลัพธ์เหมือนกัน ขอให้ผู้อ่านลองเขียนแบบที่สามด้วยตนเอง

```
01 import java.util.Scanner;
02 // โปรแกรมคำนวณเลขโดดตรวจสอบตามมาตรฐาน EAN-13
03 public class EAN13 {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("ตัวเลข 12 หลัก = ");
07     String d = kb.nextLine();
08     int s = 0;
09     for (int k=0; k<=11; k++) {
10       int v = Integer.parseInt(d.substring(k, k+1));
11       if (k % 2 == 0) {
12         s += v;    // ตำแหน่งคู่
13       } else {
14         s += 3*v; // ตำแหน่งคี่
15       }
16     }
17     System.out.println("Check Digit คือ " + (10-(s%10))%10);
18   }
19 }
```

รหัสที่ 5-9 โปรแกรมคำนวณเลขโดดตรวจสอบตามมาตรฐาน EAN-13 (แบบใช้วงวน)

พาลินโดรม

พาลินโดรม (palindrome) คือคำ วลี ประโยค หรือจำนวน ที่มีตัวอักษรหรือตัวเลขจากซ้ายไปขวาเหมือนกับจากขวามาซ้าย (โดยทั่วไปจะละเลยเครื่องหมายหรือเว้นวรรคในประโยคหรือวลี) เช่น civic, Was it a rat I saw?, ทายาท, กาฝาก, นางดงงดงาน, 11311 (จำนวนเฉพาะที่เป็นพาลินโดรม), 20-02-2002 (วันเดือนปีที่เป็นพาลินโดรม) ขอเขียนโปรแกรมแบบง่ายที่ตรวจสอบว่าคำ (ขอเน้นว่าตรวจสอบเฉพาะคำเท่านั้น) ที่ได้รับทางแป้นพิมพ์คือพาลินโดรมหรือไม่ ดังรหัสที่ 5-10

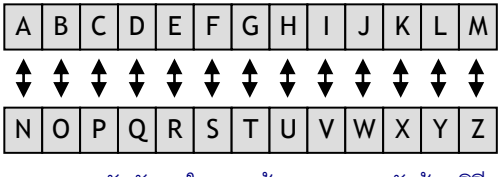
โปรแกรมในรหัสที่ 5-10 เริ่มด้วยการรับคำทางแป้นพิมพ์ (บรรทัดที่ 7) จากนั้นเตรียมตัวแปร `m` เก็บความยาวของสตริง, `n` เก็บดัชนีตัวหลังสุดของครึ่งซ้ายของสตริง เช่น ถ้า `word = "0123210"` ซึ่งยาว 7, ดังนั้น `n` จะมีค่า 7/2 – 1 = 2, และมีตัวแปร `k` เป็นดัชนีของสตริงเริ่มจาก 0 เพิ่มทีละหนึ่งในแต่ละรอบ จนถึงค่า `n` เพื่อจับคู่เปรียบเทียบตัวที่ `k` กับตัวที่ `m-k-1` (บรรทัดที่ 11 ถึง 13) ถ้าเป็นพาลินโดรมต้องเหมือนกัน การเปรียบเทียบนั้นใช้ `equalsIgnoreCase` เพราะเราไม่สนใจความแตกต่างของตัวอังกฤษเล็กหรือตัวใหญ่ เก็บผลการเปรียบเทียบใส่ตัวแปร `isPalindrome` เพื่อเอาไว้ทดสอบในเงื่อนไขของวงวน `for` ซึ่งถ้าเป็นเท็จ คือให้ออกจากวงวน ก็เมื่อพบตัวที่ไม่เหมือนกัน (`isPalindrome` เป็นเท็จ) หรือไม่ก็ `k>n` (คือได้เปรียบเทียบครบทุกคู่แล้วว่าเหมือนกัน)

```
01 import java.util.Scanner;
02 // โปรแกรมตรวจสอบคำที่ได้รับว่าเป็นพาลินโดรมหรือไม่
03 public class Palindrome {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("คำ = ");
07     String word = kb.nextLine();
08     int m = word.length(), n = m/2 - 1;
09     boolean isPalindrome = true;
10     for (int k=0; isPalindrome && k <= n; k++) {
11       String c1 = word.substring(k, k+1);
12       String c2 = word.substring(m-k-1, m-k);
13       isPalindrome = c1.equalsIgnoreCase(c2);
14     }
15     if (!isPalindrome) {
16       System.out.print("ไม่");
17     }
18     System.out.println("เป็นพาลินโดรม");
19   }
20 }
```

รหัสที่ 5-10 โปรแกรมตรวจสอบคำที่ได้รับว่าเป็นพาลินโดรมหรือไม่

การเข้ารหัสลับแบบ ROT-13

ROT-13 เป็นกลวิธีในการเข้ารหัสหรืออำพรางข้อความที่แสดงกันในอินเทอร์เน็ต (สมัยก่อน) ทั้งนี้ไม่ได้มีจุดประสงค์ไม่ให้ผู้อ่านถอดรหัสไม่ได้ เพียงแต่ไม่ให้เห็นข้อความดังกล่าวทันที ถ้าอยากรู้ต้องถอดรหัสซึ่งกระทำได้ง่ายๆ เพราะใช้วิธีเดียวกับการเข้ารหัส จึงมักใช้กับข้อความประเภทเฉลยปริศนา ข้อความที่ถ้าอ่านก่อนอาจเสียความรู้สึก หรืออ่านแล้วอาจกระทบกระเทือนจิตใจ เป็นต้น ROT-13 เป็นวิธีที่คล้ายกับการเข้ารหัสลับข้อความของซีซาร์ในยุคโรมันโบราณ ซึ่งอาศัยการแทนแต่ละตัวอักษรในข้อความต้นฉบับด้วยตัวอักษรอื่นตามกฎที่ตกลงกันก่อน ROT-13 อาศัยการแทนตัวอักษรดังรูปที่ 5-5 โดยแทนตัวอักษรหนึ่งด้วยตัวอักษรที่ห่างจากตัวนั้นไป 13 ตัวในลำดับ ABCD... (ในกรณีที่นับเลย Z ก็ให้หมุนวนมานับต่อที่ A, ROT-13 ย่อมาจาก “rotate by 13 places”) เช่น "Java Programming" เปลี่ยนเป็น "Wnin Cebtenzzvat" และ "Wnin Cebtenzzvat" ก็เปลี่ยนเป็น "Java Programming" เป็นต้น



รูปที่ 5-5 การแทนตัวอักษรในการเข้าและถอดรหัสด้วยวิธี ROT-13

เราจะมาเขียนโปรแกรมรับข้อความจากผู้ใช้ จากนั้นเข้ารหัสด้วย ROT-13 และแสดงทางจอภาพ (การถอดรหัสก็ใช้วิธีเดียวกัน) ดังรหัสที่ 5-11 เริ่มด้วยการอ่านข้อความทางแป้นพิมพ์ จากนั้นเตรียมสตริง `upperCase` และ `lowerCase` ที่เก็บตัวอักษรตั้งแต่ A ถึง Z และ a ถึง z ตามลำดับ จะใช้สตริงสองตัวนี้เพื่อการแปลงตัวอักษร มีตัวแปร `rot13` ไว้เก็บผลลัพธ์ แล้วเข้าวงวนหยิบตัวอักษรทีละตัวในข้อความที่ได้รับ (โดยใช้ `substring` ในบรรทัดที่ 13) บรรทัดที่ 14 ค้นว่า ตัวอักษรที่เพิ่งดึงมานี้อยู่ใน `upperCase` หรือไม่ ถ้าค้นพบ `indexOf` จะคืนดัชนีตัวที่ค้นพบใน `upperCase` ให้บวกตำแหน่งนั้นไปอีก 13 (โดยให้วนกลับถ้าเกินด้วยการ `%` ด้วยความยาวของ `upperCase`) แล้วนำผลไปต่อกับสตริงผลลัพธ์ในบรรทัดที่ 17 ในกรณีที่ไม่พบใน `upperCase` ก็ไปค้นใน `lowerCase` และทำในทำนองเดียวกัน (บรรทัดที่ 19 ถึง 22) และถ้าไม่เป็นทั้งตัวเล็กและใหญ่ ก็ไม่ต้องแปลง นำไปต่อท้าย `rot13` เลย (บรรทัดที่ 24) หมุนทำจนครบทุกตัวแล้ว ก็ให้หลุดออกจากวงวน เพื่อแสดงผลก่อนเลิกการทำงาน

```
01 import java.util.Scanner;
02 // โปรแกรมเข้าและถอดรหัสด้วยวิธี ROT-13
03 public class Rot13 {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("ข้อความ = ");
07     String text = kb.nextLine();
08     String upperCase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
09     String lowerCase = "abcdefghijklmnopqrstuvwxyz";
10     String rot13 = "";
11     int n = text.length();
12     for (int k=0; k<n; k++) {
13       String c = text.substring(k, k + 1);
14       int j = upperCase.indexOf(c);
15       if (j >= 0) {
16         int i = (j + 13) % upperCase.length();
17         rot13 = rot13 + upperCase.substring(i, i + 1);
18       } else {
19         j = lowerCase.indexOf(c);
20         if (j >= 0) {
21           int i = (j + 13) % lowerCase.length();
22           rot13 = rot13 + lowerCase.substring(i, i + 1);
23         } else {
24           rot13 = rot13 + c;
25         }
26       }
27     }
28     System.out.println("ROT-13 = " + rot13);
29   }
30 }
```

รหัสที่ 5-11 โปรแกรมเข้ารหัส ROT-13

การอ่านเขียนแฟ้มข้อมูล

ข้อมูลที่คอมพิวเตอร์ประมวลผลนั้นมักถูกจัดเก็บไว้ในแฟ้มข้อมูลที่หน่วยความจำสำรอง หัวข้อนี้นำเสนอวิธีการอ่านแฟ้มมาประมวลผล วิธีการเขียนผลลัพธ์เก็บลงในแฟ้ม และตัวอย่างการใช้งาน โดยแฟ้มที่อ่านหรือเขียนนี้จะเป็นแฟ้มที่เก็บสตริงหรือข้อความเท่านั้น (ซึ่งคือแฟ้มที่สามารถใช้โปรแกรม notepad เปิดอ่านหรือแก้ไขได้)

การอ่านแฟ้มข้อมูล

จาวามีคลังคำสั่งในการเปิดแฟ้มและอ่านข้อมูลจากแฟ้มได้หลายวิธี ขอนำเสนอวิธีง่าย ๆ ซึ่งใช้ `Scanner` (ตัวเดียวกับที่อ่านแป้นพิมพ์) อ่านข้อมูลจากแฟ้ม ดูรหัสที่ 5-12 ประกอบ

```
01 import java.util.Scanner;
02 import java.io.File;
03 import java.io.IOException;
04 // โปรแกรมแสดงการใช้ Scanner ในการอ่านข้อความจากแฟ้ม
05 public class ReadMe {
06   public static void main(String[] args) throws IOException {
07     Scanner in = new Scanner(new File("ReadMe.java"));
08     while (in.hasNext()) {
09       String s = in.nextLine();
10       System.out.println(s);
11     }
12   }
13 }
```

แทนสองบรรทัดนี้ด้วย java.io.* ก็ได้

main ขอไม่รับผิดชอบสิ่งผิดปกติเรื่อง I/O แต่ขอโยนให้ระบบจัดการเอง

รหัสที่ 5-12 ตัวอย่างการใช้ `Scanner` อ่านแฟ้มข้อมูล

เริ่มด้วยการสร้างตัวอ่านแฟ้มในบรรทัดที่ 7 เราเคยใช้ `new Scanner(System.in)` โดยส่ง `System.in` ซึ่งคือแป้นพิมพ์ไปสร้างตัวอ่าน คราวนี้เราส่ง `new File(` *ชื่อแฟ้ม* `)` ไปสร้างตัวอ่านข้อมูลจากแฟ้ม (ขอให้จำไปก่อนว่าต้องเขียนแบบนี้) เมื่อได้ตัวแปร `in` เป็นตัวอ่านแฟ้มแล้ว ก็สามารถอ่านข้อมูลจากแฟ้มได้เหมือนกับที่อ่านจากแป้นพิมพ์ ถ้าต้องการอ่าน `double` หนึ่งตัวจากแฟ้มก็ใช้ `in.nextDouble()` ถ้าต้องการ `int` ก็ใช้ `in.nextInt()` แต่ที่เราจะใช้กันคือ `in.nextLine()` ซึ่งอ่านหนึ่งบรรทัดจากแฟ้ม อย่างไรก็ตาม ก่อนจะอ่านต้องมั่นใจว่ามีข้อมูลให้อ่าน (เพราะอาจจะอ่านจนหมดแฟ้มไปแล้วก็ได้) ดังนั้น ควรทดสอบก่อนอ่านด้วยคำสั่ง `in.hasNext()` ซึ่งจะคืนจริง ถ้ายังมีข้อมูลเหลือ แต่ถ้าอ่านแฟ้มจนหมดแล้ว จะคืนเท็จ รหัสที่ 5-12 จึงใช้วงวน `while` วนอ่านข้อมูลในแฟ้มมาแสดงทางจอภาพ (บรรทัดที่ 9 และ 10) โดยจะวนไปเรื่อย ๆ ตราบเท่าที่ยังมีข้อมูลในแฟ้ม ซึ่งคือคำสั่ง `while(in.hasNext())` นั่นเอง โปรแกรมในรหัสที่ 5-12 คือคลาส `ReadMe` ทำการอ่านแฟ้ม `ReadMe.java` ซึ่งคือรหัสต้นฉบับของโปรแกรมตัวเองออกมาแสดงทางจอภาพ

หากผู้อ่านสังเกตที่หัวเมท็อด `main` จะพบข้อความ `throws IOException` เนื่องจากในเมท็อด `main` มีการใช้แฟ้มข้อมูล คำถามคือหากมีปัญหาในการใช้แฟ้ม เช่น ไม่มีแฟ้มให้เปิดตามชื่อที่ระบุ หรือฮาร์ดดิกส์บริเวณที่เก็บแฟ้มมีปัญหา หรือผู้ใช้ดึง flash drive ที่เก็บแฟ้มออกพอดีตอนที่กำลังจะอ่าน !!! เหตุการณ์เหล่านี้ถือว่าเป็นสิ่งผิดปกติที่มิได้มาจากการเขียนโปรแกรมผิด แต่ถ้าเกิดขึ้น จะให้โปรแกรมทำอย่างไร นักเขียนโปรแกรมอาจให้โปรแกรมจัดการสิ่งผิดปกติดังกล่าว (วิธีจัดการจะกล่าวในบทหลังๆ) หรืออาจละเลยไม่ขอรับผิดชอบ ให้ระบบจาวาฟ้องผู้ใช้เอง การเขียนวลี `throws IOException` ที่หัวเมท็อด เป็นการบอกระบบว่า `main` ไม่ขอจัดการสิ่งผิดปกติแบบ I/O (input/output) แต่ขอ "โยน" สิ่งผิดปกติดังกล่าวให้ระบบจัดการ อนึ่งโปรแกรมนี้มีการใช้คลาส `File` และคลาส `IOException` ซึ่งมีชื่อเต็มว่า `java.io.File` และ `java.io.IOException` จึงต้องแจ้งให้ตัวแปลโปรแกรมรู้ก่อนที่บรรทัดที่ 2 และ 3 ด้วย

วรรณยุกต์ใดใช้มากสุด

อยากทราบว่าวรรณยุกต์ เอก โท ตรี จัตวา ตัวใดถูกใช้มากสุด เดา ๆ เอาก็น่าจะเป็นไม้เอกหรือไม่ก็ไม้โท ตัวใดตัวหนึ่ง อย่างไรก็ตามเราจะมานับกัน แล้วจะนับจากแหล่งข้อมูลใด ขอใช้แฟ้ม riwords.txt [1] ภายในมีคำศัพท์จำนวน 32,896 คำของพจนานุกรมราชบัณฑิตยสถานฉบับปี พ.ศ. 2525 เริ่มด้วยการเปิดแฟ้ม riwords.txt จากนั้นอ่านคำศัพท์มาทีละคำในรูปของสตริง แล้วก็ใช้ `indexOf` ค้นไม้เอก ไม้โท ไม้ตรี ไม้จัตวา พบตัวใด ก็เพิ่มตัวนับที่เตรียมไว้นับจำนวนวรรณยุกต์ของตัวนั้น ทำจนครบทุกคำ ก็สรุปผล ก่อนอื่นเราต้องรู้รายละเอียดของแฟ้ม riwords.txt ว่ามีลักษณะอย่างไร รูปที่ 5-6 แสดงข้อมูลคร่าว ๆ ของแฟ้มนี้ สังเกตได้ว่าบรรทัดแรกๆ เป็นหมายเหตุอธิบายเนื้อหาของแฟ้ม บรรทัดเหล่านี้ขึ้นต้นด้วยเครื่องหมาย # ส่วนบรรทัดที่เป็นคำศัพท์จะเก็บหนึ่งบรรทัดหนึ่งคำ หรือหนึ่งวลี (และไม่ได้ขึ้นต้นด้วยเครื่องหมาย #) เพียงเท่านี้ก็สามารถเขียนโปรแกรมนับจำนวนวรรณยุกต์แต่ละตัวได้ดังรหัสที่ 5-13

```
#**************************************************************************
#* Copyright(C)2003, National Electronics and Computer Technology Center
#* and others.  All Rights Reserved.
#...
ก
ก็
...
ไฮฮี
```

บรรทัดใดไม่ใช่คำศัพท์จะขึ้นต้นด้วยเครื่องหมาย #

มีคำศัพท์มากกว่า 3 หมื่นคำ หนึ่งคำหนึ่งบรรทัด

รูปที่ 5-6 ตัวอย่างข้อมูลในแฟ้ม `riwords.txt`

[1] แฟ้มนี้อยู่ที่สารบบ c:/java101/riwords.txt (หลังการติดตั้งแผ่น CD หลังหนังสือ) หรือผู้อ่านจะใช้ Internet Explorer ไปที่ http://source.icu-project.org/repos/icu/icu/trunk/source/test/testdata/riwords.txt เพื่อดูข้อมูลในแฟ้มนี้ และใช้เมนู File -> Save As เพื่อบันทึกแฟ้มนี้ แต่อย่าลืมเลือก Encoding (ช่องล่างสุด) ให้เป็นแบบ Thai(Windows) ด้วย (ri ย่อมาจาก The Royal Institute – ราชบัณฑิตยสถาน)

```
01 import java.util.Scanner;
02 import java.io.File;
03 import java.io.IOException;
04 // โปรแกรมนับจำนวนวรรณยุกต์ในพจนานุกรม
05 public class ToneCount {
06   public static void main(String[] args) throws IOException {
07     Scanner in = new Scanner(new File("c:/java101/riwords.txt"));
08     int c1 = 0, c2 = 0, c3 = 0, c4 = 0;
09     String t1 = "";  // ไม้เอก
10     String t2 = "";  // ไม้โท
11     String t3 = "";  // ไม้ตรี
12     String t4 = "";  // ไม้จัตวา
13     while (in.hasNext()) {
14       String line = in.nextLine();
15       line = line.trim();
16       if ( line.length()>0 && !"#".equals(line.substring(0,1)) ) {
17         int k = -1;
18         while (true) {  // นับจำนวนไม้เอกใน line
19           k = line.indexOf(t1, k+1);
20           if (k < 0) break;
21           c1++;
22         }
23         k = -1;
24         while (true) {  // นับจำนวนไม้โทใน line
25           k = line.indexOf(t2, k+1);
26           if (k < 0) break;
27           c2++;
28         }
29         k = -1;
30         while (true) {  // นับจำนวนไม้ตรีใน line
31           k = line.indexOf(t3, k+1);
32           if (k < 0) break;
33           c3++;
34         }
35         k = -1;
36         while (true) {  // นับจำนวนไม้จัตวาใน line
37           k = line.indexOf(t4, k+1);
38           if (k < 0) break;
39           c4++;
40         }
41       }
42     }
43     System.out.println("ไม้เอกมี\t " + c1 + " ตัว");
44     System.out.println("ไม้โทมี\t " + c2 + " ตัว");
45     System.out.println("ไม้ตรีมี\t " + c3 + " ตัว");
46     System.out.println("ไม้จัตวามี\t " + c4 + " ตัว");
47   }
48 }
```

นับวรรณยุกต์อีกแบบ โดยค่อย ๆ เปรียบเทียบทีละตัวในสตริงก็ได้ ดังแสดงข้างล่างนี้

```
for (int k=0; k<line.length( ); k++)
  if (line.substring(k,k+1).equals(t1)) c1++;
```

รหัสที่ 5-13 โปรแกรมนับจำนวนวรรณยุกต์ในพจนานุกรม

โปรแกรมนี้ยาวกว่าโปรแกรมอื่น ๆ ที่เคยเขียนมา เริ่มด้วยการสร้างตัวอ่านในบรรทัดที่ 7 เตรียมตัวแปร c1, c2, c3, และ c4 ไว้เก็บจำนวนไม้เอก ไม้โท ไม้ตรี และไม้จัตวาที่จะนับตามลำดับ โดยเริ่มต้นตัวแปรทั้งสี่มีค่าเป็นศูนย์ จากนั้นเตรียมสตริงสี่ตัวเก็บของวรรณยุกต์ไม้เอก ไม้โท ไม้ตรี และไม้จัตวาไว้ในตัวแปร t1, t2, t3, และ t4 (บรรทัดที่ 9 ถึง 12) เมื่อเตรียมทุกทุกอย่างพร้อม ก็เข้าวงวน while(in.hasNext()) ซึ่งจะวนอ่านข้อมูลในแฟ้ม ตราบเท่าที่ตัวอ่านแฟ้มยังมีข้อความให้อ่าน โดยสนใจบรรทัดที่ยาวเกิน 0 (line.length()>0) และตัวแรกไม่ใช่ # (!"#".equals(line.substring(0,1)) กลุ่มคำสั่งหลัง if ของบรรทัดที่ 16 ประกอบด้วยวงวน 4 วง แยกกันทำตามลำดับ วงแรกนับจำนวนไม้เอก วงต่อมานับจำนวนไม้โท ไม้ตรี และไม้จัตวาตามลำดับ แต่ละวงใช้ตัวแปร k เป็นตัวกำหนดดัชนีก่อนตำแหน่งที่สนใจ จึงเริ่มด้วย k = -1 แล้วเริ่มค้นที่ k+1 ด้วย indexOf หลังค้นถ้าไม่พบก็หลุดจากวงวน ถ้าพบก็เพิ่มตัวนับ แล้ววนกลับไปค้นต่อ เมื่อใดออกจากวงวนนอกสุดก็รายงานผลในตัวนับวรรณยุกต์ทั้งสี่ (ก่อนเริ่มทำงานอย่าลืมตรวจสอบว่า มีแฟ้ม riwords.txt วางไว้ ณ ตำแหน่งที่เขียนไว้ในโปรแกรม)

หลังจากสั่งโปรแกรมนี้ทำงาน จะได้ผลแสดงทางจอภาพ (ผู้อ่านต้องลองสั่งงานดูถึงจะรู้ว่าจำนวนวรรณยุกต์ที่นับได้เป็นเท่าใด) แต่สิ่งที่น่าสงสัยมากก็คือ จำนวนที่นับได้ถูกต้องหรือไม่ เราจะรู้ได้ก็ต้องลองสร้างแฟ้มเล็กๆ ใส่คำหลาย ๆ คำ หลาย ๆ บรรทัดที่มีวรรณยุกต์ทั้งสี่ตามที่ต่าง ๆ ลองสั่งงานดูว่าได้ผลเทียบกับการนับด้วยตาว่าเหมือนกันหรือไม่ เพื่อสร้างความมั่นใจในโปรแกรมที่เขียนก่อนใช้งานจริง แต่ถ้าหากไม่ต้องการสร้างแฟ้มเพื่อทดสอบ เราสามารถใช้คุณสมบัติของ Scanner ที่สามารถสร้างให้อ่านจากสตริงที่กำหนดได้ เช่น หากแทนคำสั่งในบรรทัดที่ 7 ด้วยคำสั่งข้างล่างนี้

```
Scanner in = new Scanner("#ก่าก้าก๊าก่ำ\n  ก่าก้าก๊าก่ำ\nก้าก๊าก่ำ\nก๊าก่ำ\nก่ำ");
```

แล้วสั่งงานดู โปรแกรมจะทำงานเสมือนอ่าน 5 บรรทัด (ให้สังเกตว่ามี \n ซึ่งแทนรหัสขึ้นบรรทัดใหม่ในสตริง) โดยไม่สนใจบรรทัดแรกเพราะขึ้นต้นด้วย # แต่นับวรรณยุกต์ในบรรทัดที่เหลือ ได้ไม้เอก โท ตรี และจัตวาเป็นจำนวน 1, 2, 3, และ 4 ตามลำดับ (ต้องขอบอกว่า ปกติแล้วเราจะไม่ลบบรรทัดที่ 7 ออก แล้วใส่คำสั่งข้างบนนี้แทน แต่จะใช้วิธีทำให้บรรทัดที่ 7 เดิมเป็นหมายเหตุก่อนโดยเติม // ไว้ข้างหน้าบรรทัด จากนั้นจึงเพิ่มคำสั่งใหม่)

การเขียนแฟ้มข้อมูล

เราใช้ Scanner ในการอ่านจากแฟ้มข้อมูลในลักษณะเดียวกับการอ่านจากแป้นพิมพ์ หัวข้อนี้ขอนำเสนอการใช้คำสั่ง print และ println เพื่อเขียนแฟ้มข้อมูลในลักษณะเดียวกับการแสดงผลทางจอภาพ คือแทนที่จะใช้ print และ println กับ System.out (ซึ่งหมายถึงจอภาพ) เราจะสร้างตัวเขียนแฟ้ม (เรียกว่า PrintStream) ซึ่งสามารถเรียกทั้งสองเมท็อดนี้ได้

เช่นกัน (จะว่าไปแล้ว System.out ก็ถือว่าเป็น PrintStream) ดังแสดงเป็นตัวอย่างในรหัสที่ 5-14 ข้างล่างนี้

```
01 import java.util.Scanner;
02 import java.io.*;
03 // โปรแกรมสร้างแฟ้มเก็บผลการตัดเกรดของนักเรียน
04 public class GradeReport {
05   public static void main(String[] args) throws IOException {
06     Scanner in = new Scanner(new File("c:/java101/scores.txt"));
07     PrintStream out = new PrintStream(
08                             new File("c:/java101/grades.txt"));
09     while (in.hasNext()) {
10       String line = in.nextLine();
11       line = line.trim();
12       int j = line.indexOf(" ", 0);
13       if (j >= 0) {
14         String id = line.substring(0, j);
15         String s = line.substring(j, line.length());
16         double score = Double.parseDouble(s);
17         String grade;
18         if (score >= 80) grade = "A";
19         else if (score >= 70) grade = "B";
20         else if (score >= 60) grade = "C";
21         else if (score >= 50) grade = "D";
22         else grade = "F";
23         out.println(id + "\t " + grade);
24       }
25     }
26     out.close();
27   }
28 }
```

สร้างตัวเขียนลงแฟ้ม

ซ้ายสุดถึงก่อนช่องว่างแรก

ช่องว่างแรกถึงขวาสุดสตริง

แปลงสตริงเป็น double

เขียนลงแฟ้มด้วย println

อย่าลืมปิดแฟ้ม ให้ระบบบันทึกผลให้เรียบร้อย

5130120321	74.0
5130293921	85.5
5130294121	58.0
5130338421	90.2
...	

ตัวอย่างแฟ้ม scores.txt

รหัสที่ 5-14 โปรแกรมสร้างแฟ้มเก็บผลการตัดเกรดของนักเรียน

รหัสที่ 5-14 เป็นโปรแกรมอ่านข้อมูลคะแนนของนักเรียนจากแฟ้ม scores.txt เพื่อสร้างแฟ้ม grades.txt ที่เก็บผลการตัดเกรดของนักเรียนแต่ละคน เริ่มด้วยการสร้างตัวอ่านแฟ้ม in และตัวเขียนแฟ้ม out (บรรทัดที่ 6, 7 และ 8) จากนั้นเข้าวงวนตัดเกรดนักเรียนแต่ละคน แต่ละบรรทัดที่อ่านจาก in (บรรทัดที่ 10) จะเก็บเลขประจำตัวตามด้วยคะแนนที่ได้รับ คั่นด้วยช่องว่าง ดังนั้นจึงค้นดัชนีของช่องว่างใน line ที่อ่านจากแฟ้ม หากพบช่องว่าง (บรรทัดที่ 13) จึงจะตัดเกรด เริ่มด้วยการดึงส่วนที่เป็นเลขประจำตัวโดยใช้ substring (บรรทัดที่ 14) จากนั้นดึงส่วนที่เป็นคะแนนจากตำแหน่งช่องว่างถึงขวาสุดสตริง (บรรทัดที่ 15) แล้วเปลี่ยนสตริงเป็น double ด้วย Double.parseDouble (เคยนำเสนอในบทที่ 2) ตามด้วย if หลายชั้นเพื่อให้เกรด ได้ผลเรียบร้อยก็เขียนผลลงแฟ้มผ่านตัวเขียนแฟ้ม out ด้วย println คล้ายกับการแสดงผลทางจอภาพ พอออกจากวงวนต้องอย่าลืมปิดแฟ้มข้อมูลเพื่อแจ้งให้ระบบบันทึกข้อมูลลงแฟ้มให้เรียบร้อย

โปรแกรมตรวจคำสะกด

หัวข้อนี้นำเสนอโปรแกรมอ่านข้อความภาษาไทยในแฟ้มที่ผู้ใช้กำหนด เพื่อนำมาตรวจสอบว่ามีคำใดในแฟ้มนี้ที่ไม่ปรากฏในพจนานุกรม (อาจเพราะเป็นคำเฉพาะ หรือสะกดผิด) ปัญหาที่ต้องทราบทางแก้ไขคือ จะอ่านข้อมูลจากแฟ้มออกมาเป็นคำ ๆ ได้อย่างไร และเมื่อดึงเป็นคำได้แล้วจะค้นในพจนานุกรมให้รวดเร็วได้อย่างไร ขอสนใจปัญหาที่สองก่อน สมมติว่าในแฟ้มที่สนใจมีข้อความ "**อยุธยาเมืองเก่าของเราแต่ก่อน**" อ่านคำแรกได้ **อยุธยา** นำไปค้นในแฟ้มพจนานุกรมโดยอ่านจากแฟ้มพจนานุกรม (riwords.txt) ทีละคำ (เหมือนกับขั้นตอนของโปรแกรมนับจำนวนวรรณยุกต์) มาเปรียบเทียบ ซึ่งจะต้องไม่พบ เพราะ "**อยุธยา**" เป็นคำเฉพาะ อ่านคำต่อไปจะได้ **เมือง** นำไปค้นเช่นเดิมซึ่งจะต้องพบ เพราะปรากฏในพจนานุกรม กระทำเช่นนี้กับคำทุกคำที่มี ถ้าในแฟ้มที่สนใจมี 1000 คำ ก็ต้องค้นในแฟ้มพจนานุกรม 1000 ครั้ง ซึ่งช้ามาก จะขอปรับปรุงให้ค้นเร็วขึ้น โดยนำคำต่าง ๆ ในพจนานุกรมมาต่อๆ กันให้ยาวๆ (ทุกคำนำหน้าและปิดท้ายด้วย $) เก็บไว้ในสตริงเดียว เป็น $**ก**$**ก็**$**กก**$**ก็ก**$**กกขนาก**$**กกช้าง**$... เมื่อต้องการค้นคำใด ก็ให้นำคำนั้นมาปิดหน้าหลังด้วย $ แล้วใช้ `indexOf` ค้นในสตริงยาวๆ ที่ได้สร้างขึ้นนี้ (ไม่ต้องค้นจากแฟ้ม)

วิธีสร้างสตริงยาวๆ เก็บคำทุกคำที่ได้กล่าวมา ทำได้ง่ายอย่างตรงไปตรงมาด้วยการอ่านคำมาทีละคำจากแฟ้ม riwords.txt นำไปต่อท้ายเรื่อย ๆ ด้วยการบวกสตริง ดังส่วนของโปรแกรมที่แสดงในรหัสที่ 5-15 วิธีนี้ดูแล้วก็ไม่มีปัญหาอะไร แต่ถ้าผู้อ่านลองเขียนให้เป็นโปรแกรมสมบูรณ์แล้วลองสั่งทำงานเฉพาะส่วนการต่อสตริงนี้ จะพบว่า ทำงานช้ามากๆ ช้าจนรอไม่ไหว ช้าจนคิดว่าเขียนโปรแกรมผิด (ผู้อ่านต้องลองสั่งทำงานจริงถึงรู้สึกได้) เนื่องจากลักษณะการต่อสตริงของจาวาเป็นการสร้างสตริงใหม่ซึ่งใช้เวลาแปรตามความยาวสตริง เมื่อมีการต่อสตริงบ่อยๆ ยิ่งต่อยิ่งยาว ยิ่งยาวยิ่งต่อช้า จึงเกิดเหตุการณ์ดังกล่าว

```
Scanner in = new Scanner(new File("c:/java101/riwords.txt"));
String riwords = "$";
while (in.hasNext()) {
  String line = in.nextLine();
  line = line.trim();
  if (line.length() > 0 && !"#".equals(line.substring(0, 1))) {
    riwords = riwords + line + "$";
  }
}
```

นำคำใหม่ต่อท้าย คั่นด้วย $

รหัสที่ 5-15 ส่วนของโปรแกรมอ่านคำศัพท์มาต่อกันเป็นสตริง

ขอเปลี่ยนการสร้างสตริงยาว ๆ มาใช้วิธีการนำคำศัพท์ต่างๆ ที่อ่านมาได้เขียนลงแฟ้มด้วย `print` ต่อกันไปเรื่อยๆ (ปิดหน้าปิดหลังแต่ละคำด้วย $) ลงในแฟ้มเพียงหนึ่งบรรทัด (แต่เป็นบรรทัดที่ยาวมาก) เมื่อ `print` ครบทุกคำ ก็ปิดแฟ้มให้เรียบร้อย แล้วเปิดแฟ้มนี้กลับมาอ่านด้วย `nextLine` เพียงครั้งเดียว ได้ส่วนของโปรแกรมดังแสดงในรหัสที่ 5-16 ที่ทำงานได้เร็วมาก ๆ

```
01 import java.util.Scanner;
02 import java.io.*;
03 import jlab.WordScanner;  // WordScanner เป็นคลาสพิเศษใน JLab
04 // โปรแกรมตรวจคำสะกดผิด
05 public class SpellChecking {
06   public static void main(String[] args) throws IOException {
07     Scanner in = new Scanner(new File("c:/java101/riwords.txt"));
08     PrintStream out = new PrintStream(new File("c:/java101/t.txt"));
09     out.print("$");
10     while (in.hasNext()) {
11       String line = in.nextLine();
12       line = line.trim();
13       if (line.length() > 0 && !"#".equals(line.substring(0, 1))) {
14         out.print(line + "$");
15       }
16     }
17     out.close();
18     Scanner tmp = new Scanner(new File("c:/java101/t.txt"));
19     String riwords = tmp.nextLine();
```

อ่านแต่ละคำในพจนานุกรมมาต่อๆกันเป็นบรรทัดยาว

เปิดแฟ้มแล้วอ่านบรรทัดยาวๆ เข้ามาบรรทัดเดียว

รหัสที่ 5-16 ส่วนของโปรแกรมสร้างสตริงเก็บคำศัพท์ต่างๆ

กลับมาพิจารณาวิธีการอ่านข้อความในแฟ้มให้ได้ทีละคำ ๆ เพื่อค้นในพจนานุกรม เราอาศัยบริการพิเศษของ JLab ผ่านคลาสชื่อเต็มว่า `jlab.WordScanner` เพื่อสร้างตัวอ่านแบบคำที่มีวิธีการใช้คล้ายกับ `Scanner` มีเมท็อด `nextWord` ที่คืนคำถัดไป เขียนโปรแกรมต่อจากรหัสที่ 5-16 ได้ดังรหัสที่ 5-17 เริ่มทำต่อด้วยการอ่านชื่อแฟ้มทางแป้นพิมพ์ เพื่อมาสร้างตัวอ่านจากแฟ้มแบบคำ (บรรทัดที่ 23) แล้วเข้าวงวนดึงคำเพื่อนำไปปิดหน้าปิดหลังด้วย `"$"` ส่งให้ `indexOf` ค้นในสตริง `riwords` (บรรทัดที่ 28) หากได้ผลเป็นลบ แสดงว่าค้นไม่พบ ก็แสดงผลให้ผู้ใช้ทราบ

```
20     Scanner kb = new Scanner(System.in);
21     System.out.print("แฟ้ม = ");
22     String inputFile = kb.nextLine();  // ลอง c:/java101/anthem.txt
23     WordScanner file = new WordScanner(new File(inputFile));
24     System.out.println("คำที่ไม่พบในพจนานุกรมมีดังนี้ : ");
25     while (file.hasNext()) {
26       String word = file.nextWord();
27       word = word.trim();
28       if(riwords.indexOf("$"+word+"$")<0) {
29         System.out.println(word);
30       }
31     }
32   }
33 }
```

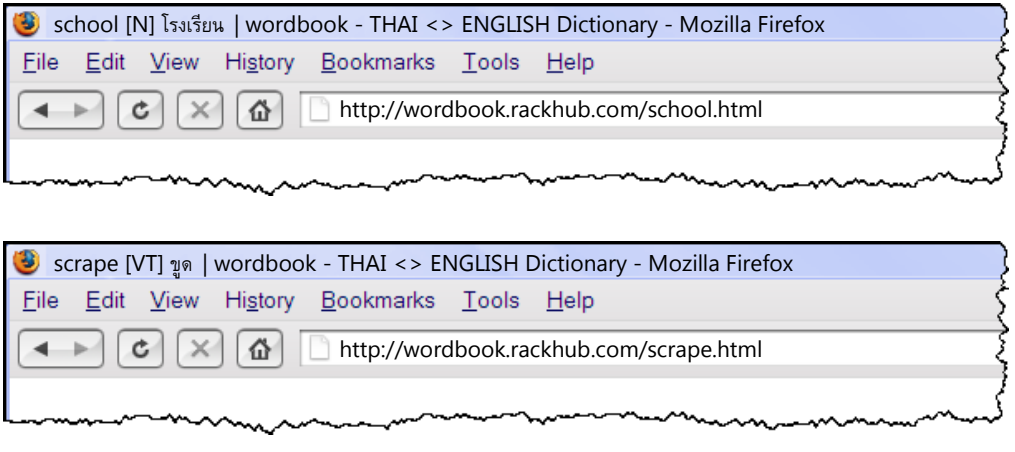
ถ้าค้นไม่พบ ให้แสดงทางจอภาพ

รหัสที่ 5-17 โปรแกรมตรวจคำสะกดผิด (ต่อจากรหัสที่ 5-16)

โปรแกรมแปลคำอังกฤษเป็นไทย

บริการแปลคำอังกฤษเป็นไทย หรือคำไทยเป็นอังกฤษมีอยู่หลายที่ในอินเทอร์เน็ต (ผู้อ่านสามารถค้นวลี "English Thai Dictionary" ด้วยกูเกิลดู) เราจะมาเขียนโปรแกรมรับคำอังกฤษจากผู้ใช้ทางแป้นพิมพ์ จากนั้นเรียกใช้บริการที่ http://wordbook.rackhub.com [2] เพื่อค้นคำแปลเป็นภาษาไทย ได้ผลคืนกลับมา แล้วนำไปแสดงให้ผู้ใช้อีกทอดหนึ่ง คำถามที่ตามมาคือจะเขียนโปรแกรมต่อกับเว็บไซต์ได้อย่างไร จะส่งคำที่ต้องการหาคำแปลได้อย่างไร และผลที่ได้กลับมาเป็นอย่างไร คำแปลที่ได้อยู่ที่ไหน ?

ขอเริ่มที่สองคำถามแรก ถ้าเราต้องการทราบว่า "school" แปลว่าอะไร ก็ให้เปิดบราวเซอร์แล้วไปที่ http://wordbook.rackhub.com/school.html หากต้องการคำแปลของ "scrape" ต้องไปที่ http://wordbook.rackhub.com/scrape.html จะได้ผลดังรูปที่ 5-7 (คำแปลปรากฏที่หัววินโดว์) ดูสองตัวอย่างนี้แล้ว ผู้อ่านเดาได้หรือยังว่า ถ้าต้องการหาคำอื่นจะไปที่ใด ปัญหาคือจะให้โปรแกรมของเราไปที่เว็บเพจที่ต้องการได้อย่างไร ไม่ยาก ใช้ `Scanner` เช่นเคย แทนที่จะส่ง `System.in` เพื่ออ่านจากแป้นพิมพ์ หรือส่ง `new File(...)` เพื่ออ่านจากแฟ้ม คราวนี้ส่ง `new URLStream(url)` เพื่ออ่านจากเว็บแทน [3] ดังตัวอย่างในรหัสที่ 5-18 เป็นการเชื่อมไปยังเว็บเพจคำแปลของคำอังกฤษในตัวแปร `word` เมื่อสร้างตัวอ่านเว็บชื่อ `web` เสร็จ ก็สามารถใช้เมท็อด `web.nextLine()` ของ `Scanner` อ่านออกมาทีละบรรทัดเช่นที่เคยทำมา



รูปที่ 5-7 ตัวอย่างการใช้บริการแปลคำอังกฤษเป็นไทยด้วย wordbook

[2] เหตุผลที่เลือกเว็บไซต์นี้ไม่มีอะไรนอกจากความง่ายในการเขียนโปรแกรมเพื่อส่งคำไป และตีความคำแปลที่ได้กลับมา และต้องขอเน้นว่าเนื่องจากโปรแกรมนี้เรียกใช้เว็บไซต์ภายนอกเครื่องที่สั่งทำงาน ดังนั้นจึงต้องเชื่อมต่อกับอินเทอร์เน็ตก่อนใช้งาน และก็ไม่แน่เสมอไปว่าจะทำงานได้ เพราะเว็บไซต์อาจไม่พร้อมให้บริการก็เป็นได้

[3] `URLStream` เป็นคลาสพิเศษของ JLab มีชื่อเต็มว่า `jlab.URLStream`

```
String site = "http://wordbook.rackhub.com/";
String url = site + word + ".html";
Scanner web = new Scanner(new URLStream(url));
...
```

รหัสที่ 5-18 ส่วนของโปรแกรมอ่านเว็บเพจ

คำถามถัดมาคือข้อมูลที่อ่านได้จากตัวอ่านเว็บที่เราสร้างขึ้นมีรูปแบบเช่นไร รูปที่ 5-8 แสดงบรรทัดต้นๆ ของข้อความในเว็บเพจที่อ่านได้เมื่อไปหาคำแปลของคำว่า “school” ข้อมูลที่ได้มาอยู่ในรูปแบบที่เรียกว่า HTML (HyperText Markup Language) [4] มีป้ายข้อความ (ซึ่งอยู่ภายในเครื่องหมาย < >) กำกับเพื่อบอกความหมายหรือลักษณะการนำเสนอของเนื้อความในเอกสาร

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http ...
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<base href="http://wordbook.rackhub.com/" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>school [N] โรงเรียน | wordbook - THAI &lt;&gt; ENGLISH Diction ...
...
```

รูปที่ 5-8 ตัวอย่างข้อความที่อ่านได้เมื่อต่อไปยังเว็บเพจของ wordbook

และก็มาถึงคำถามสุดท้ายคือ แล้วคำแปลอยู่ที่ใด ? ถ้ากลับไปดูในรูปที่ 5-7 พบว่า โรงเรียน ซึ่งเป็นคำแปลของ school ปรากฏที่ส่วนหัวของวินโดว์ของบราวเซอร์ [5] ข้อความที่ปรากฏที่ส่วนหัวนี้คือข้อความเดียวกันกับที่ปรากฏหลังคำว่า `<title>` ที่อ่านได้จากตัวอ่านเว็บ (ดูบรรทัดที่ 6 ในรูปที่ 5-8) และจะเป็นเช่นนี้กับคำแปลทุกคำที่อ่านได้จากเว็บไซต์นี้ ดังนั้น ภาระส่วนใหญ่ของการเขียนโปรแกรมนี้ก็คือ การหาคำแปลที่ปรากฏหลัง `<title>` นี่เอง [6]

รหัสที่ 5-19 แสดงโปรแกรมสมบูรณ์เพื่อแปลคำอังกฤษเป็นคำไทย เริ่มด้วยการรับคำจากผู้ใช้ จากนั้นใช้ `trim` เพื่อขจัดช่องว่างซ้ายและขวา ตามด้วยการเปลี่ยนเป็นตัวเล็ก เก็บใส่ตัวแปรเดิม ในบรรทัดที่ 9 ใช้ `eng.trim()` ซึ่งคืนสตริงกลับมาตามด้วย `.toLowerCase()` ต่อได้เลย บรรทัดที่ 10 สร้างสตริงที่เก็บตำแหน่งของเว็บเพจที่มีคำแปล แล้วนำไปสร้างตัวอ่านเว็บ ขอให้สังเกตว่าที่ท้ายคำสั่ง เราส่งสตริง `"UTF-8"` ไปด้วย ตรงนี้หมายความว่า เว็บเพจที่จะอ่านนั้นถูกเข้ารหัสแบบที่เรียกว่า UTF-8 (การเข้ารหัสข้อความในเอกสารที่หลายแบบ ให้สังเกตที่ปลายบรรทัดที่ 5 ของรูปที่ 5-8 เขียนไว้ว่า `charset=utf-8` เป็นการระบุว่าเอกสารนี้เข้ารหัสข้อความ

---

[4] หลังจากใช้บราวเซอร์ไปที่เว็บเพจแล้ว หากต้องการดูข้อมูล HTML ที่บรรยายหน้านั้น ให้คลิกเมาส์ปุ่มขวาที่หน้าเอกสาร แล้วเลือกเมนู View Source

[5] รูปที่ 5-7 ไม่ได้แสดงบราวเซอร์เต็มจอภาพ แต่ถ้าผู้อ่านลองต่อเว็บไปดูที่หน้านี้ จะพบว่า มีรายละเอียดคำแปลมากกว่าหนึ่งความหมายในหน้านี้

[6] เนื่องจากเว็บเพจถูกออกแบบมาเพื่อนำเสนอเนื้อหาให้คนดู แต่นี่เราต้องการเขียนโปรแกรมให้ “เลือกดู” ส่วนที่สนใจในเว็บเพจ ซึ่งต้องอาศัยการค้นหาข้อมูลในเว็บเพจ เรียกกลวิธีแบบนี้ว่า *การขูดเว็บ* (web scraping)

แบบ UTF-8 ทำให้เราต้องส่งให้ตัวอ่านเว็บรู้ด้วยว่าจะถอดรหัสข้อความแบบใด) ก่อนเข้าวงวนค้นคำแปล เราเตรียมตัวแปร `thai` ให้ค่าเริ่มต้นเป็น `""` ไว้เก็บคำแปล โดยจะค้นไปเรื่อยๆ ในวงวนตราบเท่าที่ยังไม่พบบรรทัดที่มีคำแปล (`thai` ยังเป็น `""`) และตัวอ่านเว็บยังอ่านไม่หมด (`web.hasNext()` เป็นจริง) ภายในวงวนอ่านบรรทัดใหม่เข้ามา แล้วใช้ `indexOf` ค้นสตริง `<title>` ถ้าพบ ก็ค้นหาเครื่องหมาย `]` ต่อ (บรรทัดที่ 19) โดยเริ่มค้นหลังตำแหน่งของเครื่องหมาย `>` ของ `<title>` (ซึ่งคือ 7 ตัวหลังตำแหน่งที่พบ `<title>`) ถ้าพบ `]` บรรทัดที่ 21 จะค้นเครื่องหมาย `|` ต่อ ถ้าพบอีกแสดงว่าคำแปลคือสตริงย่อยที่อยู่หลัง `]` ที่พบ ไปจนถึงก่อน `|` ที่พบ (บรรทัดที่ 22) หากค้นไม่พบ ก็จะวนกลับไปอ่านบรรทัดถัดไป โดยที่ `thai` ยังเป็น `""` ดังนั้นถ้าหลุดจากวงวนโดยมี `thai` ไม่ใช่ `""` แสดงว่าเก็บคำแปล ก็ให้แสดงคำแปลนั้นทางจอภาพ (บรรทัดที่ 27) ไม่เช่นนั้นให้แจ้งให้ผู้ใช้ทราบว่าค้นไม่พบ

```
01 import java.util.Scanner;
02 import jlab.URLStream;
03 // โปรแกรมแปลคำอังกฤษเป็นไทย
04 public class English2Thai {
05   public static void main(String[] args) {
06     Scanner kb = new Scanner(System.in);
07     System.out.print("คำ = ");
08     String eng = kb.nextLine();
09     eng = eng.trim().toLowerCase();
10     String url = "http://wordbook.rackhub.com/" + eng + ".html";
11     Scanner web = new Scanner(new URLStream(url), "UTF-8");
12     String thai = "";
13     while (thai.equals("") && web.hasNext()) {
14       String t = web.nextLine();
15       //ต้องการหา <title>school [N] โรงเรียน | wordbook...
16       t = t.toLowerCase();
17       int j = t.indexOf("<title>", 0);
18       if (j >= 0) {
19         j = t.indexOf("]", j+7);
20         if (j >= 0) {
21           int k = t.indexOf("|", j+1);
22           if (k >= 0) thai = t.substring(j+1, k);
23         }
24       }
25     }
26     if (!thai.equals("")) {
27       System.out.println("แปลว่า " + thai.trim());
28     } else {
29       System.out.println("ไม่พบคำแปล");
30     }
31   }
32 }
```

รหัสที่ 5-19 โปรแกรมแปลคำอังกฤษเป็นไทย

การเปรียบเทียบสตริง

ตามที่ได้เคยนำเสนอมาแล้วว่า ตัวดำเนินการสัมพันธ์ < <= > >= ที่ใช้เปรียบเทียบความน้อยกว่ามากกว่านั้น ใช้ได้กับจำนวน แต่ใช้กับสตริงไม่ได้ และถึงแม้ว่าการใช้ == กับสตริงจะไม่ผิดหลักไวยากรณ์ของภาษา แต่ก็ไม่แนะนำให้ใช้ เพราะบางครั้งน่าจะเท่าแต่กลับไม่เท่า เช่น ให้ `s = "Abc"` และ `t = "abc"` การเปรียบเทียบ `s.toLowerCase()==t` จะได้ `false` ทั้งๆ ที่น่าจะได้ `true` (ทั้งนี้มีสาเหตุมาจากการสร้างสตริงของจาวา ซึ่งขอไม่กล่าวถึง) จึงแนะนำเสมอว่าให้ใช้ `equals` แทน (จากตัวอย่างข้างต้น จึงควรใช้ `s.toLowerCase().equals(t)`)

การเปรียบเทียบความน้อยกว่ามากกว่าของสตริง สามารถทำได้ด้วยเมท็อดพิเศษของสตริงชื่อ `compareTo` ถ้าต้องการเปรียบเทียบสตริง `s` กับ `t` คำสั่ง `s.compareTo(t)` จะให้ผลคืนกลับมาเป็นจำนวนเต็ม ถ้ามีค่าน้อยกว่า 0 แสดงว่า `s` มีค่าน้อยกว่า `t` , ถ้ามีค่ามากกว่า 0 แสดงว่า `s` มีค่ามากกว่า `t`, และถ้ามีค่าเป็น 0 แสดงว่า `s` เหมือนกับ `t` ดังตัวอย่างการใช้งานในตารางที่ 5-2 `compareTo` อาศัยการเปรียบเทียบทีละอักขระในสตริงเริ่มจากดัชนี 0 ไปเรื่อย ๆ โดยความน้อยกว่ามากกว่าของอักขระเป็นไปตามลำดับในพจนานุกรม ตัวอย่างเช่น

- `"ant"` มากกว่า `"ann"` การเปรียบเทียบสองตัวแรกคือ `a` กับ `n` นั้นเท่ากัน แต่ตัวถัดมา `t` นั้นอยู่หลัง `n` ในลำดับตัวอักษร จึงถือว่าตัว `t` มากกว่า `n`
- `"mod"` มากกว่า `"four"` เพราะดูตัวแรกก็พบว่า `m` มากกว่า `f` ดังนั้น สตริงยาวกว่าไม่ได้หมายความว่ามากกว่า
- `"แขน"` มากกว่า `"คอ"` เพราะดูตัวแรก สระมากกว่าพยัญชนะ ถ้าดูตามลำดับในพจนานุกรม `"แขน"` อยู่หมวด ข อยู่ก่อน `"คอ"` ดังนั้น `compareTo` ไม่มีความรู้เรื่องคำไทย จึงต้องระวังการใช้ `compareTo` กับสตริงไทย (อ่านหัวข้อเพิ่มเติม ถ้าต้องการเปรียบเทียบสตริงไทยตามลำดับแบบพจนานุกรม)
- `"java"` มากกว่า `"Ruby"` เพราะตัวอังกฤษตัวเล็กมากกว่าตัวใหญ่ !!! ลำดับตัวอักษรภาษาอังกฤษคือ `A < B < ... < Z < a < b < ... < z` ถ้าต้องการเปรียบเทียบโดยไม่สนใจเรื่องเล็กใหญ่ให้ใช้ `compareToIgnoreCase`

ตารางที่ 5-2 ตัวอย่างการใช้งานเมท็อด compareTo ของสตริง

การเรียกใช้	ความหมาย
`if ( s.compareTo(t) < 0 )`	ถ้า s น้อยกว่า t
`if ( s.compareTo(t) == 0 )`	ถ้า s เท่ากับ t
`if ( s.compareTo(t) > 0 )`	ถ้า s มากกว่า t

เพิ่มเติม

มาตรฐานยูนิโคด

สตริงแปลว่าสายอักขระ ประกอบด้วยอักขระ (character) เรียงต่อเนื่องกันไป ภายในสตริงจึงประกอบด้วยอักขระตั้งแต่ศูนย์ตัวเป็นต้นไป จาวารองรับการประมวลผลข้อความที่เป็นนานาชาติ หลายภาษาที่ใช้กันในโลก จาวาจึงเก็บอักขระแต่ละตัวตามมาตรฐานการเข้ารหัสที่เรียกว่า *ยูนิโคด* (Unicode) แต่ละตัวใช้เนื้อที่ 2 ไบต์ (จึงเป็นรหัสที่แทนอักขระได้ทั้งหมด 2^{16} = 65,536 ตัว) ตารางที่ 5-3 และตารางที่ 5-4 แสดงยูนิโคดของอักขระอังกฤษและไทยตามลำดับ เช่น ตัว Z แทนด้วยรหัส $(005A)_{16}$, ฐ แทนด้วยรหัส $(0E10)_{16}$ ซึ่งเขียนในระบบเลขฐานสิบหก (เลขฐานสิบหกประกอบด้วยเลขโดด 0, 1, …, 9, A, B, C, D, E, F ซึ่งแทนค่า 0 ถึง 15 ในฐานสิบตามลำดับ หนึ่งหลักในฐาน 16 ใช้เนื้อที่ 4 บิต ดังนั้น รหัส $(005A)_{16}$ มี 4 หลัก ใช้เนื้อที่ 4×4 = 16 บิต = 2 ไบต์)

ตารางที่ 5-3 ยูนิโคดของอักขระอังกฤษ

	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	002A	002B	002C	002D	002E	002F
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0030	0031	0032	0033	0034	0035	0036	0037	0038	0039	003A	003B	003C	003D	003E	003F
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0040	0041	0042	0043	0044	0045	0046	0047	0048	0049	004A	004B	004C	004D	004E	004F
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	005A	005B	005C	005D	005E	005F
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0060	0061	0062	0063	0064	0065	0066	0067	0068	0069	006A	006B	006C	006D	006E	006F
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
0070	0071	0072	0073	0074	0075	0076	0077	0078	0079	007A	007B	007C	007D	007E	007F

ตารางที่ 5-4 ยูนิโคดของอักขระไทย

	ก	ข	ฃ	ค	ฅ	ฆ	ง	จ	ฉ	ช	ซ	ฌ	ญ	ฎ	ฏ
0E00	0E01	0E02	0E03	0E04	0E05	0E06	0E07	0E08	0E09	0E0A	0E0B	0E0C	0E0D	0E0E	0E0F
ฐ	ฑ	ฒ	ณ	ด	ต	ถ	ท	ธ	น	บ	ป	ผ	ฝ	พ	ฟ
0E10	0E11	0E12	0E13	0E14	0E15	0E16	0E17	0E18	0E19	0E1A	0E1B	0E1C	0E1D	0E1E	0E1F
ภ	ม	ย	ร	ฤ	ล	ฦ	ว	ศ	ษ	ส	ห	ฬ	อ	ฮ	ฯ
0E20	0E21	0E22	0E23	0E24	0E25	0E26	0E27	0E28	0E29	0E2A	0E2B	0E2C	0E2D	0E2E	0E2F
ะ	ั	า	ำ	ิ	ี	ึ	ื	ุ	ู	ฺ					฿
0E30	0E31	0E32	0E33	0E34	0E35	0E36	0E37	0E38	0E39	0E3A	0E3B	0E3C	0E3D	0E3E	0E3F
เ	แ	โ	ใ	ไ	ๅ	ๆ	็	่	้	๊	๋	์	ํ	๎	๏
0E40	0E41	0E42	0E43	0E44	0E45	0E46	0E47	0E48	0E49	0E4A	0E4B	0E4C	0E4D	0E4E	0E4F
๐	๑	๒	๓	๔	๕	๖	๗	๘	๙	๚	๛				
0E50	0E51	0E52	0E53	0E54	0E55	0E56	0E57	0E58	0E59	0E5A	0E5B	0E5C	0E5D	0E5E	0E5F

ให้สังเกตว่า รหัสที่แทนอักขระต่าง ๆ นั้นมีค่าเรียงตามลำดับอักษรในพจนานุกรมด้วย หากมองอักขระเป็นจำนวนตามค่าของรหัสแล้ว ผลการเปรียบเทียบรหัสจะสะท้อนลำดับของอักขระ เช่น ก แทนด้วยรหัส $(0E01)_{16} = (3585)_{10}$ ส่วน ฮ แทนด้วยรหัส $(0E2E)_{16} = (3630)_{10}$ ถ้าเปรียบเทียบ ก กับ ฮ ด้วยการเปรียบเทียบรหัส จะได้ 3585 < 3630 แสดงว่า ก อยู่หน้า ฮ ในลำดับตัวอักษร นอกจากนี้หากเรานำรหัสมาลบกัน เช่น 3630 – 3585 = 45 ก็สรุปได้ว่า ฮ ห่างจาก ก 45 ตัวในลำดับตัวอักษร (ลำดับนี้รวมตัว ฤ และ ฦ ด้วย)

เราสามารถเขียนอักขระในสตริง โดยใช้รหัสยูนิโคดของอักขระนั้นได้ ด้วยการใช้อักขระหลีก `\u` ตามด้วยรหัสเลขฐานสิบหก 4 หลักของอักขระนั้น เช่น `"\u0e01"` คือ "ก" ผู้อ่านอาจสงสัยว่าจะเขียนแบบนี้ไปทำไม เราก็กดปุ่มทางแป้นพิมพ์ได้อยู่แล้ว ก็ต้องบอกว่าไม่แน่ เพราะอักขระหลายตัวไม่มีให้กดบนแป้นพิมพ์ เช่น ◉ ฯ ๛ เป็นต้น จึงต้องใช้อักขระหลีก `\u` ตามด้วยรหัสยูนิโคด ดังตัวอย่างคำสั่งข้างล่างนี้ ใช้แสดงตัวฟองมัน ◉ อังคั่นคู่ ฯ และ โคมูตร ๛

```
System.out.println("\u0e4f\u0e5a\u0e5b");
```

ประเภทข้อมูล char

จาวามีประเภทข้อมูลพื้นฐานอีกแบบชื่อ `char` มีไว้เก็บอักขระหนึ่งตัว การเขียนค่าคงตัวของอักขระกระทำโดยเขียนอักขระหนึ่งตัวไว้ภายในเครื่องหมายอัญประกาศเดี่ยว ' เช่น `char c = 'ก';` ตัวแปรแบบ `char` ต้องเก็บอักขระหนึ่งตัวพอดี ดังนั้น จะเขียน `char c = 'AB';` หรือ `char c = '';` ไม่ได้ (กรณีหลังนี้เป็นการเขียน ' ติดกันสองตัว) การประมวลผลอักขระทำได้ง่ายกว่าการประมวลสตริง เพราะจาวามองอักขระเป็นจำนวน (มองเป็นจำนวนบวกเท่านั้น) จึงสามารถใช้ตัวดำเนินการสัมพันธ์ `==` `!=` `<` `>` `<=` `>=` ในการเปรียบเทียบอักขระได้[7] นอกจากนี้ยังสามารถใช้ตัวดำเนินการคำนวณ `+` `-` `*` `/` `%` กับอักขระได้ด้วย ผู้อ่านอาจสงสัยว่าเราจะนำอักขระมาเปรียบเทียบ หรือคำนวณทำไม ไม่น่ามีประโยชน์อะไร แต่นักเขียนโปรแกรมสามารถใช้การประมวลผลอักขระให้เป็นประโยชน์ได้ เช่น

- `(char)('ก' + 1)` จะได้ 'ข' เพราะถัดจาก ก ไปหนึ่งตัวคือ ข ให้สังเกตด้วยว่าการเขียน `'ก' + 1` จะได้ผลเป็น `int` เพราะ `char` จะบวกกับ `int` ได้ ต้องมอง `char` เป็นจำนวน ดังนั้น จึงเปลี่ยน `char` เป็น `int` โดยนำรหัสของอักขระนั้นมาใช้แทน เมื่อบวกกันแล้วได้ผลเป็น `int` จึงต้องเปลี่ยนกลับเป็น `char` ด้วยการใช้ `(char)` นำหน้าจำนวนเต็มที่ต้องการมองกลับเป็นอักขระ
- `(char)('A' + 32)` จะได้ 'a' เพราะรหัสของตัวใหญ่จะน้อยกว่าตัวเล็กอยู่ $(0020)_{16} = (32)_{10}$ และในทำนองเดียวกัน `(char)('k' - 32)` ก็ย่อมได้ 'K'

[7] ด้วยเหตุที่จาวามอง `char` เป็นจำนวนเต็ม จึงสามารถใช้ข้อมูลแบบ `char` ในคำสั่ง `switch-case` ได้ด้วย

- `'9' - '0'` ได้ 9 เพราะรหัสยูนิโคดของเลขโดดเรียงติดกันจาก `'0'` ถึง `'9'` ดังนั้น ถ้า `c` เก็บอักขระที่แทนเลขโดด เราสามารถแปลง `c` มาเป็นจำนวนเต็มของเลขโดดนั้นด้วยนิพจน์ `c - '0'`
- `'0'<=c && c<='9'` ทดสอบว่า `c` เป็นอักขระที่แทนเลขโดดหรือไม่
- `'a'<=c && c<='z'` ทดสอบว่า `c` เป็นอักขระอังกฤษตัวเล็กหรือไม่ (ดูตารางที่ 5-3)
- `'ก'<=c && c<='๙'+2` ทดสอบว่า `c` เป็นอักขระไทยหรือไม่ (ดูตารางที่ 5-4)

สตริงเก็บอักขระไว้ภายใน สตริงจึงมีเมท็อดที่เกี่ยวกับ `char` เช่น `s.charAt(k)` คืนอักขระตัวที่ดัชนี `k` ของสตริง `s`, เมท็อด `s.indexOf(c,i)` เหมือน `indexOf` ที่ได้นำเสนอมา แต่รับอักขระ `c` แทน, หรือเมท็อด `s.replace(c1, c2)` จะคืนสตริงใหม่เหมือน `s` แต่เปลี่ยนอักขระภายในทุกตัวที่เหมือน `c1` ด้วย `c2` โปรแกรมต่าง ๆ ที่เราได้เขียนมาสามารถเปลี่ยนมาใช้การประมวลผลด้วย `char` ก็ได้ ถ้าย้อนไปดูรหัสที่ 5-9 มีการเปลี่ยนอักขระของเลขโดดให้เป็นจำนวนเต็ม ด้วยคำสั่ง `v = Integer.parseInt(d.substring(k, k+1));` ก็สามารถใช้คำสั่ง `v = d.charAt(k)-'0';` แทนได้ หรือวงวนหลักของรหัสที่ 5-11 เพื่อเข้ารหัส ROT-13 สามารถแทนได้ด้วยวงวนที่สั้น (และเร็วกว่า) ในรหัสที่ 5-20 โดยไม่ต้องใช้สตริง `upperCase` และ `lowerCase` ของรหัสที่ 5-11

```
for (int k=0; k<n; k++) {      // แทนวงวนหลักในรหัสที่ 5-11
  char c = text.charAt(k);
  if      ('a' <= c && c <= 'm') c = (char)(c + 13);
  else if ('n' <= c && c <= 'z') c = (char)(c - 13);
  else if ('A' <= c && c <= 'M') c = (char)(c + 13);
  else if ('N' <= c && c <= 'Z') c = (char)(c - 13);
  rot13 = rot13 + c;  // ใช้บวกต่อสตริงกับ char ได้
}
```

รหัสที่ 5-20 ส่วนของโปรแกรมเข้ารหัสแบบ ROT-13 ที่ใช้ประมวลผลอักขระ

การเปรียบเทียบสตริงไทย

`String` ไม่มีความรู้ภาษาไทย คำสั่ง `"เขย".compareTo("สะใภ้")` จึงได้ผลเป็นจำนวนบวก เพราะรหัสของ เ ซึ่งคือ $(0E40)_{16}$ มีค่ามากกว่าของ ส ซึ่งคือ $(0E2A)_{16}$ แต่ผู้อ่านก็คงรู้เหมือนผู้เขียนว่า เขย เป็นคำที่มาก่อน สะใภ้ ในพจนานุกรม จึงใช้ `compareTo` เปรียบเทียบไม่ได้ อย่างไรก็ตาม จาวาต้องการเป็นระบบที่รองรับหลากหลายภาษาในโลก จึงเพิ่มบริการพิเศษที่ใส่ความรู้ของสารพัดภาษาลงไปในคลังคำสั่งมาตรฐานของระบบ เช่น คลาสมาตรฐานที่มีชื่อเต็มว่า `java.text.Collator` (collate แปลว่าเรียง) มีความสามารถในการเปรียบเทียบคำไทยได้ ดังตัวอย่างในรหัสที่ 5-21 บรรทัดที่ 6 สร้างตัวเรียงด้วยเมท็อด `Collator.getInstance` โดยสิ่งที่เขียนในวงเล็บระบุภาษาที่จะใช้เรียง ในรหัสที่ 5-21 คือ `new Locale("th")` แทน

ภาษาไทยนั่นเอง เพียงเท่านี้ก็สามารถเปรียบเทียบได้ถูกต้องตามภาษาที่ตั้งไว้ด้วยการเรียกเมท็อด compare ของตัวเรียง บรรทัดที่ 7 แสดงการใช้ตัวเรียง thai เปรียบเทียบ "เขย" กับ "สะใภ้" ได้ผลเป็นจำนวนเต็มลบ (แสดงว่า "เขย" มาก่อน "สะใภ้" ในลำดับของพจนานุกรม ซึ่งถูกต้อง) ในขณะที่บรรทัดที่ 8 ได้จำนวนเต็มบวก (ซึ่งผิด)

```
01 import java.text.Collator;
02 import java.util.Locale;
03 // โปรแกรมแสดงตัวอย่างการเปรียบเทียบสตริงไทย
04 public class ThaiString {
05   public static void main(String[] args) {
06     Collator thai = Collator.getInstance(new Locale("th"));
07     System.out.println( thai.compare("เขย", "สะใภ้") );   // ได้จำนวนลบ
08     System.out.println( "เขย".compareTo("สะใภ้") );       // ได้จำนวนบวก
09   }
10 }
```

รหัสที่ 5-21 โปรแกรมแสดงตัวอย่างการเปรียบเทียบสตริงไทย

แบบฝึกหัด

1. โปรแกรมข้างล่างนี้ยังผิดอยู่ จงแก้ไขให้ถูกต้อง

```
public class Java {
  public static void main(String[] args) {
    System.out.println(" _____                          ");
    System.out.println("(___  )                         ");
    System.out.println("    | |   _ _  _   _   _ _       ");
    System.out.println(" _  | | /'_  )( ) ( ) /'_  )      ");
    System.out.println("( )_| |( (_| || \_/ |( (_| |      ");
    System.out.println(" \___/  \__,_) \___/  \__,_)      ");
  }
}
```

2. จงเปลี่ยนโปรแกรมในรหัสที่ 5-14 โดยใช้ in.next() (จะได้โปรแกรมที่สั้นและง่ายกว่า)

3. จงเขียนโปรแกรมรับสตริงทางแป้นพิมพ์หนึ่งบรรทัด เพื่อนำมาแยกออกเป็น “คำคำ” นิยามให้หนึ่งคำ คือ ลำดับอักขระที่ติดกันที่ไม่มีช่องว่าง เช่น "What a wonderful world" แยกได้เป็น "What", "a", "wonderful" และ "world"

4. จงเขียนโปรแกรมนับจำนวนคำของข้อความในแฟ้มที่กำหนดให้(นิยาม “คำ” เหมือนข้อที่แล้ว)

5. จงเขียนโปรแกรมกลับลำดับของคำในสตริงที่รับจากผู้ใช้ เช่น "love and marriage" กลับลำดับแล้วจะได้ "marriage and love" (นิยาม “คำ” เหมือนข้อที่แล้ว) ข้อแนะนำ : อ่านรายละเอียดการใช้งานเมท็อด lastIndexOf ของสตริงที่ทำงานคล้ายเมท็อด indexOf แต่เป็นการค้นจากขวามาซ้าย

6. จงเปลี่ยนโปรแกรมคำนวณเลขโดดตรวจสอบตามมาตรฐาน EAN-13 ในรหัสที่ 5-9 ให้ใช้วงวนสองวง วนหนึ่งสำหรับเลขตำแหน่งคู่ อีกวงสำหรับเลขตำแหน่งคี่

7. หนังสือทั้งหลายในโลกที่พิมพ์ขายตั้งแต่วันที่ 1 มกราคม พ.ศ. 2550 จะมีหมายเลขกำกับ 13 หลักเรียกว่า ISBN-13 โดยมีลักษณะเดียวกับ EAN-13 ที่ได้นำเสนอไปแล้ว แต่สำหรับหนังสือเก่า ๆ ที่พิมพ์ก่อนปีพ.ศ. 2550 มักใช้ ISBN แค่ 10 หลัก เช่นหนังสือ “การออกแบบและวิเคราะห์อัลกอริทึม” เขียนโดยสมชาย ประสิทธิ์จูตระกูล จัดพิมพ์โดยสำนักงานพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ มี ISBN-10 คือ 974-229-026-1 ในกรณีที่หนังสือเล่มเก่ามีการจัดพิมพ์ใหม่ สำนักพิมพ์จะเปลี่ยนไปใช้ ISBN-13 โดยให้เติมเลข 978 นำหน้าเลขเก่า ตัดเลขหลักขวาสุดออก (ที่เป็นเลขโดดตัวตรวจสอบของ ISBN แบบเก่า) แล้วเติมเลขโดดตรวจสอบใหม่ตามแบบ EAN-13 ทางขวาสุดแทน เช่น 974-229-026-1 จะเปลี่ยนเป็น 978-974-229-026-9 จงเขียนโปรแกรมรับ ISBN-10 เพื่อเปลี่ยนเป็น ISBN-13 ด้วยวิธีข้างต้น

8. ISBN-10 นั้นมีเลขโดดหลักที่สิบ (หลักขวาสุด) เป็นเลขโดดไว้ตรวจสอบความถูกต้องของอีก 9 ตัวทางซ้าย กำหนดให้ d_k แทนตัวเลขหลักที่ k (d_1 คือเลขหลักซ้ายสุด, d_{10} คือเลขหลักขวาสุด) จะได้ว่า

$$d_{10} = \left(\sum_{k=1}^{9} k \times d_k \right) \bmod 11$$

เนื่องจากผลที่ได้จากการ mod ด้วย 11 จะได้เลข 1 ถึง 10 ในกรณีที่ได้ 10 จะใช้ตัวอักษร X แทนตำแหน่งของเลขโดดตรวจสอบ เช่น 974-229-026-1 มี 1 เป็นเลขโดดตัวสอบของ 974-229-026 จงเขียนโปรแกรมรับเลข 9 หลัก เพื่อคำนวณและแสดงเลขโดดตรวจสอบทางจอภาพ

9. รหัสประจำตัวบัตรประชาชนไทยก็มีเลขหลักที่ 13 (ตัวขวาสุด) เป็นเลขโดดตรวจสอบ ซึ่งมีสูตรการคำนวณคือ

$$d_{13} = \left(11 - \left(\sum_{k=1}^{12} (14-k) \times d_k \right) \bmod 11 \right) \bmod 10$$

เช่น 3720200252613 มีเลข 3 ตัวขวาสุดเป็นเลขโดดตรวจสอบของเลข 12 ตัวทางซ้าย จงเขียนโปรแกรมรับเลข 12 หลัก เพื่อคำนวณและแสดงเลขโดดตรวจสอบ

10. พาลินโดรมของวลีคล้ายกับพาลินโดรมของคำที่ไม่ต้องพิจารณาช่องว่างและเครื่องหมายใด ๆ ในวลี เช่น "A man, a plan, a canal, Panama." เป็นพาลินโดรม จงปรับปรุงรหัสที่ 5-10 ให้สามารถตรวจสอบความเป็นพาลินโดรมของวลี

11. เว็บไซต์ wordbook ที่ใช้ในรหัสที่ 5-19 สามารถแปลบางวลีได้ เช่น คำแปลของ on one's toe อยู่ที่เว็บเพจ http://wordbook.rackhub.com/on+one's+toes.html จงแก้ไขรหัสที่ 5-19 ให้สามารถใช้กับวลีได้

12. เว็บเพจหลายแห่งนอกจากจะมีข้อมูลให้ผู้ใช้ชมแล้ว มักมีการเผยแพร่ข้อมูลในรูปแบบที่เรียกว่า RSS (หรือ XML) ที่ทำให้เราเขียนโปรแกรมอ่านข้อมูลนั้นมาประมวลผลได้สะดวก เช่น ที่ http://news.mcot.net/news_rss/politic.xml เป็นสรุปหัวข้อข่าวการเมืองของสำนักข่าวไทย หรือที่ http://rss.cnn.com/rss/cnn_world.rss เป็นสรุปหัวข้อข่าวทั่วโลกของสำนักข่าวซีเอ็นเอ็น เป็นต้น โดยถ้าอ่านข้อมูลภายในที่ได้จากเว็บเพจเหล่านี้ จะมีโครงสร้างคร่าว ๆ ดังแสดงข้างล่างนี้

```
...
<item>
...
  <title>
    หัวข้อของเรื่องที่นำเสนอ
  </title>
...
</item>
...
```

```
...
<item> ...
  <title>  หัวข้อของเรื่องที่นำเสนอ
</title> ... </item>
... <item> ...
  <title>  หัวข้อของเรื่องที่นำเสนอ
</title> ... </item>
...
```

ข้อมูลที่อ่านได้อาจถูกจัดรูปแบบให้มีระเบียบแบบทางซ้าย หรือไม่มีระเบียบแบบทางขวาก็ได้ ถ้าต้องการหัวข้อข่าวต่าง ๆ ก็เพียงแค่หาข้อความที่ถูกครอบด้วย `<title>` กับ `</title>` และถูกครอบด้วย `<item>` กับ `</item>` อีกชั้นหนึ่ง จงเขียนโปรแกรมรับตำแหน่งของเว็บเพจ เพื่อเชื่อมต่อไปยังเว็บเพจนั้น (ด้วยการใช้ `URLStream` และ `Scanner` เหมือนกับในรหัสที่ 5-19) แล้วอ่านข้อมูลที่ได้มา เพื่อค้นหาและแสดงหัวข้อต่าง ๆ ทางจอภาพ

13. จงเขียนโปรแกรมรับสตริงสองตัวจากผู้ใช้ ให้สตริงตัวแรกเก็บในตัวแปร `t` สตริงตัวที่สองเก็บในตัวแปร `p` โปรแกรมนี้มีหน้าที่นับว่า มีสตริงย่อยใน `t` กี่แห่งที่มีค่าเหมือน `p` เช่น ถ้า `t` = **"กินถามกินว่า กินจะกินก่อนหรือไม่กินก่อน ถ้ากินไม่กินก่อน กินจะกินก่อน"** และ `p` = **"กิน"** จะนับได้ 9

14. จงเขียนโปรแกรมนับจำนวนสตริงย่อยของข้อความในแฟ้มว่า มีกี่ตัวที่เหมือนกับสตริงที่ผู้ใช้กำหนด (คล้ายข้อที่แล้ว แต่เป็นการนับข้อความทั้งหมดในแฟ้ม)

15. จงเขียนโปรแกรมรับสตริงสามตัวจากผู้ใช้ เก็บในตัวแปร `t`, `from` และ `to` เพื่อสร้างและแสดงสตริงใหม่ที่ได้จากการแทนสตริงย่อยใน `t` ที่เหมือน `from` ให้เป็น `to` เช่น `t` = **"กินถามกินว่า กินจะกินก่อนหรือไม่กินก่อน ถ้ากินไม่กินก่อน กินจะกินก่อน"**, `from` = **"กิน"**, และ `to` = **"นอน"** จะได้สตริงใหม่เป็น **"นอนถามนอนว่า นอนจะนอนก่อนหรือไม่นอนก่อน ถ้านอนไม่นอนก่อน นอนจะนอนก่อน"**

16. จงเขียนโปรแกรมแทนสตริงย่อยของข้อความในแฟ้มจากสตริงหนึ่งไปเป็นอีกสตริงหนึ่งตามที่ผู้ใช้กำหนด (คล้ายข้อที่แล้ว แต่เป็นการแทนข้อความในแฟ้ม)

17. จงเขียนโปรแกรมรับคำนามภาษาอังกฤษในรูปเอกพจน์ มาเปลี่ยนและแสดงคำนามนี้ในรูปพหูพจน์ เช่น รับ box แสดง boxes เป็นต้น กำหนดให้ใช้กฎง่าย ๆ ดังนี้
    - ถ้าเป็นคำนามที่ลงท้าย s, x หรือ ch, ให้ทำเป็นพหูพจน์ด้วยการให้เติม es ต่อท้าย (เช่น box → boxes, witch → witches เป็นต้น)
    - ถ้าลงท้ายด้วย y และตัวอักษรก่อน y ไม่ใช่สระ, ให้เปลี่ยน y เป็น i แล้วเติม es ต่อท้าย (เช่น fly → flies, memory → memories เป็นต้น)
    - ถ้าไม่ตรงกับกฎสองข้อข้างบนนี้, ให้ต่อท้ายด้วย s เลย (เช่น computer → computers, boy → boys เป็นต้น)

18. ถ้าตั้งใจพิมพ์คำว่า ผิด โดยตั้งแป้นพิมพ์เป็นอังกฤษ จะได้คำว่า zbf เมื่อใดที่ตั้งภาษาของแป้นพิมพ์ผิด (TH / EN) โดยเวลาพิมพ์มองแต่แป้นพิมพ์ พอเลื่อนตาขึ้นไปดูจอ กลับพบว่า ได้ข้อความแปลก ๆ เป็นเหตุการณ์ที่เกิดขึ้นบ่อยมาก จนรู้สึกว่า คอมพิวเตอร์น่าจะช่วยแก้ปัญหานี้ให้หน่อย จงเขียนโปรแกรมเพื่อรับสตริงที่ผู้ใช้คิดว่ากำลังป้อนข้อความไทย แต่แท้จริงแล้วใช้แป้นพิมพ์อังกฤษ ให้นำสตริงที่รับมาไปแปลงเป็นข้อความไทยที่ควรเป็น แล้วแสดงทางจอภาพ (ข้อแนะนำ : ส่วนของโปรแกรมที่เขียนในแบบฝึกหัดข้อที่ 15 น่าจะใช้ให้เป็นประโยชน์ได้ในข้อนี้)

19. ในทางกลับกัน ถ้าตั้งใจพิมพ์คำว่า wrong แต่ตั้งแป้นพิมพ์เป็นไทย จะได้คำว่า ไพนืเ จงเขียนโปรแกรมเพื่อรับสตริงที่ผู้ใช้คิดว่ากำลังป้อนข้อความอังกฤษ แต่แท้จริงแล้วใช้แป้นพิมพ์ไทย ให้นำสตริงที่รับมาไปแปลงเป็นข้อความอังกฤษที่ควรเป็น แล้วแสดงทางจอภาพ

20. จงปรับปรุงรหัสที่ 5-13 โดยเปลี่ยนวงวน `while` สี่วงที่ใช้ในการนับวรรณยุกต์ทั้งสี่ มาเป็นวงวน `for` สี่วงตามที่นำเสนอในรหัสที่ 5-13 จากนั้นปรับปรุงต่อ โดยรวมวงวน `for` ทั้งสี่ให้เหลือเพียงวงเดียว

21. จงปรับปรุงรหัสที่ 5-13 ต่อจากข้อที่แล้ว โดยแต่ละบรรทัดที่อ่านจากแฟ้ม ให้ใช้วงวน `for` ดึงอักขระออกมาทีละตัวด้วยเมท็อด `charAt` จากนั้นใช้คำสั่ง `switch-case` แยกพิจารณาเป็นกรณีกรณี เพื่อเพิ่มตัวนับวรรณยุกต์ให้ตรงกับวรรณยุกต์ที่พบในแต่ละ `case`

22. รหัสที่ 5-16 และรหัสที่ 5-17 เป็นโปรแกรมตรวจคำสะกดที่ค้นคำในสตริงที่เก็บคำศัพท์ จงเขียนโปรแกรมตรวจคำสะกดที่ค้นคำแบบใช้วิธีค้นคำในแฟ้มศัพท์ ดูสิว่า จะทำงานช้าเพียงใด

23. โปรแกรมตรวจสะกดคำผิดที่ได้นำเสนอในหน้าที่ 111 มีขั้นตอนการนำคำศัพท์ทั้งหมดที่อ่านได้จากแฟ้ม riwords.txt มาต่อๆ กันไป (คั่นด้วยเครื่องหมาย `$`) ให้เป็นสตริงเดียว เพื่อจะได้

นำไปค้นด้วย indexOf ได้สะดวก ปัญหาอยู่ที่ว่า การใช้การ + สตริงเพื่อต่อสตริงจะใช้เวลานานมาก จาวามีคลาสมาตรฐานชื่อ StringBuffer (หรือจะใช้ StringBuilder ก็ได้) เพื่อใช้ต่อสตริง ส่วนของโปรแกรมข้างล่างนี้ แสดงตัวอย่างการต่อสตริง แบบบนใช้การ + แบบล่างใช้ StringBuffer ให้ผู้อ่านเขียนโปรแกรมเพื่อเปรียบเทียบเวลาการทำงานของวิธีการต่อสตริงทั้งสอง จากนั้นเปลี่ยนรหัสที่ 5-16 ให้สร้างสตริงเก็บคำศัพท์ทั้งหมดด้วย StringBuffer (แทนวิธีการเขียนคำศัพท์ต่อกันไปในแฟ้ม แล้วค่อยอ่านกลับมาเก็บในสตริง ที่เขียนในรหัสที่ 5-16)

```
String t = "";
for(int i=0; i<30000; i++) {
  t = t + "a";
}
...
```

```
StringBuffer sb = new StringBuffer();
for(int i=0; i<30000; i++) {
  sb.append("a");
}
String t = sb.toString();
...
```

6 แยกย่อย

การแยกโปรแกรมที่ใหญ่ออกเป็นส่วนย่อย ๆ เป็นหลักการพื้นฐานในการพัฒนาโปรแกรมที่ลดความซับซ้อนและซ้ำซ้อน ให้ได้โปรแกรมที่จัดการ ปรับเปลี่ยน และบำรุงรักษาได้ง่าย โดยทั่วไปเราแบ่งโปรแกรมออกเป็นหลาย ๆ เมท็อด แต่ละเมท็อดมีชื่อที่สื่อความหมายตามหน้าที่ที่เมท็อดนั้นทำ เพื่อให้ผู้อื่นเรียกใช้ได้สะดวก และได้ใช้แล้วใช้อีก บทนี้นำเสนอองค์ประกอบ การเรียกใช้ และหลักการเขียนเมท็อดที่ดี ปิดท้ายด้วยการเขียนเมท็อดแบบเรียกซ้ำซึ่งเหมาะกับขั้นตอนการทำงานที่มีลักษณะของการแบ่งปัญหาใหญ่ออกเป็นปัญหาย่อย เพื่อหาคำตอบย่อยๆ ที่สามารถนำมารวมเป็นคำตอบของปัญหาใหญ่ได้

องค์ประกอบของเมท็อด

โปรแกรมประกอบด้วยคลาส คลาสประกอบด้วยเมท็อด และเมท็อดประกอบด้วยคำสั่ง โปรแกรมต่างๆ ที่ได้นำเสนอมาตั้งแต่บทแรกเป็นต้นมามีหนึ่งคลาส และคลาสที่เขียนมาก็มีเพียงหนึ่งเมท็อด เป็นเมท็อดชื่อ `main` มีหัวเมท็อดเขียนเหมือนกัน เพราะเป็นข้อตกลงของระบบจาวาว่า ถ้าสั่งคลาสใดทำงาน จะเริ่มทำงานที่คำสั่งในเมท็อด `main` ของคลาสนั้น หลักปฏิบัติสำคัญประการหนึ่งที่ใช้กันในวงการนักเขียนโปรแกรมคือ การแยกเขียนกลุ่มคำสั่งที่มีภาระการทำงานที่เด่นชัดให้เป็นเมท็อดใหม่ ตั้งชื่อให้สื่อความหมาย แล้วให้เรียกใช้ได้จากเมท็อดอื่น ขอแสดงโปรแกรมคำนวณดัชนีมวลกายที่ได้เขียนในบทที่ 2 ให้ดูอีกครั้งในรหัสที่ 6-1 โปรแกรมนี้รับจำนวนจริงจากผู้ใช้ 2 จำนวน นำมาคำนวณ และแสดงผล เราอาจปรับโปรแกรมโดยแยกส่วนที่รับจำนวนจริงทางแป้นพิมพ์ออกเป็นเมท็อดชื่อ `readDouble` ได้ดังรหัสที่ 6-2 ทำให้สามารถเรียกใช้ใน `main` ที่บรรทัดที่ 6 และ 7 เพื่อรับน้ำหนักและความสูงได้กะทัดรัดและสื่อความหมายมากขึ้น

```
01 import java.util.Scanner;
02 // โปรแกรมคำนวณดัชนีมวลกาย
03 public class BodyMassIndex {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     System.out.print("น้ำหนัก (kg.) = ");
07     double weight = kb.nextDouble();
08     System.out.print("ความสูง (cm.) = ");
09     double height = kb.nextDouble();
10     double hm = height / 100.0;
11     double bmi = weight / (hm * hm);
12     System.out.println("ดัชนีมวลกาย = " + bmi);
13   }
14 }
```

รหัสที่ 6-1 โปรแกรมคำนวณดัชนีมวลกาย

```
01 import java.util.Scanner;
02 // โปรแกรมคำนวณดัชนีมวลกาย
03 public class BodyMassIndex {
04   public static void main(String[] args) {
05     Scanner kb = new Scanner(System.in);
06     double weight = readDouble(kb, "น้ำหนัก (kg.) = ");
07     double height = readDouble(kb, "ความสูง (cm.) = ");
08     double hm = height / 100.0;
09     double bmi = weight / (hm * hm);
10     System.out.println("ดัชนีมวลกาย = " + bmi);
11   }
12   public static double readDouble(Scanner kb, String prompt) {
13     System.out.print(prompt);
14     double value = kb.nextDouble();
15     return value;
16   }
17 }
```

เรียกใช้เมท็อด readDouble ที่เขียนเป็นเมท็อดข้างล่าง

หัวเมท็อด

ตัวเมท็อด

รหัสที่ 6-2 ตัวอย่างการแยกคำสั่งรับจำนวนจากผู้ใช้มาเขียนเป็นเมท็อด

เมื่อใดที่มีการเรียกใช้เมท็อด การทำงานจะย้ายไปที่เมท็อดนั้น ทำจนเสร็จ ก็กลับมาทำต่อ ณ คำสั่งที่เรียกเมท็อดนั้น ตัวอย่างเช่น ถ้าเราสั่งโปรแกรมในรหัสที่ 6-2 ทำงาน จะเริ่มทำที่บรรทัดที่ 5 สร้างตัวอ่านแป้นพิมพ์ พอถึงบรรทัดที่ 6 เรียกเมท็อด `readDouble` จึงกระโดดไปทำบรรทัดที่ 13, 14, 15 สิ้นสุดการทำงานที่เมท็อดนี้ กลับไปทำต่อจากจุดที่เรียกคือบรรทัดที่ 6 ให้ค่ากับ `weight` เรียบร้อย ก็ทำบรรทัดที่ 7 ต่อ ซึ่งเรียก `readDouble` อีกแล้ว จึงกระโดดไปทำบรรทัดที่ 13, 14, 15 แล้วก็กลับมาไปทำต่อที่บรรทัดที่ 7 ให้ค่ากับ `height` แล้วทำบรรทัดที่ 8, 9, 10 และ 11 เป็นอันจบการทำงาน (การทำงานของบรรทัดที่ 5, 10, 13, และ 14 ก็เรียกเมท็อดเช่นกัน แต่เป็นเมท็อดของคลาสอื่น การทำงานก็จะย้ายไปที่เมท็อดของคลาสนั้น และกลับมาเมื่อทำเสร็จในทำนองเดียวกัน)

หัวเมท็อด

เมท็อดประกอบด้วยส่วนหัวและส่วนตัว ส่วนหัวของเมท็อดประกอบด้วยชื่อ ประเภทของผลลัพธ์ และรายการของพารามิเตอร์ ซึ่งคือข้อมูลที่รับเข้ามาใช้ มีรายละเอียดการเขียนดังนี้[1]

```
public static ประเภทของผลลัพธ์ ชื่อเมท็อด( รายการของพารามิเตอร์ )
```

- ชื่อเมท็อด : คือคำที่อยู่ข้างหน้าวงเล็บเปิดในหัวเมท็อด เมท็อดของบรรทัดที่ 12 ในรหัสที่ 6-2 ชื่อ `readDouble` การตั้งชื่อเมท็อดมีกฎเกณฑ์เหมือนกับการตั้งชื่อตัวแปร โดยทั่วไปมักตั้งให้เป็น “กริยา” เพราะเมท็อดมีไว้ให้ “ทำ” อะไรบางอย่าง เช่น `draw`, `print` ถ้าตั้งเป็น “นาม” ก็มักเป็นเมท็อดที่คืนค่าของสิ่งที่มีชื่อนั้น เช่น `width`, `x`, `y` เป็นต้น (นักเขียนโปรแกรมบางกลุ่มชอบเขียน `getWidth`, `getX`, `getY` แทน)
- ประเภทของผลลัพธ์ : ข้างหน้าของชื่อเมท็อดคือประเภทของข้อมูลที่จะคืนกลับเป็นผลลัพธ์ เมท็อด `readDouble` ในรหัสที่ 6-2 ระบุว่า คืน `double` เพราะฉะนั้นใครที่เรียกใช้ `readDouble` ก็ควรเตรียมนำ `double` ไปใช้ ถ้าจะนำไปเก็บก็ต้องเก็บในตัวแปร `double` ในกรณีที่เมท็อดไม่มีอะไรคืนกลับไป เช่น `main` จะใส่คำว่า `void`[2]
- รายการของพารามิเตอร์ : *พารามิเตอร์* (parameter) คือตัวแปรของตัวเมท็อดที่รับข้อมูลจากผู้เรียกมาประมวลผล ถือว่าเป็นข้อมูลขาเข้าของเมท็อด เปรียบเสมือนตัวกำหนดพฤติกรรมของเมท็อด รายการของพารามิเตอร์ปรากฏในวงเล็บหลังชื่อเมท็อด เขียนเหมือนกับการประกาศตัวแปร แต่เป็นการประกาศตัวแปรทีละตัว คั่นการประกาศด้วยเครื่องหมาย , เมท็อด `readDouble` มีพารามิเตอร์สองตัว คือ ตัวอ่าน `Scanner` ชื่อ `kb` และสตริงชื่อ `prompt` หากเราเรียก `readDouble(kb, "a = ")` `prompt` จะมีค่า `"a = "` ตอนเริ่มทำงานในเมท็อด
- ทัศนวิสัยของเมท็อด : คำแรกของหัวเมท็อด `public` แปลว่า สาธารณะ หมายความว่า คลาสใด ๆ ก็เห็นเมท็อดนี้ เรียกใช้ได้จากทุกที่ หากเป็น `private` แปลว่า ส่วนตัว คลาสอื่นไม่เห็น จึงเรียกใช้ไม่ได้ จะใช้ได้เฉพาะภายในคลาสที่เมท็อดนี้ปรากฏอยู่เท่านั้น

มาดูตัวอย่างการเขียนหัวเมท็อดดังต่อไปนี้

- เมท็อดที่คืนจำนวนเต็มสุ่มมีค่าตั้งแต่ `a` ถึง `b` : ให้ชื่อเมท็อดว่า `random` เป็นเมท็อดที่คืน `int` และรับพารามิเตอร์สองตัวชื่อว่า `a` และ `b` เป็น `int` เขียนหัวเมท็อดได้ดังนี้

```
public static int random(int a, int b)
```

---

[1] ยังขอไม่อธิบายความหมายของคำว่า `static` ที่นำหน้าหัวเมท็อด ขอให้จำและเขียนแบบนี้ไปก่อน

[2] เมท็อดคืนผลลัพธ์ได้เพียงค่าเดียว ถ้าต้องการคืนข้อมูลหลายตัวต้องใช้วิธีการรวมข้อมูลทั้งหลายเป็นข้อมูลประกอบ ซึ่งจะอธิบายในบทถัด ๆ ไป

- เมท็อดที่นับว่าภายในสตริง `text` เก็บสตริงย่อยที่มีค่าเหมือนสตริง `pattern` อยู่กี่ตำแหน่ง : ตั้งชื่อเมท็อดว่า `count` คืน `int` และมีพารามิเตอร์เป็นสตริงสองตัว ดังนี้

```
public static int count(String text, String pattern)
```

- เมท็อดที่คืนชื่อเดือนปัจจุบัน : ตั้งชื่อว่า `getCurrentMonth` ซึ่งคืนสตริง ไม่ต้องรับพารามิเตอร์ใดๆ เขียนหัวเมท็อดได้ดังนี้

```
public static String getCurrentMonth()
```

- เมท็อดที่มีไว้เรียกเมื่อต้องการแจ้งข้อผิดพลาดให้ผู้ใช้ทราบทางจอภาพ ดังนี้

```
public static void showErrorMessage(String msg)
```

จาวาอนุญาตให้ตั้งชื่อเมท็อดซ้ำกันได้ ถึงแม้จะอยู่ในคลาสเดียวกัน โดยเมท็อดที่มีชื่อซ้ำกันต้องมีรายการของประเภทพารามิเตอร์ต่างกัน ที่เขียนว่าประเภทพารามิเตอร์นี้หมายถึงไม่ต้องสนใจชื่อของพารามิเตอร์ สนใจเฉพาะประเภทข้อมูลของพารามิเตอร์ และไม่ต้องสนใจประเภทของผลลัพธ์ เช่น `int f(int x)` , `String f(double x)` , `int f(int x, int y)` เป็นสามเมท็อดที่อยู่ร่วมกันได้ เพราะส่วนที่เป็นรายการของประเภทพารามิเตอร์ต่างกัน ในขณะที่ `int f(int x)` , `int f(int b)` , `double f(int x)` ไม่ถูกต้อง

ตัวเมท็อด

ส่วนของตัวเมท็อดประกอบด้วยคำสั่งต่าง ๆ ที่เขียนอยู่ภายในเครื่องหมาย `{` และ `}` เมื่อใดเมท็อดทำงานเสร็จแล้ว จะกลับไปทำงานต่อจากคำสั่งที่เรียกเมท็อดนี้ เราได้เขียนคำสั่งต่าง ๆ ในเมท็อด `main` มาตั้งแต่บทแรก เมื่อ `main` ทำงานเสร็จ ถือว่าเป็นการสิ้นสุดของโปรแกรม มีประเด็นที่ต้องทราบเกี่ยวกับคำสั่งที่เขียนในตัวเมท็อดดังนี้

- การประกาศตัวแปรภายในเมท็อดใด ใช้ได้เฉพาะในเมท็อดนั้น จึงเป็นที่มาของการเรียกตัวแปรที่ประกาศภายในเมท็อดว่า *ตัวแปรเฉพาะที่* (local variable)[3] ตัวแปรของคนละเมท็อดจะไม่เกี่ยวกันเลย ถึงแม้จะตั้งชื่อเดียวกันแต่ถ้าอยู่คนละเมท็อดก็ถือว่าเป็นคนละตัว เพราะตัวแปรเฉพาะที่ของเมท็อดจะถูกสร้างก็เมื่อเมท็อดนั้นถูกเรียก โดยระบบจะจัดเตรียมเนื้อที่ใหม่ในหน่วยความจำหลักให้กับตัวแปรเฉพาะที่ และจะยกเลิกการใช้เนื้อที่เหล่านั้นเมื่อเมท็อดสิ้นสุดการทำงาน
- ในกรณีที่เมท็อดไม่มีการคืนผลลัพธ์ การทำงานของเมท็อดจะสิ้นสุดเมื่อทำจนถึงเครื่องหมาย `}` สุดท้ายของตัวเมท็อด หรือไม่ก็เมื่อทำคำสั่ง `return` แต่ถ้าเป็นเมท็อดที่มีการคืนผลลัพธ์ การทำงานจะสิ้นสุดเมื่อทำคำสั่ง `return v` โดยที่ `v` คือผลลัพธ์ที่

[3] เราสามารถประกาศตัวแปรไว้นอกเมท็อดได้ ซึ่งจะอธิบายในบทถัดๆ ไป

ต้องการคืนกลับไปให้ผู้เรียก และ v ต้องมีค่าที่เป็นประเภทเดียวกับประเภทของผลลัพธ์ที่ได้เขียนไว้ที่หัวเมท็อด หรือเป็นแบบที่เปลี่ยนได้อย่างอัตโนมัติ เช่น รหัสที่ 6-3 ด้านบนแสดงเมท็อดที่คืนผลลัพธ์เป็น double ถึงแม้ว่าบรรทัดที่ 2 หลัง if จะคืน 0 ที่เป็น int ก็ไม่เป็นไร เพราะ int เปลี่ยนเป็น double ได้อย่างอัตโนมัติ ในขณะที่รหัสด้านล่าง หลัง if คืน 0.0 แต่ที่หัวเมท็อดประกาศว่าคืน int จะผิด

```
public static double clip(double x) {
  if (x < 0) return 0;
  return x;
}
```

คืน int เปลี่ยนเป็น double ให้

```
public static int clip(double x) {
  if (x < 0) return 0.0;
  return x;
}
```

คืน double ไม่เปลี่ยนเป็น int ให้

รหัสที่ 6-3 ตัวอย่างการคืนผลลัพธ์ (ด้านบนถูก ด้านล่างผิด)

ตัวอย่างผิด ๆ

ขอให้ผู้อ่านพิจารณาเมท็อดต่าง ๆ ในรหัสที่ 6-4 แล้วหาว่า แต่ละเมท็อดผิดที่ใด[4]

```
public static print(String t) {
  System.out.println( t );
}
public static int abs(int a) {
  if ( a < 0 ) return -a;
}
public static double max(double x, y) {
  if ( x > y ) return x; else return y;
}
public static double length(double dx, double dy) {
  double dx = Math.abs(dx);
  return Math.sqrt( dx*dx + dy*dy );
}
public static int b(int b) {
  while (true)
    System.out.println("hello");
  return 0;
}
```

รหัสที่ 6-4 ตัวอย่างเมท็อดที่เขียนผิด

4 เมท็อด print ไม่ได้ระบุประเภทของผลลัพธ์ (ต้องใส่ void), เมท็อด abs ไม่ได้คืนผลลัพธ์กรณีที่ไม่เข้า if (ต้องใส่ return a; ต่อท้าย if), เมท็อด max เขียนประกาศพารามิเตอร์ผิด แม้พารามิเตอร์ทั้งหมดจะเป็น double เหมือนกัน ก็ต้องเขียนเป็น max(double a, double b), เมท็อด length มีการประกาศตัวแปรชื่อ dx ซึ่งซ้ำกับชื่อของพารามิเตอร์ ซึ่งทำไม่ได้, เมท็อด b สุดท้าย ไม่มีอะไรผิดที่ตั้งชื่อเมท็อดซ้ำกับชื่อตัวแปร หรือชื่อพารามิเตอร์ก็ได้ แต่เนื่องจากเมท็อด b ต้องคืน int แต่เมท็อดนี้วนทำงานไม่รู้จบ (ไม่มี break อยู่ในวงวน) ดังนั้นจึงไม่มีทางไปถึงคำสั่ง (คือ return 0 จึงไม่มีทางได้ทำงาน ผิด

การเรียกใช้เมท็อด

จะเรียกใช้เมท็อดใด ต้องรู้ว่าเมท็อดนั้นทำอะไร ตรงตามที่เราต้องการใช้บริการหรือไม่ และมีหัวเมท็อดอย่างไร จะได้เรียกให้ถูกต้องตามหลักไวยากรณ์ของภาษา ผู้เรียกใช้เมท็อดต้องเขียนชื่อให้ถูกต้อง และส่งข้อมูลไปยังพารามิเตอร์ต่าง ๆ ให้ครบและถูกประเภท สำหรับผลลัพธ์ที่คืนกลับมานั้น ภาษาจาวาไม่ได้บังคับว่าต้องนำไปใช้ แต่ถ้านำไปใช้หรือนำไปเก็บ ต้องให้ถูกต้องตามประเภทข้อมูลที่คืนกลับมา สิ่งที่ต้องสนใจเป็นพิเศษคือการส่งข้อมูลไปให้พารามิเตอร์ ซึ่งมีประเด็นที่ต้องเข้าใจดังนี้

- เราสามารถส่งผลของนิพจน์ไปให้เมท็อดได้ เพราะสิ่งที่เมท็อดต้องการคือค่าของข้อมูลที่จะนำไปตั้งเป็นค่าเริ่มต้นให้กับพารามิเตอร์ เช่น `a = Math.max(2*a, b-7)` คือการให้คำนวณค่า `2*a` และ `b-7` ก่อน แล้วค่อยนำผลที่ได้ส่งไปให้พารามิเตอร์ของเมท็อด `max` ถ้าเขียน `Math.max(Math.max(a, b), c)` จะส่งค่าของ `a` และ `b` ให้ `max` (ตัวใน) ไปหาค่ามากได้ผลคืนกลับมา ก็ส่งไปให้ `max` (ตัวนอก) พร้อมกับ `c` อีกครั้งเพื่อหาค่ามาก โดยสรุปคือต้องการหาค่ามากสุดของ `a`, `b`, และ `c`
- ขอเน้นอีกครั้งว่า สิ่งที่เราส่งไปให้เมท็อดคือ ค่าของตัวแปรหรือนิพจน์ ซึ่งถูกส่งไปเป็นค่าเริ่มต้นของพารามิเตอร์ ดังนั้น หากภายในเมท็อดมีการเปลี่ยนแปลงค่าของพารามิเตอร์ ย่อมไม่มีผลใดๆ ต่อตัวแปรที่เราใช้ส่งค่าไปให้ เช่น รหัสที่ 6-5 แสดงเมท็อด `reset` ที่ทำคำสั่งเดียวคือ ตั้งค่าให้ `x` (ซึ่งคือพารามิเตอร์) มีค่าเป็น `0` จากตัวอย่างการเรียกใช้ `reset` เราเรียก `reset(a)` เป็นการส่งค่าของ `a` ที่มีค่า `23` ไปให้ `x` ถึงแม้ `x` จะถูกเปลี่ยนค่าเป็น `0` ก็ไม่เกี่ยวอะไรกับ `a` และเช่นเดียวกันกับคำสั่ง `reset(x)` ถึงแม้ว่าจะมีชื่อ `x` เหมือนพารามิเตอร์ `x` ก็ไม่ได้หมายความว่าเป็นตัวแปรเดียวกัน เพราะเป็นของคนละเมท็อด การเปลี่ยน `x` ที่เมท็อด `reset` ก็ไม่กระทบ `x` ของผู้เรียก

```
int a = 23, x = 36;
reset(a);
reset(x);
System.out.println(a + ", " + x); // แสดง 23, 36
```

```
public static void reset(double x) {
  x = 0;
}
```

รหัสที่ 6-5 ตัวอย่างแสดงการเปลี่ยนค่าของพารามิเตอร์ในเมท็อดไม่มีผลต่อตัวแปรของผู้เรียก

- ในกรณีที่มีการเรียกเมท็อดที่มีชื่อซ้ำกัน ตัวแปลโปรแกรมจะเลือกเมท็อดที่มีประเภทของข้อมูลที่ส่ง กับประเภทของพารามิเตอร์ที่ตรงกันมากที่สุดและไม่กำกวม เช่น รหัสที่ 6-6 มีเมท็อด `min` ที่มีชื่อซ้ำกันสองเมท็อด เมท็อดแรกรับ `double` สองตัว อีกเมท็อด

รับ int สองตัว ถ้ามีการเรียก min(1.5,3.3) ย่อมต้องไปทำที่เมท็อดแรก ถ้าเรียก min(1,3) ก็ไปทำเมท็อดหลัง (ให้สังเกตคำสั่งในบรรทัดที่ 2 เรานำผลที่คืนมาเป็น int เก็บใส่ตัวแปรแบบ double ซึ่งไม่ผิด) และคำสั่งที่สามเราเรียก min(2,2.0) ตัวแปลภาษาจะเลือกไปเรียก min แบบรับสอง double เพราะข้อมูลทั้งสองส่งไปให้ double ทั้งสองตัวได้ แต่จะส่งไปให้ min แบบรับสอง int ไม่ได้

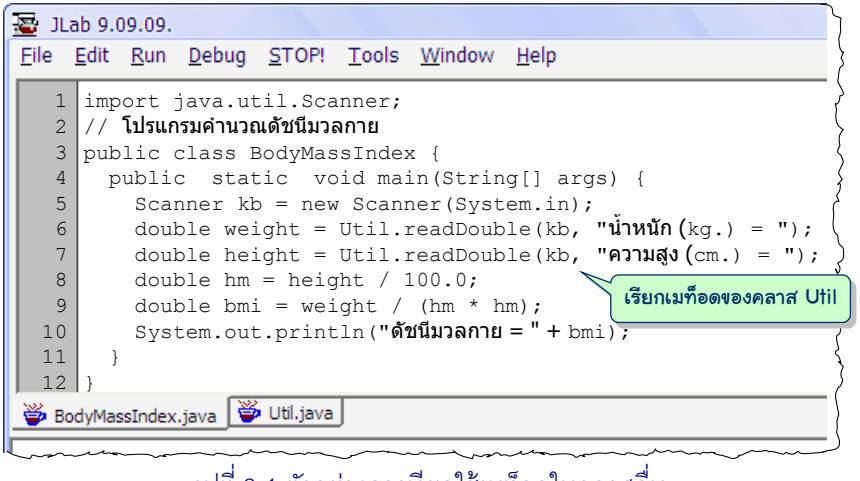
```
double a = min(1.5, 3.3);
double b = min(1, 3);
double c = min(2, 2.0);
```

```
public static double min(double a, double b) {
  if (a < b) return a; else return b;
}
public static int min(int a, int b) {
  if (a < b) return a; else return b;
}
```

รหัสที่ 6-6 ตัวอย่างการเรียกใช้เมท็อดที่มีชื่อซ้ำ

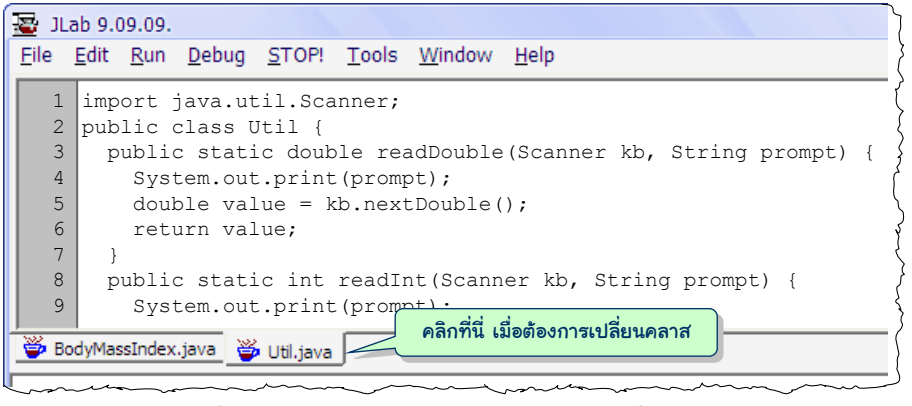
การเรียกใช้เมท็อดของคลาสอื่น

ขณะที่นักเขียนโปรแกรมกำลังพัฒนาซอฟต์แวร์สำหรับระบบหนึ่ง ก็มักเขียนเมท็อดเพิ่มที่ตัวเองเรียกใช้เอง เมื่อเขียนเมท็อดมากขึ้น ๆ บางเมท็อดก็เหมาะกับงานที่กำลังพัฒนาอยู่ แต่ก็มีหลายเมท็อดที่คาดว่า สามารถใช้ได้อีกในอนาคตกับงานอื่น ๆ เมื่อมีความรู้สึกดังกล่าว ก็มักย้ายเมท็อดเหล่านี้ไปอยู่รวม ๆ กันในคลาสใหม่ และตั้งชื่อที่สื่อว่าเป็นคลาสอรรถประโยชน์ เช่น Util (ย่อมาจาก utilities ที่แปลว่าอรรถประโยชน์) เมท็อด readDouble ในรหัสที่ 6-2 จัดเป็นเมท็อดที่มีลักษณะดังกล่าว เมื่อใดต้องการเรียกเมท็อดสาธารณะของคลาสอื่น ให้นำหน้าชื่อเมท็อดด้วยชื่อคลาส มีเครื่องหมายจุด . คั่นกลาง เช่น เรียกใช้ Util.readDouble(...) ดังรูปที่ 6-1



รูปที่ 6-1 ตัวอย่างการเรียกใช้เมท็อดในคลาสอื่น

ใน JLab เราสามารถสร้างคลาสใหม่ด้วยการเลือกเมนู File → New → Java Class ระบบก็จะแสดงกล่องโต้ตอบให้เติมชื่อคลาส จะเพิ่มคลาสใหม่กี่คลาสก็ได้ ระบบจะเตรียมเนื้อที่และเพิ่มพื้นที่ใหม่ให้เขียนคลาสใหม่ เมื่อต้องการดูหรือแก้ไขคลาสใด ก็เพียงคลิกที่ชื่อคลาสหลังรูปถ้วยกาแฟ อย่าลืมว่าจะสั่ง `main` ของคลาสใดทำงาน ให้เลือกคลาสนั้นก่อน แล้วค่อยสั่งทำงาน



รูปที่ 6-2 คลาสอรรถประโยชน์เก็บเมท็อดให้ผู้อื่นเรียกใช้

หลักการเขียนเมท็อด

จาวากำหนดให้เราเขียนเมท็อด `main` เพื่อเป็นจุดเริ่มต้นของการทำงาน แล้วเมื่อไรเราจึงเขียนเมท็อดอื่น โดยทั่วไปนักเขียนโปรแกรมควรออกแบบขั้นตอนการทำงานของโปรแกรมให้ประกอบด้วยขั้นตอนการทำงานย่อย เป็นเมท็อดย่อย ๆ ที่แต่ละเมท็อดมีภาระที่ต้องทำชัดเจนหนึ่งอย่าง โปรแกรมที่เราเขียนกันมาส่วนมากตกอยู่ในรูปแบบที่แบ่งขั้นตอนออกเป็น การรับข้อมูล การประมวลผลข้อมูล และการแสดงผลลัพธ์ ในขั้นตอนการประมวลผลก็อาจแบ่งเป็นขั้นตอนย่อยๆ ได้อีก ผู้อ่านที่เริ่มหัดเขียนโปรแกรมอาจยังขาดประสบการณ์ในการแบ่งกระบวนการประมวลผลออกเป็นเมท็อดย่อย ๆ ซึ่งก็ต้องค่อยๆ ฝึกฝนไป สิ่งที่ควรสังเกตระหว่างการเขียนโปรแกรม คือ เมื่อใดเริ่มรู้สึกว่า เมท็อด `main` หรือเมท็อดอื่น ที่เขียนอยู่ยาวผิดปกติ (หมายความว่า มีจำนวนบรรทัดจำนวนคำสั่งมาก [5]) หรือไม่ก็เริ่มเห็นว่า มีกลุ่มคำสั่งซ้ำ ๆ กันในโปรแกรม [6] นั่นเป็นสัญญาณให้หยุดคิดได้แล้วว่า ควรแยกกลุ่มคำสั่งที่เห็นซ้ำๆ ออกมาเขียนเป็นเมท็อดใหม่จะดีไหม หรือไม่ก็ลองแบ่งส่วนของโปรแกรมที่เห็นยาวนี้เป็นกลุ่มๆ ที่เมื่อจับกลุ่มแล้ว รู้ว่ากลุ่มนี้มีหน้าที่อะไร กลุ่มนั้นมีหน้าที่อะไร หากเราแยกแต่ละกลุ่ม ๆ ออกเป็นเมท็อด ตั้งชื่อให้ตรงกับหน้าที่ที่ทำแล้วเขียนใหม่ให้เรียกใช้เมท็อดใหม่นี้ ทำให้เราอ่านโปรแกรมง่ายขึ้นหรือไม่ ถ้าคิดว่าทำแล้วดีขึ้น

[5] เมท็อดยาวไปมักหมายถึงเมท็อดที่มีจำนวนบรรทัดเกินที่จะเห็นได้ในหนึ่งจอภาพ (ด้วยตัวอักษรขนาดปกติ ☺)

[6] ในวิศวกรรมซอฟต์แวร์เรียกเหตุการณ์เช่นนี้ว่า โปรแกรมมี “กลิ่นตุๆ” (bad smell) อันเป็นสัญญาณที่บอกนักเขียนโปรแกรมว่าควรสนใจดูตัวโปรแกรมว่ามีอะไรต้องปรับปรุงหรือประกอบใหม่หรือไม่

คือโปรแกรมอ่านง่ายขึ้น หรือเมท็อดใหม่เขียนแล้วได้ใช้แล้วใช้อีกในหลายสถานการณ์ ก็ควรแยกเขียนเป็นเมท็อด ถ้าลองกลับไปดูโปรแกรมนับวรรณยุกต์ในรหัส 5-13 จะพบว่า มีวงวนซ้ำกันถึง 4 วง แต่ละวงต่างกันตรงที่จะให้นับวรรณยุกต์อะไร จึงควรแยกกลุ่มคำสั่งที่วนนับนี้ ออกเป็นเมท็อดใหม่ ดังแสดงในรหัสที่ 6-7 เราแยกออกมาเป็นเมท็อดชื่อ `countTone` แล้วส่งคำกับวรรณยุกต์ที่จะนับในคำนั้นไปเป็นพารามิเตอร์ของเมท็อด ได้ผลกลับมาเป็นจำนวนวรรณยุกต์ที่พบ ให้สังเกตว่า เมื่ออ่านบรรทัด `c1 += countTone(line,t1)` จะสื่อความหมายว่า “ทำอะไร” ไม่ได้บอกว่า “ทำอย่างไร” ซึ่งเป็นหลักสำคัญของการทำความเข้าใจขั้นตอนการทำงานในภาพรวม

```
01 import java.util.Scanner;
02 import java.io.File;
03 import java.io.IOException;
04 // โปรแกรมนับจำนวนวรรณยุกต์ในพจนานุกรม
05 public class ToneCount {
06   public static void main(String[] args) throws IOException {
07     Scanner in = new Scanner(new File("c:/java101/riwords.txt"));
08     int c1 = 0, c2 = 0, c3 = 0, c4 = 0;
09     String t1 = "";  // ไม้เอก
10     String t2 = "";  // ไม้โท
11     String t3 = "";  // ไม้ตรี
12     String t4 = "";  // ไม้จัตวา
13     while (in.hasNext()) {
14       String line = in.nextLine();
15       line = line.trim();
16       if ( line.length()>0 && !"#".equals(line.substring(0,1)) ) {
17         c1 += countTone(line, t1);
18         c2 += countTone(line, t2);
19         c3 += countTone(line, t3);
20         c4 += countTone(line, t4);
21       }
22     }
23     System.out.println("ไม้เอกมี\t " + c1 + " ตัว");
24     System.out.println("ไม้โทมี\t " + c2 + " ตัว");
25     System.out.println("ไม้ตรีมี\t " + c3 + " ตัว");
26     System.out.println("ไม้จัตวามี\t " + c4 + " ตัว");
27   }
28   // countTone คืนจำนวน tone ที่ปรากฏใน word
29   public static int countTone(String word, String tone) {
30     int k = -1, count = 0;
31     while ((k=word.indexOf(tone, k+1)) >= 0) {
32       count++;
33     }
34     return count;
35   }
36 }
```

เรียกใช้เมท็อดเพื่อนับวรรณยุกต์ที่ปรากฏในคำที่อ่านเข้ามา

นิพจน์ (k=word.indexOf(tone,k+1)) ได้ผลเท่ากับค่าที่ใส่ให้ k ซึ่งหมายความว่าหลังจากค้น tone ใน word เริ่มที่ k+1 แล้วได้ผลเก็บใน k ซึ่งถ้า >= 0 ก็ให้ทำในวงวนต่อ

รหัสที่ 6-7 โปรแกรมนับจำนวนวรรณยุกต์ในพจนานุกรม (ปรับจากรหัส 5-13)

แล้วเราจะทราบได้อย่างไรว่า เมท็อดที่เขียนสั้นพอ ดีพอ ขอให้คำแนะนำดังนี้

- เมท็อดควรมีภาระที่ต้องทำเพียงอย่างเดียวตามชื่อของเมท็อด และจะยิ่งดีขึ้นถ้าเป็นภาระที่น่าจะถูกใช้หลายๆ ที่ ในหลายๆ สถานการณ์ สมมติว่ามีส่วนของโปรแกรมที่อ่านภาพจากแฟ้มข้อมูลเข้ามาในหน่วยความจำตามด้วยการทำภาพนั้นให้คมชัดขึ้น เราก็ไม่ควรเขียนเป็นเมท็อดชื่อ `loadAndSharpenImage` แต่ควรแยกเป็นสองเมท็อด คือ `loadImage` กับ `sharpen` ซึ่งทั้งสองเมท็อดนี้น่าจะถูกเรียกใช้งานในหลากหลายสถานการณ์กว่าการรวมสองภาระเข้าด้วยกัน
- เมท็อดควรสั้นกะทัดรัดอ่านเข้าใจง่าย : ข้อแนะนำโดยทั่วไปก็คือเมท็อดไม่ควรมีจำนวนบรรทัดยาวเกินหนึ่งหน้าจอภาพ คุณ Jerry Weinberg หนึ่งในผู้เชี่ยวชาญด้านการพัฒนาโปรแกรมเสนอว่า ควรเป็นเมท็อดที่มีขนาดพอเหมาะที่นักเขียนโปรแกรมทั่วไปอ่านสักพัก เมื่อเข้าใจแล้ว สามารถเขียนใหม่ได้ โดยไม่ต้องกลับไปดูรหัสต้นฉบับเดิม
- เมท็อดที่ดีไม่ควรมีจำนวนพารามิเตอร์มากเกินไป (มักกำหนดไว้ว่าอย่าเกินสาม) เนื่องจากลำดับของพารามิเตอร์นั้นสำคัญต่อการเรียกใช้ หากมากเกินไป อาจเข้าใจสับสน สำหรับกรณีที่เมท็อดใช้ชื่อซ้ำกัน รายการพารามิเตอร์ก็ไม่ควรคล้ายกันมาก เช่น `g(int a, double b)` กับ `g(double a, int b)` ซึ่งไม่ผิด แต่จะจำผิด
- ควรเขียนหมายเหตุกำกับเมท็อดเสมอในรหัสต้นฉบับว่าเมท็อดนี้ทำอะไร ข้อมูลที่รับต้องมีลักษณะอย่างไร และจะคืนอะไรกลับไป ข้อจำกัดของการทำงานมีอะไรบ้าง ทั้งนี้เพื่อให้ผู้เรียกเข้าใจเงื่อนไขการเรียกใช้ และการนำผลไปใช้
- ควรตรวจสอบความถูกต้องของพารามิเตอร์ที่รับมาว่า เป็นไปตามข้อกำหนดของการให้บริการของเมท็อดหรือไม่ โดยเฉพาะเมท็อดที่เป็น `public` ในฐานะผู้เขียนเมท็อดให้ผู้อื่นเรียก (หรือแม้แต่เขียนเองใช้เองก็ตาม) ไม่ควรไว้ใจว่าจะได้รับข้อมูลที่ถูกต้องตามข้อกำหนด เช่น หากเรากำหนดว่าจะให้บริการวาดรูปวงกลมโดยรับรัศมีที่ต้องเป็นจำนวนบวกเท่านั้น ก็ควรตรวจสอบก่อนวาดด้วย ถ้าไม่ตรงตามข้อกำหนดของเมท็อด ก็ควรแสดงข้อผิดพลาดทันที (หรือใช้การ “โยน” สิ่งผิดปกติซึ่งจะกล่าวในภายหลัง)

การแปลงจำนวนเต็มเป็นข้อความ

ขอนำเสนอตัวอย่างโปรแกรมที่ซับซ้อนขึ้น ซึ่งถ้าเขียนในเมท็อดเดียว จะยาวและอ่านลำบาก เริ่มด้วยเมท็อดการแปลงจำนวนเต็มเป็นข้อความ มีหัวเมท็อดดังนี้

```
public static String int2text(int i)
```

ตัวอย่างเช่น `int2text(321)` จะได้ **"สามร้อยยี่สิบเอ็ด"** จะเขียนเมท็อดนี้ได้ต้องมีเมท็อดย่อยแปลงเลขโดดให้เป็นคำไทย คือ 0 เป็น **ศูนย์**, 1 เป็น **หนึ่ง**, ..., 9 เป็น **เก้า** และต้องมีการแปลงเลข

หลักเป็นคำไทยด้วย คือ 0 เป็น **หน่วย**, 1 เป็น **สิบ**,..., 6 เป็น **ล้าน** รหัสที่ 6-8 แสดงเมท็อดแปลงเลขโดดเป็นคำไทย โดยใช้คำสั่ง `if-else` ไล่เปรียบเทียบค่า ถ้าอยู่นอกช่วงจะคืนสตริง `"???"` ส่วนรหัสที่ 6-9 แสดงเมท็อดแปลงเลขหลักเป็นคำไทย อาศัยคำสั่ง `if-else` ในทำนองเดียวกัน

```
// เมท็อดแปลงเลขโดดเป็นคำไทย, digit มีค่าตั้งแต่ 0,1,2,...,9
public static String digit2text(int digit) {
  String t = "";
  if (digit == 0) t = "ศูนย์";
  else if (digit == 1) t = "หนึ่ง";
  else if (digit == 2) t = "สอง";
  else if (digit == 3) t = "สาม";
  else if (digit == 4) t = "สี่";
  else if (digit == 5) t = "ห้า";
  else if (digit == 6) t = "หก";
  else if (digit == 7) t = "เจ็ด";
  else if (digit == 8) t = "แปด";
  else if (digit == 9) t = "เก้า";
  else t = "???";
  return t;
}
```

รหัสที่ 6-8 เมท็อดแปลงเลขโดดเป็นคำไทย

```
// เมท็อดแปลงหลักเป็นคำไทย, pos มีค่าตั้งแต่ 0,1,2,...,6
public static String pos2text(int pos) {
  String t = "";
  if (pos == 0) t = "หน่วย";
  else if (pos == 1) t = "สิบ";
  else if (pos == 2) t = "ร้อย";
  else if (pos == 3) t = "พัน";
  else if (pos == 4) t = "หมื่น";
  else if (pos == 5) t = "แสน";
  else if (pos == 6) t = "ล้าน";
  else t = "???";
  return t;
}
```

รหัสที่ 6-9 เมท็อดแปลงเลขหลักเป็นคำไทย (0 เป็นหน่วย, 1 เป็นสิบ, ..., 6 เป็นล้าน

เมื่อต้องการแปลงจำนวนเต็มในตัวแปร `i` เป็นข้อความ (ขอจำกัดให้ `int2text` รับพารามิเตอร์ `i` ซึ่งมีค่าตั้งแต่ 0 ถึง 9,999,999 เท่านั้น) ต้องไล่พิจารณาเลขโดดทีละตัวใน `i` ไล่ตั้งแต่หลักหน่วยไปเรื่อย ๆ นำสตริงของผลการแปลงทั้งเลขโดดและเลขหลักที่ได้มาต่อทางซ้าย ดังตัวอย่างในตารางที่ 6-1 ต้องการแปลง `i=6400321` ในแต่ละรอบเราดึงหลักหน่วยของ `i` (ด้วย `i%10`) มาเก็บใน `digit` แล้วส่งไปแปลงด้วย `digit2text` จากนั้นตัดหลักหน่วยทิ้ง (ด้วย `i = i/10`) และใช้ตัวแปร `pos` เก็บเลขหลักเริ่มที่ 0 เพิ่มขึ้นทีละหนึ่งในแต่ละรอบ เพื่อส่งไปแปลงด้วย `pos2text` คอลัมน์ขวาสุดแสดงผลลัพธ์ที่ได้จากการนำข้อความของเลขโดดและเลขหลักมาต่อทางซ้ายของผลลัพธ์ สังเกตว่าข้อความสุดท้ายที่ได้คือ **หกล้านสี่แสนสามร้อยสองสิบหนึ่งหน่วย** ซึ่ง

ยังไม่ถูกต้อง ต้องเปลี่ยน **สองสิบ** เป็น **ยี่สิบ** และ **หนึ่งหน่วย** เป็น **เอ็ด** ก็จะได้ข้อความสุดท้ายที่ถูกต้อง (และต้องเปลี่ยน **หนึ่งสิบ** เป็น **สิบ** ด้วย และลบ **หน่วย** ออกถ้ามี)

ตารางที่ 6-1 ตัวอย่างการทำงานของเมท็อดการแปลงจำนวนเต็มเป็นข้อความ

i	digit	digit2text	pos	pos2text	ข้อความผลลัพธ์
6400321					
640032	1	หนึ่ง	0	หน่วย	หนึ่งหน่วย
64003	2	สอง	1	สิบ	สองสิบหนึ่งหน่วย
6400	3	สาม	2	ร้อย	สามร้อยสองสิบหนึ่งหน่วย
640	0		3		สามร้อยสองสิบหนึ่งหน่วย
64	0		4		สามร้อยสองสิบหนึ่งหน่วย
6	4	สี่	5	แสน	สี่แสนสามร้อยสองสิบหนึ่งหน่วย
0	6	หก	6	ล้าน	หกล้านสี่แสนสามร้อยสองสิบหนึ่งหน่วย

รหัสที่ 6-10 แสดงเมท็อดแปลงจำนวนเต็มเป็นข้อความตามขั้นตอนที่ได้บรรยายตอนต้น ด้วยวงวน `for` มี `pos` เป็นตัวนับ จะวนทำใน `for` ตราบเท่าที่ `i` ยังมากกว่า `0` โดยจะสนใจแปลงเลขโดดและเลขหลักเมื่อเลขโดดที่พิจารณามีค่ามากกว่า `0` หลังจากวนทำทุกหลักแล้ว ให้ใช้เมท็อด `replace` ของสตริง (ยังไม่เคยนำเสนอเมท็อดของสตริงนี้มาก่อน) เพื่อแทนสตริงย่อยจาก**สองสิบ** เป็น**ยี่สิบ** จาก**หนึ่งสิบ**เป็น**สิบ** จาก**หนึ่งหน่วย**เป็น**เอ็ด** และลบ**หน่วย**ทิ้ง เช่น `t.replace("สองสิบ", "ยี่สิบ")` คืนสตริงใหม่ที่เหมือน `t` แต่จะแทนทุกสตริงย่อยที่เป็น `"สองสิบ"` ด้วย `"ยี่สิบ"` เมท็อด `int2text` มีข้อจำกัดหลายประการตามข้อกำหนด ขอละไว้เป็นแบบฝึกหัดให้ผู้อ่านปรับปรุงเพื่อให้สามารถแปลงทุก ๆ ค่าของ `int` ได้ เช่น แปลง –1311480221 เป็น **ลบหนึ่งพันสามร้อยสิบเอ็ดล้านสี่แสนแปดหมื่นสองร้อยยี่สิบเอ็ด** ได้

```
// เมท็อดแปลงจำนวนเต็มเป็นข้อความ จำกัดให้จำนวนที่แปลงมีค่าได้ตั้งแต่ 0 ถึง 9,999,999
public static String int2text(int i) {
  String t = "";
  if (i < 10) return digit2text(i);
  for (int pos = 0; i > 0; pos=pos+1) {
    int digit = i % 10;
    if (digit > 0) t = digit2text(digit) + pos2text(pos) + t;
    i = i / 10;
  }
  t = t.replace("สองสิบ", "ยี่สิบ");
  t = t.replace("หนึ่งสิบ", "สิบ");
  t = t.replace("หนึ่งหน่วย", "เอ็ด");
  t = t.replace("หน่วย", "");  // เช่น หนึ่งร้อยสองหน่วย → หนึ่งร้อยสอง
  return t;
}
```

เขียนเป็นกรณีพิเศษสำหรับเลขหลักเดียว

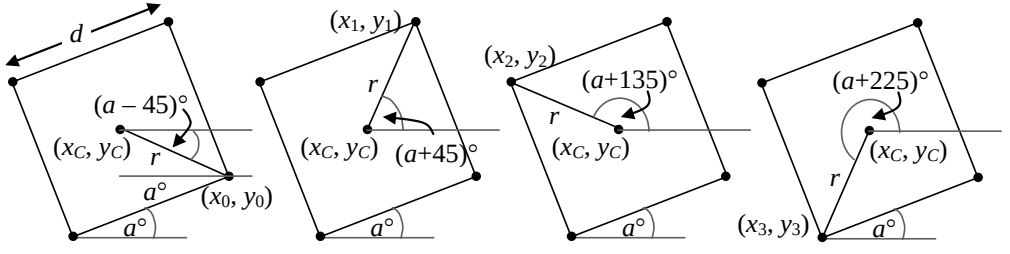
เรียกใช้เมท็อดเพื่อเปลี่ยนเลขโดด และเลขหลักเป็นคำมาต่อด้านหน้าของผลลัพธ์

รหัสที่ 6-10 เมท็อดแปลงจำนวนเต็มเป็นข้อความ

การวาดรูปหลายเหลี่ยมด้านเท่า

คลาส `DWindow` มีเมท็อด `drawEllipse`, `fillEllipse` และ `drawLine` เพื่อวาดวงรีและลากเส้นตรง ในกรณีที่ต้องการวาดสี่เหลี่ยม คลาสนี้มีเมท็อด `drawRect` ให้ใช้ แต่วาดได้เฉพาะสี่เหลี่ยมที่มีด้านขนานกับแนวนอนหรือแนวตั้งเท่านั้น คลาสนี้ไม่มีเมท็อดให้เรียกเพื่อวาดรูปสี่เหลี่ยมเอียงๆ หากต้องการ ก็ต้องลากเส้นตรงต่อเอง ขอเริ่มด้วยการเขียนเมท็อดเพื่อวาดรูปสี่เหลี่ยมจัตุรัสที่เอียงได้บนวินโดว์ `w` ด้วยการกำหนดจุดตรงกลางสี่เหลี่ยม `xc`, `yc`, ความยาวด้าน `d` และมุม `a°` ที่สี่เหลี่ยมนี้หมุนทวนเข็มโดยใช้จุด (`xc`, `yc`) เป็นจุดหมุน เขียนเป็นหัวเมท็อดดังนี้

```
public static void drawSquare(DWindow w, int xc, int yc, int d, int a)
```



รูปที่ 6-3 การคำนวณพิกัดมุมของสี่เหลี่ยมจัตุรัสที่หมุนทวนเข็ม $a°$

พิจารณารูปที่ 6-3 ก่อนจะวาดรูปสี่เหลี่ยมจัตุรัสที่หมุนทวนเข็มเป็นมุม $a°$ เราต้องหาความยาว r ของเส้นที่ลากจากจุด (x_C, y_C) ไปยังมุมของสี่เหลี่ยม จากรูปจะได้ $r\cos(45°) = d/2$ ดังนั้น $r = d / 2\cos(45°)$ เมื่อรู้ความยาว r ก็สามารถหาพิกัดของมุมทั้งสี่ ด้วยการหมุนเส้น r ไปยังมุมต่างๆ กำหนดให้เส้น r มีจุดหมุนที่ (x_C, y_C) เมื่อปลายของเส้น r อยู่ที่ (x_0, y_0) เส้น r ทำมุมกับแกนนอน $(a – 45)°$ (ดูรูปที่ 6-3 ซ้ายสุด) เมื่อหมุนเส้น r ทวนเข็มไปอีกทีละ 90° จะได้ปลายของเส้น r อยู่ที่มุมของสี่เหลี่ยมดังแสดงในรูป ดังนั้น สามารถคำนวณพิกัดของมุมทั้งสี่ได้ดังนี้ [7]

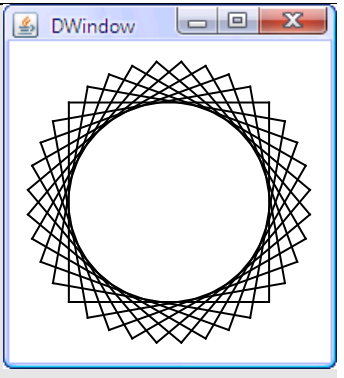
$$x_0 = x_C + r\cdot\cos(a - 45), \qquad y_0 = y_C - r\cdot\sin(a - 45), \qquad r = d / 2\cos(45°)$$
$$x_1 = x_C + r\cdot\cos(a + 45), \qquad y_1 = y_C - r\cdot\sin(a + 45)$$
$$x_2 = x_C + r\cdot\cos(a + 135), \qquad y_2 = y_C - r\cdot\sin(a + 135)$$
$$x_3 = x_C + r\cdot\cos(a + 225), \qquad y_3 = y_C - r\cdot\sin(a + 225)$$

เมื่อรู้วิธีการคำนวณพิกัดของมุมทั้งสี่ ก็สามารถเขียนเป็นเมท็อดได้ดังรหัสที่ 6-11 โดยใช้เมท็อดเสริม `cos` และ `sin` ที่รับพารามิเตอร์เป็นมุมมีหน่วยเป็นองศา (ซึ่งจะแปลงเป็นเรเดียนแล้วเรียกเมท็อดของคลาส `Math`) มีเมท็อด `main` เพื่อทดสอบการทำงานโดยวาดสี่เหลี่ยมจัตุรัสจำนวน 9 รูปซ้อนกันหมุนทวนเข็มเป็นมุม 0°, 10°, 20°, ..., 80° (ถ้าหมุนเป็นมุม 90° จะได้รูปเดียวกับ 0°)

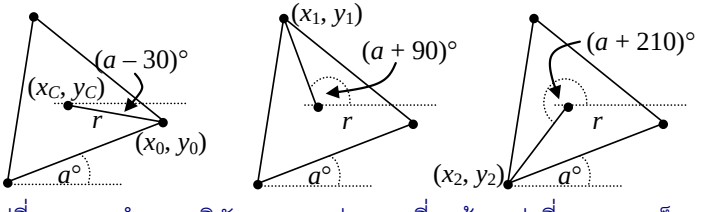
---

[7] ขอทวนความจำเรื่องพิกัดบนวินโดว์ว่า ค่า x จะเพิ่มขึ้นเมื่อไปทางขวา และค่า y จะเพิ่มขึ้นเมื่อลงด้านล่าง โดยหัวมุมซ้ายบนของพื้นที่ที่วาดได้ในวินโดว์คือพิกัด (0,0) จึงเป็นที่มาของเครื่องหมายลบในสูตรหาค่า y

```
01 import jlab.graphics.DWindow;
02
03 public class Drawing {
04   public static void main(String[] args) {
05     DWindow w = new DWindow(200, 200);
06     for (int a = 0; a < 90; a = a + 10) {
07       drawSquare(w, 100, 100, 120, a);
08     }
09   }
10   public static double sin(double a) {
11     return Math.sin(Math.toRadians(a));
12   }
13   public static double cos(double a) {
14     return Math.cos(Math.toRadians(a));
15   }
16   public static void drawSquare(DWindow w, int xc, int yc,
17                                 int d, int a) {
18     double r = d / (2 * cos(45));
19     double angle = a - 45;
20     double x0 = xc + r * cos(angle);
21     double y0 = yc - r * sin(angle);
22     for (int i = 0; i < 4; i++) {
23       angle = angle + 90;
24       double x1 = xc + r * cos(angle);
25       double y1 = yc - r * sin(angle);
26       w.drawLine(x0, y0, x1, y1);
27       x0 = x1; y0 = y1;
28     }
29   }
```



รหัสที่ 6-11 เมท็อดวาดรูปสี่เหลี่ยมจัตุรัส



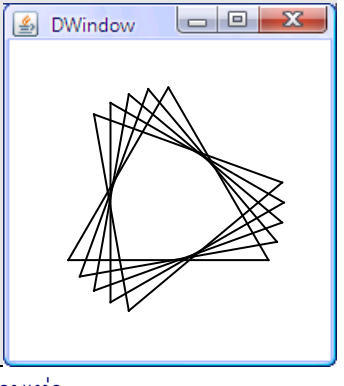
รูปที่ 6-4 การคำนวณพิกัดมุมของรูปสามเหลี่ยมด้านเท่าที่หมุนทวนเข็ม $a°$

ถ้าเปลี่ยนมาเขียนเป็นเมท็อดวาดสามเหลี่ยมด้านเท่าในลักษณะเดียวกัน จะมีสูตรการคำนวณพิกัดของมุมดังรูปที่ 6-4 ให้ (x_C, y_C) คือจุดที่เกิดจากการตัดกันของเส้นแบ่งครึ่งมุมทั้งสาม (ซึ่งคือจุดศูนย์กลางของวงกลมล้อมสามเหลี่ยม) ให้ r คือความยาวของเส้นที่ลากจากจุด (x_C, y_C) ไปยังมุมของสามเหลี่ยม เส้นนี้ยาว $d / 2\cos(30°)$ โดยที่ d คือความยาวด้านของสามเหลี่ยม ดูรูปที่ 6-4 ซ้ายสุดพบว่า เมื่อปลายของเส้น r อยู่ที่ (x_0, y_0) เส้น r ทำมุมกับแกนนอน $(a - 30)°$ เมื่อหมุนเส้น r ทวนเข็มไปอีกทีละ 120° จะได้ปลายของเส้น r อยู่ที่มุมของสามเหลี่ยมดังแสดงในรูป ดังนั้นสามารถคำนวณพิกัดของมุมทั้งสามได้ดังนี้

$$x_0 = x_C + r\cdot\cos(a-30), \qquad y_0 = y_C - r\cdot\sin(a-30), \quad r = d / 2\cos(30°)$$
$$x_1 = x_C + r\cdot\cos(a+90), \qquad y_1 = y_C - r\cdot\sin(a+90)$$
$$x_2 = x_C + r\cdot\cos(a+210), \qquad y_2 = y_C - r\cdot\sin(a+210)$$

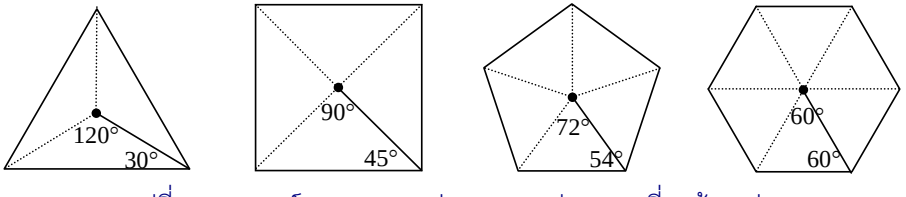
รหัสที่ 6-12 แสดงเมท็อดวาดรูปสามเหลี่ยมด้านเท่าซึ่งมีกระบวนการทำงานเหมือนการวาดรูปสี่เหลี่ยมจัตุรัสในรหัสที่ 6-11 พร้อมกับรูปที่ได้เมื่อวาดสามเหลี่ยมด้านเท่าจำนวน 6 รูปซ้อนกัน หมุนทวนเข็มเป็นมุม 0°, 10°, ..., 50° (ถ้าวาดต่ออีก 6 รูป จะได้รูปครบวงพอดี)

```
30   public static void drawEquilateralTriangle(DWindow w,
31                            int xc, int yc, int d, int a) {
32     double r = d / (2 * cos(30));
33     double angle = a - 30;
34     double x0 = xc + r * cos(angle);
35     double y0 = yc - r * sin(angle);
36     for (int i = 0; i < 3; i++) {
37       angle = angle + 120;
38       double x1 = xc + r * cos(angle);
39       double y1 = yc - r * sin(angle);
40       w.drawLine(x0, y0, x1, y1);
41       x0 = x1; y0 = y1;
42     }
43   }
```



รหัสที่ 6-12 เมท็อดวาดรูปสามเหลี่ยมด้านเท่า

นักเขียนโปรแกรมทั่วไปเมื่อเห็นความคล้ายกันของการวาดสี่เหลี่ยมจัตุรัสในรหัสที่ 6-11 และการวาดสามเหลี่ยมด้านเท่าในรหัสที่ 6-12 ที่ต่างกันแค่ค่าคงตัวต่าง ๆ ที่ใช้ในการคำนวณและจำนวนด้าน ก็อดไม่ได้ที่จะเขียนเป็นเมท็อดใหม่ที่ทำงานได้สารพัดประโยชน์มากขึ้น ในกรณีนี้ก็คือเขียนเป็นเมท็อดเพื่อวาดรูปหลายเหลี่ยมปกติ (ด้านเท่ากัน มุมภายในเท่ากัน) โดยรับพารามิเตอร์เพิ่มอีกตัวที่ระบุจำนวนด้าน สังเกตว่า ถ้าเป็นสามเหลี่ยม จุดแรกคำนวณได้จากมุม $(a-30)°$ เพิ่มมุม (หมุนทวนเข็ม) ครั้งละ 120°, ถ้าเป็นสี่เหลี่ยม จุดแรกคำนวณได้จากมุม $(a-45)°$ เพิ่มมุมครั้งละ 90° หากพิจารณารูปที่ 6-5 ประกอบจะพบว่า ถ้าเป็นห้าเหลี่ยม จุดแรกคำนวณได้จากมุม $(a-54)°$ เพิ่มมุมครั้งละ 72° สรุปว่า ถ้าเป็น n เหลี่ยม จุดแรกย่อมคำนวณจากมุม $a-(180-360/n)/2$ เพิ่มมุมครั้งละ $(360/n)°$ เขียนได้ดังรหัสที่ 6-13



รูปที่ 6-5 จุดศูนย์กลางและมุมต่าง ๆ ของรูปหลายเหลี่ยมด้านเท่า

```
44   public static void drawRegularPolygon(DWindow w,
45                           int xc, int yc, int d, int a, int n) {
46     double b = (180 - 360.0/n) / 2;
47     double r = d / (2 * cos(b));
48     double angle = a - b;
49     double x0 = xc + r * cos(angle);
50     double y0 = yc - r * sin(angle);
51     for (int i = 0; i < n; i++) {
52       angle = angle + 360.0/n;
53       double x1 = xc + r * cos(angle);
54       double y1 = yc - r * sin(angle);
55       w.drawLine(x0, y0, x1, y1);
56       x0 = x1; y0 = y1;
57     }
58   }
59 }
```

เพิ่มพารามิเตอร์ระบุจำนวนด้าน, เช่น n=3 คือสามเหลี่ยมด้านเท่า

รหัสที่ 6-13 เมท็อดวาดรูปหลายเหลี่ยมปกติ

การทดสอบความถูกต้องของพารามิเตอร์

ข้อกำหนดของเมท็อด `int2text` ที่เราได้เขียนในรหัสที่ 6-10 กำหนดไว้ว่าจะทำงานถูกต้องเมื่อพารามิเตอร์ที่รับมาเป็นจำนวนเต็มบวกที่ต่ำกว่าสิบล้าน หากมีผู้เรียกเมท็อดนี้แล้วส่งค่าที่ไม่ตรงตามข้อกำหนด ก็จะทำงานผิด เช่น หากเรียก `int2text(116000000)` จะได้ "หนึ่ง???หนึ่ง???หกล้าน" คืนกลับมา คำถามที่น่าสนใจคือ เราในฐานะผู้เขียนเมท็อด ได้เขียนข้อกำหนดไว้แล้ว แต่ก็ยังมีผู้ไม่ปฏิบัติตาม ด้วยอาจเป็นเพราะผู้นั้นไม่ได้อ่านข้อกำหนด หรือเป็นเพราะโปรแกรมของผู้เรียกมีข้อผิดพลาด ถ้าเราคืนสตริงที่ผิดๆ ไปเช่นนี้ จะมีผลต่อระบบงานเช่นไร หากผู้เรียกนำผลไปเก็บในฐานข้อมูล หรือส่งไปตามเครือข่ายให้ผู้อื่น ผลที่ผิดๆ นี้อาจไม่ถูกค้นพบในทันทีที่มีการเรียกใช้ที่ผิดข้อกำหนด ทางที่ดีเมท็อดของเราควรแจ้งให้ผู้เรียกทราบทันทีว่ามีข้อผิดพลาดเกิดขึ้น เพื่อให้นักเขียนโปรแกรมทราบและแก้ไข ดีกว่าปล่อยให้ข้อผิดพลาดที่เกิดขึ้นถูกหมกเม็ดไว้ไม่มีใครรู้ แล้วเราจะทำให้ผู้เรียกเมท็อดเราที่ส่งข้อมูลผิดๆ ทราบได้อย่างไร จาวามีคำสั่งให้เราสร้างสิ่งผิดปกติ และโยนสิ่งผิดปกตินี้ให้ผู้เรียกเมท็อดของเรา ดังตัวอย่างในรหัสที่ 6-14

```
// เมท็อดแปลงจำนวนเต็มเป็นข้อความ จำกัดให้จำนวนที่แปลงมีค่าได้ตั้งแต่ 0 ถึง 9,999,999
public static String int2text(int i) {
  if (i < 0 || i > 9999999) {
    throw new IllegalArgumentException("ค่า " + i + " อยู่นอกช่วง");
  }
  String t = "";
  ... // กลุ่มคำสั่งเดิมที่แปลง i เป็นข้อความเก็บใน t
}
```

สร้างสิ่งผิดปกติ และโยนให้ผู้เรียกเมท็อดทราบ

รหัสที่ 6-14 การโยนสิ่งผิดปกติให้กับผู้เรียกเมท็อด

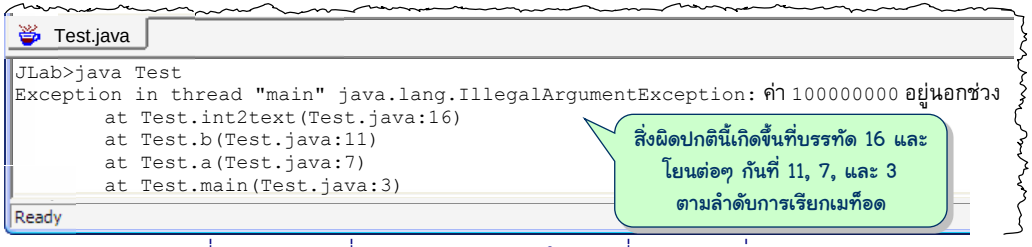
คำสั่ง `throw new IllegalArgumentException()` ที่ต้นเมท็อดในรหัสที่ 6-14 สั่งให้สร้าง (`new`) และโยน (`throw`) [8] สิ่งผิดปกติแบบ `IllegalArgumentException` ซึ่งเป็นแบบที่ใช้กันเพื่อบอกว่า ผู้เรียกส่งข้อมูลที่ผิดกฎมาให้ [9] เมื่อโยนแล้วเกิดอะไรขึ้น ขอใช้รหัสที่ 6-15 เป็นตัวอย่าง เมื่อให้โปรแกรมเริ่มทำงาน ก็ไปทำ `main`, บรรทัดที่ 3 เรียก `a()`, ก็ไปทำบรรทัดที่ 7 เรียก `b()`, ก็ไปทำบรรทัดที่ 11 เรียก `int2text` แต่ส่งหนึ่งร้อยล้านไปให้ ทำให้การตรวจสอบใน `int2text` เป็นจริงเกิดการสร้างและโยนสิ่งผิดปกติ เมื่อมีการโยนเช่นนี้ การทำงานของ `int2text` จะสิ้นสุดลงโดยอัตโนมัติ จากนั้นสิ่งผิดปกติจะถูกโยนกลับไปสู่ผู้เรียกคือ `b()` ซึ่งไม่มีคำสั่งเพื่อจัดการกับสิ่งผิดปกติ (วิธีการจัดการสิ่งผิดปกติจะอธิบายในบทที่ 10 ก่อนถึงบทนั้นเราจะไม่มีการจัดการสิ่งผิดปกติที่เมท็อดใดๆ เลย) การทำงานของ `b()` ก็สิ้นสุดลงเช่นกัน สิ่งผิดปกติที่ได้รับมาจาก `int2text` ก็จะถูกโยนต่อ (ระบบเป็นผู้โยนต่อ) กลับไปให้ผู้เรียกอีก ในตัวอย่างนี้ก็คือ `a()` และอีกเช่นกัน `a()` เมื่อได้รับสิ่งผิดปกติที่โยนกลับมา การทำงานของ `a()` ก็สิ้นสุดลง และสิ่งผิดปกตินี้ก็ถูกโยนต่อขึ้นไปอีก กลับมาที่ `main` คราวนี้จะถูกโยนกลับสู่ระบบ เป็นการสิ้นสุดการทำงานของโปรแกรม ซึ่งจะแสดงสิ่งผิดปกติที่ระบบได้รับทางจอภาพ ดังรูปที่ 6-6 ให้สังเกตข้อความที่ปรากฏในรูป แสดงให้เห็นว่าสิ่งผิดปกติถูกโยนที่บรรทัด 16 ตามด้วยบรรทัดที่ 11, 7, และ 3 ตามลำดับ ซึ่งมาจากลำดับการโยนสิ่งผิดปกติต่อเป็นทอด ๆ โดยสตริงที่เราเขียนว่าค่า `i` อยู่นอกช่วงตอนโยน (บรรทัดที่ 16) จะปรากฏให้เห็นทางจอด้วย

```
01 public class Test {
02   public static void main(String[] args) {
03     a();  <-- เรียก a แล้วเกิดการโยนสิ่งผิดปกติ ก็โยนต่อ (4)
04     System.out.println("----a----");
05   }
06   public static void a() {
07     b();  <-- เรียก b แล้วเกิดการโยนสิ่งผิดปกติ ก็โยนต่อ (3)
08     System.out.println("----b----");
09   }
10   public static void b() {
11     String t = int2text(100000000);  <-- เรียก int2text แล้วเกิดการโยนสิ่งผิดปกติ ก็โยนต่อ (2)
12     System.out.println(t);
13   }
14   public static int2text(int i) {
15     if (i < 0 || i > 9999999) {  <-- สร้างและโยนสิ่งผิดปกติที่นี่ (1)
16       throw new IllegalArgumentException("ค่า " + i + " อยู่นอกช่วง");
17     }
..     ...
```

รหัสที่ 6-15 ผลของการเรียกเมท็อดที่โยนสิ่งผิดปกติ

[8] ให้สังเกตว่า `throw` นี้ไม่ใช่ `throws` (มีตัว `s`) ที่เราเคยเขียนไว้ที่หัวเมท็อด `main` ที่มีการเปิดแฟ้มข้อมูล

[9] เราเรียกค่าที่ส่งมาให้พารามิเตอร์ว่า argument, ส่วนคำว่า illegal แปลว่าผิดกฎ, exception แปลว่าสิ่งผิดปกติ `IllegalArgumentException` เป็นชื่อคลาสมาตรฐานของจาวาที่ผู้ออกแบบตั้งใจตั้งชื่อนี้ เพื่อสื่อความหมายถึงสิ่งผิดปกติแบบนี้ เราจะได้ศึกษาอย่างละเอียดในบทท้ายๆ



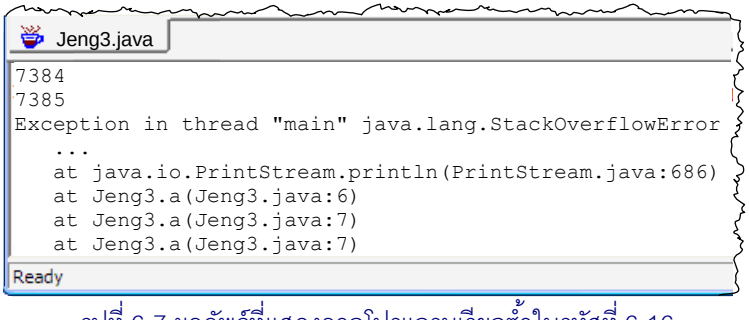
รูปที่ 6-6 ผลลัพธ์ที่แสดงจากโปรแกรมในรหัสที่ 6-15 โยนสิ่งผิดปกติ

เมท็อดแบบเรียกซ้ำ

การทำงานภายในเมท็อดหนึ่งสามารถเรียกใช้เมท็อดอื่นได้ และภายในเมท็อดอื่นที่ถูกเรียกก็อาจเรียกใช้เมท็อดอื่นต่อไปได้อีกเป็นทอดๆ ส่งผลให้เมท็อดที่เคยเขียน ถูกเรียกใช้แล้วใช้อีก จะเรียกใช้เมท็อดใดที่ใดก็ได้ จึงไม่ใช่เรื่องแปลกที่จะเรียกเมท็อดต่อกันเป็นทอดๆ ไปเรื่อยๆ แล้วอาจวนกลับมาเรียกเมท็อดเดิมตอนต้นๆ เช่น ภายในเมท็อด `a` มีการเรียกเมท็อด `b` และภายใน `b` มีการเรียกกลับมาที่ `a` หรือว่าภายในเมท็อด `a` มีการเรียกใช้ตัวเอง ก็เป็นไปได้ การทำงานในลักษณะเช่นนี้เรียกว่า *การเรียกซ้ำ* (recursion) ซึ่งมีผลคล้ายกับการทำงานแบบวงวน แต่ต่างกันตรงที่อาจเกิดข้อผิดพลาดได้ ถ้าใช้อย่างไม่ระมัดระวัง เช่น หากผู้อ่านสั่งโปรแกรมในรหัสที่ 6-16 ทำงาน โปรแกรมจะแสดง 0, 1, 2, ..., ไปสักพักแล้วจะเกิดข้อผิดพลาดดังรูปที่ 6-7

```
01 public class Jeng3 {
02   public static void main(String[] args) {
03     a(0);
04   }
05   private static void a(int i) {
06     System.out.println(i);
07     a(i + 1);
08   }
09 }
```

รหัสที่ 6-16 ตัวอย่างโปรแกรมใช้เมท็อดแบบเรียกซ้ำที่มีปัญหา



รูปที่ 6-7 ผลลัพธ์ที่แสดงจากโปรแกรมเรียกซ้ำในรหัสที่ 6-16

มาลองติดตามการทำงานของโปรแกรมในรหัสที่ 6-16 เริ่มที่ main เรียก a(0) เป็นการส่ง 0 ไปให้ i บรรทัดที่ 6 จึงแสดง 0 แล้วเรียก a(1) ก็ไปทำที่บรรทัดที่ 6 อีก แสดง 1 แล้วเรียก a(2) ทำเช่นนี้ไปเรื่อยๆ ต้องทวนอีกครั้งว่า ทุกครั้งที่เมท็อดหนึ่งถูกเรียก ระบบจะจัดเนื้อที่ในหน่วยความจำเพื่อเป็นที่เก็บข้อมูลให้กับตัวแปรเฉพาะที่และตัวแปรที่เป็นพารามิเตอร์ของเมท็อด ดังนั้น การเรียก a จาก main ในครั้งแรก จะมีการจองหน่วยความจำของตัวแปร i ไว้ที่หนึ่ง การเรียก a จาก a ในครั้งที่สองก็จะจองหน่วยความจำของตัวแปร i อีกที่เก็บหนึ่ง ดังนั้น การเรียก a แต่ละครั้ง ระบบจะต้องเตรียมหน่วยความจำให้กับตัวแปร i ตัวใหม่เสมอ (จะเลิกใช้และคืนเนื้อที่ของตัวแปรนี้ ก็เมื่อเมท็อดทำงานเสร็จ) เนื่องจากโปรแกรมนี้มีการเรียก a ไปเรื่อยๆ และไม่มีทีท่าว่าจะเลิก เพราะจะเลิกได้ต้องทำบรรทัดที่ 8 แต่จะถึงบรรทัดที่ 8 ต้องผ่านบรรทัดที่ 7 ซึ่งคือการเรียก a ดังนั้น โปรแกรมนี้จึงพิมพ์ค่า 0, 1, 2, ... เพิ่มขึ้นเรื่อยๆ ทีละหนึ่ง แต่ปัญหามันอยู่ตรงที่เนื้อที่ในหน่วยความจำก็ถูกจัดสรรให้กับตัวแปร i ตัวใหม่เพิ่มขึ้นเรื่อยๆ ด้วย เนื่องจากระบบการทำงานของจาวาได้จำกัดปริมาณเนื้อที่หน่วยความจำที่ใช้ในลักษณะนี้ เมื่อเกินขีดจำกัดของปริมาณหน่วยความจำที่ระบบตั้งไว้ ระบบจะโยนข้อผิดพลาดแบบที่มีชื่อว่า StackOverflowError ออกมา (หน่วยความจำที่จัดเก็บตัวแปรเฉพาะที่และพารามิเตอร์ระหว่างการเรียกเมท็อดนี้มีชื่อว่า stack การใช้เนื้อที่ที่ล้นเกินความจุคือ stack overflow จึงเป็นที่มาของชื่อข้อผิดพลาดดังกล่าว)

ถ้าเราเขียนเมท็อด b คล้าย a ที่เขียนมา ไม่มีการเรียกซ้ำใน b แต่ใช้วงวนไม่รู้จบใน main วนเรียก b ไปเรื่อยๆ ดังรหัสที่ 6-17 จะพบว่าโปรแกรมจะแสดง 0,1,2,... ไปเรื่อย ๆ ทำงานไม่รู้จบ และไม่มีข้อผิดพลาดเรื่องหน่วยความจำแต่อย่างใด ทั้งนี้เพราะถึงแม้จะมีการจัดสรรหน่วยความจำให้ตัวแปร i ใหม่ทุกครั้งที่เรียก b แต่หน่วยความจำนี้ก็จะคืนกลับให้ระบบทุกครั้งที่ b ทำงานเสร็จ

```
01 public class MaiJeng3 {
02   public static void main(String[] args) {
03     int i = 0;
04     while (true) {
05       b(i);
06       i++;
07     }
08   }
09   private static void b(int i) {
10     System.out.println(i);
11   }
12 }
```

เมท็อดเริ่มทำงาน : มีการจองหน่วยความจำให้ตัวแปร i

เมท็อดเลิกทำงาน : คืนหน่วยความจำของตัวแปร i

รหัสที่ 6-17 ตัวอย่างโปรแกรมใช้วงวนเรียกเมท็อดซ้ำๆ

ถึงตรงนี้ผู้อ่านคงสงสัยว่า แล้วมีเหตุผลอันใดต้องมาเขียนเมท็อดแบบเรียกตัวเองซ้ำๆ ด้วย ต้องขอตอบว่า การเขียนโปรแกรมแบบเรียกซ้ำในบางโอกาสจะได้โปรแกรมที่สั้น เขียนบรรยายโปรแกรมแบบตรงไปตรงมาตามนิยามทางคณิตศาสตร์ (ช่วยให้มั่นใจในความถูกต้องของโปรแกรมที่เขียน) นอกจากนี้ยังได้โปรแกรมที่สวย (ประเด็นนี้ขึ้นกับรสนิยม) อ่านเข้าใจง่าย ขอเริ่มนำเสนอ

ด้วยตัวอย่างแบบง่ายก่อน ซึ่งเขียนแบบวงวนธรรมดาอาจง่ายกว่า ตามด้วยตัวอย่างที่ซับซ้อนขึ้น เพื่อให้เห็นแนวทางและสร้างความคุ้นเคยกับการเขียนโปรแกรมแบบเรียกซ้ำ

ผลบวก 1 + 2 +... + n

ผู้อ่านที่ได้ฝ่าฟันการเขียนโปรแกรมมาถึงจุดนี้แล้ว คาดว่าคงสามารถเขียนเมท็อดหาผลบวก `int sum(int n)` เพื่อคืนผลบวก 1+2+...+n ได้ไม่ยาก ในกรณีที่เราไม่รู้สูตรสำเร็จของการหาผลบวกนี้ (ซึ่งคือ $n(n+1)/2$) ก็คงเขียนเมท็อดด้วยการใช้วงวน (จะเป็นวงวนแบบใดก็ได้ แต่สำหรับผู้เขียนเอง คงเลือกใช้วงวน `for`) คราวนี้เราจะมาเขียนเมท็อดแบบเรียกซ้ำ ให้ S_n คือ 1+2+...+n หากเราเขียน S_{n-1} ก็ย่อมแทน 1+2+...+$(n-1)$ สรุปได้ว่า $S_n = S_{n-1} + n$ เมื่อ $n > 0$ และ $S_0 = 0$ ด้วยนิยามที่เขียนนี้สามารถเขียนเป็นเมท็อดแบบเรียกซ้ำสั้น ๆ ได้ดังรหัสที่ 6-18

```
public static int sum(int n) {
  if (n == 0) return 0;      // sum(0) มีค่า 0
  int s = sum(n-1) + n;
  return s;                  // sum(n) หาได้จาก sum(n-1) + n
}
```

รหัสที่ 6-18 เมท็อดหาผลบวก 1+2+...+n (แบบเรียกซ้ำ) โดยที่ $S_n = S_{n-1} + n$ เมื่อ $n > 0$ และ $S_0 = 0$

หลักการสำคัญของการทำงานแบบเรียกซ้ำคือ ถ้าต้องการแก้ไขปัญหา P ที่ "ใหญ่" ให้แก้ไขปัญหาย่อยของ P ที่มีขนาด "เล็ก" กว่า แล้วจึงนำคำตอบของปัญหาย่อยๆ นั้นมาหาคำตอบของปัญหา P สิ่งที่ต้องคิดก่อนคือ อะไรเป็นตัวกำหนดขนาดของปัญหา ปัญหาใหญ่สะท้อนให้เห็นว่าต้องใช้เวลาคำนวณมากกว่าปัญหาเล็ก ในกรณีของ `sum(n)` ขนาดก็คือค่า `n` มีค่ามากก็ต้องบวกหลายตัวกว่ากรณี `n` มีค่าน้อย ความสัมพันธ์ $S_n = S_{n-1} + n$ ชี้ให้เห็นว่า เราสามารถหา `sum(n)` ได้จากการหาคำตอบของ `sum(n-1)` ซึ่งมีขนาดของปัญหาที่เล็กกว่า และเนื่องจากปัญหามีขนาดเล็กลง ๆ ทุกครั้งที่เรียกซ้ำ จึงต้องคอยตรวจสอบว่า ปัญหามีขนาด "เล็กพอ" ที่เรารู้คำตอบโดยไม่จำเป็นต้องย่อยให้เล็กลงอีกหรือยัง ในกรณีของ `sum(n)` ที่เขียนในรหัสที่ 6-18 กำหนดให้เล็กพอเมื่อ `n=0` เพราะรู้ทันทีว่า `sum(0)` เท่ากับ 0 ถ้าจะกำหนดให้เล็กพอคือ `n=1` จะเขียนได้รหัสที่ 6-19 (ผู้อ่านเห็นข้อจำกัดของรหัสที่ 6-19 เมื่อเทียบกับรหัสที่ 6-18 หรือไม่ [10])

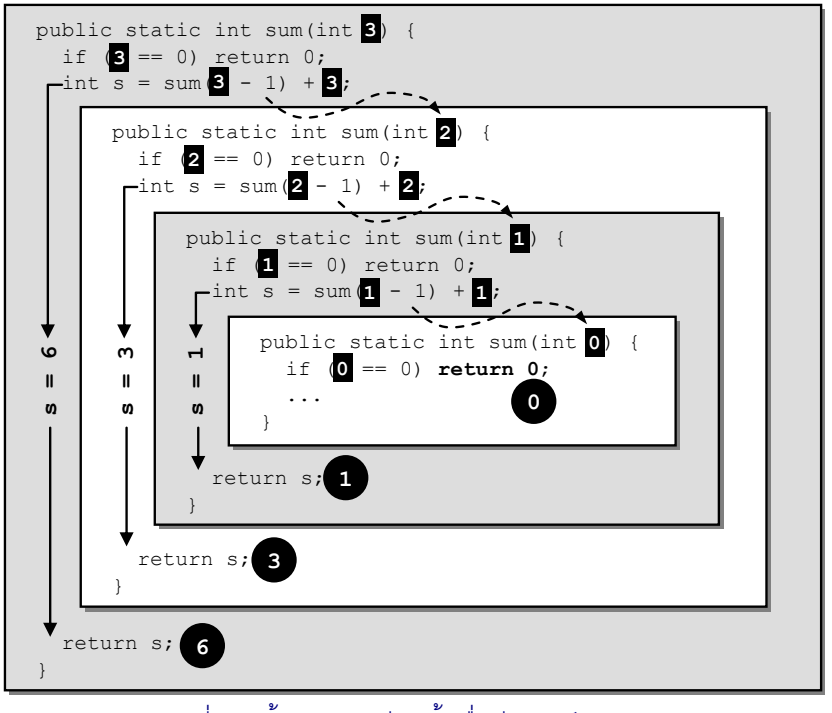
```
public static int sum(int n) {
  if (n == 1) return 1;      // sum(1) มีค่า 1
  int s = sum(n-1) + n;
  return s;                  // sum(n) หาได้จาก sum(n-1) + n
}
```

รหัสที่ 6-19 เมท็อดหาผลบวก 1+2+...+n (แบบเรียกซ้ำ) โดยที่ $S_n = S_{n-1} + n$ เมื่อ $n > 1$ และ $S_1 = 1$

---

10 [illegible] sum(0) [illegible] StackOverflowError ([illegible] n [illegible] n [illegible] 10000)

รูปที่ 6-8 แสดงการทำงานของรหัสที่ 6-18 เมื่อเรียก `sum(3)` ได้ `n` มีค่าเป็น 3 เกิดการเรียก `sum(3-1)` ระบบสร้าง `n` ตัวใหม่มีค่า 2 แล้วก็เรียก `sum(2-1)` ได้ `n` เพิ่มอีกตัวมีค่า 1 ตามด้วยการเรียก `sum(1-1)` ซึ่งก็เช่นกันได้ `n` ตัวใหม่มีค่า 0 ให้สังเกตว่า `n` แต่ละตัวที่เกิดขึ้นนั้นจะซ้อนๆ กัน ตัวใหม่เกิดก็ซ้อนทับตัวเก่า มองได้ว่าระบบจะสนใจ `n` ตัวบนสุด เมื่อเมท็อดทำงานเสร็จ ก็จะทิ้ง `n` ตัวปัจจุบัน กลับไปทำต่อ และไปใช้ `n` ตัวล่าสุดก่อนหน้านี้ เมื่อ `n=0` จะคืน 0 กลับไป (แล้ว `n` ตัวนี้ก็หายไป) เป็นผลของ `sum(1-1)` นำไปใช้ในบรรทัด `s=sum(1-1)+n` โดยที่ `n` ตัวนี้มีค่าเป็น 1 จึงได้ `s=0+1` คืน 1 กลับไปเป็นผลของ `sum(2-1)` นำไปใช้คำนวณ `s` (โดยตอนนี้ `n` มีค่าเป็น 2) ได้ `s=1+2` คืน 3 กลับไปเป็นผลของ `sum(3-1)` ตอนนี้ `n` มีค่าเป็น 3 จึงได้ `s=3+3` ได้ 6 คืนกลับไปเป็นผลของ `sum(3)` ที่เรียกไว้ครั้งแรก



รูปที่ 6-8 ขั้นตอนการเรียกซ้ำเมื่อเรียกเมท็อด sum(3)

จำนวนฟิโบนักชี

จำนวนฟิโบนักชี (Fibonacci number) คือจำนวนในลำดับ 0, 1, 1, 2, 3, 5, 8, 13, ... ให้ f_n คือจำนวนฟิโบนักชีตัวที่ n ในลำดับ จะได้ว่า $f_n = f_{n-1} + f_{n-2}$ เมื่อ $n \geq 2$ และ $f_0 = 0, f_1 = 1$ รหัสที่ 6-20 แสดงเมท็อด `fib(n)` ที่คืนจำนวนฟิโบนักชีตัวที่ `n` โดยใช้วงวน `for` หมุนคำนวณค่า `fk` จาก `fk1` และ `fk2` โดยให้ `fk` เก็บจำนวนฟิโบนักชีตัวที่ `k` ส่วน `fk1` และ `fk2` เก็บสองตัวก่อนหน้า วงวนนี้หมุนเริ่มค่า `k` ที่ 2 เพิ่มรอบละหนึ่ง เมื่อ `k>n` ก็ออกจากวงวนเพื่อคืนผล ก่อนจะขึ้น

รอบใหม่ ต้องย้ายค่าจาก fk ไปเก็บใน fk1 และย้ายค่าจาก fk1 ไปเก็บใน fk2 เนื่องจากวงวนนี้ไม่ครอบคลุมกรณี n เป็น 0 กับ 1 จึงเขียนทดสอบสองกรณีพิเศษนี้ก่อนเข้าทำงานในวงวน ส่วนบรรทัดแรกของเมท็อดตรวจสอบความถูกต้องของ n ว่าต้องไม่ติดลบ

```
public static int fib(int n) {
  if (n < 0) throw new IllegalArgumentException(n + " เป็นลบ");
  if (n == 0) return 0;
  if (n == 1) return 1;
  int fk2 = 0, fk1 = 1, fk = 1;
  for (int k=2; k<=n; k++) {
    fk = fk1 + fk2;
    fk2 = fk1;
    fk1 = fk;
  }
  return fk;
}
```

k	fk2	fk1	fk
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
..	..	..	..

รหัสที่ 6-20 เมท็อดหาจำนวนฟิโบนักชี (แบบวงวน)

นิยาม $f_n = f_{n-1} + f_{n-2}$ เมื่อ $n \geq 2$ และ $f_0 = 0, f_1 = 1$ บอกว่า เราสามารถนำคำตอบของปัญหาเล็ก (ซึ่งคือ f_{n-1} และ f_{n-2}) มาประกอบกันเป็นคำตอบของปัญหาที่ใหญ่ขึ้น (ซึ่งคือ f_n) หรือมองในมุมกลับคือ เราสามารถหาคำตอบของปัญหาใหญ่ด้วยการหาคำตอบของปัญหาเล็ก ลักษณะของปัญหาที่เป็นเช่นนี้ เหมาะที่จะแปลงเป็นเมท็อดแบบเรียกซ้ำ ดังรหัสที่ 6-21 เริ่มด้วยการตรวจสอบความถูกต้องของ n ตามด้วยกรณีเล็กๆ ของ n เรายุบรวมสอง if(n==0)return 0 กับ if(n==1) return 1 ในรหัสที่ 6-20 เป็น if(n<=1) return n หากเป็นกรณีที่ n เกิน 1 ก็ให้หา fib(n-1) กับ fib(n-2) นำผลที่ได้มาบวกกัน แล้วคืนค่ากลับไป

```
public static int fib(int n) {
  if (n < 0) throw new IllegalArgumentException(n + " เป็นลบ");
  if (n <= 1) return n;
  return fib(n-1) + fib(n-2);
}
```

$f_n = f_{n-1} + f_{n-2}$

รหัสที่ 6-21 เมท็อดหาจำนวนฟิโบนักชี (แบบเรียกซ้ำ)

พาลินโดรม (อีกครั้ง)

รหัส 5-10 ในบทที่แล้วคือโปรแกรมซึ่งรับคำทางแป้นพิมพ์ และตรวจสอบว่า คำนั้นเป็นพาลินโดรมหรือไม่ ปัญหาการตรวจสอบคำว่าเป็นพาลินโดรมหรือไม่สามารถแก้ไขได้ในลักษณะของการแก้ไขปัญหาใหญ่ด้วยปัญหาย่อย โดยนิยามว่า w เป็นพาลินโดรม ก็เมื่อตัวซ้ายสุดของ w เท่ากับตัวขวาสุดของ w และสตริงย่อยของ w ที่เหลือ (ไม่นับตัวซ้ายสุดและขวาสุด) ต้องเป็นพาลินโดรมด้วย และต้องอย่าลืมนิยามกรณีเล็กๆ ของปัญหาที่รู้คำตอบทันที ในที่นี้ก็คือกรณีที่ w เป็นสตริงว่าง หรือมีอักขระตัวเดียว ก็ถือว่าเป็นพาลินโดรม เขียนเป็นเมท็อดได้ดังรหัสที่ 6-22 เริ่มด้วย

การตรวจสอบว่า ถ้าสตริงที่ได้รับยาวน้อยกว่าสอง ก็ถือได้ว่าเป็นพาลินโดรม ถ้ายาวตั้งแต่สองตัวขึ้นไป ก็ให้ตรวจสอบตัวซ้ายสุดว่าเท่ากับตัวขวาสุดหรือไม่ ถ้าไม่เท่า คืนเท็จได้ทันที ถ้าเท่าก็เรียกซ้ำให้ตรวจสอบสตริงย่อยที่เหลือ ได้ผลอย่างไร ก็คืนผลนั้นกลับไป

```
public static boolean isPalindrome(String word) {
  int n = word.length();
  if (n < 2) return true;
  String c1 = word.substring(0, 1);
  String c2 = word.substring(n - 1, n);
  if (!c1.equalsIgnoreCase(c2)) return false;
  return isPalindrome(word.substring(1, n - 1));
}
```

เป็นพาลินโดรม ถ้าเป็นสตริงว่าง หรือมีตัวเดียว

ไม่ใช่พาลินโดรม ถ้าตัวซ้ายสุดไม่เท่ากับขวาสุด

ตรวจสอบส่วนที่เหลือ

รหัสที่ 6-22 เมท็อดการตรวจสอบพาลินโดรม (แบบเรียกซ้ำ)

ผู้อ่านคิดว่าหากปรับการตรวจสอบ `if(n<2)` ให้เป็น `if(n==0)` การทำงานจะยังถูกต้องหรือไม่ ถ้าผิด จะผิดที่บรรทัดใด [11]

การหาค่า $a^b \bmod m$

การหาค่าของ $a^b \bmod m$ เป็นหนึ่งในการคำนวณที่ต้องทำเพื่อเข้าและถอดรหัสลับข้อมูล (ไม่ขอลงในรายละเอียด ผู้อ่านที่สนใจสามารถค้นคำว่า RSA ด้วยกูเกิล) สามารถเขียนเป็นเมท็อดที่ทำงานแบบวงวนได้ไม่ยาก หรือเขียนแบบเรียกซ้ำก็ได้เพราะ $a^b \bmod m = a(a^{b-1} \bmod m) \bmod m$ (โดยกรณีเล็กคือ $a^0 \bmod m = 1$) ดังรหัสที่ 6-23

```
public static int powerMod(int a, int b, int m) {
  if (b == 0) return 1;
  return (a*powerMod(a, b-1, m)) % m;
}
```

รหัสที่ 6-23 เมท็อดคำนวณค่า $a^b \bmod m$ (มีปัญหาเมื่อ b มีค่ามาก)

ถ้าต้องการคำนวณ $2^{50001} \bmod 10$ ต้องเรียก `powerMod(2, 50001, 10)` แต่ถ้าผู้อ่านลองสั่งทำงานดูจะพบว่า เกิดปัญหาเรื่อง `StackOverflowError` เพราะต้องมีการเรียกซ้ำๆ แบบยังไม่คืนผลลงไปเรื่อย ๆ ถึงห้าหมื่นครั้ง แล้วจะทำอย่างไร สำหรับปัญหานี้เราหาทางแก้ได้ สมมติเรารู้ว่า $2^{25000} \bmod 10$ มีค่าเป็น 6 แล้ว $2^{50001} \bmod 10$ มีค่าเท่าไร เนื่องจาก 25000 เป็นครึ่งหนึ่งของ 50000 ก็สามารถหาค่า $2^{50000} \bmod 10 = (2^{25000} \bmod 10)^2 \bmod 10 = 6^2 \bmod 10 = 6$ จะได้ $2^{50001} \bmod 10 = (2\times(2^{50000} \bmod 10)) \bmod 10 = (2\times 6) \bmod 10 = 2$ ดังนั้น สามารถเขียนสูตรการหา $a^b \bmod m$ ได้อีกแบบดังนี้

---

11 บรรทัดถูกต้อง เพราะ[illegible] substring(1, 0) [illegible] if (n==0) [illegible] substring [illegible]

$$a^b \bmod m = \begin{cases} 1 & b = 0 \\ \left(a^{b/2} \bmod m\right)^2 \bmod m & b \text{ เป็นจำนวนคู่} \\ a\left(a^{\lfloor b/2 \rfloor} \bmod m\right)^2 \bmod m & b \text{ เป็นจำนวนคี่} \end{cases}$$

สูตรนี้บอกว่า ถ้าต้องการหา a^{50} จะต้องหา a^{25}, a^{12}, a^6, a^3, a (ขอเขียนสั้น ๆ ไม่มี mod) ให้สังเกตว่าเลขยกกำลังลดลงทีละครึ่ง (ปัดเศษทิ้ง) เมื่อได้ผลลัพธ์ของกรณีเลขยกกำลังน้อย ก็นำมาคำนวณของกรณีเลขยกกำลังมากขึ้น เช่น เมื่อได้ a ก็ได้ $a^3 = a(a)^2$, ได้ $a^6 = (a^3)^2$, ได้ $a^{12} = (a^6)^2$, ได้ $a^{25} = a(a^{12})^2$, และได้ $a^{50} = (a^{25})^2$ นำแนวคิดนี้มาเขียนได้ดังรหัสที่ 6-24

```
public static int powerMod(int a, int b, int m) {
  if (b == 0) return 1;
  int p = powerMod(a, b / 2, m);
  p = (p * p) % m;
  if (b % 2 == 1) p = (a * p) % m;
  return p;
}
```

คำนวณ $a^{\lfloor b/2 \rfloor} \bmod m$

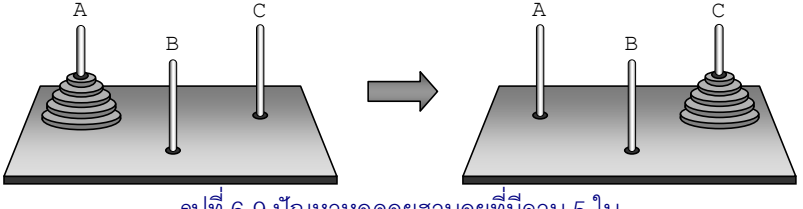
ถ้า b เป็นเลขคี่ คูณ a อีกครั้ง

รหัสที่ 6-24 เมท็อดคำนวณค่า $a^b \bmod m$ (แบบเร็ว)

การเรียก `powerMod(2, 50001, 10)` ของรหัสที่ 6-24 ไม่มีข้อผิดพลาดเรื่อง `StackOverflowError` เพราะมีการเรียกซ้ำ ๆ ซ้อนกันไม่กี่ครั้งก็คืนผลกลับแล้ว จากตัวอย่าง `b=50001` จะมีลำดับการลดลงของค่า `b` ทีละครึ่งคือ 50001, 25000, 12500, 6250, 3125, 1562, 781, 390, 195, 97, 48, 24, 12, 6, 3, 1, 0 รวมแล้วมีการเรียก `powerMod` ซ้อนๆ กันเพียง 17 ครั้งเท่านั้น

หอคอยฮานอย

ขอปิดท้ายหัวข้อนี้ด้วยโปรแกรมที่แก้ปัญหา*หอคอยฮานอย* (Tower of Hanoi) ปัญหานี้มีอยู่ว่า มีเสา 3 ต้นปักอยู่บนกระดานไม้ ให้ชื่อว่า `A`, `B`, และ `C` มีจานกลมอยู่ n ใบ ขนาดไม่เท่ากัน แต่ละใบมีรูตรงกลางพอดีให้เสาสอดผ่านได้ เริ่มต้นจานทั้ง n ใบสอดที่เสา `A` เรียงซ้อนกันใบใหญ่อยู่ล่างใบเล็กอยู่บน รูปที่ 6-9 ทางซ้ายแสดงตำแหน่งของจานตอนเริ่มต้น (มีจาน 5 ใบ) ปัญหานี้ถามว่าจะย้ายจานอย่างไร เพื่อให้จานทุกใบไปอยู่ที่เสา `C` ดังรูปที่ 6-9 ทางขวา โดยมีเงื่อนไขว่า ให้ย้ายจานได้ทีละใบ (จากเสาใดไปเสาใดก็ได้) และห้ามไม่ให้ใบใหญ่กว่าทับใบเล็กกว่า



รูปที่ 6-9 ปัญหาหอคอยฮานอยที่มีจาน 5 ใบ

รูปที่ 6-10 แสดงขั้นตอนการย้ายจานกรณีที่มีจานเพียง 3 ใบ ใช้การย้ายเป็นจำนวน 7 ครั้ง หากพิจารณารูปที่ 6-10 เราสามารถมองได้ว่าประกอบด้วยขั้นตอนย่อย 3 ขั้นตอนคือ

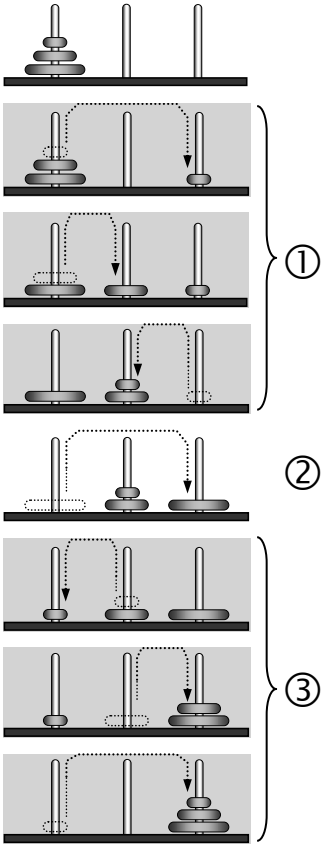
① ย้ายจาน 2 ใบบนจากเสาซ้ายไปเสากลาง
② ย้ายจานใบใหญ่จากเสาซ้ายไปที่เสาขวา
③ ย้ายจาน 2 ใบจากเสากลางไปเสาขวา

วิธีการย้ายเช่นนี้แสดงให้เห็นถึงแนวคิดการแก้ปัญหาใหญ่ด้วยการแก้ปัญหาย่อย วิธีการย้ายจาน 3 ใบประกอบด้วยการย้านจาน 2 ใบ และหากมองลึกลงไป การย้ายจาน 2 ใบก็มีขั้นตอนย่อยในทำนองเดียวกัน

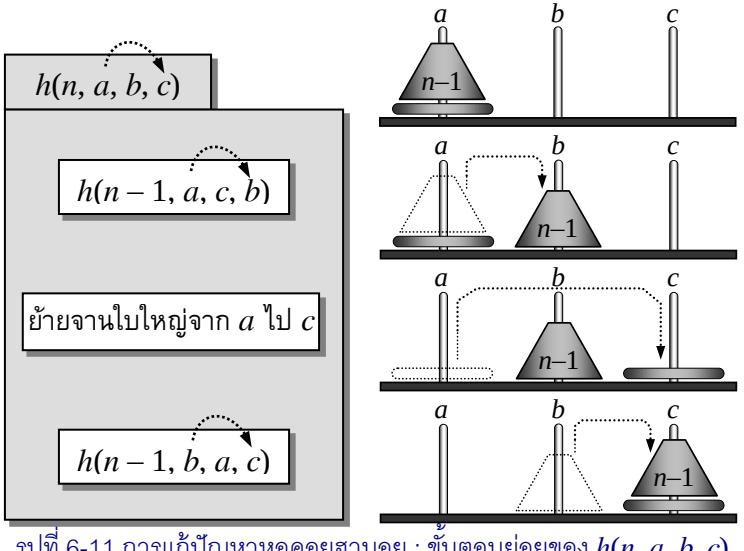
ดังนั้น เราสามารถขยายแนวคิดนี้มาใช้กับกรณีทั่วไป นิยามให้ $h(n, a, b, c)$ คือวิธีการย้ายจาน n ใบจากเสา a ไปเสา c โดยอาศัยเสา b เป็นเสาพักชั่วคราว เราสามารถแบ่งการทำงาน $h(n, a, b, c)$ ออกเป็น 3 ขั้นตอนย่อยคือ (ดูรูปที่ 6-11 ประกอบ)

- ย้ายจาน $n-1$ ใบจาก a ไป b ซึ่งคือ $h(n-1, a, c, b)$
- ย้ายจานใบใหญ่สุดจาก a ไป c
- ย้ายจาน $n-1$ ใบจาก b ไป c ซึ่งคือ $h(n-1, b, a, c)$

สามารถเขียนเป็นโปรแกรมได้ดังรหัสที่ 6-25



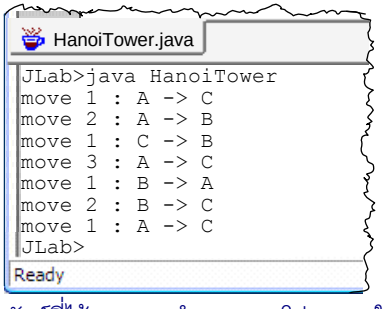
รูปที่ 6-10 กรณี n=3



รูปที่ 6-11 การแก้ปัญหาหอคอยฮานอย : ขั้นตอนย่อยของ $h(n, a, b, c)$

```
01 public class HanoiTower {
02   public static void main(String[] args) {
03     hanoi(3, "A", "B", "C");
04   }
05   public static void hanoi(int n, String a, String b, String c) {
06     if (n == 0) return;
07     hanoi(n - 1, a, c, b);
08     System.out.println("move " + n + " : " + a + " -> " + c);
09     hanoi(n - 1, b, a, c);
10   }
11 }
```

รหัสที่ 6-25 โปรแกรมแสดงการย้ายจานในหอคอยฮานอย



รูปที่ 6-12 ผลลัพธ์ที่ได้จากการทำงานของโปรแกรมในรหัสที่ 6-25

สิ่งที่โปรแกรมในรหัสที่ 6-25 ทำคือ แสดงขั้นตอนการย้ายจานทางจอภาพ (เมื่อสั่งโปรแกรมนี้ทำงานจะได้ผลดังรูปที่ 6-12) หรือจะดัดแปลงบรรทัดที่ 8 ให้เป็นคำสั่งแสดงภาพเคลื่อนไหวของจานบนจอภาพ หรือจะสั่งแขนหุ่นยนต์ให้ย้ายจานจริง ๆ ก็ทำได้ โดยกำหนดให้จานแต่ละใบมีหมายเลขกำกับ หมายเลข 1 คือจานใบเล็กสุด ขนาดใหญ่ขึ้น หมายเลขก็มากขึ้น ใบใหญ่สุดจึงมีหมายเลข n เมท็อด `hanoi(n, a, b, c)` ในรหัสที่ 6-25 ทำ 3 ขั้นตอนตามที่กล่าวไปข้างต้น ทำงานแบบเรียกซ้ำ โดยขนาดของปัญหาก็คือค่าของ `n` ซึ่งคือจำนวนจาน ทุกครั้งที่เรียกซ้ำ จำนวนจานจะลดลงหนึ่ง ดังนั้น เราต้องกำหนดกรณีเล็กสุดที่เราจะแก้ปัญหาได้ง่าย ๆ โดยไม่ต้องเรียกซ้ำ ในกรณีนี้เราอาจเลือกกรณี `n=1` นั่นคือต้องการย้ายจานใบเดียวจาก `a` ไป `c` ก็ย้ายเลยไม่ต้องคิดมาก แต่รหัสที่ 6-25 ใช้อีกวิธีที่ง่ายกว่าคือ ให้เป็นกรณี `n=0` การย้ายจาน 0 ใบ หมายความว่าไม่ต้องทำอะไร `return` ได้เลย [12]

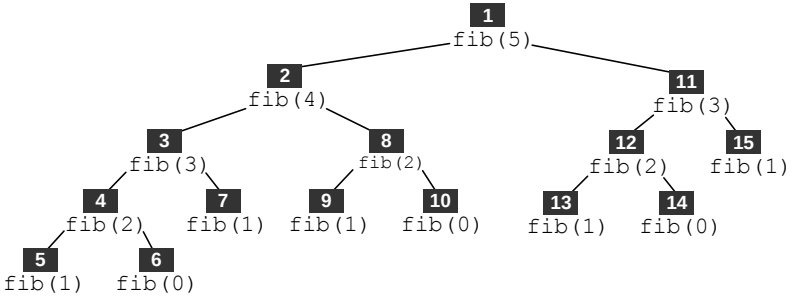
การเขียนเมท็อดแบบเรียกซ้ำบรรยายขั้นตอนการทำงานในลักษณะของการแก้ปัญหาใหญ่ด้วยการแก้ปัญหาย่อย ๆ ทำให้ได้โปรแกรมที่สั้นและสวย หากใช้วงวนอาจเขียนได้ลำบาก (เช่นหอคอยฮานอย) อย่างไรก็ตามต้องอย่าลืมว่า การเรียกเมท็อดแต่ละครั้งมีการจองหน่วยความจำไปจนกว่าเมท็อดจะทำงานเสร็จ จึงต้องระวังอย่าให้มีการเรียกซ้อนๆ กันมากเกินไป

[12] ขั้นตอนวิธีการย้ายจานที่นำเสนอนี้ใช้การย้าย $2^n - 1$ ครั้ง (เมื่อมีจาน n ใบ) ซึ่งเป็นจำนวนการย้ายที่น้อยสุด ผู้อ่านที่สนใจสามารถดูวิธีพิสูจน์ได้จากหนังสือคณิตศาสตร์ดีสครีต (หรือภิทนคณิตศาสตร์)

เพิ่มเติม

การซ้อนเหลื่อมของปัญหาย่อย

ถ้าผู้อ่านเขียนโปรแกรมเรียก `fib(47)` แบบใช้วงวน (รหัสที่ 6-20) และแบบเรียกซ้ำ (รหัสที่ 6-21) จะพบประเด็นน่าสนใจสองประเด็น ประเด็นที่หนึ่งคือ ทั้งสองเมท็อดให้ผลลัพธ์ที่ผิด ถ้าเรียก `fib(45)` และ `fib(46)` จะได้ 1134903170 และ 1836311903 (ซึ่งถูก) แต่พอเรียก `fib(47)` กลับได้ –1323752223 เป็นจำนวนลบ เหตุการณ์นี้ย้ำเตือนข้อจำกัดของการเก็บจำนวนเต็มด้วย `int` ซึ่งได้นำเสนอในบทที่ 2 ว่า `int` เก็บจำนวนเต็มได้ในช่วงประมาณบวกลบสองพันล้านเท่านั้น หากเกินช่วงนี้จะได้ผลผิด ในกรณีนี้ $f_{47} = f_{46} + f_{45}$ = 1134903170 + 1836311903 = 2971215073 ซึ่งเกิน ใช้ `int` เก็บจึงผิด



รูปที่ 6-13 ลำดับการเรียก `fib(5)` (หมายเลขสีขาวพื้นดำแสดงเลขลำดับของการเรียก)

อีกประเด็นที่น่าสนใจคือ การเรียก `fib` แบบวงวนจะได้ผลลัพธ์เร็วกว่าการเรียก `fib` แบบเรียกซ้ำ ซึ่งทำงานช้ามากแบบรู้สึกได้ `fib(46)` ใช้เวลาประมาณครึ่งนาทีบนเครื่องคอมพิวเตอร์ของผู้เขียนที่มีความถี่สัญญาณนาฬิกา 1.3GHz เมื่อเทียบกับแบบวงวนซึ่งได้ผลลัพธ์ทันทีที่กดปุ่มสั่งทำงาน หากลองติดตามการทำงานของ `fib` จะเห็นลักษณะการทำงานที่เรียกซ้ำมากเกินไป รูปที่ 6-13 แสดงลำดับการเรียก `fib` (หมายเลขสีขาวพื้นดำแสดงลำดับที่ของการเรียก) เมื่อเรียก `fib(5)` เรารู้ว่าต้องมีการเรียก `fib(4)` กับ `fib(3)` แต่ตัวโปรแกรมจะเรียก `fib(4)` ก่อน ยังไม่เรียก `fib(3)` จนกว่า `fib(4)` จะทำงานเสร็จ (เพราะเราเขียนให้เรียก `fib(n-1)` ก่อน) พอมาเรียก `fib(4)` ก็เรียก `fib(3)` ไล่ลงไปเรื่อยจนถึงการเรียกลำดับที่ 5 คือ `fib(1)` ซึ่งไม่มีการเรียกซ้ำแล้ว คืนผล กลับขึ้นไปเรียก `fib(0)` เป็นลำดับที่ 6 และดำเนินการเรียก `fib` ไปเรื่อยๆ ตามลำดับที่แสดงในรูป รวม ๆ แล้วมีการเรียกทั้งสิ้น 15 ครั้ง หากสังเกตดีๆ จะพบว่า การเรียก `fib` หลายครั้งเป็นการเรียกเพื่อหาคำตอบที่เคยเรียกไปแล้ว เช่น `fib(3)` ถูกเรียกในลำดับที่ 3 แล้วยังต้องมาเรียกอีกครั้งในลำดับที่ 11 หรือกรณี `fib(2)` มีการเรียกในลำดับที่ 4, 8, และ 12 เป็นลักษณะที่เรียกว่า เกิด*การซ้อนเหลื่อมของปัญหาย่อย* ตัวโปรแกรมไม่ได้จำว่า เคยหา

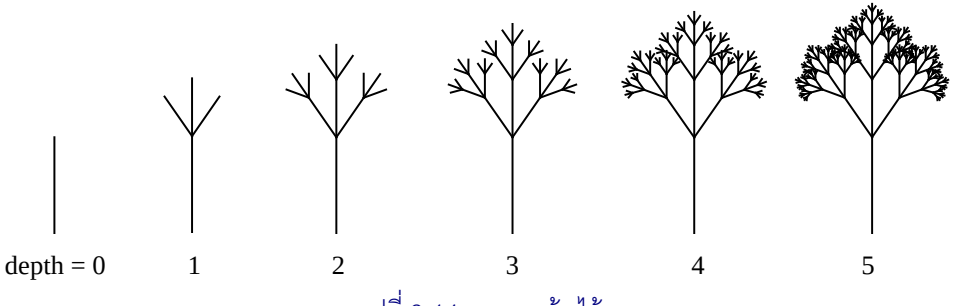
คำตอบมาก่อนหรือเปล่า ก็ต้องเรียกใหม่ทุกครั้งที่ต้องการคำตอบ ทำให้ทำงานช้ามากๆ อย่างไรก็ตาม การทำงานซ้ำมาก ๆ ในลักษณะเช่นนี้ไม่ได้เกิดกับทุกโปรแกรมที่เขียนแบบเรียกซ้ำ แต่เกิดเฉพาะกับการแบ่งปัญหาย่อยที่เกิดการซ้อนเหลื่อมกัน

การวาดสาทิสรูปด้วยโปรแกรมแบบเรียกซ้ำ

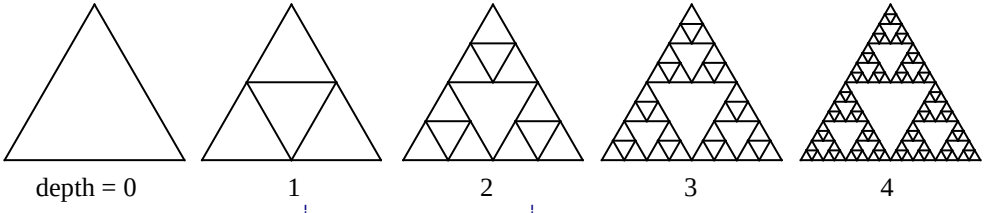
หัวข้อนี้นำเสนอการวาดรูปสวยๆ เรียกว่า*สาทิสรูป* หรือ*แฟร็กทัล* (fractal) สาทิส- แปลว่าเหมือนกัน คล้ายกัน สาทิสรูปหมายถึงรูปที่ประกอบด้วยรูปย่อยที่เหมือนหรือคล้ายกับรูปใหญ่ ดังตัวอย่างในรูปที่ 6-14 ถึง รูปที่ 6-17 เราวาดรูปเหล่านี้ด้วยโปรแกรมที่เขียนแบบเรียกซ้ำได้ง่าย ๆ เนื่องจากโครงสร้างของรูปวาดเหล่านี้มีลักษณะที่องค์ประกอบใหญ่ประกอบด้วยองค์ประกอบย่อยๆ ที่มีโครงสร้างในลักษณะเดียวกัน

เริ่มด้วยต้นไม้ในรูปที่ 6-14 จะเห็นว่าการวาดต้นไม้นี้ประกอบด้วยการวาดเส้นตรงหนึ่งเส้น (แทนลำต้นหรือกิ่ง) จากนั้นแตกกิ่งไปวาดต้นไม้เล็กๆ สามต้นสามทิศ ต้นหนึ่งตรงไปทิศเดียวกับเส้นที่วาด อีกต้นเอียงไปทางซ้าย และอีกต้นเอียงไปทางขวา การวาดต้นไม้ย่อยก็ใช้แนวทางเดียวกัน โดยต้องกำหนดว่า ให้แตกกิ่งและวาดต้นย่อยในลักษณะนี้ไปกี่ครั้ง เขียนได้ดังรหัสที่ 6-26

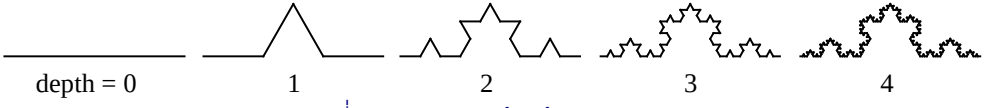
เมท็อด `drawTree` รับวินโดว์ `w`, รับพิกัด `(x0,y0)` ของโคนต้นไม้ รับความยาวลำต้น `d` ก่อนแตกกิ่ง รับมุมเอียง `a°` เทียบกับพื้น (`a=90` คือต้นไม้ตั้งฉากกับพื้น) และสุดท้ายรับความลึก `depth` ในการเรียกซ้ำ เมื่อใดมีค่า 0 คือให้หยุดแตกกิ่ง บรรทัดที่ 12 และ 13 ลากเส้นเริ่มต้นที่ `(x0,y0)` ยาว `d` เอียง `a` ถ้า `depth` ไม่มากกว่าศูนย์ ก็เลิกวาด แต่ถ้ายังมากกว่าศูนย์ ก็วาดต่อบรรทัดที่ 15 วาดต้นย่อยเริ่มที่ `(x1,y1)` ซึ่งคือจุดปลายของเส้นที่ลากตอนต้น ในทิศเดียวกับที่ได้รับมีความยาวเป็น 60% ของ `d` ที่ได้รับ จากนั้นวาดต้นสอง (บรรทัดที่ 16) เริ่มจุดเดียวกัน หมุนทวนเข็มไป 35° แต่ให้ความยาวเป็น 50% ตามด้วยต้นที่สาม (บรรทัดที่ 17) หมุนตามเข็มไป 35° เพื่อให้ได้ต้นไม้ที่ได้ดุลทางซ้ายและขวา ให้สังเกตว่าการเรียก `drawTree` ซ้ำเพื่อวาดต้นย่อยนั้น เราส่งค่า `depth-1` ไปให้เป็นค่า `depth` ใหม่ ดังนั้น `drawTree` ที่ถูกเรียกซ้ำและซ้อนกันจะมีค่า `depth` ลดลง ๆ รูปที่ 6-14 แสดงผลที่ได้จากค่า `depth` เริ่มต้นที่ต่างกัน



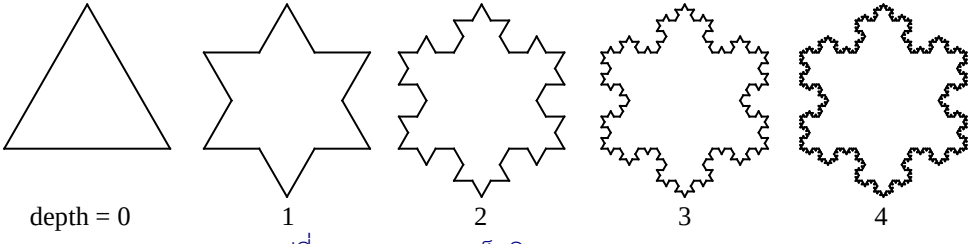
รูปที่ 6-14 การวาดต้นไม้



รูปที่ 6-15 การวาดสามเหลี่ยมของ Sierpinski

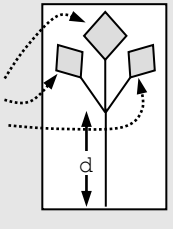


รูปที่ 6-16 การวาดเส้นโค้ง von Koch



รูปที่ 6-17 การวาดเกล็ดหิมะของ von Koch

```
01 import jlab.graphics.DWindow;
02
03 public class Fractal {
04   public static void main(String[] args) {
05     DWindow w = new DWindow(200, 200);
06     drawTree(w, 100, 200, 80, 90, 6);       // วาดต้นไม้
07     // spskiTriangle(w, 10, 190, 180, 6);  // วาดสามเหลี่ยม Sierpinski
08     // kochSnowFlake(w, 20, 150, 160, 4);  // วาดเกล็ดหิมะ von Koch
09   }
10   public static void drawTree(DWindow w, double x0, double y0,
11                               double d, double a, int depth) {
12     double x1 = x0 + d * cos(a), y1 = y0 - d * sin(a);
13     w.drawLine(x0, y0, x1, y1);
14     if (depth <= 0) return;
15     drawTree(w, x1, y1, 0.6 * d, a,      depth - 1);
16     drawTree(w, x1, y1, 0.5 * d, a + 35, depth - 1);
17     drawTree(w, x1, y1, 0.5 * d, a - 35, depth - 1);
18   }
19   public static double sin(double a) {
20     return Math.sin(Math.toRadians(a));
21   }
22   public static double cos(double a) {
23     return Math.cos(Math.toRadians(a));
24   }
```

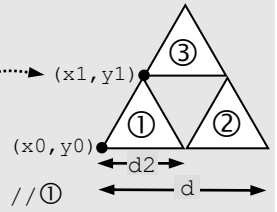


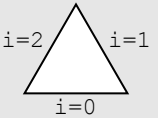
รหัสที่ 6-26 เมท็อดวาดรูปต้นไม้

คราวนี้มาดูการวาดรูปสามเหลี่ยมของ Sierpinski ดังรูปที่ 6-15 ตามชื่อของนักคณิตศาสตร์ชาวโปแลนด์ที่ได้นำเสนอการสร้างรูปนี้ในปี ค.ศ. 1915 เริ่มด้วยสามเหลี่ยมด้านเท่าหนึ่งรูป จากนั้นเปลี่ยนสามเหลี่ยมนี้ให้เป็นสามเหลี่ยมด้านเท่า 3 รูป แต่ละรูปมีฐานยาวเป็นครึ่งหนึ่งของฐานเดิม นั่นคือเปลี่ยนจาก △ เป็น  แล้วก็เริ่มเปลี่ยนสามเหลี่ยมทุกรูปให้เป็นสามเหลี่ยมย่อยๆ ในลักษณะเดียวกัน เขียนเป็นเมท็อดได้รหัสที่ 6-27

เมท็อด `spskiTriangle` รับวินโดว์ `w`, รับพิกัด `(x0,y0)` ของมุมซ้ายล่างของสามเหลี่ยมที่มีฐานยาว `d` และรับความลึก `depth` ในการเรียกซ้ำ ถ้า `depth` เป็น 0 ให้วาดรูปสามเหลี่ยมและคืนการทำงานทันที (บรรทัดที่ 28) ถ้า `depth` มากกว่า 0 จะไม่วาดสามเหลี่ยม แต่จะเปลี่ยนเป็นสามเหลี่ยมเล็ก 3 รูป และเรียกซ้ำต่อ บรรทัดที่ 33 เรียกซ้ำกับสามเหลี่ยมเล็กล่างซ้าย ส่วนบรรทัดที่ 34 และ 35 เรียกซ้ำกับสามเหลี่ยมเล็กล่างขวาและบนตามลำดับ และก็เหมือนตอนวาดต้นไม้ จะส่งค่า `depth-1` ให้กับ `depth` ใหม่ของการเรียกซ้ำ

```
25   public static void spskiTriangle(DWindow w,
26                       double x0, double y0, double d, int depth) {
27     if (depth <= 0) {
28       drawTriangle(w, x0, y0, d);
29     } else {
30       double d2 = d / 2;
31       double x1 = x0 + d2 * cos(60);
32       double y1 = y0 - d2 * sin(60);
33       spskiTriangle(w, x0,    y0, d2, depth - 1); //①
34       spskiTriangle(w, x0+d2, y0, d2, depth - 1); //②
35       spskiTriangle(w, x1,    y1, d2, depth - 1); //③
36     }
37   }
38   public static void drawTriangle(DWindow w, double x0, double y0,
39                                   double d) {
40     double a = 0;
41     for (int i = 0; i < 3; i++) {
42       double x1 = x0 + d * cos(a), y1 = y0 - d * sin(a);
43       w.drawLine(x0, y0, x1, y1);
44       a = a + 120;
45       x0 = x1; y0 = y1;
46     }
47   }
```





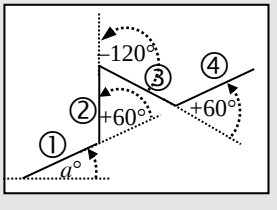
รหัสที่ 6-27 เมท็อดวาดสามเหลี่ยมของ Sierpinski

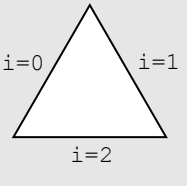
และรูปสุดท้ายคือเกล็ดหิมะของ von Koch (รูปที่ 6-17) ของนักคณิตศาสตร์ชาวสวีเดน เมื่อปี ค.ศ. 1904 ซึ่งสร้างมาจากเส้นโค้ง von Koch (รูปที่ 6-16) ขอนำเสนอการวาดเส้นโค้งนี้กันก่อน เริ่มต้นด้วยเส้นตรงหนึ่งเส้น แบ่งออกเป็นสามส่วนเท่ากัน เปลี่ยนส่วนกลางเป็นสามเหลี่ยมด้านเท่าแบบไม่มีฐาน โดยสรุปคือให้เปลี่ยนจาก เป็น จากนั้นเปลี่ยนส่วนของเส้นตรงทั้งสี่เส้นให้เป็นเส้นตรงย่อยๆ ในลักษณะเดียวกัน เขียนเป็นเมท็อดได้ดังรหัสที่ 6-28

เมท็อด kochCurve รับวินโดว์ w, รับพิกัด (x,y) ซึ่งเป็นจุดเริ่มของเส้นตรงที่มีความ d หมุนทวนเข็มเป็นมุม a° และรับความลึก depth ในการเรียกซ้ำ ถ้า depth เป็น 0 ก็ลากเส้นตรงที่ได้รับและเลิกการเรียกซ้ำ คืนการทำงานเลย ถ้า depth มากกว่า 0 จะเรียกซ้ำให้ทำกับเส้นตรงสี่เส้น ทุกเส้นมีความยาวเป็นหนึ่งในสามของ d (บรรทัดที่ 53) เส้นที่หนึ่งทำมุม a° เหมือนที่ได้รับมา เส้นต่อมาหมุนทวนเข็มจาก a° ไปอีก 60° (บรรทัดที่ 56) เส้นต่อไปหมุนตามเข็มวกกลับมา 120° (บรรทัดที่ 59) และเส้นสุดท้ายหมุนทวนเข็ม 60° อีกครั้ง (บรรทัดที่ 62) กลับมาทำมุมเหมือนเส้นแรก

เมื่อรู้วิธีการวาดเส้นโค้ง von Koch ก็สามารถวาดเกล็ดหิมะ von Koch ได้ง่าย ๆ ด้วยการวาดเส้นโค้ง von Koch สามเส้น แต่ละเส้นคือด้านของสามเหลี่ยมด้านเท่า ดังแสดงในเมท็อด kochSnowFlake ของรหัสที่ 6-28 ซึ่งรับวินโดว์ w จุดมุมซ้ายล่างของสามเหลี่ยม ความยาวด้าน และความลึกในการเรียกซ้ำ เมท็อดนี้ก็เพียงแต่ใช้วงวน for หมุนสามรอบเรียก kockCurve เพื่อวาดเส้นโค้ง von Koch สามเส้นแทนแต่ละด้านของสามเหลี่ยมที่ได้รับ

```
48   public static void kochCurve(DWindow w, double x, double y,
49                                double d, double a, int depth) {
50     if (depth <= 0) {
51       w.drawLine(x, y, x + d * cos(a), y - d * sin(a));
52     } else {
53       double d3 = d / 3;
54       kochCurve(w, x, y, d3, a, depth - 1);  //①
55       x = x + d3 * cos(a);  y = y - d3 * sin(a);
56       a = a + 60;
57       kochCurve(w, x, y, d3, a, depth - 1);  //②
58       x = x + d3 * cos(a);  y = y - d3 * sin(a);
59       a = a - 120;
60       kochCurve(w, x, y, d3, a, depth - 1);  //③
61       x = x + d3 * cos(a);  y = y - d3 * sin(a);
62       a = a + 60;
63       kochCurve(w, x, y, d3, a, depth - 1);  //④
64     }
65   }
66   public static void kochSnowFlake(DWindow w, double x, double y,
67                                    double d, int depth) {
68     double a = 60;
69     for (int i = 0; i < 3; i++) {
70       kochCurve(w, x, y, d, a, depth);
71       x = x + d * cos(a);  y = y - d * sin(a);
72       a = a - 120;
73     }
74   }
75 }
```





รหัสที่ 6-28 เมท็อดวาดเส้นโค้งและเกล็ดหิมะของ von Koch

แบบฝึกหัด

1. เมท็อดข้างล่างนี้ผิดที่ใด จงแก้ไขให้ถูกต้อง

```
public static double abs(double x) {
  if (x < 0)
    return -x;
  else if (x > 0)
    return x;
}
```

2. จงเขียนเมท็อดที่ทำหน้าที่คืนเลขสุ่มชื่อ `random` ตามข้อกำหนดต่อไปนี้
   - `random(int b)` คืนจำนวนเต็มสุ่มตั้งแต่ `0` ถึง `b`
   - `random(int a, int b)` คืนจำนวนเต็มสุ่มตั้งแต่ `a` ถึง `b`
   - `random(double a, double b)` คืนจำนวนจริงสุ่มตั้งแต่ `a` ถึง `b` (ไม่รวม `b`)

3. จงเขียนเมท็อดซึ่งปรับจากโปรแกรมต่าง ๆ ที่ได้เขียนในบทก่อนหน้านี้ ดังต่อไปนี้
   - `isPrime(int n)` ตรวจสอบว่า `n` เป็นจำนวนเฉพาะหรือไม่ (จากรหัส 3-14)
   - `dayOfWeek(int d, int m, int y)` คืนสตริงที่เป็นชื่อวันในสัปดาห์ของวันที่ `d` เดือน `m` ปี ค.ศ. `y` (จากรหัสที่ 4-4)
   - `rot13(String text)` คืนผลการเปลี่ยนข้อความ `text` ตามการเข้ารหัสแบบ ROT-13 (จากรหัสที่ 5-11)
   - `ean13(String code)` ตรวจสอบว่า รหัสที่ได้รับเป็นไปตามมาตรฐาน EAN-13 หรือไม่ (จากรหัส 5-19)

4. จงเพิ่มการตรวจสอบพารามิเตอร์ให้กับเมท็อด `digit2text` และ `pos2text` ในรหัสที่ 6-8 และรหัสที่ 6-9 ตามลำดับ ให้โยนสิ่งผิดปกติ เมื่อได้รับค่าที่ไม่ตรงตามข้อกำหนด

5. จงปรับเมท็อด `int2text` ในรหัสที่ 6-10 ให้ใช้ได้กับทุกค่าของ `int` ไม่ว่าจะเป็นจำนวนบวกหรือลบ เช่น ถ้าเรียก `int2text(-1311480221)` จะได้ **"ลบหนึ่งพันสามร้อยสิบเอ็ดล้านสี่แสนแปดหมื่นสองร้อยยี่สิบเอ็ด"**

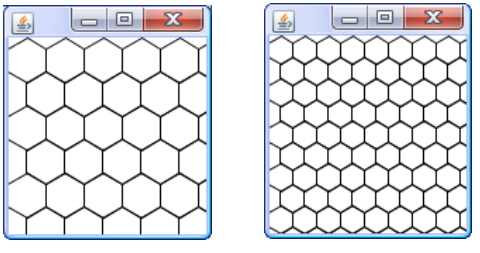
6. จงเขียนเมท็อด `double2text(double d, int p)` ที่แปลงจำนวน `d` ให้เป็นข้อความโดยใช้เลขหลังจุดทศนิยมเป็นจำนวน `p` ตำแหน่ง เช่น `double2text(-25.374, 2)` จะได้ **"ลบยี่สิบห้าจุดสามเจ็ด"**

7. จงเขียนเมท็อดเพื่อให้บริการตรวจสอบคุณสมบัติของสามเหลี่ยม (จากพิกัดของมุมทั้งสามของสามเหลี่ยมที่ได้รับ)
   - `isEquilateral` ตรวจสอบว่าเป็นสามเหลี่ยมด้านเท่าหรือไม่

- `isIsosceles` ตรวจสอบว่าเป็นสามเหลี่ยมหน้าจั่วหรือไม่
- `isRightangled` ตรวจสอบว่าเป็นสามเหลี่ยมมุมฉากหรือไม่
- `isScalene` ตรวจสอบว่าเป็นสามเหลี่ยมด้านไม่เท่าหรือไม่

8. จงเขียนเมท็อดเพื่อให้บริการตรวจสอบคุณสมบัติของสี่เหลี่ยม (จากพิกัดของมุมทั้งสี่ของสี่เหลี่ยมที่ได้รับ)
   - `isSquare` ตรวจสอบว่าเป็นสี่เหลี่ยมจัตุรัสหรือไม่
   - `isRectangle` ตรวจสอบว่าเป็นสี่เหลี่ยมผืนผ้าหรือไม่
   - `isRhombus` ตรวจสอบว่าเป็นสี่เหลี่ยมขนมเปียกปูนหรือไม่
   - `isParallelogram` ตรวจสอบว่าเป็นสี่เหลี่ยมด้านขนานหรือไม่
   - `isTrapezoid` ตรวจสอบว่าสี่เหลี่ยมคางหมูหรือไม่

9. จงเขียนโปรแกรมเรียกใช้เมท็อด `drawEquilateralPolygon` ในรหัสที่ 6-13 ให้แสดงภาพรังผึ้งซึ่งมาจากการวาดรูปหกเหลี่ยมด้านเท่าหลาย ๆ รูปต่อ ๆ กันไปดังแสดงข้างล่างนี้ (ปรับขนาดได้ตามพารามิเตอร์ที่ผู้ใช้ระบุ)



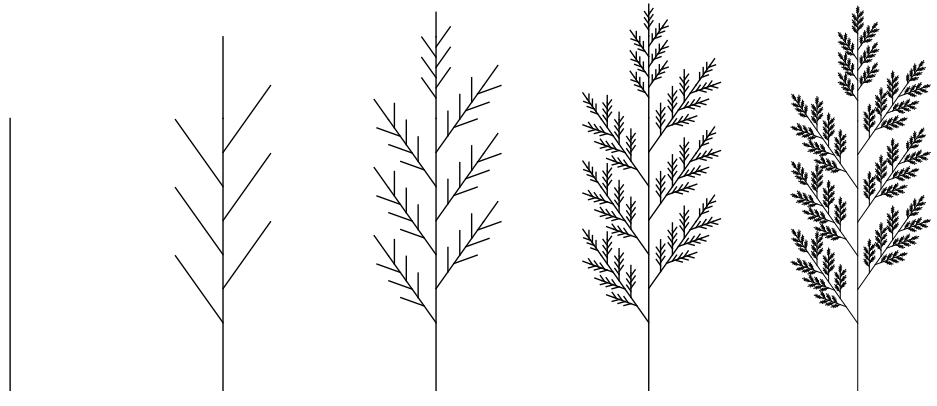
10. สองเมท็อดที่เป็นแบบ `public` ข้างล่างนี้ทำอะไร

```
public static String what1(int x) {
  if (x == 0) return "";
  return what1(x / 2) + (x % 2);
}
```

```
public static void what2(int k) {
  what2("", k);
}
private static void what2(String d, int k) {
  if (k == d.length()) {
    System.out.println(d);
  } else {
    what2(d + "0", k);
    what2(d + "1", k);
  }
}
```

11. จงเขียนเมท็อดแบบเรียกซ้ำเพื่อให้บริการต่าง ๆ ดังต่อไปนี้
    - `factorial(int n)` คืนค่า `n!` ซึ่งคือแฟกทอเรียลของ `n` จากนิยาม $n! = n \times (n-1)!$ เมื่อ $n > 0$, $0! = 1$
    - `numberOfRegions(int n)` คืน L_n ซึ่งคือจำนวนบริเวณมากสุดที่เป็นไปได้จากการขีดเส้นตรง `n` เส้นตัดกันบนกระดาษ โดยที่ $L_n = L_{n-1} + n$ สำหรับ $n > 0$, $L_0 = 1$
    - `reverse(String t)` คืนสตริงที่มีลำดับของอักขระกลับกับ `t` ที่ได้รับ เช่น `reverse("ABCDE")` จะได้ `"EDCBA"`
    - `log2(int n)` คืนจำนวนเต็มมากสุดที่น้อยกว่า $\log_2$ `n` เช่น `log2(1000)` ได้ `9` (โดยไม่ใช้บริการของคลาส `Math`)
    - `gcd(int n, int m)` คืนค่าหารร่วมมากของ `n` กับ `m` จากนิยาม $gcd(n, m) = gcd(m, n \% m)$ ถ้า n และ m ไม่เป็น 0, $gcd(0, m) = m$ และ $gcd(n, 0) = n$

12. จงเขียนโปรแกรมเพื่อวาดรูปใบไม้ในลักษณะที่คล้าย ๆ กับที่แสดงข้างล่างนี้



แถวลำดับ

เมื่อต้องประมวลผลข้อมูลที่เก็บในหน่วยความจำเป็นจำนวนมาก การใช้ตัวแปรหนึ่งตัวเก็บข้อมูลหนึ่งตัวคงไม่สะดวกแน่ จึงเป็นที่มาของการใช้แถวลำดับหรือเรียกว่าอาเรย์ ซึ่งคือโครงสร้างการจัดเก็บกลุ่มข้อมูลเป็นแถว ๆ แต่ละแถวมีชื่อกำกับ สามารถเข้าใช้ข้อมูลเหล่านี้ได้ด้วยเลขลำดับของข้อมูลในแถว ทำให้การประมวลผลข้อมูลในอาเรย์กระทำได้ง่าย อีกทั้งเหมาะกับการใช้วงวนเป็นกลไกหลักในการควบคุมการทำงาน ทำให้เขียนโปรแกรมได้กะทัดรัดและสวย บทนี้นำเสนอการสร้าง การใช้งาน และการเขียนเมท็อดที่จัดการเกี่ยวกับอาเรย์ ตลอดจนตัวอย่างการใช้งานอาเรย์ เช่น การค้นข้อมูล การเรียงลำดับข้อมูล การวาดกราฟเส้นจากข้อมูลอนุกรมเวลา การประมวลผลเมทริกซ์ การประมวลผลรูปภาพ เป็นต้น

การสร้างและการใช้งาน

แถวลำดับ หรือ*อาเรย์* (array) คือ แถวที่เก็บข้อมูลเรียงต่อกันเป็นลำดับ เสมือนเป็นช่องเก็บข้อมูลหลายๆ ช่องเรียงต่อกันไป อาเรย์ทั้งแถวมีชื่อกำกับ การใช้ข้อมูลในช่องใดอาศัยเลขลำดับของช่องนั้น เช่น `d` 30 | 15 | 78 | 43 แสดงอาเรย์ชื่อ `d` มี 4 ช่อง เก็บจำนวนเต็ม 4 ตัว แต่ละช่องมีเลขลำดับเรียกว่า *ดัชนี* (index) จากซ้ายไปขวาเป็น 0, 1, 2, 3 เมื่อต้องการใช้ข้อมูลช่องที่ `k` ใน `d` ให้เขียน `d[k]` ดังนั้น จากตัวอย่าง `d[0]` คือ 30 และ `d[3]` คือ 43 หากเปรียบเทียบกับสตริง ก็อาจมองได้ว่า สตริง (ที่แปลว่าสายอักขระ) เป็นอาเรย์ของอักขระ มีดัชนีกำกับอักขระแต่ละตัวในสตริง แต่จาวาไม่ได้กำหนดให้สตริงเป็นอาเรย์ จึงไม่สามารถใช้ `s[k]` เพื่ออ้างอิงอักขระที่ดัชนี `k` ของสตริง `s` ได้ (ต้องใช้เมท็อดของสตริงแทน เช่น `s.substring(k,k+1)`) อาเรย์จึงเหมาะกับการเก็บกลุ่มข้อมูล โดยใช้ดัชนีระบุตำแหน่งของข้อมูลในกลุ่มเพื่อการประมวลผล

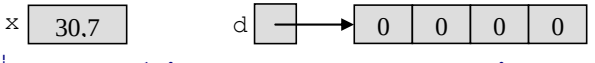
ก่อนจะใช้อาเรย์ ต้องรู้ว่า จะใช้เก็บข้อมูลประเภทใด `int`, `double`, `String` หรือประเภทอื่น และจะสร้างอาเรย์กี่ช่อง คำสั่ง `new` มีไว้สร้างอาเรย์ซึ่งต้องระบุประเภทข้อมูลและจำนวนช่องที่จะสร้าง เช่น ต้องการสร้างอาเรย์ชื่อ `d` มี 4 ช่องไว้เก็บ `int` ก็ใช้คำสั่ง

```
int[] d = new int[4];
```

คำสั่งนี้ประกอบด้วยส่วนย่อยสามส่วน

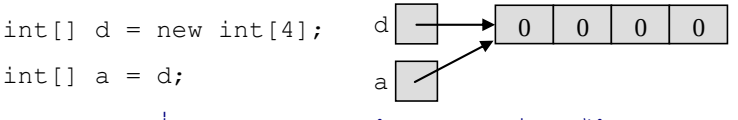
- `int[] d` คือการประกาศตัวแปร `d` ว่า เป็นตัวแปรที่อ้างอิงอาเรย์แบบ `int`
- `new int[4]` คือการสร้างอาเรย์แบบ `int` มีขนาด 4 ช่อง
- `=` แทนการให้ตัวแปร `d` (ทางซ้ายของ =) อ้างอิงอาเรย์ที่สร้างได้ (ทางขวาของ =)

ขอขยายความคำว่าอ้างอิงอาเรย์ ที่ผ่านมาหากเราประกาศตัวแปร `double x = 30.7;` ระบบจะเตรียมที่เก็บชื่อ `x` และใส่ `30.7` ไว้ในที่เก็บนั้นเลย (ดูรูปที่ 7-1 ทางซ้าย) แต่ในกรณีของอาเรย์ตัวแปร `d` ในตัวอย่างข้างบนนี้ ไม่ได้เป็นที่เก็บของตัวอาเรย์ทั้งแถว แต่เป็นที่เก็บข้อมูลขนาดเล็กๆ ซึ่งแทนตำแหน่งที่อยู่ของอาเรย์ (ดูรูปที่ 7-1 ทางขวา) จึงเรียกว่า `d` อ้างอิง หรือบางทีเรียกว่า “ชี้” ไปยังตัวอาเรย์ ดังนั้น ไม่ว่าอาเรย์จะมีขนาดเท่าใดก็ตาม ตัวแปรที่อ้างอิงอาเรย์นั้นจะมีขนาดเท่ากันหมด



รูปที่ 7-1 ตัวแปรเก็บข้อมูลแบบ `double` และตัวแปรอ้างอิงอาเรย์

ด้วยการอ้างอิงอาเรย์เช่นนี้ จึงสามารถให้ตัวแปรหลายตัวอ้างอิงอาเรย์เดียวกันได้ ดังตัวอย่างในรูปที่ 7-2 แสดงคำสั่งการให้ค่า `a = d;` คือการนำค่าใน `d` ซึ่งคือตำแหน่งที่อยู่ของอาเรย์ ไปให้กับ `a` ทำให้ตัวแปร `a` อ้างอิงอาเรย์เดียวกับที่ `d` อ้างอิงด้วย เมื่อตัวแปร `a` และ `d` อ้างอิงอาเรย์เดียวกัน การอ้างอิงข้อมูลในอาเรย์ที่ดัชนีเดียวกัน เช่น `a[2]` กับ `d[2]` ก็คือช่องที่เก็บข้อมูลเดียวกัน



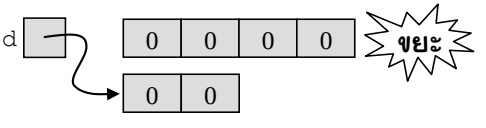
รูปที่ 7-2 ตัวแปรหลายตัวอ้างอิงอาเรย์เดียวกันได้

รูปที่ 7-2 แสดงอาเรย์ที่ทุกช่องมีค่า 0 หลังการสร้างนั้น ไม่ใช่เรื่องบังเอิญ แต่เป็นความตั้งใจของระบบจาวาที่ตั้งค่าเริ่มต้นให้กับทุกช่องในอาเรย์โดยอัตโนมัติหลังการสร้างอาเรย์ด้วย `new` โดยจะเติมค่าศูนย์ให้กับอาเรย์แบบจำนวน เติมค่า `false` ให้กับอาเรย์แบบ `boolean` และเติมค่า `null` ให้กับอาเรย์แบบสตริง โดยที่ `null` เป็นค่าคงตัวพิเศษที่จะอธิบายในภายหลัง แต่ตอนนี้ขอให้เข้าใจเพียงว่า อาเรย์ของสตริงที่เก็บ `null` หมายถึงทุกช่องยังไม่มีสตริงใดๆ เก็บอยู่เลย

ก่อนจะนำเสนอตัวอย่างการใช้งานอาเรย์ ขอสรุปประเด็นที่น่าสนใจในการสร้างอาเรย์ดังนี้

- การประกาศตัวแปรแบบอาเรย์ ไม่ได้สร้างอาเรย์ เช่น คำสั่ง `int[] d;` เป็นการเตรียมที่เก็บสำหรับตัวแปร `d` ซึ่งพร้อมอ้างอิงอาเรย์แบบ `int` แต่ยังไม่มีตัวอาเรย์ ระบบจะสร้างอาเรย์ก็เมื่อใช้คำสั่ง `new`
- ตัวแปรจะอ้างอิงอาเรย์ใด ต้องประกาศให้เป็นประเภทเดียวกับที่อาเรย์นั้นเป็น เช่น คำสั่ง `int e = new int[9];` ผิดเพราะ `e` เป็นตัวแปรแบบ `int` อ้างอิงอาเรย์ไม่ได้ (ต้องเป็น `int[]`), คำสั่ง `double[] b = new int[10];` ก็ผิด เพราะ `b` เป็นตัวแปรที่อ้างอิงอาเรย์คนละแบบกับ `int[]` หรือจากตัวอย่างในรูปที่ 7-2 ถ้าเพิ่มคำสั่ง `String[] s = d;` ก็ผิดด้วยสาเหตุเดียวกัน
- จำนวนช่องของอาเรย์ที่จะให้สร้างนั้น กำหนดเป็นค่าคงตัว (`new int[5]`) เป็นตัวแปร (`new int[n]`) หรือเป็นนิพจน์คำนวณ (`new int[2*n+10]`) ก็ได้ แต่จำนวนที่ได้ต้องเป็นจำนวนเต็มไม่ติดลบ (ดังนั้น สามารถสร้าง 0 ช่องก็ได้)
- หลังจากที่สร้างอาเรย์แล้ว จาวาไม่มีคำสั่งให้เพิ่มหรือลดขนาดของอาเรย์นั้น หากต้องการเพิ่มหรือลดขนาด อาจทำได้ด้วยการสร้างอาเรย์แถวใหม่ให้มีขนาดตามต้องการ จากนั้นทำสำเนาข้อมูลในอาเรย์เก่าไปเก็บในอาเรย์ใหม่ (จะนำเสนอรายละเอียดวิธีทำในหัวข้อถัดๆ ไป)
- อาเรย์ใดที่สร้างแล้ว ไม่มีตัวแปรใดอ้างอิง ถือว่าเป็น “ขยะ” เนื้อที่ของอาเรย์นี้จะถูกคืนกลับสู่ระบบ เช่น รูปที่ 7-3 แสดงผลที่ตัวแปร `d` จากเคยอ้างอิงอาเรย์หนึ่ง แล้วเปลี่ยนใจไปอ้างอิงแถวอื่น อาเรย์เดิมจะกลายเป็น “ขยะ”

```
int[] d = new int[4];
d = new int[2];
```



รูปที่ 7-3 อาเรย์ที่ไม่มีตัวแปรอ้างอิงจะถูกคืนกลับสู่ระบบ

หัวข้อย่อยต่อไปนี้นำเสนอตัวอย่างเพื่อให้เห็นถึงแรงจูงใจของการใช้อาเรย์ เมื่อใดที่เราต้องจัดเก็บกลุ่มข้อมูลที่มีความหมายเดียวกัน ก็ให้คิดถึงอาเรย์ ที่มักใช้ควบคู่กับการทำงานแบบวงวน ซึ่งทำให้ได้โปรแกรมที่สั้นและสวย

โปรแกรมเก็บสถิติคะแนนสอบ

กำหนดให้มีแฟ้มข้อมูลเก็บคะแนนของนักเรียน แต่ละบรรทัดในแฟ้มเก็บรหัสประจำตัวนักเรียน ตามด้วยคะแนน (คั่นด้วยช่องว่าง) คะแนนที่ให้เป็นจำนวนเต็มตั้งแต่ 0 ถึง 10 เราต้องการเขียนโปรแกรมที่อ่านแฟ้มนี้เพื่อแจกแจงจำนวนนักเรียนที่ได้คะแนนต่าง ๆ (เช่น ได้ 0 คะแนนมี 8

คน, ได้ 1 คะแนนมี 2 คน, ...) โปรแกรมนี้ใช้ Scanner เปิดแฟ้ม มีตัวแปรไว้นับจำนวนนักเรียนที่ได้คะแนนต่าง ๆ เราอาจเลือกที่จะใช้ตัวแปร 11 ตัว ให้ชื่อว่า p0, p1, ..., p10 โดยใช้ p0 เก็บจำนวนนักเรียนที่ได้ 0 คะแนน, p1 เก็บจำนวนนักเรียนที่ได้ 1 คะแนน, ... จากนั้นใช้วงวนอ่านข้อมูลจากแฟ้มทีละบรรทัด ใช้ if-else เปรียบเทียบว่า คะแนนที่อ่านเข้ามานั้นจะให้ไปเพิ่มตัวแปรใดใน 11 ตัว

แต่พอเห็นว่าต้องใช้ตัวแปรหลายตัวที่มีความหมายเหมือนกัน นั่นคือเป็นตัวแปรไว้เก็บจำนวนนักเรียน และตัวแปรดังกล่าวมีหมายเลขกำกับ จึงเหมาะมากที่จะใช้อาเรย์ โดยสร้างเป็นอาเรย์ชื่อ p จำนวน 11 ช่องไว้เก็บจำนวนเต็ม โดยกำหนดให้ p[v] เก็บจำนวนนักเรียนที่ได้ v คะแนน เพียงเท่านี้ก็ไม่ต้องใช้ if-else เพราะใช้คะแนนเป็นดัชนีเพื่อเพิ่มค่าในช่องที่ต้องการได้ทันที เขียนโปรแกรมนี้ได้ดังรหัสที่ 7-1 บรรทัดที่ 7 สร้างอาเรย์ 11 ช่อง โดยระบบตั้งค่าในแต่ละช่องให้เป็น 0 ซึ่งตรงกับความต้องการของเราพอดี ที่ต้องการให้ตอนเริ่มต้นจำนวนนักเรียนเป็น 0 ทุกช่อง เข้าวงวนทำซ้ำตราบเท่าที่ยังมีข้อมูลเหลือให้อ่าน เนื่องจากหนึ่งบรรทัดในแฟ้มประกอบด้วยรหัสประจำตัว ตามด้วยช่องว่าง และตามด้วยคะแนน ซึ่งข้อมูลทั้งสองเป็นจำนวน จึงใช้ Scanner อ่านเป็นจำนวนได้เลย แต่เนื่องจากรหัสประจำตัวเป็นจำนวนที่มีค่าเกินกว่าที่ int จะรับได้ จึงหันไปใช้ nextDouble() เพื่ออ่านเข้ามา (บรรทัดที่ 9) โดยไม่ได้เก็บค่านี้ในตัวแปรใด ๆ เพราะไม่ได้ใช้ ตามด้วย nextInt() อ่านได้เป็นคะแนนเก็บใน v (บรรทัดที่ 10) ถ้าเป็นค่าที่อยู่ในช่วง 0 ถึง 10 ก็สามารถเพิ่มค่าของ p[v] หลังจากอ่านและประมวลผลจนหมดแฟ้มออกจากวงวน while ก็เข้าวงวน for หมุนไล่หยิบแต่ละช่องในอาเรย์ออกมาแสดงผลทางจอภาพ นักเขียนโปรแกรมใช้วงวน for คู่กับการประมวลผลข้อมูลในอาเรย์กันมาก ซึ่งเราจะได้เห็นการใช้งานในลักษณะนี้อีกหลายตัวอย่าง

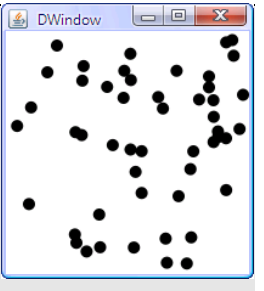
```
01 import java.util.Scanner;
02 import java.io.*;
03 // โปรแกรมแสดงสถิติจำนวนนักเรียนที่ได้คะแนนต่างๆ
04 public class Stat {
05   public static void main(String[] args) throws IOException {
06     Scanner in = new Scanner(new File("c:/java101/java101.txt"));
07     int[] p = new int[11];
08     while (in.hasNext()) {
09       in.nextDouble();       // อ่านรหัสประจำตัวทิ้งไป
10       int v = in.nextInt(); // อ่านคะแนน (เป็นจำนวนเต็ม)
11       if (0 <= v && v <= 10) p[v]++;
12     }
13     for (int v = 0; v < 11; v++) {
14       System.out.println(p[v] + " คนได้ " + v + " คะแนน");
15     }
16   }
17 }
```

```
4531001321  3
4531003621  8
4531004221  0
4531005921  8
4531006521  8
4531101421 10
4531102021 10
```

รหัสที่ 7-1 โปรแกรมแสดงสถิติจำนวนนักเรียนที่ได้คะแนนต่างๆ

ลูกบอลหลายลูก

ในบทที่ 4 เราได้นำเสนอโปรแกรมแสดงลูกบอลหนึ่งลูกเคลื่อนที่กระทบผนังไปมาในวินโดว์ ตัวแปรที่ใช้บรรยายลูกบอลคือ ตำแหน่ง x, y และความเร็วแทนด้วย dx และ dy ซึ่งคือการเปลี่ยนตำแหน่งตามแนวนอนและแนวตั้ง หากเราต้องการให้ลูกบอล 50 ลูกเคลื่อนที่บนวินโดว์ ก็คงต้องเขียนโปรแกรมที่ยาวมาก เพราะต้องใช้ตัวแปรถึง 200 ตัวเพื่อเก็บตำแหน่งและความเร็วของลูกบอลแต่ละลูก ทำไมไม่ใช้อาเรย์ 4 แถว ให้ชื่อว่า x, y, dx, และ dy โดยให้หมายเลข 0 ถึง 49 กับลูกบอล เพื่อให้ลูกบอลลูกที่ i อยู่ที่พิกัด x[i], y[i] และมีความเร็ว dx[i] และ dy[i] เพียงเท่านี้จะได้โปรแกรมที่สั้น ดังแสดงในรหัสที่ 7-2




```
01 import jlab.graphics.DWindow;
02 public class BouncingBalls {
03   public static void main(String[] args) {
04     DWindow w = new DWindow(200, 200);
05     int n = 50;
06     double[] x = new double[n], y = new double[n];
07     double[] dx = new double[n],dy = new double[n];
08     for (int i = 0; i < n; i++) {
09       x[i] = 100; dx[i] = random(-5, 5);
10       y[i] = 100; dy[i] = random(-5, 5);
11     }
12     w.setRepaintDuringSleep(true); //ให้ w แสดงภาพเมื่อ sleep เท่านั้น
13     while (true) {
14       w.clearBackground();
15       for (int i = 0; i < n; i++) {
16         x[i] += dx[i];
17         y[i] += dy[i];
18         if (x[i] <= 5 || x[i] >= 195) dx[i] = -dx[i]; //ชนผนังกลับทิศ
19         if (y[i] <= 5 || y[i] >= 195) dy[i] = -dy[i]; //ชนผนังกลับทิศ
20         x[i] = Math.min(Math.max(x[i], 5), 195);    //ปรับให้อยู่ในวินโดว์
21         y[i] = Math.min(Math.max(y[i], 5), 195);    //ปรับให้อยู่ในวินโดว์
22         w.fillEllipse(w.BLACK, x[i], y[i], 10, 10);
23       }
24       w.sleep(50);
25     }
26   }
27   private static double random(double a, double b) {
28     return a + (b - a) * Math.random();
29   }
30 }
```

เปลี่ยน 195 เป็น w.getWidth() – 5 และ w.getHeight() – 5 จะดีกว่านี้ จริงไหม ?

รหัสที่ 7-2 โปรแกรมแสดงลูกบอล 50 ลูกเคลื่อนที่ไปมาในวินโดว์

บรรทัดที่ 6 และ 7 สร้างอาเรย์ทั้งสี่แถว เข้าวงวน for เพื่อตั้งให้ลูกบอลทุกลูกอยู่ที่กลางวินโดว์ และสุ่มตั้งความเร็วโดยเรียกเมท็อด random จากนั้นเข้าวงวนไม่รู้จบเริ่มด้วยการลบวินโดว์ (บรรทัดที่ 14) ตามด้วยวงวน for เพื่อเลื่อนลูกบอล (บรรทัดที่ 16 และ 17) ตรวจสอบการ

ชนผนังของลูกบอล (บรรทัดที่ 18 และ 19) ปรับให้ลูกบอลมาแตะผนังถ้าออกนอกวินโดว์ (บรรทัดที่ 20 และ 21) และวาดลูกบอล (บรรทัดที่ 22) จนครบทุกลูก แล้วหยุดชั่วขณะเพื่อกำหนดความเร็ว วนทำเช่นนี้ไปเรื่อยๆ ไม่รู้จบ เห็นลูกบอลเคลื่อนที่ไปมา 50 ลูกพร้อม ๆ กันบนวินโดว์

โดยทั่วไปวินโดว์ `w` แสดงภาพใหม่ทุกครั้งที่มีการวาดรูป ดังนั้น การวาดวงกลมที่ตำแหน่งใหม่ 50 รูป จะทำให้วินโดว์ต้องแสดงภาพใหม่ 50 ครั้งในแต่ละรอบของวงวน `while(true)` การวาดภาพใหม่ติด ๆ กันหลาย ๆ ครั้งเช่นนี้ ทำให้เกิดภาพกระพริบ ภาพเคลื่อนไหวของลูกบอลแลดูไม่สวย เราสามารถแก้ไขอาการกระพริบนี้ได้โดยสั่งให้วินโดว์อย่าใจร้อนแสดงภาพทุกครั้งที่วาดรูป แต่ให้รอไปแสดงภาพใหม่ทีเดียวในเมท็อด `sleep` ซึ่งทำงานที่ปลายวงวน ด้วยเมท็อด `setRepaintDuringSleep` (บรรทัดที่ 12) เมื่อรับค่า `true` จะหมายถึงให้วาดใหม่เฉพาะเมื่อ `sleep` แต่ถ้ารับ `false` จะแสดงภาพทันทีที่วาดตามปกติ เพียงเท่านี้ภาพจะไม่กระพริบ ผู้อ่านควรลองสั่งโปรแกรมนี้ทำงานดู อาจลองเปลี่ยนเป็น 200 ลูก (แก้ไขเพียงบรรทัดเดียวเท่านั้น ผู้อ่านทราบไหมว่าบรรทัดใด) แล้วสังเกตผลที่ได้ หรือจะลองเปลี่ยนคำสั่ง `w.clearBackground()` ที่บรรทัดที่ 14 เป็น `w.fade(0.4)` จะทำให้ลูกบอลเคลื่อนที่แบบมีหาง

การใช้ดัชนีนอกช่วงของอาเรย์

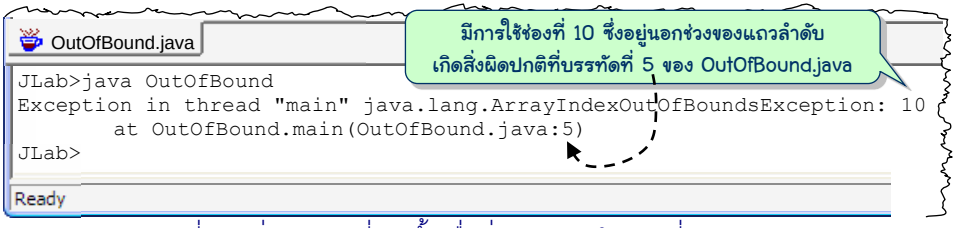
บ่อยครั้งที่นักเขียนโปรแกรมเปลี่ยนแปลงค่าดัชนีจนออกนอกช่วงที่อาเรย์มี จะทำให้เกิดข้อผิดพลาด อย่าลืมว่าอาเรย์ที่มี n ช่อง จะใช้ช่องที่ 0 ถึง $n-1$ ได้เท่านั้น รหัสที่ 7-3 แสดงตัวอย่างของข้อผิดพลาดดังกล่าว บรรทัดที่ 3 สร้างอาเรย์ขนาด 10 ช่อง จากนั้นต้องการตั้งค่าเริ่มต้นให้ช่องที่ `i` มีค่า `i` โดยใช้วงวน `for` แต่ใส่เงื่อนไขใน `for` ผิดเป็น `i<=10` แสดงว่าเมื่อ `i` มีค่า 10 ก็ยังทำในวงวน จึงเกิดการใช้ช่องที่ 10 ซึ่งอาเรย์ `d` ไม่มีให้ใช้ ระบบจาวาจะโยนสิ่งผิดปกติประเภท `ArrayIndexOutOfBoundsException` ได้ผลการทำงานดังรูปที่ 7-4 ให้สังเกตว่า ข้อความที่แสดงสิ่งผิดปกติที่เกิดขึ้นนั้นบอกว่า ผิดที่บรรทัดที่ 5 และยังบอกอีกว่า ดัชนีที่ผิดนั้นมีค่าเป็น 10 ขณะที่ทำงานผิด

สิ่งผิดปกติที่เกิดขึ้นนี้มีลักษณะคล้ายกับกรณีที่เราต้องการสตริงย่อยที่อยู่นอกช่วงที่สตริงมีให้ใช้ เช่น คำสั่ง `s.substring(4,5)` คือการคืนสตริงย่อยตัวที่ดัชนี 4 ออกมา ถ้า `s` มีค่า `"0123"` คำสั่งนี้จะโยนสิ่งผิดปกติ `StringIndexOutOfBoundsException`

```
01 public class OutOfBound {
02   public static void main(String[] args) {
03     int[] d = new int[10];
04     for (int i=0; i<=10; i++) {
05       d[i] = i;
06     }
07   ...
```

เกิดข้อผิดพลาดเมื่อ i เป็น 10

รหัสที่ 7-3 ตัวอย่างโปรแกรมที่ใช้ดัชนีนอกช่วงของอาเรย์



รูปที่ 7-4 สิ่งผิดปกติที่เกิดขึ้นเมื่อสั่งโปรแกรมในรหัสที่ 7-3 ทำงาน

การตั้งค่าเริ่มต้น

ขอเขียนเมท็อดแปลงเลขวันของสัปดาห์เป็นชื่อ คือ 0 เป็น "เสาร์", 1 เป็น "อาทิตย์", ..., 6 เป็น "ศุกร์" (ซึ่งนำไปใช้ได้กับโปรแกรมหาวันของสัปดาห์จากวันเดือนปีที่ได้รับในรหัสที่ 4-4) โดยใช้อาเรย์แบบ `String` ชื่อ `dow` สร้างไว้ 7 ช่อง เก็บชื่อวันในแต่ละช่อง ต้องการชื่อของเลขวัน `d` ก็ใช้ `dow[d]` ได้เลย ไม่ต้องใช้ `if-else` ให้ยืดยาว ดังรหัสที่ 7-4

```
public static String getDayOfWeek(int d) {
  if (d < 0 || d > 6) throw new IllegalArgumentException("d="+d);
  String[] dow = new String[7];
  dow[0] = "เสาร์";   dow[1] = "อาทิตย์";
  dow[2] = "จันทร์";  dow[3] = "อังคาร";
  dow[4] = "พุธ";     dow[5] = "พฤหัสบดี";
  dow[6] = "ศุกร์";
  return dow[d];
}
```

รหัสที่ 7-4 เมท็อดแปลงเลขวันของสัปดาห์เป็นชื่อ

การตั้งค่าเริ่มต้นในลักษณะนี้เกิดขึ้นบ่อย จาวาจึงมีวิธีลัดในการสร้างและตั้งค่าเริ่มต้น ด้วยการเขียนเป็น*รายการเริ่มต้น* ดังรหัสที่ 7-5 หากต้องการสร้างอาเรย์แบบ `int` ขนาด 3 ช่อง ให้มีค่าเริ่มต้น 1, 3, 7 ก็เขียน `new int[]{1,3,7}` โดยไม่ต้องระบุจำนวนช่อง ตัวแปลโปรแกรมรู้ได้เองจากจำนวนข้อมูลภายใน `{ }` ถ้าเขียน `new int[3]{1,3,7}` จะผิด นอกจากนี้จาวายังอนุญาตให้เขียนลัดให้น้อยลงอีก คือให้เขียน `int[] d = {1,3,7};` ได้เลย แต่ต้องเขียนในบรรทัดที่ประกาศตัวแปรเท่านั้น จะเขียนแยกเป็นสองคำสั่ง `int[] d; d={1,3,7};` ไม่ได้

```
public static String getDayOfWeek(int d) {
  if (d < 0 || d > 6) throw new IllegalArgumentException("d="+d);
  String[] dow =
    new String[] {"เสาร์","อาทิตย์","จันทร์","อังคาร","พุธ","พฤหัสบดี","ศุกร์"};
  return dow[d];
}
```

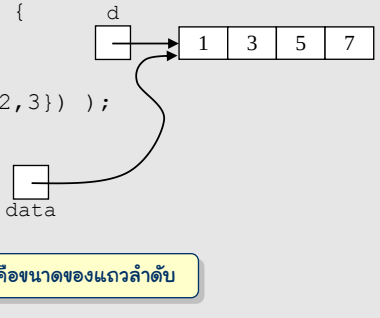
การสร้างและตั้งค่าด้วยรายการเริ่มต้น

รหัสที่ 7-5 การสร้างและตั้งค่าเริ่มต้นให้กับอาเรย์ด้วยรายการเริ่มต้น

เมท็อดที่รับอาเรย์

เมท็อดรับข้อมูลผ่านมาทางพารามิเตอร์เพื่อประมวลผล แล้วคืนผลลัพธ์กลับให้ผู้เรียก จึงไม่แปลก ถ้าเมท็อดจะรับอาเรย์เข้ามาประมวลผล หรือคืนผลลัพธ์เป็นอาเรย์ให้ผู้เรียก แต่การรับและคืนอาเรย์มีข้อแตกต่างจากข้อมูลประเภทอื่น ๆ ที่เราได้นำเสนอมา ขอเริ่มด้วยการรับพารามิเตอร์เป็นอาเรย์ ต้องขอย้ำว่า เราใช้งานอาเรย์หนึ่งผ่านตัวแปรที่อ้างอิงอาเรย์นั้น โดยตัวแปรนี้เป็นเพียงที่เก็บเล็กๆ ซึ่งเก็บตำแหน่งที่อยู่ของอาเรย์ที่อ้างอิง ตัวแปรนี้ไม่ได้เก็บตัวอาเรย์ เหมือนอย่างที่ตัวแปร `int` เก็บจำนวนเต็ม พิจารณาตัวอย่างในรหัสที่ 7-6 บรรทัดที่ 4 เรียก `sum` และส่ง `d` ไปให้ ถ้าดูที่หัวเมท็อด `sum` จะพบว่า `data` รับค่าของ `d` ซึ่งคือตำแหน่งที่อยู่ของอาเรย์ที่ `d` อ้างอิง นั่นหมายความว่า `data` ของ `sum` ก็จะอ้างอิงอาเรย์ตัวเดียวกับที่ `d` ของ `main` อ้างอิง ภายในเมท็อดจึงสามารถใช้งานอาเรย์เดียวกับที่ผู้เรียกใช้งาน เนื่องจากภาระของ `sum` คือการหาผลรวมของข้อมูลทุกตัวในอาเรย์ ก็เพียงแต่ไล่หยิบข้อมูลในแต่ละช่องมารวมกัน โดยใช้วงวน `for` (บรรทัดที่ 9) หยิบ `data[i]` มารวมในตัวแปร `sum` (บรรทัดที่ 10)

```
01 public class TestMethod {
02   public static void main(String[] args) {
03     int[] d = {1,3,5,7};
04     System.out.println( sum(d) );
05     System.out.println( sum(new int[]{1,2,3}) );
06   }
07   public static int sum(int[] data) {
08     int sum = 0;
09     for(int i=0; i<data.length; i++) {
10       sum += data[i];
11     }
12     return sum;
13   }
14 }
```



รหัสที่ 7-6 เมท็อดหาผลรวมของจำนวนเต็มในอาเรย์ที่ได้รับ

คำถามที่น่าสนใจคือ แล้วจะให้วนกี่รอบ นั่นคือจะรู้ได้อย่างไรว่า อาเรย์มีกี่ช่อง หลายคนอาจบอกว่า ถ้าดูโปรแกรมก็รู้แล้วว่า ผู้เรียกเขาส่ง `d` ที่อ้างอิงอาเรย์ที่มี 4 ช่อง ก็ให้วน 4 รอบ (`i=0,1,2,3`) แต่ต้องอย่าลืมว่า เราเขียนเมท็อด `sum` ไม่ได้เพื่อให้ บรรทัดที่ 4 เรียกที่เดียว บรรทัดที่ 5 ก็เรียก `sum` เหมือนกัน แต่ใช้วิธีสร้างอาเรย์ ตั้งค่าเริ่มต้น และส่งไปให้ `sum` เลย (ที่ส่งไปนั้นไม่ใช่ตัวอาเรย์ แต่เป็นตำแหน่งที่อยู่ของอาเรย์) ซึ่งในกรณีนี้มี 3 ช่อง อาเรย์ในจาวามีลักษณะพิเศษคือ หากต้องการรู้จำนวนช่องของอาเรย์ใด ให้ใส่ `.length` ตามหลังตัวแปรที่อ้างอิงอาเรย์นั้น ดังแสดงในบรรทัดที่ 9 เราเขียน `data.length` แทนจำนวนช่องของอาเรย์ `data` เพื่อกำหนดจำนวนรอบ การเรียก `sum` ในบรรทัดที่ 4 `data` อ้างอิงอาเรย์ที่มี 4 ช่อง `data.length` ก็มีค่าเป็น 4 ส่วนการเรียก `sum` ในบรรทัดที่ 5 `data.length` มีค่าเป็น 3

ต้องขอเน้นว่า สำหรับสตริง s จาวาใช้ s.length() ซึ่งเป็นเมท็อด (เพราะมีวงเล็บ) ที่คืนความยาวของสตริง s แต่ใช้ d.length ซึ่งให้ค่าความยาวของอาเรย์ d

การเปรียบเทียบอาเรย์

ขอนำเสนอตัวอย่างการเขียนเมท็อดเพื่อการประมวลผลอาเรย์ที่พบบ่อย เริ่มด้วยเมท็อดเพื่อเปรียบเทียบว่า อาเรย์สองแถวที่ได้รับเหมือนกันหรือไม่ เหมือนกันในที่นี้หมายความว่า มีขนาดเท่ากัน และมีข้อมูลแต่ละช่องเหมือนกันแบบช่องต่อช่องด้วย เนื่องจากเป็นเมท็อดที่ถามว่าเหมือนกันหรือไม่ จึงเป็นเมท็อดที่ได้ผลเป็น boolean และก็คงหนีไม่พ้นการใช้วงวน for เพื่อไล่เปรียบเทียบข้อมูลทีละตัว ตั้งแต่ช่องที่ 0 ไปเรื่อยๆ จนกว่าจะพบข้อมูลที่ไม่เหมือน จะคืน false แต่ถ้าพบว่า เหมือนหมดทุกช่อง จะคืน true ดังรหัสที่ 7-7

```
public static boolean equals(int[] a, int[] b) {
  if (a.length != b.length) return false;
  for (int i=0; i<a.length; i++) {
    if (a[i] != b[i]) return false;
  }
  return true;
}
```

รหัสที่ 7-7 เมท็อดเปรียบเทียบสองอาเรย์ว่าเหมือนกันหรือไม่

การหาค่ามากสุดในอาเรย์

ขอเขียนเมท็อดหาค่ามากสุดในอาเรย์สองแบบ แบบแรกให้คืนค่ามากสุด (เมท็อด max) อีกแบบให้คืนดัชนีของอาเรย์ที่เก็บค่ามากสุด (เมท็อด maxIndex) เช่น | 30 | 15 | 78 | 43 | มีค่ามากสุดคือ 78 และดัชนีที่เก็บค่ามากสุดคือ 2 เริ่มด้วยเมท็อด max ในรหัสที่ 7-8 ขอตั้งข้อกำหนดก่อนว่า หาก max ได้รับอาเรย์ที่มี 0 ช่อง จะถือว่าผิด (เพราะไม่รู้ว่าจะคืนอะไรดี) ดังนั้น จึงต้องตรวจสอบเงื่อนไขนี้ก่อน ในกรณีไม่ผิด เรามีตัวแปร max ไว้คอยเก็บค่ามากสุด เนื่องจากเราจะดูข้อมูลในอาเรย์จากซ้ายไปขวา จึงขอตั้งค่าเริ่มต้นให้ max=d[0] และใช้วงวนดูตั้งแต่ตัว d[1] ถึงตัวสุดท้าย หากตัวใหม่ d[i] มีค่ามากกว่า max (ซึ่งเก็บค่ามากสุดตั้งแต่ d[0] ถึง d[i-1]) ก็ควรเปลี่ยน max=d[i] วนเปรียบเทียบเช่นนี้จนครบ ออกจากวงวน max ย่อมเก็บค่ามากสุด

```
public static double max(double[] d) {
  if (d.length == 0) throw new IllegalArgumentException("0 ช่อง");
  double max = d[0];
  for (int i=1; i<d.length; i++) {
    if (d[i] > max) max = d[i];
  }
  return max;
}
```

รอบที่ i, max เก็บค่ามากสุดของข้อมูล d[0] ถึง d[i]

รหัสที่ 7-8 เมท็อดคืนค่ามากสุดในอาเรย์

เมท็อด maxIndex ในรหัสที่ 7-9 ก็มีหลักการทำงานคล้ายกัน แต่แทนที่จะใช้ตัวแปร max เก็บค่ามากสุด จะใช้ตัวแปร maxI เก็บดัชนีของช่องที่เก็บค่ามากสุด ซึ่งหมายความว่า d[maxI] ก็ย่อมเป็นค่ามากสุด ดังนั้น ในแต่ละรอบจึงเปรียบเทียบข้อมูลตัวใหม่ d[i] กับตัวที่มากสุดที่ผ่านมาคือ d[maxI] หากตัวใหม่มากกว่าก็จะเปลี่ยน maxI ให้เป็น i แทน ออกจากวงวนก็ให้คืน maxI กลับไป

```
public static int maxIndex(double[] d) {
  if (d.length == 0) throw new IllegalArgumentException("0 ช่อง");
  int maxI = 0;
  for (int i=1; i<d.length; i++) {
    if (d[i] > d[maxI]) maxI = i;
  }
  return maxI;
}
```

รหัสที่ 7-9 เมท็อดคืนดัชนีที่เก็บค่ามากสุดในอาเรย์

การค้นข้อมูลในอาเรย์

เมื่อมีการเก็บกลุ่มข้อมูล ก็ต้องมีการค้นข้อมูล เมท็อด search ในรหัสที่ 7-10 คืนดัชนีของอาเรย์ s ที่เก็บสตริงที่มีค่าเหมือนกับ x ในกรณีที่ค้นไม่พบ x ใน s ให้คืน -1 เราใช้วงวน for เพื่อไล่หยิบสตริงในแต่ละช่องมาเปรียบเทียบด้วย equals ว่าเหมือน x หรือไม่ (อย่าลืมว่าเราไม่ใช้ == เปรียบเทียบสตริง) ถ้าเหมือน ก็คืนดัชนี i ทันที ถ้าออกจากวงวน for แสดงว่า ค้นไม่พบ ก็คืน -1 ตามข้อกำหนด

```
public static int search(String[] s, String x) {
  for (int i=0; i<s.length; i++) {
    if (s[i].equals(x)) return i;
  }
  return -1;
}
```

รหัสที่ 7-10 เมท็อดค้นข้อมูลในอาเรย์

การสลับข้อมูลสองตัวในอาเรย์

ขอย้ำอีกครั้งว่า ตัวแปรอาเรย์ที่เมท็อดได้รับนั้นอ้างอิงไปยังอาเรย์แถวเดียวกับที่ผู้เรียกอ้างอิง ดังนั้น หากเราเปลี่ยนแปลงที่ช่องใดในอาเรย์ ก็จะเปลี่ยนแปลงอาเรย์ที่ผู้เรียกใช้อยู่ด้วย ในที่นี้เราต้องการเขียนเมท็อด swap ที่ให้บริการสลับข้อมูลตำแหน่ง i กับ j ในอาเรย์ d เช่น ถ้า d อ้างอิงอาเรย์ 30 | 15 | 43 การเรียก swap(d,0,2) จะทำให้อาเรย์เปลี่ยนเป็น 43 | 15 | 30 เขียนได้ดังรหัสที่ 7-11 เริ่มด้วยการทำสำเนาค่าของ d[i] ไปเก็บใน t จากนั้นย้าย d[j] มาเก็บ

แทนที่ d[i] แล้วจึงนำค่าของ d[i] ก่อนหน้านี้ที่เก็บใน t กลับมาใส่ใน d[j] เป็นการสลับ d[i] กับ d[j] นั่นเอง

```
public static void swap(double[] d, int i, int j) {
  double t = d[i];
  d[i] = d[j];
  d[j] = t;
}
```

รหัสที่ 7-11 เมท็อดสลับข้อมูลสองตัวในอาเรย์

หากเราเขียนเมท็อดสลับข้อมูลดังรหัสที่ 7-12 ซึ่งรับพารามิเตอร์ a กับ b สลับข้อมูลใน a กับ b ด้วยสามคำสั่งในทำนองเดียวกับที่ทำในรหัสที่ 7-11 ซึ่งจะสลับค่าของ a กับ b ได้จริง แต่จะไม่มีผลใด ๆ กับตัวแปรของผู้เรียก จากตัวอย่างที่แสดงในรหัสที่ 7-12 การส่งข้อมูล d[0] กับ d[2] ไปให้ swap สลับ เป็นการส่งค่าที่เก็บในช่อง 0 และ 2 ลงไปให้พารามิเตอร์ a กับ b ถึงแม้ a กับ b จะเปลี่ยน แต่ d[0] กับ d[2] ของผู้เรียกไม่เปลี่ยน

```
public static void main(String[] args) {
  double[] d = {10, 32, 11, 23};
  swap(d[0], d[2]);
}
public static void swap(double a, double b) {
  double t = a;
  a = b;
  b = a;
}
```

รหัสที่ 7-12 เมท็อดสลับข้อมูล (ที่ผิด)

การล้างข้อมูลในอาเรย์

คราวนี้ขอเขียนเมท็อด clear(int[] d) ที่รับอาเรย์ d มาเพื่อตั้งให้ทุกค่าในอาเรย์นี้เป็น 0 ให้หมด ก็ทำได้ไม่ยากด้วยการใช้วงวน for ใส่แต่ละช่องให้เป็น 0 ดังรหัสที่ 7-13

```
public static void clear(int[] d) {
  for (int i=0; i<d.length; i++) d[i] = 0;
}
```

รหัสที่ 7-13 เมท็อดล้างข้อมูลในอาเรย์

แต่ถ้ามาคิดดูให้ดี ทำไมเราไม่สร้างอาเรย์ใหม่ที่มีขนาดเท่ากับของ d ก็สิ้นเรื่อง เพราะการสร้างอาเรย์ใหม่จะได้ทุกช่องในแถวเป็น 0 โดยอัตโนมัติ ดังแสดงในรหัสที่ 7-14

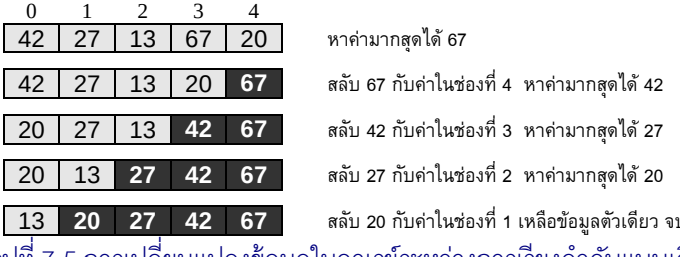
```
public static void clear(int[] d) {
  d = new int[d.length];
}
```

รหัสที่ 7-14 เมท็อดล้างข้อมูลในอาเรย์ (ที่ผิด)

ต้องบอกเลยว่า รหัสนี้ผิด ทำเช่นนี้ไม่ได้ การที่พารามิเตอร์ d เปลี่ยนค่า ไม่มีผลต่อตัวแปรอาเรย์ของผู้เรียก ถ้าผู้เรียกสร้างอาเรย์ `int[] a = {1,2,4}` จากนั้นเรียก `clear(a)` ซึ่งคือการส่งค่าใน a ให้กับ d ที่เป็นพารามิเตอร์ของ `clear` ตามที่ได้อธิบายมาก่อนหน้านี้ เรารู้ว่า a ของผู้เรียก กับ d ของเมท็อดจะอ้างอิงไปยังอาเรย์เดียวกัน การเปลี่ยนแปลง `d[i]` ใดๆ จะส่งผลต่อ `a[i]` ของผู้เรียก แต่ถ้าเราไปตั้งค่าใหม่ให้ `d = new int[d.length]` นั่นหมายความว่า d ได้เปลี่ยนการอ้างอิงจากที่เคยอ้างอิงอาเรย์เดียวกับของ a กลายเป็น d ไปอ้างอิงอาเรย์ใหม่ (แต่ a ก็ยังอ้างอิงแถวเดิม) เมื่อเมท็อด `clear` คืนการทำงาน ตัวแปร d ก็หาย อาเรย์ใหม่ที่สร้างก็ไม่มีใครอ้างอิง กลายเป็นขยะ แต่อาเรย์ที่อ้างอิงโดย a ก็ยังเหมือนเดิม ไม่เปลี่ยนแปลงแต่อย่างใด

การเรียงลำดับข้อมูลในอาเรย์

การเรียงลำดับข้อมูลเป็นกระบวนการที่สำคัญและต้องทำเป็นประจำในการประมวลผลข้อมูล เนื่องจากข้อมูลที่เรียงลำดับอย่างมีระเบียบ ทำให้การตีความ การหาความสัมพันธ์ของข้อมูลต่าง ๆ ทำได้ง่ายขึ้น ตัวอย่างเช่น การเรียงลำดับข้อมูลในอาเรย์ 30 | 15 | 78 | 43 จะได้ 15 | 30 | 43 | 78 เป็นผลลัพธ์ การเรียงลำดับข้อมูลในอาเรย์กระทำได้หลากหลายวิธี ขอนำเสนอวิธีที่ง่าย เหมาะกับปริมาณข้อมูลไม่มากนักชื่อว่า *การเรียงลำดับแบบเลือก* (selection sort) วิธีนี้อาศัยการเลือกข้อมูลตัวที่มีค่ามากสุดในกลุ่ม นำไปสลับกับข้อมูลตัวขวาสุดในกลุ่ม กระทำเช่นนี้ไปเรื่อย ๆ โดยลดขนาดของกลุ่มลงทีละหนึ่ง จนเหลือเพียงตัวเดียวก็เป็นอันเรียงลำดับเสร็จ ดังตัวอย่างในรูปที่ 7-5 แถวแรกแสดงข้อมูลของอาเรย์ตอนเริ่มต้น มีข้อมูลอยู่ 5 ตัว หาตัวมากสุดได้ค่า 67 อยู่ช่องที่ 3 ก็นำไปสลับกับช่องที่ 4, เหลือข้อมูล 4 ตัว หาตัวมากสุดได้ค่า 42 อยู่ช่องที่ 0 นำไปสลับกับช่องที่ 3, กระทำการหาตัวมากสุดเพื่อสลับกับตัวท้ายของกลุ่มที่เหลือ จะได้ผลลัพธ์ดังแถวล่างสุด



รูปที่ 7-5 การเปลี่ยนแปลงข้อมูลในอาเรย์ระหว่างการเรียงลำดับแบบเลือก

รหัสที่ 7-15 แสดงโปรแกรมสาธิตการเรียงลำดับข้อมูล ประกอบด้วยเมท็อด `main` ที่สร้างอาเรย์ สั่งให้เรียงลำดับ ตามด้วยการแสดงผล, เมท็อด `maxIndex(d,left,right)` คืนดัชนี (ระหว่าง `left` ถึง `right`) ของอาเรย์ d ที่เก็บค่ามากสุด, เมท็อด `swap(d,i,j)` ทำหน้าที่สลับข้อมูลที่ดัชนี i กับ j ในอาเรย์ d, เมท็อด `print(d)` แสดงข้อมูลทั้งหมดในอาเรย์ d ออกทางจอภาพ, และสุดท้ายคือเมท็อด `selectionSort(d)` ทำหน้าที่เรียงลำดับข้อมูลในอาเรย์ d, การทำงานของเมท็อด `selectionSort` อาศัยวงวน `for` ในบรรทัดที่ 8 มีตัวแปร k เก็บ

ดัชนีของอาเรย์ d เริ่มต้นที่ช่องขวาสุด ลดลงรอบละหนึ่งตำแหน่งมาทางซ้าย จนจบที่ช่องก่อนซ้ายสุด ในแต่ละรอบ ข้อมูลใน d ที่สนใจคือ ตั้งแต่ช่องที่ 0 ถึง k การเรียงลำดับแบบเลือกทำการเลือกตัวมากสุดในกลุ่มเพื่อสลับกับตัวขวาสุดในกลุ่ม นั่นคือใช้ maxIndex หาดัชนีของตัวมากสุดในกลุ่ม (บรรทัดที่ 9) ตามด้วยการเรียก swap เพื่อสลับข้อมูลมากสุดในช่องนั้นกับตัวขวาสุดของกลุ่ม ซึ่งคือตัวที่ช่อง k จบการเรียงลำดับเมื่อเหลือข้อมูลเพียงตัวเดียว (k=0) [1]

```
01 public class Sorting {
02   public static void main(String[] args) {
03     double[] data = {9,3,2,4,1,5,6};  // สร้างข้อมูลทดสอบ
04     selectionSort(data);              // เรียงลำดับข้อมูลใน data
05     print(data);                      // แสดงผลการเรียงลำดับ
06   }
07   public static void selectionSort(double[] d) {
08     for (int k=d.length-1; k>0; k--) {
09       int maxI = maxIndex(d, 0, k);
10       swap(d, maxI, k);
11     }
12   }
13   private static int maxIndex(double[] d, int left, int right) {
14     if (left > right) throw new IllegalArgumentException("0 ช่อง");
15     int maxI = left;
16     for (int i=left+1; i<=right; i++) {
17       if (d[i] > d[maxI]) maxI = i;
18     }
19     return maxI;
20   }
21   private static void swap(double[] d, int i, int j) {
22     double t = d[i];
23     d[i] = d[j];
24     d[j] = t;
25   }
26   private static void print(double[] d) {
27     System.out.print("[");
28     for (int i=0; i<d.length; i++) {
29       System.out.print(d[i]);
30       if (i<d.length-1) System.out.print(",");
31     }
32     System.out.println("]");
33   }
34 }
```

การเรียงลำดับข้อมูลแบบเลือก

คืนดัชนีระหว่าง left ถึง right ของช่องที่เก็บค่ามากสุดใน d

สลับข้อมูลช่องที่ i กับ j ในแถวลำดับ d

แสดงข้อมูลทั้งหมดของแถวลำดับทางจอภาพ

รหัสที่ 7-15 เมท็อดเรียงลำดับข้อมูลในอาเรย์

[1] จาวามีคลาสมาตรฐานชื่อ Arrays ที่มีเมท็อดประจำคลาสชื่อ sort ให้บริการเรียงลำดับข้อมูล อ่านรายละเอียดได้ในหัวข้อเพิ่มเติมท้ายบท

เมท็อดที่คืนอาเรย์

เราได้ทราบมาแล้วว่า ตัวแปรเฉพาะที่ต่าง ๆ ที่ประกาศในเมท็อดจะมีให้ใช้โดยอัตโนมัติเมื่อเมท็อดถูกเรียก และจะหายไปโดยอัตโนมัติเช่นกันเมื่อเมท็อดทำงานเสร็จ แต่สำหรับอาเรย์ที่เราสร้างในเมท็อดนั้น จะไม่หาย ถ้าเราคืนอาเรย์นั้นเป็นผลลัพธ์ของเมท็อดกลับไปให้ผู้เรียกใช้ตัวแปรอ้างอิงมัน ความต้องการคืนอาเรย์เป็นผลลัพธ์ของเมท็อดมีหลากหลายสถานการณ์ดังตัวอย่างที่จะนำเสนอในหัวข้อย่อยต่อไปนี้

การขยายอาเรย์

การสร้างอาเรย์ต้องกำหนดประเภทข้อมูลและขนาด โดยหลังจากสร้างแล้วไม่สามารถเพิ่มหรือลดขนาดของอาเรย์ได้ แต่ถ้ามีความจำเป็นต้องเปลี่ยนขนาด ต้องใช้วิธีการสร้างใหม่ ทำสำเนา และเปลี่ยนการอ้างอิง รหัสที่ 7-16 แสดงเมท็อด `doubleCapacity(int[] d)` ที่เริ่มด้วยการสร้างอาเรย์ใหม่ชื่อ `a` มีขนาดเป็นสองเท่าของ `d` จากนั้นทำสำเนาทุกตัวใน `d` ไปเก็บใน `a` ตามด้วยการคืน `a` เป็นผลลัพธ์ของเมท็อด สมมติว่า เรามี `int[] x = {1,2,3};` แล้วเรียก `x = doubleCapacity(x)` ทำให้ `x` ไปอ้างอิงอาเรย์ใหม่ที่คืนมามีขนาด 6 ช่อง สามช่องซ้ายเก็บค่าเหมือนของเดิม (อาเรย์เดิมไม่มีใครอ้างอิง ก็จะกลายเป็นขยะ)

```
public static int[] doubleCapacity(int[] d) {
  int[] a = new int[2*d.length];
  for (int i=0; i<d.length; i++) {
    a[i] = d[i];
  }
  return a;
}
```

สร้างแถวลำดับใหม่ใหญ่เป็นสองเท่า

ทำสำเนาจากแถวลำดับเก่าไปเก็บในของใหม่

รหัสที่ 7-16 เมท็อดขยายขนาดของอาเรย์เป็นสองเท่า

การคืนผลลัพธ์หลายตัว

เราเคยเขียนโปรแกรมหารากของสมการกำลังสอง $ax^2 + bx + c = 0$ ในบทที่ 2 และบทที่ 4 ถ้า $a \neq 0$ และ $b^2 > 4ac$ รากของสมการจะเป็น $(-b \pm \sqrt{b^2 - 4ac}) / (2a)$ นั่นคือ มีราก 2 ค่า หากเราต้องการเขียนเมท็อด `quadraticRoot(a,b,c)` หาราก และต้องการให้คืนรากทั้งสองให้ผู้เรียกจะทำอย่างไร เพราะจาวาให้คืนได้ค่าเดียว ในกรณีเช่นนี้ อาจใช้อาเรย์ช่วยได้สองวิธีดังนี้

วิธีที่หนึ่ง เขียนให้เมท็อดนี้รับพารามิเตอร์เพิ่มอีกตัวเป็นอาเรย์ 2 ช่อง เพื่อให้เมท็อดเติมรากทั้งสองไว้ในนั้น และไม่ต้องคืนอะไร เพราะผู้เรียกเป็นผู้ส่งอาเรย์มาให้ ผู้เรียกจึงสามารถหยิบอาเรย์ที่เก็บรากไปใช้ได้เมื่อเมท็อดทำงานเสร็จ ดังรหัสที่ 7-17

วิธีที่สอง เราสร้างเมท็อดที่รับ a, b, c และคืนอาเรย์แบบ double ภายในเมท็อดสร้างอาเรย์แบบ double ขนาด 2 ช่องเพื่อเก็บรากในแต่ละช่อง และคืนกลับไปเป็นผลลัพธ์ ดังรหัสที่ 7-18 โดยผู้เรียกต้องนำผลเก็บใส่ตัวแปรอ้างอิงอาเรย์แบบ double เช่น ทำคำสั่ง double[] r = quadraticRoot(1,-5,6); จะได้ r[0] มีค่า 3 และ r[1] มีค่า 2

```
public static void quadraticRoot(double a, double b, double c,
                                  double[] root) {
  double t = Math.sqrt(b*b - 4*a*c);
  root[0] = (-b + t)/(2*a);
  root[1] = (-b - t)/(2*a);
}
```

ผู้เรียกต้องส่งแถวลำดับเพื่อเก็บผลลัพธ์มาให้

รหัสที่ 7-17 เมท็อดหารากของสมการ $ax^2 + bx + c = 0$ (แบบที่ 1)

```
public static double[] quadraticRoot(double a,double b,double c){
  double t = Math.sqrt(b*b - 4*a*c);
  double[] root = new double[2];
  root[0] = (-b + t)/(2*a);
  root[1] = (-b - t)/(2*a);
  return root;
}
```

สร้างแถวลำดับเก็บผลลัพธ์สำหรับรากทั้งสอง

รหัสที่ 7-18 เมท็อดหารากของสมการ $ax^2 + bx + c = 0$ (แบบที่ 2)

การกรองข้อมูล

ในบางสถานการณ์ เราอาจมีความต้องการที่จะนำกลุ่มข้อมูลในอาเรย์มาคัดกรองข้อมูลที่ตรงตามเงื่อนไขที่กำหนดไว้ล่วงหน้า เช่น ต้องการเลือกเฉพาะข้อมูลในอาเรย์ที่มีค่าเป็นบวกมาใช้ ก็สามารถทำได้ไม่ยากโดยอาศัยอาเรย์อีกแถวไว้เก็บผลลัพธ์ เพราะข้อมูลที่ตรงตามเงื่อนไขอาจมีเกิน 1 ตัว หรืออาจไม่มีเลยก็ได้ เขียนเป็นเมท็อดได้ดังรหัสที่ 7-19

```
public static double[] getOnlyPositive(double[] d) {
  int n = 0;
  for(int i=0; i<d.length; i++) {
    if (d[i] > 0) n++;
  }
  double[] out = new double[n];
  for(int i=0, k=0; i<d.length; i++) {
    if (d[i] > 0) {
      out[k] = d[i];
      k++;
    }
  }
  return out;
}
```

วนนับจำนวนข้อมูลที่มีค่าเป็นบวก

สร้างแถวลำดับเก็บผลลัพธ์

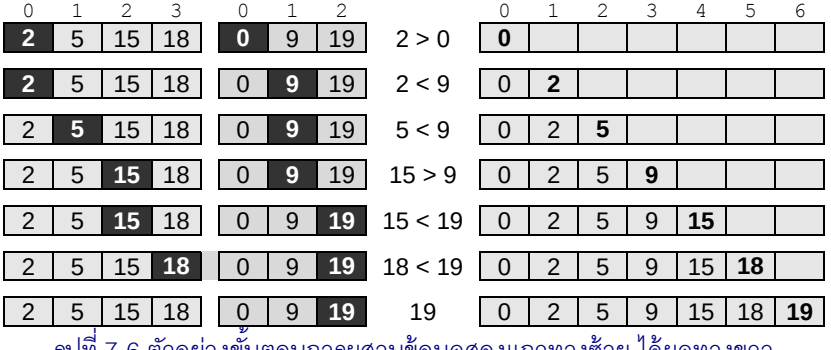
วนเก็บใส่แถวลำดับผลลัพธ์ เมื่อพบข้อมูลที่เป็นบวก

รหัสที่ 7-19 เมท็อดเลือกข้อมูลในอาเรย์ที่มีค่าเป็นบวก

พิจารณารหัสที่ 7-19 จะพบว่า มีการใช้วงวน for วนในอาเรย์ที่ได้รับสองรอบ ทั้งนี้เพราะการสร้างอาเรย์เพื่อเก็บผลลัพธ์นั้นต้องระบุขนาด แต่เรายังไม่ทราบเลยว่า มีข้อมูลที่เป็นจำนวนบวกอยู่กี่ตัว ดังนั้น จึงใช้วงวน for แรกวนนับจำนวนข้อมูลที่เป็นบวก เก็บในตัวแปร n เมื่อทราบแล้ว ก็สร้างอาเรย์ out เพื่อเก็บผลลัพธ์จำนวน n ช่อง จากนั้นวนใหม่อีกรอบ คราวนี้ข้อมูลตัวใดเกิน 0 ก็เก็บใส่อาเรย์ out โดยมีตัวแปร i เป็นดัชนีของ d ซึ่งเพิ่มตามวงวน for ในขณะที่ตัวแปร k เป็นดัชนีของ out เพิ่มทุกครั้งที่มีข้อมูลเก็บใส่ out

การผสานข้อมูล

กำหนดให้มีอาเรย์สองแถว d1 กับ d2 ข้อมูลในสองแถวนี้เรียงลำดับจากน้อยไปมาก สิ่งที่ต้องการทำคือ นำข้อมูลในสองแถวนี้มารวมกันในอาเรย์ใหม่ชื่อ d3 ให้ข้อมูลเรียงจากน้อยไปมากด้วย เช่น มี d1 [2|5|15|18] กับ d2 [0|9|19] ผสานแล้วจะได้ d3 [0|2|5|9|15|18|19] รูปที่ 7-6 แสดงตัวอย่างขั้นตอนการผสานข้อมูลสองแถวทางซ้าย ได้ผลเก็บในแถวทางขวา เริ่มจากนำตัวซ้ายของแต่ละชุดมาเปรียบเทียบกัน นำตัวน้อยกว่าไปใส่ในผลลัพธ์ แล้วเลื่อนไปสนใจข้อมูลตัวถัดไป (ตัวที่มากกว่ายังคงไว้) กระทำการเปรียบเทียบ และนำตัวน้อยไปต่อท้ายผลลัพธ์เช่นนี้จนชุดใดชุดหนึ่งหมด แล้วนำข้อมูลที่เหลือในอีกชุดไปต่อท้ายผลลัพธ์ให้หมด เขียนเป็นเมท็อดได้ดังรหัสที่ 7-20



รูปที่ 7-6 ตัวอย่างขั้นตอนการผสานข้อมูลสองแถวทางซ้าย ได้ผลทางขวา

รหัสที่ 7-20 เริ่มด้วยการสร้างอาเรย์ใหม่ d3 มีขนาดเท่ากับของ d1 รวมกับ d2 จากนั้นตั้งตัวแปร i1 และ i2 เพื่อเป็นดัชนีของ d1 และ d2 ตามลำดับ ให้ทั้งคู่มีค่า 0 เพราะจะเริ่มจากซ้ายของ d1 และ d2 เข้าวงวน for เพื่อผสานข้อมูล โดยมี i3 เป็นดัชนีของ d3 เริ่มที่ 0 เพิ่มค่าทีละหนึ่ง จนถึงขวาสุดของ d3 การทำงานในวงวนมีสามแบบขึ้นกับค่าของ i1 กับ i2 ดังนี้

- กรณีที่ 1 : i1 และ i2 ยังไม่เกินตัวขวาของ d1 และ d2 ให้นำตัวน้อยของ d1[i1] กับ d2[i2] ไปใส่ในผลลัพธ์ d3[i3] เมื่อนำข้อมูลของแถวใดไปใส่ในผลลัพธ์ให้เพิ่มดัชนีของแถวนั้น เพื่อขยับไปทางขวาของแถวนั้น ให้สังเกตการเขียน d1[i1++] กับ d2[i2++] ที่เราเขียนลัด คือให้นำค่าตัวแปรดัชนีมาใช้ก่อน แล้วค่อยเพิ่มค่า

- กรณีที่ 2 : `i2` มีค่าเกินตัวขวาของ `d2` แต่ `i1` ยังไม่เกินของ `d1` ให้นำ `d1[i1]` ไปใส่ใน `d3[i3]` แล้วก็เพิ่ม `i1` สำหรับรอบถัดไป
- กรณีที่ 3 : กรณีนี้ตรงข้ามกับกรณีที่ 2 คือ `i1` มีค่าเกินตัวขวาของ `d1` แต่ `i2` ยังไม่เกินของ `d2` ให้นำ `d2[i2]` ไปใส่ใน `d3[i3]` แล้วก็เพิ่ม `i2` สำหรับรอบถัดไป

```
public static double[] merge(double[] d1, double[] d2) {
  double[] d3 = new double[d1.length + d2.length];
  int i1 = 0, i2 = 0;
  for (int i3 = 0; i3 < d3.length; i3++) {
    if (i1 < d1.length && i2 < d2.length) {
      if (d1[i1] < d2[i2]) {
        d3[i3] = d1[i1++];
      } else {
        d3[i3] = d2[i2++];
      }
    } else {
      if (i1 < d1.length) {
        d3[i3] = d1[i1++];
      } else {
        d3[i3] = d2[i2++];
      }
    }
  }
  return d3;
}
```

ทั้ง i1 และ i2 ยังไม่เกินตัวขวาของ d1 และ d2

กรณีที่ 1 : ย้ายตัวน้อยของ d1[i1] กับ d2[i2] ไปใส่ d3[i3] และเลื่อนไปทางขวาหนึ่งตำแหน่ง

กรณีที่ 2 : d2 หมดแล้ว ย้ายของ d1 ไป d3

กรณีที่ 3 : d1 หมดแล้ว ย้ายของ d2 ไป d3

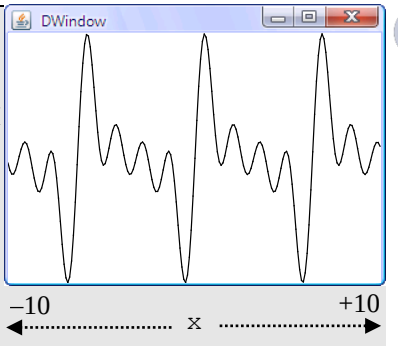
รหัสที่ 7-20 เมท็อดผสานข้อมูลในอาเรย์สองแถวให้เป็นหนึ่ง

โปรแกรมวาดกราฟเส้น

รหัสที่ 7-21 แสดงตัวอย่างการใช้งานเมท็อด `plot(w, h, x, y)` ที่นำข้อมูลในอาเรย์ `x` และ `y` ไปวาดกราฟเส้นบนวินโดว์ขนาด `w×h` เมท็อด `main` สร้างอาเรย์ `x` มี 200 ช่องเก็บค่าตั้งแต่ -10 ถึง 10 (ห่างกันจุดละ 0.1) สร้างอาเรย์ `y` อีก 200 ช่อง ให้ `y[i]` มีค่าเป็นผลรวมค่าไซน์ที่ความถี่ต่างกัน 4 ค่า ด้วยฟังก์ชัน $y_i = \sin(x_i)+\sin(2x_i)+\sin(3x_i)+\sin(4x_i)$ (บรรทัดที่ 10 ถึง 12) ที่เลือกฟังก์ชันนี้ไม่มีอะไรนอกจากว่าได้กราฟเส้นที่สวยดี จากนั้นใช้เมท็อด `plot` วาดบนวินโดว์ขนาดกว้าง 300 สูง 200 จุดภาพ (บรรทัดที่ 14)

สิ่งที่เมท็อด `plot` ทำมีสองส่วน ส่วนแรกสร้างวินโดว์และตั้งค่าให้กับตัวแปรเสริมจำนวนหนึ่งซึ่งเกี่ยวกับการวาดเส้นกราฟ (บรรทัดที่ 18 ถึง 22) ส่วนหลังคือการลากเส้นกราฟจาก `(x[0],y[0])` ไปยัง `(x[1],y[1])`, จาก `(x[1],y[1])` ไปยัง `(x[2],y[2])`, ต่อ ๆ กันในลักษณะนี้จนถึงเส้นสุดท้าย ด้วยวงวน `for` (บรรทัดที่ 23 ถึง 29)

```
01 import jlab.graphics.DWindow;
02
03 public class LineGraph {
04   public static void main(String[] args) {
05     int n = 200;
06     double[] x = new double[n];
07     double[] y = new double[n];
08     for (int i = 0; i < x.length; i++) {
09       x[i] = -10 + i * 0.1;
10       for (int k = 1; k <= 4; k++) {
11         y[i] += Math.sin(k * x[i]);
12       }
13     }
14     plot(300, 200, x, y);
15   }
16   public static void plot(int width, int height,
17                           double[] x, double[] y) {
18     DWindow w = new DWindow(width, height);
19     double xmin = min(x);
20     double xmax = max(x);
21     double ymin = min(y);
22     double ymax = max(y);
23     for (int i = 0; i < x.length - 1; i++) {
24       double sx0 = toScreen(x[i],   xmin, xmax, width);
25       double sx1 = toScreen(x[i+1], xmin, xmax, width);
26       double sy0 = toScreen(y[i],   ymax, ymin, height);
27       double sy1 = toScreen(y[i+1], ymax, ymin, height);
28       w.drawLine(w.BLACK, sx0, sy0, sx1, sy1);
29     }
30   }
31   private static double toScreen(double v, double min,
32                                  double max, double length) {
33     return (v - min) * (length-1) / (max - min);
34   }
..   public static double max(double[] d) { ... } // รหัสที่ 7-8
..   public static double min(double[] d) { ... } // ลองเขียนเอง
.. }
```



$$y[i] = \sum_{k=1}^{4} \sin(kx[i])$$

เปลี่ยนพิกัดของข้อมูลที่ได้รับ เป็นพิกัดบนวินโดว์

รหัสที่ 7-21 โปรแกรมวาดกราฟเส้น

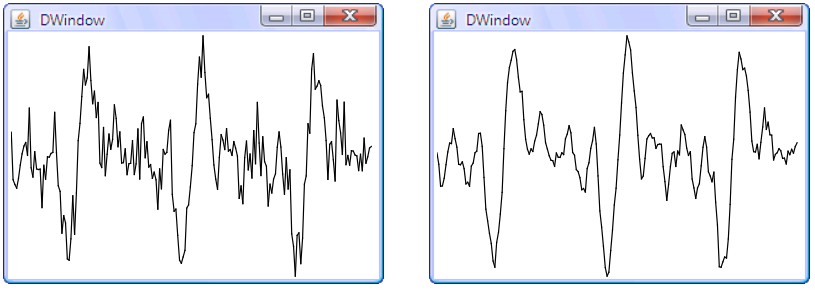
เนื่องจากเราต้องการให้ `plot` รู้จักปรับขนาดของค่า `x` และ `y` ให้อยู่ในช่วงกว้างและสูงของวินโดว์ได้อย่างพอดี ซึ่งทำได้ดังนี้ กำหนดให้ x_{min} และ x_{max} เป็นค่าน้อยสุดและมากสุดในอาเรย์ `x` และให้ y_{min} และ y_{max} เป็นค่าน้อยสุดและมากสุดในอาเรย์ `y` ให้ *width* และ *height* เป็นความกว้างและความสูงของพื้นที่วาดรูปในวินโดว์ ดังนั้น เราต้องหาฟังก์ชันเปลี่ยน x_{min} ให้เป็น 0, x_{max} ให้เป็น $width - 1$, y_{min} ให้เป็น $height - 1$, และสุดท้าย y_{max} ให้เป็น 0 (อย่าลืมว่าพิกัดของจุดในวินโดว์นั้น 0 อยู่ด้านบน เพิ่มค่าเมื่อลงล่าง) ดังนั้น x_i และ y_i ใด ๆ สามารถเปลี่ยนเป็นพิกัดวินโดว์ได้ดังนี้

$$sx_i = (x_i - x_{min}) \times (width - 1) / (x_{max} - x_{min}) \quad \text{และ} \quad sy_i = (y_i - y_{max}) \times (height - 1) / (y_{min} - y_{max})$$

บรรทัดที่ 19 ถึง 22 ตั้งค่าให้กับตัวแปร `xmin`, `xmax`, `ymin`, `ymax` โดยอาศัยเมท็อด `min` และ `max` ซึ่งคืนค่าน้อยสุดและมากสุดของอาเรย์ที่ได้รับ ค่าเหล่านี้จะใช้ในการแปลงค่าบนกราฟเส้นเป็นพิกัดบนวินโดว์ เราเขียนแยกเป็นเมท็อด `toScreen(v, min, max, length)` ที่เปลี่ยนค่า `v` ที่อยู่ในช่วง `min` ถึง `max` ไปเป็นช่วง `0` ถึง `length-1` ซึ่งใช้ได้ทั้งกับพิกัด `x` และ `y` ค่า `length` คือความกว้างวินโดว์เมื่อส่ง `x` ให้กับ `v` และคือความสูงวินโดว์เมื่อส่ง `y` ให้กับ `v` สำหรับกรณีที่แปลงค่า `y` ให้สลับการส่ง `ymax` ให้ `min` และ `ymin` ให้ `max` ก็จะได้ตรงตามสูตรข้างต้น บรรทัดที่ 24 ถึง 27 ใช้ `toScreen` แปลงค่าจุดบนกราฟเส้น เพื่อนำไปลากเส้นด้วย `drawLine` ในบรรทัดที่ 28 เพียงเท่านี้ก็ได้เมท็อดที่ใช้วาดกราฟเส้นที่รับอาเรย์ `x` และ `y` ซึ่งมีขนาดกี่จุดก็ได้ ไม่จำกัดช่วงของค่าของข้อมูล เพราะจะปรับให้พอดีกับขนาดของวินโดว์โดยอัตโนมัติ

ค่าเฉลี่ยเคลื่อนที่

ข้อมูลอนุกรมเวลาคือข้อมูลที่วัดได้ต่อเนื่องตามเวลาในช่วงเวลาที่เท่า ๆ กัน เช่น ราคาหุ้นรายวัน ปริมาณสินค้าที่ขายได้ในแต่ละสัปดาห์ จำนวนประชากรในรอบร้อยปีที่ผ่านมา สัญญาณเสียง เป็นต้น การวิเคราะห์ข้อมูลอนุกรมเวลามักอาศัยการหา*ค่าเฉลี่ยเคลื่อนที่* (moving average) เพื่อให้เห็นแนวโน้มการเปลี่ยนแปลงของข้อมูล เสมือนกับการปรับให้กราฟเส้นของข้อมูลอนุกรมเวลาให้เรียบขึ้น ในงานการประมวลผลสัญญาณ (เช่น เสียง ภาพ สัญญาณไฟฟ้า) การหาค่าเฉลี่ยเคลื่อนที่ทำหน้าที่เสมือนตัวกรองผ่านต่ำ (low-pass filter) ที่กันสัญญาณความถี่สูง ยอมให้แต่สัญญาณความถี่ต่ำผ่านเท่านั้น ดังตัวอย่างในรูปที่ 7-7 แสดงกราฟเส้นทางซ้ายที่แทนข้อมูลต้นฉบับ ซึ่งเมื่อนำไปหาค่าเฉลี่ยเคลื่อนที่จะได้กราฟเส้นดังรูปขวา



รูปที่ 7-7 ตัวอย่างการหาค่าเฉลี่ยเคลื่อนที่ของข้อมูลอนุกรมเวลา

ค่าเฉลี่ยเคลื่อนที่มีหลายแบบ ขอเสนอแบบที่ง่ายสุด คือการนำข้อมูลที่ติดกัน 3 จุดมาเฉลี่ย ถ้าข้อมูลเก็บในอาเรย์ `y` ค่าเฉลี่ยเคลื่อนที่ที่ตำแหน่ง `i` จะมีค่า `(y[i-1]+y[i]+y[i+1])/3` สำหรับตำแหน่งแรกกับตำแหน่งท้าย มีข้อมูลไม่ครบสามตัว ก็หาค่าเฉลี่ยสองตัวพอ นั่นคือตำแหน่ง `0` จะได้ `(y[0]+y[1])/2` ส่วนตำแหน่ง `n-1` จะได้ค่า `(y[n-2]+y[n-1])/2` เมื่อ `n` คือจำนวนข้อมูล เขียนเป็นเมท็อด `getMovingAverage` ในรหัสที่ 7-22 เมท็อดนี้รับอาเรย์ `y` และ

คืนอาเรย์ใหม่ ซึ่งเก็บค่าเฉลี่ยเคลื่อนที่ของ y เริ่มด้วยการสร้างอาเรย์ out เก็บผลลัพธ์มีขนาดเท่ากับของ y จากนั้นใช้วงวน for หาค่าเฉลี่ยเคลื่อนที่ของช่องดัชนีที่ 1 ถึง n-2 (แบบใช้ 3 ค่า) ออกจากวงวนคำนวณของดัชนีที่ 0 กับที่ n-1 (แบบใช้ 2 ค่า) แล้วก็คืนผล

```
public static double[] getMovingAverage(double[] y) {
  int n = y.length;
  double[] out = new double[n];
  for (int i = 1; i <= n-2; i++) {
    out[i] = (y[i - 1] + y[i] + y[i + 1]) / 3;
  }
  out[0] = (y[0] + y[1]) / 2;
  out[n - 1] = (y[n - 2] + y[n - 1]) / 2;
  return out;
}
```

หาค่าเฉลี่ยของข้อมูล 3 ตัวที่ติดกัน

สำหรับข้อมูลซ้ายสุดกับขวาสุด

รหัสที่ 7-22 เมท็อดหาค่าเฉลี่ยเคลื่อนที่

รหัสที่ 7-23 แสดงการทดสอบด้วยการสร้างชุดข้อมูลเหมือนในหัวข้อที่แล้ว แต่เติมจำนวนสุ่มไปเล็กน้อยด้วย Math.random() เสมือนเป็นสัญญาณรบกวน จากนั้นไปหาค่าเฉลี่ยเคลื่อนที่ได้ผลมาก็ไปวาดกราฟเส้น (และวาดกราฟเส้นของข้อมูลชุดเดิมด้วย) ได้ผลคล้ายกับรูปที่ 7-7 ที่เห็นก่อนหน้านี้

```
public static void main(String[] args) {
  double[] x = new double[200];
  double[] y = new double[200];
  for (int i = 0; i < x.length; i++) {
    x[i] = -10 + i * 0.1;
    for (int k = 1; k <= 4; k++) {
      y[i] += Math.sin(k*x[i]);
    }
    y[i] += Math.random(); // เพิ่มสัญญาณรบกวน
  }
  plot(300, 200, x, y);                        // วาดเส้นที่มีสัญญาณรบกวน
  plot(300, 200, x, getMovingAverage(y)); // วาดเส้นค่าเฉลี่ยเคลื่อนที่
}
```

รหัสที่ 7-23 ตัวอย่างโปรแกรมวาดกราฟเส้น และเส้นค่าเฉลี่ยเคลื่อนที่

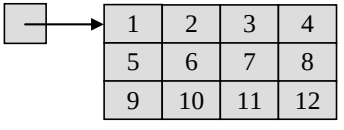
อาเรย์หลายมิติ

นอกจากเราจะสามารถเก็บข้อมูลตามช่องต่าง ๆ ที่เรียงกันในอาเรย์แล้ว เรายังสามารถเก็บข้อมูลในอาเรย์ที่มีช่องต่าง ๆ เรียงกันเป็นตาราง หรือที่เรียกว่า อาเรย์สองมิติก็ได้ (จึงเรียกอาเรย์ที่อธิบายตั้งแต่ต้นบทว่า อาเรย์หนึ่งมิติ) หากเรามองอาเรย์หนึ่งมิติเสมือนเป็นเวกเตอร์ เช่น เราแทนเวกเตอร์ $v = (1,3,5,7)$ ด้วยอาเรย์ $\boxed{1|3|7|5}$ เราก็คงมองอาเรย์สองมิติเสมือนเป็นเมทริกซ์

$m = \begin{bmatrix} 1 & 0 & 3 \\ 3 & 2 & 4 \end{bmatrix}$ หรือมองเสมือนเป็นแผ่นตารางทำงาน (spreadsheet) ที่ผู้อ่านคงเคยใช้ จะมีช่องต่าง ๆ เรียงตามแนวนอนและแนวตั้ง โดยที่เราสามารถเก็บข้อมูลในช่องต่าง ๆ ได้ ในกรณีของอาเรย์สามมิติ ก็อาจมองเสมือนแผ่นตารางทำงานหลาย ๆ แผ่น มิติแรกแทนว่า จะใช้แผ่นใด มิติต่อมาแทนหมายเลขแถว และมิติสุดท้ายแทนหมายเลขสดมภ์

หากต้องการสร้างอาเรย์สองมิติชื่อ `m` ขนาด 3×4 (3 แถว 4 สดมภ์) เก็บจำนวนเต็ม ก็ใช้คำสั่ง `int[][] m = new int[3][4];` และถ้าต้องการอาเรย์สามมิติชื่อ `p` ขนาด 2×3×4 เก็บจำนวนเต็ม ก็ใช้ `int[][][] p = new int[2][3][4];` การตั้งค่าเริ่มต้นทำได้ด้วยรายการเริ่มต้นเช่นเดียวกับกรณีของอาเรย์หนึ่งมิติ แต่ต้องเขียนเป็นรายการของรายการซ้อนกันดังตัวอย่างในรูปที่ 7-8 [2]

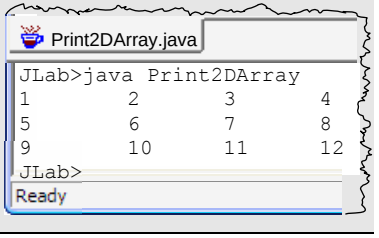
```
int[][] m = { {1,  2,  3,  4},
              {5,  6,  7,  8},
              {9, 10, 11, 12} };
```



รูปที่ 7-8 ตัวอย่างการสร้างและตั้งค่าเริ่มต้นให้กับอาเรย์สองมิติ

การใช้ข้อมูลในอาเรย์หลายมิติ ก็อาศัยดัชนีเช่นกัน แต่ต้องกำหนดให้เท่ากับจำนวนมิติ เช่น จากรูปที่ 7-8 `m[0][0]` คือแถว 0 สดมภ์ 0 เก็บค่า `1` และ `m[2][3]` คือแถว 2 สดมภ์ 3 เก็บค่า `12` เป็นต้น และอย่าลืมตั้งค่าดัชนีให้อยู่ในช่วง มิฉะนั้นจะเกิดข้อผิดพลาด ถ้าต้องการทราบจำนวนแถวของ `m` ใช้ `m.length` ส่วนจำนวนสดมภ์ของแถว `r` ของ `m` ใช้ `m[r].length` โดยทั่วไปจะเขียน `m[0].length` เพื่อระบุจำนวนสดมภ์ (เมื่อทุกแถวมีจำนวนสดมภ์เท่ากัน) [3] รหัสที่ 7-24 คือเมท็อดแสดงค่าของอาเรย์สองมิติ อาศัยวงวน `for` สองวงซ้อนกัน วงนอกเปลี่ยนดัชนีของแถว วงในเปลี่ยนดัชนีของสดมภ์เพื่อแสดงข้อมูลในแต่ละแถว

```
public static void print(int[][] m) {
  for (int i=0; i<m.length; i++) {
    for (int j=0; j<m[0].length; j++) {
      System.out.print(m[i][j] + "\t ");
    }
    System.out.println();
  }
}
```



รหัสที่ 7-24 เมท็อดแสดงข้อมูลในอาเรย์สองมิติ

[2] อาเรย์หลายมิติก็ต้องมีตัวแปรอ้างอิงเช่นกัน รายละเอียดโครงสร้างของอาเรย์หลายมิตินั้นซับซ้อนกว่าแบบหนึ่งมิติ ผู้อ่านที่สนใจต้องอ่านหัวข้อเพิ่มเติมท้ายบท

[3] จาวาอนุญาตให้สร้างอาเรย์สองมิติที่แต่ละแถวมีจำนวนสดมภ์ไม่เท่ากัน จะขออธิบายในหัวข้อเพิ่มเติมท้ายบท

การประมวลผลเมทริกซ์

เมทริกซ์แทนด้วยอาเรย์สองมิติ หัวข้อนี้นำเสนอตัวอย่างการเขียนเมท็อดที่เกี่ยวกับเมทริกซ์ ได้แก่ การบวกเมทริกซ์ การคูณเมทริกซ์กับเวกเตอร์ และการคูณเมทริกซ์กับเมทริกซ์

เริ่มด้วยการบวก ให้ A กับ B เป็นเมทริกซ์ที่มีมิติเท่ากัน ถ้า $C = A + B$ จะได้ $c_{i,j} = a_{i,j} + b_{i,j}$ เช่น $\begin{bmatrix}2 & 3 & 4\\5 & 6 & 7\end{bmatrix}+\begin{bmatrix}0 & 1 & 0\\1 & 0 & 1\end{bmatrix}=\begin{bmatrix}2+0 & 3+1 & 4+0\\5+1 & 6+0 & 7+1\end{bmatrix}=\begin{bmatrix}2 & 4 & 4\\6 & 6 & 8\end{bmatrix}$ เขียนเมท็อดการบวกได้ดังรหัสที่ 7-25 เริ่มด้วยการตรวจสอบว่าถ้ามีมิติไม่เท่ากันจะโยนสิ่งผิดปกติทันที ถ้าขนาดเท่ากัน ก็สร้างเมทริกซ์ผลลัพธ์ แล้วใช้วงวน `for` สองวงซ้อนกันเพื่อเข้าใช้ข้อมูลในเมทริกซ์แบบไล่แต่ละตัวในแถวจากซ้ายไปขวา (วงใน) และไล่แถวจากบนลงล่าง (วงนอก) อันเป็นรูปแบบที่ใช้บ่อยมาก หาผลบวกของแต่ละตำแหน่งในเมทริกซ์ ทำเสร็จก็คืนผล

```
01 public class Matrix {
02   public static double[][] add(double[][] a, double[][] b) {
03     if (a.length != b.length || a[0].length != b[0].length) {
04       throw new IllegalArgumentException("มิติไม่เท่ากัน");
05     }
06     double[][] c = new double[a.length][a[0].length];
07     for (int i = 0; i < c.length; i++) {
08       for (int j = 0; j < c[0].length; j++) {
09         c[i][j] = a[i][j] + b[i][j];
10       }
11     }
12     return c;
13   }
```

รหัสที่ 7-25 เมท็อดการบวกเมทริกซ์

ต่อด้วยเมท็อดการหาผลคูณ vA โดยที่ v คือเวกเตอร์ และ A คือเมทริกซ์ จะคูณกันได้ v ต้องมีจำนวนสมาชิกเท่ากับจำนวนแถวของ A คูณแล้วได้ผลเป็นเวกเตอร์ ถ้า $u = vA$, เวกเตอร์ u จะมีขนาดเท่ากับจำนวนสดมภ์ของ A ดังตัวอย่างข้างล่างนี้

$$\begin{bmatrix}v_0 & v_1\end{bmatrix}\begin{bmatrix}a_{0,0} & a_{0,1} & a_{0,2}\\a_{1,0} & a_{1,1} & a_{1,2}\end{bmatrix} = \begin{bmatrix}v_0a_{0,0} + v_1a_{1,0} & v_0a_{0,1} + v_1a_{1,1} & v_0a_{0,2} + v_1a_{1,2}\end{bmatrix} = \begin{bmatrix}u_0 & u_1 & u_2\end{bmatrix}$$

สรุปได้ว่า สมาชิก u_i ของผลลัพธ์คำนวณได้จาก

$$u_i = \sum_{k=0}^{n-1} v_k a_{k,i}$$

โดยที่ n คือจำนวนข้อมูลใน v เขียนเป็นเมท็อดได้ดังรหัสที่ 7-26 วงวน `for` วงนอกกำหนดดัชนีของผลลัพธ์ `u` ในขณะที่วงวน `for` วงในคำนวณค่า `u[i]` ตามสูตร (ทุกครั้งที่เห็นเครื่องหมาย $\sum$ ซึ่งคือ การหาผลรวมโดยมีจุดเริ่มและจุดจบชัดเจน ก็ให้คิดถึงวงวน `for` ได้ทันที)

```
14   public static double[] multiply(double[] v, double[][] a) {
15     if (v.length != a.length) {
16       throw new IllegalArgumentException("มิติไม่เท่ากัน");
17     }
18     double[] u = new double[a[0].length];
19     for (int i = 0; i < u.length; i++) {
20       for (int k = 0; k < v.length; k++) {
21         u[i] += v[k]*a[k][i];
22       }
23     }
24     return u;
25   }
```

$$u_i = \sum_{k=0}^{n-1} v_k a_{k,i}$$

รหัสที่ 7-26 เมท็อดการคูณเวกเตอร์กับเมทริกซ์

ปิดท้ายด้วยเมท็อดหาผลคูณ AB โดย A กับ B เป็นเมทริกซ์ที่จำนวนสดมภ์ของ A เท่ากับจำนวนแถวของ B ให้ $C = AB$ จะได้ว่า สมาชิก $c_{i,j}$ ของ C มีค่า $c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$ โดยที่ n คือจำนวนสดมภ์ของ A ดังตัวอย่างข้างล่างนี้ A มีขนาด 2×3, B มีขนาด 3×2 ได้ C มีขนาด 2×2

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \end{bmatrix} \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \\ b_{2,0} & b_{2,1} \end{bmatrix} = \begin{bmatrix} a_{0,0}b_{0,0} + a_{0,1}b_{1,0} + a_{0,2}b_{2,0} & a_{0,0}b_{0,1} + a_{0,1}b_{1,1} + a_{0,2}b_{2,1} \\ a_{1,0}b_{0,0} + a_{1,1}b_{1,0} + a_{1,2}b_{2,0} & a_{1,0}b_{0,1} + a_{1,1}b_{1,1} + a_{1,2}b_{2,1} \end{bmatrix}$$

$$= \begin{bmatrix} c_{0,0} & c_{0,1} \\ c_{1,0} & c_{1,1} \end{bmatrix}$$

เขียนเป็นเมท็อดได้ดังรหัสที่ 7-27 มีวงวน `for` สามวงซ้อนกัน ดูแล้วอย่าเพิ่งตกใจ สองวงนอกคือรูปแบบการเปลี่ยนแปลงดัชนี `i`, `j` เพื่อไล่หาค่าให้ `c` จากแถวบนลงล่าง และจากซ้ายไปขวาในแต่ละแถว ส่วนวงวน `for` วงในสุด คือ การหาค่าให้กับ `c[i][j]` ตามสูตร $c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$ ดัชนี `i`, `j`, และ `k` ในสูตรก็คือตัวเดียวกับที่ปรากฏในเมท็อด

```
26   public static double[][] multiply(double[][] a, double[][] b) {
27     if (a[0].length != b.length) {
28       throw new IllegalArgumentException("มิติไม่เท่ากัน");
29     }
30     double[][] c = new double[a.length][b[0].length];
31     for (int i = 0; i < c.length; i++) {
32       for (int j = 0; j < c[0].length; j++) {
33         for (int k = 0; k < a[0].length; k++) {
34           c[i][j] += a[i][k]*b[k][j];
35         }
36       }
37     }
38     return c;
39   }
```

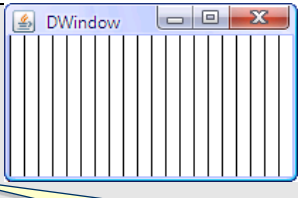
$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$$

รหัสที่ 7-27 เมท็อดการคูณเมทริกซ์กับเมทริกซ์

แผนที่จุดภาพ

คลาส DWindow ที่เราใช้สร้างวินโดว์ และวาดเส้นตรงหรือวงกลมกันมานั้น มีเมท็อดชื่อ getPixmap() ซึ่งคืนอาเรย์สองมิติที่แต่ละช่องเก็บจำนวนเต็มแทนสีของแต่ละจุดภาพบนวินโดว์ เรียกว่า *แผนที่จุดภาพ* (pixel map หรือเรียกย่อ ๆ ว่า pixmap) เช่น การสร้างวินโดว์ด้วยคำสั่ง w = new DWindow(200,100) จะสร้างวินโดว์ที่มีพื้นที่แสดงขนาดกว้าง×ยาวเป็น 200×100 จุดภาพ หากใช้คำสั่ง int[][] p = w.getPixmap() จะได้อาเรย์ของ int ขนาด 200 แถวแต่ละแถวมี 100 ช่อง ถ้าเราเติมสี (ซึ่งใน DWindow แทนสีด้วย int) ลงตามช่องต่าง ๆ จากนั้นใช้อีกคำสั่ง w.setPixmap(p) จะทำให้วินโดว์เปลี่ยนภาพที่แสดงตามข้อมูลในแผนที่จุดภาพ p ดังตัวอย่างในรหัสที่ 7-28 ให้สังเกตว่าเราใช้ x เป็นดัชนีของมิติแรก และ y เป็นดัชนีของมิติที่สอง เพื่อให้สอดคล้องกับการเขียนพิกัดของจุด (x, y) ซึ่งเขียน x ก่อน y นั่นหมายความว่าจำนวนแถว (แนวนอน) ของอาเรย์ที่แทนแผนที่จุดภาพ คือ ความกว้างของวินโดว์ และจำนวนสดมภ์ของอาเรย์ที่แทนแผนที่จุดภาพ คือ ความสูงของวินโดว์

```
01 import jlab.graphics.DWindow;
02 public class TestPixmap {
03   public static void main(String[] args) {
04     DWindow w = new DWindow(200,100);
05     int[][] p = w.getPixmap();
06     for (int x = 0; x < p.length; x += 10) {
07       for (int y = 0; y < p[0].length; y++) {
08         p[x][y] = DWindow.BLACK;
09       }
10     }
11     w.setPixmap(p);
12   }
13 }
```



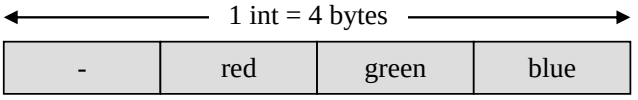
วาดเส้นตรงแนวตั้งที่ตำแหน่ง x=0, 10, 20, ...

ใส่จุดสีดำต่อ ๆ กันให้เกิดเป็นเส้นตรง

แสดงภาพใหม่ตามแผนที่จุดภาพ p

รหัสที่ 7-28 ตัวอย่างการใช้งานเมท็อด getPixmap และ setPixmap

ก่อนจะนำเสนอการจัดการแผนที่จุดภาพ ขอนำเสนอรูปแบบการแทนสีในแต่ละจุดภาพกันก่อน สีของจุดภาพแต่ละจุดเกิดจากการผสมกันของแม่สี 3 สี แดง เขียว และน้ำเงิน (ใช้อักษรย่อ R, G, และ B) แม่สีแต่ละสีมีระดับความเข้ม 256 ระดับ (ระดับ 0 แทนไม่มีสี จนถึง 255 แทนมีสีสดเต็มที่) เช่น ถ้าให้ R=0, G=0, B=0 จะได้สีดำ, R=255, G=255, B=255 ได้สีขาว, R=255, G=255, B=0 ได้สีเหลือง เป็นต้น จึงสามารถให้สีกับแต่ละจุดภาพได้ถึง 256^3 = 16,777,216 สี การแทนจำนวนเต็มที่มีค่า 0 ถึง 255 นั้น ใช้เลขฐานสองจำนวน 8 บิต (หรือ 1 ไบต์) ได้พอดี เพราะ 1 บิตแทนได้ 2 ค่า, 8 บิตจึงแทนได้ 2^8 = 256 ค่า เราจึงต้องใช้ 3 ไบต์เพื่อแทนแม่สี 3 สี เนื่องจากข้อมูล int หนึ่งตัวในจาวาใช้เนื้อที่ 4 ไบต์ ดังนั้น เราสามารถแทนสีของจุดภาพหนึ่งจุดด้วย int หนึ่งตัว โดยให้สามไบต์ทางขวาแทนแม่สี 3 สีคือ สีละไบต์ดังรูปที่ 7-9 (สำหรับไบต์ซ้ายสุดเก็บค่าความทึบของสี ซึ่งไม่ขอลงในรายละเอียด) แผนที่จุดภาพจึงแทนด้วยอาเรย์สองมิติของ int



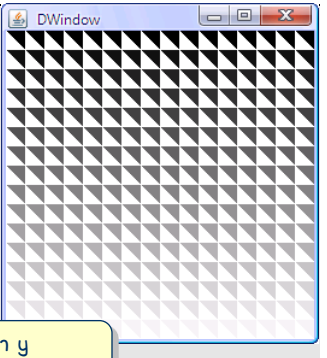
รูปที่ 7-9 การเก็บค่าของแม่สีใน `int` หนึ่งตัว

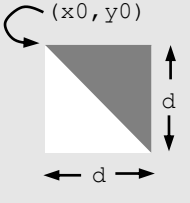
เพื่อให้การแยกแม่สีและรวมแม่สี กระทำได้ง่าย `DWindow` มีเมท็อดให้บริการดังนี้

- `getR(c), getG(c), getB(c)` คืนจำนวนเต็มที่แทนระดับของแม่สี แดง เขียว และน้ำเงินตามลำดับของสี `c` แม่สีที่ได้มีค่าตั้งแต่ 0 ถึง 255
- `mixRGB(r,g,b)` คืนสีที่เกิดจากการรวมแม่สีแดง เขียว และน้ำเงินที่ได้รับ สามารถนำค่านี้ไปตั้งสีให้กับจุดภาพได้ เช่น เรียกใช้ `DWindow.mixRGB(255,255,0)` ได้สีเหลือง เป็นต้น (ถ้าค่าของแม่สีที่ได้รับมีค่าน้อยกว่า 0 จะปรับให้เป็น 0 และถ้ามากกว่า 255 จะปรับให้เป็น 255)

นอกจากนี้ `DWindow` ยังมีสีมาตรฐานที่ผู้เขียนได้ตั้งค่าคงตัวและตั้งชื่อให้แล้วจำนวนหนึ่ง เช่น `DWindow.GREEN`, `DWindow.RED`, `DWindow.PINK` เป็นต้น รหัสที่ 7-29 แสดงตัวอย่างการใช้เมท็อดจัดการแผนที่จุดภาพและสีข้างต้น (ผู้อ่านลองศึกษาการทำงานโปรแกรมนี้ดู)

```
01 import jlab.graphics.DWindow;
02 public class FillPattern {
03   public static void main(String[] args) {
04     DWindow w = new DWindow(256,256);
05     int[][] p = w.getPixmap();
06     int d = 16;
07     for (int x = 0; x < p.length; x+=d) {
08       for (int y = 0; y < p[0].length; y+=d) {
09         int c = DWindow.mixRGB(y,y,y);
10         fillPattern(p, x, y, d, c);
11       }
12     }
13     w.setPixmap(p);
14   }
15   public static void fillPattern(int[][] p, int x0, int y0,
16                                  int d, int c) {
17     for (int x = 0; x < d; x++) {
18       for (int y = 0; y < d; y++) {
19         if (x > y)
20           p[x0 + x][y0 + y] = c;
21         else
22           p[x0 + x][y0 + y] = DWindow.WHITE;
23       }
24     }
25   }
26 }
```





รหัสที่ 7-29 ตัวอย่างการใช้เมท็อดจัดการสีและแผนที่จุดภาพ

การประมวลผลภาพ

ด้วยเมท็อดการอ่าน การเขียนแผนที่จุดภาพ และการจัดการสี ทำให้เราสามารถเขียนโปรแกรมประมวลผลภาพที่พบเห็นในซอฟต์แวร์ทั่วไปได้ไม่ยาก เช่น การปรับแสงภาพ การทำภาพให้คมชัดหรือมัว และอื่น ๆ แต่ก่อนอื่นเราต้องมีวิธีนำรูปภาพจากแฟ้มมาแสดงในวินโดว์กันก่อน DWindow มีเมท็อด loadImage(f) เพื่ออ่านแฟ้มภาพ f มาแสดง เมื่ออ่านเข้ามาแล้วก็สามารถจัดการกับจุดภาพได้ เช่น โปรแกรมในรหัสที่ 7-30 อ่านแฟ้มภาพอนุสาวรีย์ประชาธิปไตยมาแสดง (บรรทัดที่ 5), ขอแผนที่จุดภาพมาเก็บในอาเรย์ p (บรรทัดที่ 6), เรียกเมท็อดให้พลิกภาพตามแนวนอน ได้แผนที่จุดภาพใหม่คืนกลับมา (บรรทัดที่ 7) เพื่อนำไปตั้งเป็นภาพใหม่บนวินโดว์ (บรรทัดที่ 8) สำหรับเมท็อดพลิกภาพนั้น เพียงแค่กลับตำแหน่ง x ของแผนที่จุดภาพต้นฉบับ ที่อ่านจากซ้ายไปขวาของ p ไปเก็บในผลลัพธ์จากขวามาซ้าย ได้ผลดังรูปที่ 7-10

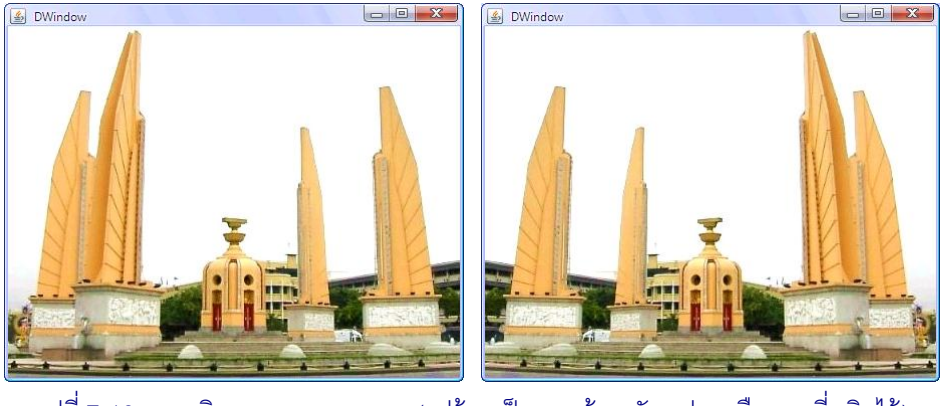
```
01 import jlab.graphics.DWindow;
02 public class ImageProc {
03   public static void main(String[] args) {
04     DWindow w = new DWindow();
05     w.loadImage("c:/java101/monument.jpg");
06     int[][] p = w.getPixmap();
07     int[][] flip = horizontalFlip(p);
08     w.setPixmap(flip);
09   }
10   public static int[][] horizontalFlip(int[][] p) {
11     int[][] out = new int[p.length][p[0].length];
12     for (int x = 0; x < p.length; x++) {
13       for (int y = 0; y < p[0].length; y++) {
14         out[x][y] = p[p.length - 1 - x][y];
15       }
16     }
17     return out;
18   }
```

อ่านแฟ้มภาพมาแสดงบนวินโดว์

พลิกแผนที่จุดภาพตามแนวนอน แล้วแสดง

ย้ายจุดจากขวาไปซ้าย ไปเก็บในผลลัพธ์จากซ้ายไปขวา

รหัสที่ 7-30 เมท็อดการพลิกภาพตามแนวนอน



รูปที่ 7-10 การพลิกภาพตามแนวนอน (รูปซ้ายเป็นภาพต้นฉบับ รูปขวาคือภาพที่พลิกได้)

มาดูอีกตัวอย่าง รหัสที่ 7-31 แสดงเมท็อดทำภาพเนกาทิฟ (negative) คือการกลับสีของแต่ละจุดให้เป็นสี “ตรงข้าม” เช่น จากขาวเป็นดำ น้ำเงินเป็นเหลือง เป็นต้น ซึ่งคือการนำค่าเดิมของแต่ละแม่สีไปลบออกจาก 255 นั่นเอง ดังนั้น ในวงวน `for` จึงอ่านแต่ละแม่สีมาลบออกจาก 255 (บรรทัดที่ 23 ถึง 25) ตามด้วยการรวมสีที่ได้ทั้งสามใส่ในจุดภาพของอาเรย์ผลลัพธ์ (บรรทัดที่ 26) หากใช้เมท็อดนี้กับแผนที่จุดภาพที่อ่านจากแฟ้มภาพอนุสาวรีย์ประชาธิปไตย จะได้ผลดังรูปที่ 7-11

```
19   public static int[][] negative(int[][] p) {
20     int[][] out = new int[p.length][p[0].length];
21     for (int x = 0; x < p.length; x++) {
22       for (int y = 0; y < p[0].length; y++) {
23         int r = 255 - DWindow.getR(p[x][y]);
24         int g = 255 - DWindow.getG(p[x][y]);
25         int b = 255 - DWindow.getB(p[x][y]);
26         out[x][y] = DWindow.mixRGB(r, g, b);
27       }
28     }
29     return out;
30   }
```

แต่ละสีมีค่าตั้งแต่ 0 ถึง 255
สีตรงข้ามของ c คือ 255 - c

รวมแม่สีทั้งสามเป็นสีที่ตั้งให้จุดภาพ

รหัสที่ 7-31 เมท็อดการทำภาพเนกาทิฟ

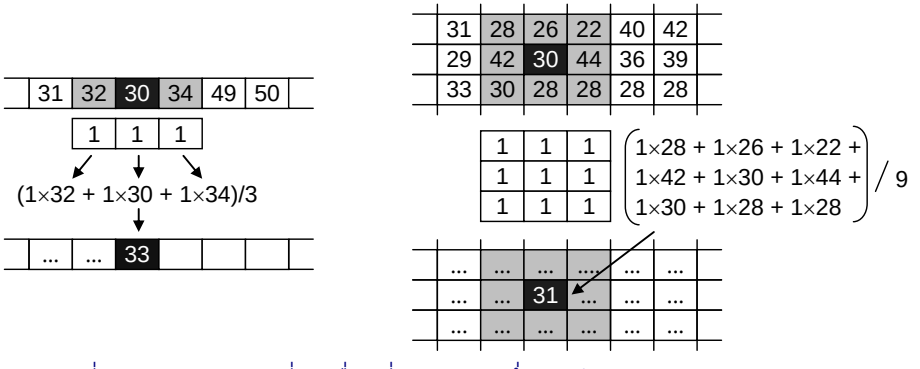
รูปที่ 7-11 ภาพเนกาทิฟของรูปที่ 7-10

คราวนี้จะขอใช้แนวคิดของการหาค่าเฉลี่ยเคลื่อนที่ที่เคยทำกับอาเรย์หนึ่งมิติมาทำกับข้อมูลในอาเรย์สองมิติ ในกรณีหนึ่งมิติ ขณะที่สนใจช่องที่ *i* ก็ให้นำข้อมูลที่ช่องนี้กับช่องก่อนหน้าและตามหลังมาหาค่าเฉลี่ย ซึ่งสามารถมองว่า (ดูรูปที่ 7-12 ทางซ้าย) เสมือนมีอาเรย์หนึ่งมิติเล็ก ๆ หนึ่งแถวที่มีสามช่อง แต่ละช่องมีค่า 1 นำไปทาบกับอาเรย์ต้นฉบับ ไล่ทาบตั้งแต่ซ้ายสุดแล้วค่อย ๆ เลื่อนไปทางขวาทีละช่องจนถึงขวาสุด เมื่อทาบที่สามช่องใด ก็นำข้อมูลช่องที่ตรงกันของทั้งสองแถวมาคูณกัน หาผลรวม แล้วหารด้วย 3 (ซึ่งก็คือการหาค่าเฉลี่ยของข้อมูลสามตัวนั่นเอง) กราฟเส้นของผลลัพธ์ที่ได้มีลักษณะคล้ายกับของอาเรย์ต้นฉบับแต่ “เรียบ” กว่า เมื่อนำแนวคิดนี้มาทำกับเมทริกซ์ (ขอใช้คำว่าเมทริกซ์แทนอาเรย์สองมิติ เพราะสิ่งที่จะทำต่อไปนี้เรียกว่า matrix convolution) คือใช้เมทริกซ์เล็กขนาด 3×3 แต่ละช่องเก็บ 1 เมื่อทาบที่ตำแหน่งใด ให้คูณช่องที่

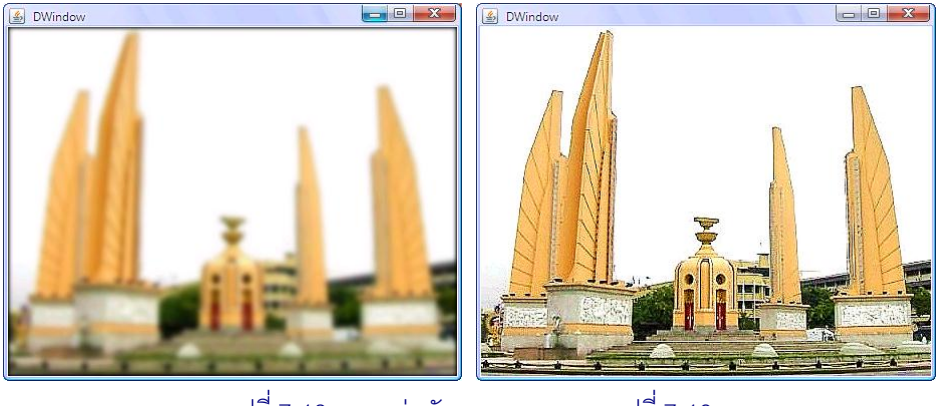
ตรงกัน หาผลรวม แล้วหารด้วย 9 (ดูรูปที่ 7-12 ทางขวา) ซึ่งก็คือการหาค่าเฉลี่ยของช่องที่สนใจกับช่องรอบข้างอีก 8 ช่องนั่นเอง โดยนำเมทริกซ์เล็กไล่ทาบเมทริกซ์ต้นฉบับให้ครบทุกตำแหน่ง ผู้อ่านอาจสงสัยว่า จะทำแบบนี้ไปทำไม ต้องบอกว่าเหตุผลที่ทำแบบนี้ ก็เพราะว่า ถ้าเมทริกซ์ใหญ่คือแผนที่จุดภาพ การทำ matrix convolution กับทีละแม่สี ด้วยเมทริกซ์เล็กที่มีค่าเริ่มต้นต่างกันจะได้การประมวลผลภาพหลายแบบ เช่น ถ้าเป็น $\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$ จะได้ภาพพร่ามัว ถ้าเป็น $\begin{bmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{bmatrix}$ จะได้ภาพคมขึ้น เป็นต้น ให้ f คือเมทริกซ์เล็ก (เรียกว่า filter matrix) และ p คือเมทริกซ์ของแผนที่จุดภาพ จะได้เมทริกซ์ผลลัพธ์ q คำนวณได้ตามสูตรข้างล่างนี้

$$q_{x,y} = \frac{1}{\sum f}\left(\sum_{i=0}^{2}\sum_{j=0}^{2} f_{i,j}\cdot p_{x+i-1,y+i-1}\right) \quad if \quad \sum f \neq 0$$

โดยที่ $\sum f$ คือผลรวมของทุกช่องใน f รูปที่ 7-13 แสดงผลที่ได้จากนำรูปที่ 7-10 มาทำภาพพร่ามัว และภาพคม (ภาพพร่ามัวที่แสดง ทำ matrix convolution ห้าครั้งติดต่อกัน)



รูปที่ 7-12 การหาค่าเฉลี่ยเคลื่อนที่บนอาเรย์หนึ่งมิติ (ซ้าย) กับสองมิติ (ขวา)



รูปที่ 7-13 ภาพพร่ามัวและภาพคมของรูปที่ 7-10

สำหรับกรณีที่ $\sum f$ มีค่าเป็น 0 คงนำ $\sum f$ ไปหารตามสูตรข้างต้นไม่ได้ แต่เนื่องจากเมทริกซ์ที่มี $\sum f = 0$ นั้นเกิดกับเมทริกซ์ที่มีไว้ประมวลผลภาพเพื่อดักตรวจบริเวณที่มีการเปลี่ยนแปลงของสีอย่างฉับพลันในภาพ (คือเปลี่ยนจากเข้มไปจาง หรือจากจางไปเข้ม) ซึ่งคือบริเวณขอบของวัตถุในภาพ ตัวอย่างเช่น ถ้าใช้ $f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ จะได้ผลลัพธ์แสดงเส้นขอบสีเข้มขาวดำของวัตถุในภาพต้นฉบับ รูปที่ 7-14 ทางซ้ายแสดงผลที่ได้ ถ้าเราเพิ่มผลที่ได้ด้วย 255 ให้กับทุกแม่สีของจุดภาพในผลลัพธ์ จะเปลี่ยนจากภาพพื้นดำเส้นขอบขาว เป็นภาพพื้นขาวเส้นขอบดำ ดังแสดงในรูปที่ 7-14 ทางขวา บางคนอาจเพิ่มแค่ 128 จะได้พื้นเทาแลดูแปลกตา (ผู้อ่านลองทำดู) ดังนั้น เมื่อ $\sum f$ มีค่าเป็น 0 จะใช้สูตรข้างล่างนี้ โดยที่ b (เรียกว่าค่า bias) เป็นค่าเพิ่มที่กล่าวถึง สามารถเขียนเมท็อดทำ matrix convolution ได้ดังรหัสที่ 7-32

$$q_{x,y} = b + \left(\sum_{i=0}^{2} \sum_{j=0}^{2} f_{i,j} \cdot p_{x+i-1,y+i-1} \right) \quad if \quad \sum f = 0$$

```
31   public static int[][] convolute(int[][] p, int[][] f, int bias) {
32     int[][] out = new int[p.length][p[0].length];
33     int sum = 0;
34     for (int i = 0; i < f.length; i++)
35       for (int j = 0; j < f[0].length; j++)
36         sum += f[i][j];
37     for (int x = 1; x < p.length - 1; x++) {
38       for (int y = 1; y < p[0].length - 1; y++) {
39         int r = 0, g = 0, b = 0;
40         for (int i = 0; i <= 2; i++) {
41           for (int j = 0; j <= 2; j++) {
42             r += f[i][j] * DWindow.getR(p[x+i-1][y+j-1]);
43             g += f[i][j] * DWindow.getG(p[x+i-1][y+j-1]);
44             b += f[i][j] * DWindow.getB(p[x+i-1][y+j-1]);
45           }
46         }
47         if (sum != 0) {
48           r /= sum; g /= sum; b /= sum;
49         } else {
50           r += bias; g += bias; b += bias;
51         }
52         out[x][y] = DWindow.mixRGB(r, g, b);
53       }
54     }
55     return out;
56   }
```

sum = $\sum f$

สองวงวนนอกแจกแจงแต่ละจุด (x,y) ในแผนที่จุดภาพ

สองวงวนในทำ convolution

ทำ convolution กับแม่สีทั้งสาม

รวมแม่สีทั้งสามเก็บใส่ผลลัพธ์

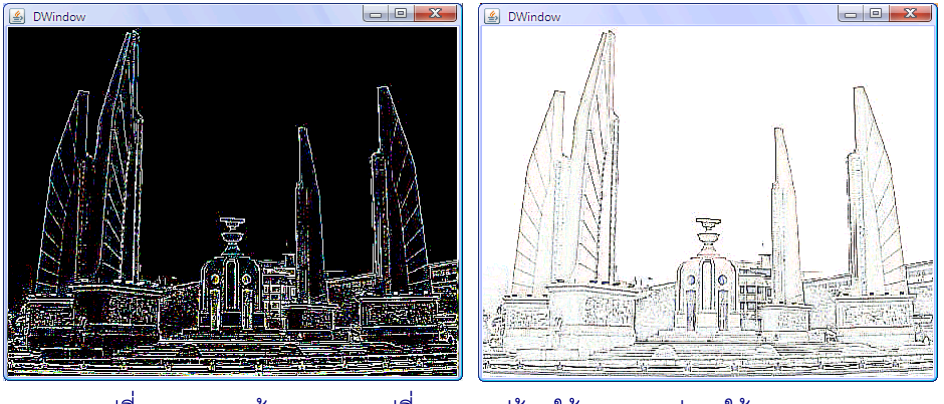
รหัสที่ 7-32 เมท็อดการทำ matrix convolution

รหัสที่ 7-33 แสดงเมท็อดการประมวลผลภาพหลากหลายแบบขึ้นกับค่าของเมทริกซ์ f รวมถึงการทำภาพดุนนูนที่ยังไม่ได้กล่าวถึงสองแบบ ผู้อ่านควรลองป้อนโปรแกรม และสั่งทำงานดู

อย่าลืมใส่ชื่อของแฟ้มภาพที่ต้องการประมวลผลให้กับเมท็อด `loadImage` ให้ถูกต้อง นอกจากนี้ถ้าต้องการบันทึกผลก็สามารถใช้เมท็อด `saveImage` ส่งชื่อแฟ้มไป ก็จะบันทึกผลให้ ดังตัวอย่างส่วนของโปรแกรมที่แสดงในรหัสที่ 7-34

```
57    public static int[][] blurFilter(int[][] p) {
58      int[][] f = {{1,1,1},{1,1,1},{1,1,1}};
59      return convolute(p, f, 0);
60    }
61    public static int[][] sharpenFilter(int[][] p) {
62      int[][] f = {{0,-1,0},{-1,5,-1},{0,-1,0}};
63      return convolute(p, f, 0);
64    }
65    public static int[][] edgeFilter(int[][] p) {
66      int[][] f = {{1,1,1},{1,-8,1},{1,1,1}};
67      return convolute(p, f, 255); // ตั้ง bias = 255 ได้พื้นขาวขอบดำ
68    }
69    public static int[][] embossFilter(int[][] p) {
70      int[][] f = {{-2,-1,0},{-1,1,1},{0,1,2}};
71      return convolute(p, f, 0);
72    }
73    public static int[][] embossGrayFilter(int[][] p) {
74      int[][] f = {{2,0,0},{0,-1,0},{0,0,-1}};
75      return convolute(p, f, 128); // ตั้ง bias = 128 ได้พื้นเทา
76    }
77  }
```

รหัสที่ 7-33 เมท็อดการประมวลผลภาพ



รูปที่ 7-14 ภาพเส้นขอบของรูปที่ 7-10 (รูปซ้ายให้ b = 0, รูปขวาให้ b = 255)

```
    DWindow w = new DWindow();
    w.loadImage("c:/java101/monument.jpg");
    int[][] p = w.getPixmap();
    w.setPixmap(embossGrayFilter(horizontalFlip(p)));
    w.saveImage("c:/java101/out.jpg");
```

รหัสที่ 7-34 ตัวอย่างการนำภาพมาประมวลผลแล้วบันทึกลงแฟ้มภาพ

เพิ่มเติม

การอ่านข้อมูลจากบรรทัดคำสั่ง

ผู้อ่านก็คงสงสัยกันมาตั้งแต่บทต้น ๆ แล้วว่า หัวเมท็อดของ `main` นั้นหมายความว่าอะไร บอกให้จำ ๆ มาตลอดว่า `public static void main(String[] args)` ความหมายเป็นดังนี้

- `public` บอกว่าเป็นเมท็อดสาธารณะ ผู้อื่นรวมถึงระบบจาวาเรียกใช้ได้
- `static` บอกว่าเป็นเมท็อดประจำคลาส (ขอค้างคำนี้ไว้บทหน้า จะเข้าใจมากขึ้น)
- `void` บอกว่าเมท็อดนี้ไม่คืนผลลัพธ์
- `main` คือชื่อเมท็อด
- `String[] args` คือพารามิเตอร์ซึ่งรับมาเป็นอาเรย์ของสตริง

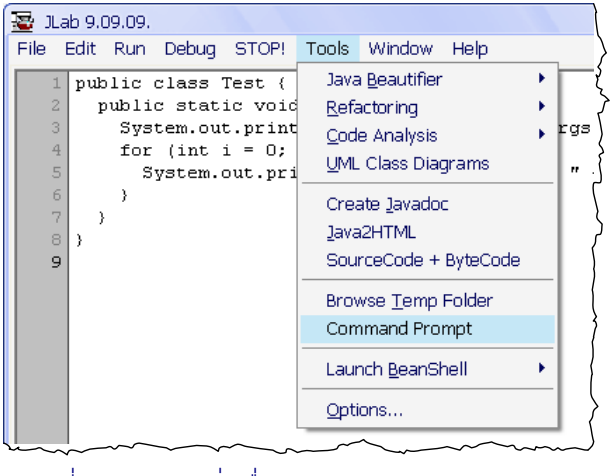
ประเด็นที่ต้องการอธิบายในหัวข้อย่อยนี้ คือ พารามิเตอร์ที่ `main` รับมา ประเด็นแรกคือ เราไม่จำเป็นต้องเขียนว่า `args` ใช้ชื่อตัวแปรอะไรก็ได้ เพราะเป็นเพียงชื่อพารามิเตอร์ ประเด็นที่สองที่น่าสนใจกว่าคือ อาเรย์ของสตริงนี้ภายในมีค่าอะไร ถ้าผู้อ่านลองสั่งทำงานโปรแกรมอะไรก็ได้ ที่ได้เขียนกันมาทั้งหมด แล้วลองเพิ่มคำสั่งแสดงความยาวของอาเรย์ `args` นี้ดู (เช่นรหัสที่ 7-35) ก็จะพบว่า มีค่าเป็น 0 ตลอด ตกลงว่าพารามิเตอร์ตัวนี้มีประโยชน์อะไร คำตอบอยู่ที่ตอนสั่งให้โปรแกรมทำงาน การใช้ซอฟต์แวร์ช่วยพัฒนาโปรแกรม เช่น ในกรณี JLab ก็คือ การกดปุ่ม F5 ระบบจะสั่งทำงานโดยส่งอาเรย์ 0 ช่องให้กับ `args` ต้องสั่งทำงานอีกแบบจึงจะส่งข้อมูลมาให้ `args` ได้

```
01 public class Test {
02   public static void main(String[] args) {
03     System.out.println("args.length = " + args.length);
04     for (int i = 0; i < args.length; i++) {
05       System.out.println("args[" + i + "] = " + args[i]);
06     }
07   }
08 }
```

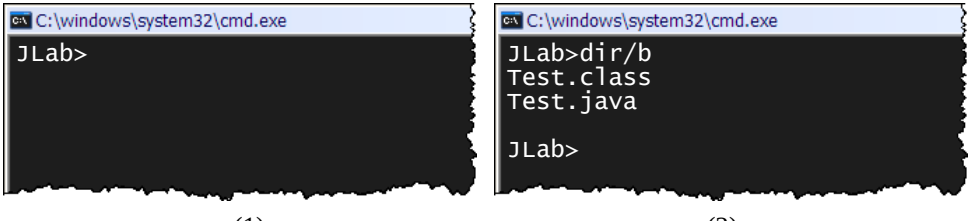
รหัสที่ 7-35 เมท็อดแสดงข้อมูลในพารามิเตอร์ของ `main`

เราสามารถสั่งคลาสทำงานได้ทาง command prompt ซึ่งเป็นซอฟต์แวร์ตัวหนึ่งของระบบปฏิบัติการของเครื่องคอมพิวเตอร์ที่ให้ผู้ใช้เครื่องสั่งโปรแกรมทำงานด้วยการพิมพ์คำสั่งทางแป้นพิมพ์โต้ตอบกับระบบ (โดยไม่ใช้เมาส์ที่ใช้กันทั่วไป) การเปิด command prompt นั้นทำได้โดยคลิกปุ่ม Start ที่มุมซ้ายล่างของจอภาพ เลือก Run... ป้อนคำว่า cmd ในช่อง แล้วกดปุ่ม OK ระบบก็จะเปิดวินโดว์เล็ก ๆ พื้นดำ ให้ป้อนคำสั่งได้ แต่ก็ยังต้องป้อนอีกหลายคำสั่งกว่าจะสั่งให้คลาสทำงานได้ (เพราะต้องไปที่ ๆ คลาสนั้นอยู่ก่อน) ใน JLab มีเมนูที่จะเปิด command prompt ให้เราใช้งานได้อย่างง่าย ๆ ด้วยการเลือกเมนู Tools->Command Prompt (ดังรูปที่ 7-15) จะได้วินโดว์

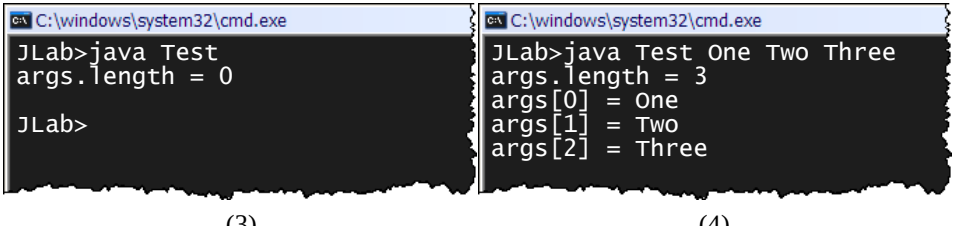
พื้นดำดังรูปที่ 7-16 (1) จากตัวอย่างเราได้สร้างคลาส `Test` ของรหัสที่ 7-35 และแปลเป็นรหัสเครื่องแล้ว ดังนั้น หากพิมพ์คำสั่ง `dir/b` จะเห็นแฟ้ม `Test.java` ซึ่งเก็บรหัสต้นฉบับ กับ `Test.class` ซึ่งเก็บรหัสเครื่อง ดังรูปที่ 7-16 (2) เมื่อต้องการสั่งคลาส `Test` ทำงาน ก็พิมพ์คำสั่ง `java Test` ใน command prompt จะได้ผลดังรูปที่ 7-16 (3) ซึ่งแสดงให้เห็นว่า `args` เป็นอาเรย์ 0 ช่อง แต่ถ้าเราพิมพ์ข้อความตามหลัง เช่น รูปที่ 7-16 (4) แสดงผลที่ได้เมื่อเราพิมพ์คำสั่ง `java Test One Two Three` พบว่าระบบจะนำข้อความที่เพิ่มตามหลังมาสร้างเป็นอาเรย์ของสตริง โดยสตริงแต่ละตัวคือแต่ละคำที่พิมพ์ตามหลัง (คั่นด้วยช่องว่าง) สำหรับตัวอย่างนี้มี 3 คำ จึงสร้างอาเรย์ 3 ช่อง ได้ผลดังปรากฏในรูปที่ 7-16 (4) (อนึ่งเมื่อเลิกใช้ command prompt ให้พิมพ์คำสั่ง `exit`)



รูปที่ 7-15 เมนูคำสั่งเพื่อเปิดวินโดว์ command prompt



(1) (2)



(3) (4)

รูปที่ 7-16 command prompt และการสั่งคลาสทำงานพร้อมกับส่งข้อมูลให้กับเมท็อด `main`

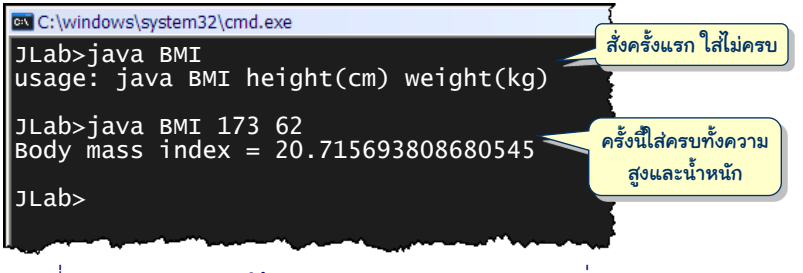
การเขียนโปรแกรมที่รับข้อมูลทางบรรทัดคำสั่งเช่นนี้ เหมาะสำหรับโปรแกรมที่ให้ผู้ใช้ตั้งพฤติกรรมของโปรแกรม ณ เวลาที่สั่งโปรแกรมทำงาน ไม่ต้องมาเสียเวลาใช้ `Scanner` มารับอีกทีหนึ่ง เช่น จากโปรแกรมคำนวณดัชนีมวลกายที่เคยรอรับความสูงและน้ำหนักทางแป้นพิมพ์ เราเขียนอีกแบบดังรหัสที่ 7-36 รับความสูงและน้ำหนักทางพารามิเตอร์ `args` แทน โดยทั่วไปมักตรวจสอบความยาวของ `args` ก่อนว่า ป้อนมาครบหรือไม่ รูปที่ 7-17 แสดงตัวอย่างการสั่งโปรแกรมนี้ทำงาน

```
01 public class BMI {
02   public static void main(String[] args) {
03     if (args.length < 2) {
04       System.out.println("usage: java BMI height(cm) weight(kg)");
05     } else {
06       double h = Double.parseDouble(args[0]) / 100;
07       double w = Double.parseDouble(args[1]);
08       double bmi = w / (h * h);
09       System.out.println("Body mass index = " + bmi);
10     }
11   }
12 }
```

ตรวจสอบก่อนว่าใส่มาครบหรือไม่

Double.parseDouble เปลี่ยนสตริงเป็น double

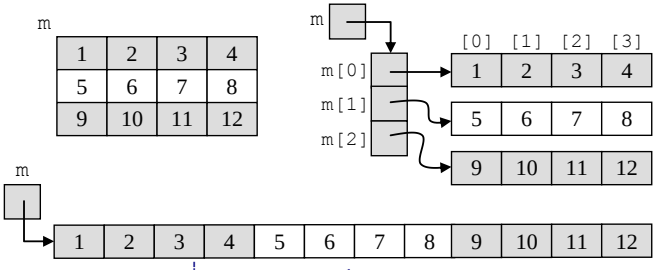
รหัสที่ 7-36 โปรแกรมคำนวณดัชนีมวลกายที่รับความสูงและน้ำหนักทางบรรทัดคำสั่ง



รูปที่ 7-17 ตัวอย่างการใช้งานคลาส `BMI` ทางบรรทัดคำสั่ง

การจัดเก็บอาเรย์หลายมิติ

อาเรย์หนึ่งมิติที่มี n ช่อง ใช้เนื้อที่ต่อเนื่องกันในหน่วยความจำที่มีขนาดเพียงพอในการเก็บข้อมูล n ตัว ทำให้การเข้าใช้ช่องใด ๆ ในอาเรย์สามารถกระทำได้อย่างรวดเร็ว เนื่องจากสามารถคำนวณตำแหน่งของช่องใด ๆ จากตำแหน่งของช่องแรกของอาเรย์ได้ง่าย (เพราะแต่ละช่องถูกจัดวางให้ติดและต่อเนื่องกันไป) สำหรับอาเรย์สองมิติที่มี $r \times c$ ช่อง (r แถว c สดมภ์) จัดเก็บกันอย่างไร รูปที่ 7-18 นำเสนอแนวทางการจัดเก็บสองรูปแบบ บางภาษาโปรแกรมจัดเก็บโดยนำแต่ละแถวของอาเรย์มาเรียงต่อกันไป (รูปล่าง) เนื่องจากเรารู้แล้วว่า แต่ละแถวมี c ช่อง หากต้องการช่องที่ [i][j] ของอาเรย์ ก็สามารถคำนวณได้ว่า ต้องอยู่ห่างจากช่องแรกไป $i \cdot c + j$ ช่อง ดังนั้นสามารถเข้าใช้ช่องใดๆ ในอาเรย์สองมิติได้รวดเร็วเช่นกัน

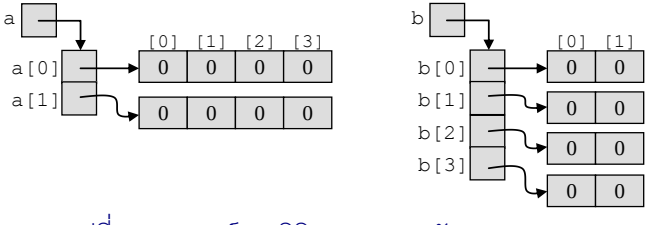


รูปที่ 7-18 การจัดเก็บอาเรย์สองมิติ

ภาษาจาวาเลือกจัดเก็บอาเรย์สองมิติในอีกรูปแบบ (รูปที่ 7-18 ขวา) คือสร้างอาเรย์สองมิติด้วยอาเรย์หนึ่งมิติหลาย ๆ แถว เรียกว่า เก็บแบบอาเรย์ของอาเรย์ จากรูปเป็นอาเรย์ขนาด 3×4 มีตัวแปร `m` อ้างอิงอาเรย์ที่มี 3 ช่อง แต่ละช่องแทนแต่ละแถวโดยเก็บข้อมูลอ้างอิงไปยังอาเรย์ที่มี 4 ช่อง สรุปได้ว่า คำสั่ง `int[][] m` มี

- `m[i][j]` เก็บข้อมูลแบบ `int`
- `m[i]` เก็บข้อมูลแบบ `int[]` คือเก็บตัวอ้างอิงไปยังอาเรย์ของ `int`
- `m` เก็บข้อมูลแบบ `int[][]` คือเก็บตัวอ้างอิงไปยังอาเรย์ของ `int[]`

ด้วยวิธีการจัดเก็บแบบนี้ การสร้างอาเรย์ด้วยคำสั่ง `new int[2][4]` กับ `new int[4][2]` มีโครงสร้างไม่เหมือนกัน และใช้เนื้อที่ไม่เท่ากันด้วย (ดูรูปที่ 7-19)

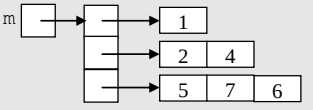


รูปที่ 7-19 อาเรย์สองมิติขนาด 2×4 กับขนาด 4×2

ผู้อ่านคงอยากถามว่า จัดเก็บแบบนี้มีอะไรดีกว่าแบบแรก ขอย้อนไปใช้อาเรย์ `m` (รูปที่ 7-18) เป็นตัวอย่าง ด้วยความที่ `m[0]`, `m[1]`, และ `m[2]` ต่างคนต่างเก็บตัวอ้างอิงไปยังอาเรย์ของ `int` ของแถวตัวเอง จึงไม่มีข้อบังคับใดว่า อาเรย์ของ `int` ทั้งสามนี้ต้องมีขนาดเดียวกัน จึงเปิดโอกาสให้เราสร้างอาเรย์สองมิติที่มี “รูปร่างแปลก ๆ” ได้ เช่น รหัสที่ 7-37 แสดงตัวอย่างการสร้างอาเรย์สองมิติที่ใช้แทนเมทริกซ์แบบสามเหลี่ยมล่าง (lower triangular matrix) ซึ่งเป็นเมทริกซ์ที่ค่าของช่องเหนือแนวทแยงมุมเป็น 0 จึงไม่จำเป็นต้องเก็บ [4] เนื่องจากจำนวนสดมภ์ไม่จำเป็นต้องเท่ากันทุกแถว อย่าลืมใช้ `a[k].length` แทนจำนวนสดมภ์ในแถวที่ `k` ของอาเรย์สองมิติ `a`

[4] เมทริกซ์แบบสามเหลี่ยมล่างมักใช้คู่กับแบบสามเหลี่ยมบนในพีชคณิตเชิงเส้น เพื่อหาคำตอบในระบบสมการเชิงเส้น ซึ่งถ้ามีสมการจำนวนมาก จะแทนด้วยเมทริกซ์ขนาดใหญ่ การจัดเก็บให้ประหยัดเนื้อที่จึงเป็นเรื่องสำคัญ

```
int[][] m = new int[3][];
m[0] = new int[]{1};
m[1] = new int[]{2,4};
m[2] = new int[]{5,7,6};
```



รหัสที่ 7-37 ตัวอย่างการสร้างเมทริกซ์แบบสามเหลี่ยมล่าง

คลาสมาตรฐานที่ให้บริการกับอาเรย์

การดำเนินการข้อมูลในอาเรย์ที่ได้นำเสนอมาหลายวิธี เช่น การค้นหา การเรียงลำดับ การทำสำเนาข้อมูลในอาเรย์ เป็นต้น เป็นการดำเนินการที่กระทำบ่อยมาก ระบบจาวาจึงมีบริการเหล่านี้บรรจุอยู่ในคลาสมาตรฐานของระบบที่สามารถเรียกใช้ได้ทันที บริการเหล่านี้อยู่ในคลาสชื่อเต็มว่า `java.util.Arrays` (คำว่า `Arrays` มี `s`) ขอนำเสนอบางเมท็อดในคลาสนี้ให้เห็นเป็นตัวอย่าง ดังนี้ (ผู้อ่านสามารถอ่านรายละเอียดได้ใน Java help file)

- `binarySearch(a,x)` : ค้น `x` ว่าอยู่ที่ดัชนีใดในอาเรย์ `a` โดยมีข้อกำหนดว่าข้อมูลในอาเรย์ `a` ต้องเรียงลำดับแล้ว เมท็อดนี้ใช้วิธีการค้นข้อมูลที่มีชื่อว่า *การค้นแบบทวิภาค* (binary search) ที่ค้นได้เร็วมาก ดังตัวอย่างข้างล่างนี้

```
int[] a = {11,21,25,40,90};          // ต้องเรียงลำดับแล้ว
int i = Arrays.binarySearch(a,40); // ได้ i = 3
int j = Arrays.binarySearch(a,38); // ได้ j = -4 (ไม่พบได้ติดลบ)
```

- `copyOf(a,n)` : สร้างและคืนอาเรย์ใหม่ขนาด `n` ช่องที่มีข้อมูลทางซ้ายเหมือน `a` ดังตัวอย่างข้างล่างนี้

```
int[] a = {1,2,3,4};
int[] a6 = Arrays.copyOf(a,6); // ได้ a6 = {1,2,3,4,0,0}
int[] a3 = Arrays.copyOf(a,3); // ได้ a3 = {1,2,3}
```

- `equals(a,b)` : คืนผลการเปรียบเทียบอาเรย์ `a` และ `b` ว่าเท่ากันทุกตัวหรือไม่ ดังตัวอย่างข้างล่างนี้

```
int[] a = {1,2,3,4}, b = {1,2,3,4}, c = {1,2,3,5}, d={1};
boolean b1 = Arrays.equals(a,b); // ได้ b1 = true
boolean b2 = Arrays.equals(a,c); // ได้ b2 = false
boolean b3 = Arrays.equals(a,d); // ได้ b3 = false
```

- `sort(a)` : เรียงลำดับข้อมูลใน `a` ดังตัวอย่างข้างล่างนี้

```
int[] a = {4,5,1,2};
Arrays.sort(a);    // ได้ a = {1,2,4,5}
```

- `toString(a)` : คืนสตริงที่แทนข้อมูลใน `a` พร้อมนำไปแสดงได้ ดังตัวอย่างข้างล่างนี้

```
int[] a = {4,5,1};
String s = Arrays.toString(a);    // ได้ s = [4, 5, 1]
```

เมท็อดที่นำเสนอข้างต้นนี้ใช้กับอาเรย์ของข้อมูลประเภทอื่นก็ได้ แต่ใช้กับอาเรย์หนึ่งมิติเท่านั้น สำหรับอาเรย์หลายมิติ มีเมท็อด `deepEquals` กับ `deepToString` ดังตัวอย่างข้างล่างนี้

```
int[][] a = { {1,2}, {3,4}, {5,6} };
String s = Arrays.deepToString(a); //[[1, 2], [3, 4], [5, 6]]
```

และยังมีอีกเมท็อดที่ใช้บ่อยคือ `System.arraycopy(a,i,b,j,n)` ซึ่งทำสำเนาจากอาเรย์ `a` เริ่มที่ดัชนี `i` ไปยังอาเรย์ `b` เริ่มที่ดัชนี `j` เป็นจำนวน `n` ช่องติดกันไป ดังตัวอย่างข้างล่างนี้

```
int[] a = {1,2,3,4}, b = {11,12,13,14};
System.arraycopy(a, 2, b, 1, 2); // ได้ b = {11, 3, 4, 14}
```

ตัวดำเนินการระดับบิต

หัวข้อย่อยนี้มีเนื้อหาที่ไม่เกี่ยวกับอาเรย์ แต่เกี่ยวกับตัวดำเนินการที่จัดการข้อมูลระดับบิตที่ไม่เคยนำเสนอมาก่อน ได้แก่ การเลื่อนบิตข้อมูล (shift) และการนำจำนวนเต็มมาแอนด์หรือออร์ (and, or) กันในระดับบิต เราได้นำเสนอการใช้จำนวนเต็ม `int` ในการแทนสี ภายในประกอบด้วยระดับความเข้มของแม่สี 3 สี ๆ ละ 1 ไบต์ ดังรูปที่ 7-9 | แดง | เขียว | น้ำเงิน | โดยมีเมท็อดใน `DWindow` ที่ให้บริการ `getR`, `getG`, `getB` เพื่อดึงแม่สีที่สนใจ รวมทั้งบริการ `mixRGB` ที่รวมแม่สีทั้งสามให้เป็น `int` เพื่อใส่ในแผนที่จุดภาพ อยากทราบว่า เมท็อดเหล่านี้ทำงานอย่างไร ถึงได้ดึงข้อมูลออกมาเป็นไบต์ ๆ ได้

ก่อนจะนำเสนอการทำงานระดับบิต ขอนำเสนอวิธีการเขียนจำนวนเต็มแบบฐานสิบหก (เลขโดดในเลขฐานสิบหกคือ 0, 1, 2,..., 9, A, B, C, D, E, F ซึ่งมีค่าเท่ากับ 0, 1, ..., 15 ในฐานสิบ) เราเขียนค่าคงตัวของจำนวนเต็มแบบฐานสิบหกในจาวาได้ด้วยการเขียน `0x` นำหน้า เช่น `0xF` แทน $(15)_{10}$, `0x2AC` = $2\times16^2 + 10\times16^1 + 12\times16^0 = (684)_{10}$, หรือเขียนแบบเต็มที่ 4 ไบต์ เช่น `0xFFFFFFFF` ซึ่งคือ 1 หมดทั้ง 32 บิต มีค่าเป็น $(-1)_{10}$ (ถ้าผู้อ่านสงสัยว่า ทำไมเท่ากับ –1 ขอให้กลับไปอ่านหัวข้อเพิ่มเติมในบทที่ 2 ว่าด้วยการแทนจำนวนเต็มในระบบเลขฐานสอง)

เราได้เคยใช้ตัวดำเนินการตรรกะ `&&`, `||`, และ `!` กันมากับค่าตรรกะ `true` และ `false` สำหรับกรณีของจำนวนเต็ม เราสามารถนำจำนวนเต็มมาแอนด์ ออร์ และนอตได้ด้วยตัวดำเนินการ `&` `|` และ `!` โดยการแอนด์ ออร์ นอตในระดับบิตนั้น ให้มองเสมือน 1 แทนจริง 0 แทนเท็จ ดังตัวอย่างต่อไปนี้ (ดูรูปที่ 7-20) `0x73 & 0x0F` ได้ `0x03` เพราะ `0x73` = $(01110011)_2$, `0x0F` = $(00001111)_2$ นำมา `&` กัน บิตใดตรงกันและเป็น 1 ทั้งคู่ แอนด์กันจะได้ 1 ได้ผลเป็น $(00000011)_2$ ซึ่งคือ `0x03` ถ้านำมาออร์กัน `0x73 | 0x0F` ได้ `0x7F` เพราะถ้าบิตใดบิตหนึ่งที่ตรงกันเป็น 1 ออร์กันจะได้ 1

```
0111 0011          0111 0011
0000 1111   &      0000 1111   |
-----------        -----------
0000 0011          0111 1111
```

รูปที่ 7-20 ตัวอย่างการใช้ตัวดำเนินการ `&` กับ `|`

การแอนด์บิตจึงมักนำมาใช้ในการสกัดบิตที่สนใจมาใช้ เช่น เราสนใจไบต์ขวาสุดของจำนวนเต็มในตัวแปร `c` ก็ใช้ `c & 0x000000FF` หรือเขียนให้สั้นด้วย `c & 0xFF` เป็นการบอกว่าขอค่าของ `c` ที่มีสามไบต์ซ้ายเป็น 0 หมด เหลือไบต์ขวา มีค่าเท่าใด ก็ขอค่านั้น ถ้าต้องการไบต์ที่สองจากขวาของ `c` ก็ใช้ `c & 0xFF00`, และถ้าสนใจไบต์ที่สามจากขวาก็ใช้ `c & 0xFF0000`

ส่วนการออร์บิตก็มักนำมาใช้ในการรวมข้อมูลระดับบิต เช่น ถ้าต้องการนำไบต์ขวาสุดของ `a` กับไบต์ที่สองจากขวาของ `b` มาวางใน `c` ก็เขียน `c = (a & 0xFF) | (b & 0xFF00)` แล้วถ้าต้องการไบต์ขวาสุดของ `r`, ไบต์ขวาสุดของ `g`, และไบต์ขวาสุดของ `b` มาวางต่อเรียงกันที่สามไบต์ขวาในตัวแปร `rgb` จะทำอย่างไร ความต้องการนี้ต้องอาศัยการเลื่อนบิต

จาวามีตัวดำเนินการเลื่อนบิตของจำนวนเต็มสามแบบคือ << แทนการเลื่อนบิตไปทางซ้าย และ >> กับ >>> แทนการเลื่อนบิตไปทางขวา (ซึ่งต่างกันเล็กน้อย) ดังนี้

- `c << k` คือผลของการนำค่าของตัวแปร `c` มาเลื่อนบิตไปทางซ้ายเป็นจำนวน `k` บิต ขอเน้นว่าค่าของตัวแปร `c` ไม่เปลี่ยน เพราะเป็นเพียงการนำค่าของ `c` มาใช้ เช่น คำสั่ง `int c = 2; int a = c << 1;` จะได้ `c` มีค่า `2` เช่นเดิม และ `a` มีค่า `4` เพราะ $(2)_{10} = (0010)_2$ เลื่อนไปทางซ้ายหนึ่งบิต จะได้ $(0100)_2$ ซึ่งมีค่าเป็น $(4)_{10}$ ให้สังเกตว่าหลังจากเลื่อนไปทางซ้ายแล้ว บิตที่เติมให้ที่หลักขวาสุดจะเป็น 0

- `c >> k` ทำงานในทางกลับกัน คือจะได้ผลของการนำค่าของตัวแปร `c` มาเลื่อนบิตไปทางขวาเป็นจำนวน `k` บิต เช่น คำสั่ง `int c = 6; int a = c >> 1;` จะได้ `a` มีค่า `3` เพราะ $(6)_{10} = (0110)_2$ เลื่อนไปทางขวาหนึ่งบิต จะได้ $(0011)_2 = (3)_{10}$ ในกรณีของการเลื่อนแบบนี้ บิตที่เติมที่หลักซ้ายสุดจะมีค่าเดียวกับบิตซ้ายสุดเดิม เช่น `-4>>1` จะได้ –2 เพราะ $(–4)_{10} = (1100)_2$ ดังนั้น เลื่อน $(1100)_2$ ไปทางขวา 1 บิตจะได้ $(1110)_2$ ซึ่งมีค่าเป็น $(–2)_{10}$ การเลื่อนไปทางขวา 1 บิต จึงเสมือนกับการหารด้วย 2 ปัดเศษทิ้ง

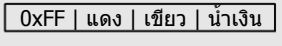
- `c >>> k` ทำงานเหมือน >> ทุกประการ ต่างกันที่บิตใหม่ที่เติมให้ที่หลักซ้ายสุดนั้นจะเป็น 0 เสมอ ดังนั้น ถ้า `c ≥ 0` ค่าของ `c >> k` กับ `c >>> k` จะเหมือนกัน แต่ถ้า `c` เป็นจำนวนลบ จะต่างกันมโหฬาร เช่น `-4>>>1` จะได้ค่า 2147483646 เพราะ $(–4)_{10}$ $= (1100)_2 = (11111111\ 11111111\ 11111111\ 11111100)_2$ เมื่อเก็บแบบ 32 บิต (ซึ่งคือขนาดของ `int`) ได้ผลการเลื่อนเป็น $(01111111\ 11111111\ 11111111\ 11111110)_2 = (2147483646)_{10}$ จึงเรียก << และ >> ว่า *การเลื่อนแบบคงเครื่องหมาย* (signed shift) ในขณะที่ >>> เป็นแบบไม่สนใจเครื่องหมายลบของข้อมูลเดิม (unsigned shift)

ถึงตรงนี้ก็สามารถเขียนเมท็อด `getR`, `getG`, `getB`, และ `mixRGB` ได้ไม่ยาก เพราะเรารู้วิธีการจัดการระดับบิตของจำนวนเต็มกันแล้ว รหัสที่ 7-38 แสดงเมท็อดทั้งสี่ในคลาส `DWindow` จากการจัดเก็บในรูปแบบ `| แดง | เขียว | น้ำเงิน` การดึงแม่สีออกจากตัวแปร `c` กระทำโดย

เลื่อนแม่สีที่ต้องการให้มาอยู่ที่ไบต์ขวาสุด ตามด้วยการแอนด์ให้เหลือเฉพาะไบต์ขวาสุดนั้น เช่น `getR(c)` เรารู้ว่า ข้อมูลของแม่สีแดงอยู่ไบต์ที่สามจากขวา ก็ให้เลื่อนข้อมูลของ `c` ไปทางขวา 16 บิต ทำให้ไบต์ที่สามทางขวาย้ายมาอยู่ไบต์ขวาสุด จากนั้นก็แอนด์ด้วย `0xFF` สำหรับ `getG(c)` ก็ให้เลื่อนมาทางขวา 8 บิตก่อน แล้วค่อยแอนด์ด้วย `0xFF` และ `getB(c)` ไม่ต้องเลื่อนเลย เพราะอยู่ไบต์ขวาสุดอยู่แล้ว ก็แอนด์ไบต์อื่นให้หมดไปได้ผลที่ต้องการทันที

และสุดท้ายคือ `mixRGB(r,g,b)` ก่อนอื่นเพื่อป้องกันไม่ให้ข้อมูลในพารามิเตอร์อยู่นอกช่วง 0 ถึง 255 เราจะอาศัยเมท็อดส่วนตัวชื่อ `clip` ที่เขียนขึ้นเพื่อปัดค่าให้อยู่ในช่วง นั่นคือถ้าได้ค่าแม่สีที่น้อยกว่า 0 ก็ให้เป็น 0 หรือถ้าค่าแม่สีเกิน 255 ก็ให้เป็น 255 กระทำการ `clip` แม่สีทั้งสาม แล้วเลื่อนบิตไปทางซ้ายให้ตรงตำแหน่ง สีแดงก็เลื่อน 16 บิต เขียวเลื่อน 8 บิต น้ำเงินไม่ต้องเลื่อน นำค่าแม่สีที่เลื่อนบิตตรงตำแหน่งแล้วมาออร์กัน เป็นอันผสมสีโดยการวางแม่สีให้ตรงตำแหน่งได้เรียบร้อย หากดูที่เมท็อด `mixRGB` จะพบว่า เราใส่ค่า `0xFF` ไว้ที่ไบต์ซ้ายสุดด้วย ตรงนี้เป็นเรื่องรายละเอียดเล็กน้อยเกี่ยวกับเรื่องสี ไบต์ซ้ายสุดนี้คือค่าระดับความทึบของสี ถ้าเป็น 0 คือโปร่งใส, `0xFF` หรือ 255 คือทึบแสง นั่นเอง

องค์ประกอบแม่สีที่เก็บใน 1 int




```
public class DWindow {
  ...
  public static int getR(int c) {
    return (c >> 16) & 0xFF;
  }
  public static int getG(int c) {
    return (c >> 8) & 0xFF;
  }
  public static int getB(int c) {
    return c & 0xFF;
  }
  public static int mixRGB(int r, int g, int b) {
    return 0xFF000000 | (clip(r)<<16) | (clip(g)<<8) | clip(b);
  }
  private static int clip(int v) { // ปัดค่าของ v ให้อยู่ในช่วง 0 - 255
    if (v < 0) v = 0;
    if (v > 255) v = 255;
    return v;
  }
  ...
```

รหัสที่ 7-38 เมท็อดให้บริการดึงแม่สีและผสมแม่สี

แบบฝึกหัด

1. จงเขียนคำสั่งที่สร้างอาเรย์ 3 ตัว ที่มีค่าเริ่มต้นเป็น $\begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 1 \end{bmatrix}$, $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$, $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

2. จงปรับโปรแกรมลูกบอลหลายลูกในรหัสที่ 7-2 ให้แต่ละลูกมีรัศมีและสีของตัวเอง โดยตั้งค่าเริ่มต้นของรัศมีและสีแบบสุ่ม (ข้อแนะนำ : การตั้งสีแบบสุ่ม คือการตั้งค่าสุ่มระหว่าง 0 ถึง 255 ให้กับแม่สีทั้งสาม แล้วใช้เมท็อด `DWindow.mixRGB` ผสมแม่สีทั้งสาม)

3. จงเขียนเมท็อดดังต่อไปนี้ พร้อมกับเขียนโปรแกรมเพื่อทดสอบด้วยว่า เมท็อดที่เขียนทำงานถูกต้องตามข้อกำหนด ตัวเมท็อดควรตรวจสอบค่าของพารามิเตอร์ด้วยว่าเป็นค่าที่ยอมรับได้หรือไม่

   - `int count(int[] d, int x)` คืนจำนวนข้อมูลในอาเรย์ `d` ที่มีค่าเท่ากับ `x`
   - `int mode(int[] d)` คืนข้อมูลในอาเรย์ `d` ที่มีปรากฏซ้ำกันเป็นจำนวนครั้งมากสุด
   - `boolean majority(int[] d)` ตรวจสอบว่าอาเรย์ `d` มีข้อมูลใดที่ซ้ำกันเกินครึ่งหนึ่งของจำนวนข้อมูลทั้งหมดหรือไม่
   - `void insert(int[] d, int i, int x)` แทรกข้อมูล `x` ที่ดัชนี `i` ในอาเรย์ `d` โดยให้ข้อมูลเดิมจากดัชนี `i` จนถึงตัวก่อนขวาสุดเลื่อนไปทางขวาหนึ่งตำแหน่ง (ตัวขวาสุดก็หายไปจากอาเรย์)
   - `void remove(int[] d, int i)` ลบข้อมูลตัวที่ดัชนี `i` ทิ้งไป โดยให้ข้อมูลจาก `d[i+1]` จนถึงตัวขวาสุดเลื่อนไปทางซ้ายหนึ่งตำแหน่ง (หลังเลื่อนเสร็จ ให้เติมตัวขวาสุดของอาเรย์ด้วย 0)
   - `void shuffle(int[] d)` สลับข้อมูลในอาเรย์ `d` อย่างสุ่ม ๆ
   - `int[] shuffle(int[] d)` คืนอาเรย์ใหม่ที่ได้จากการนำข้อมูลทั้งหมดของอาเรย์ `d` มาสลับข้อมูลอย่างสุ่ม ๆ (อาเรย์ `d` ไม่เปลี่ยนแปลง)
   - `boolean isSorted(double[] d)` ตรวจสอบว่าจำนวนในอาเรย์ `d` เรียงลำดับจากน้อยไปหามากหรือไม่
   - `boolean isSorted(String[] d)` ตรวจสอบว่าสตริงในอาเรย์ `d` เรียงลำดับจากน้อยไปหามากหรือไม่
   - `double[] inRange(double[] d, double min, double max)` คืนอาเรย์ใหม่ที่เก็บข้อมูลทั้งหมดของอาเรย์ `d` เฉพาะตัวที่อยู่ในช่วงตั้งแต่ `min` ถึง `max`

- `void arraycopy(int[] a, int i, int[] b, int j, int len)` ทำสำเนาข้อมูลในอาเรย์ `a` ตั้งแต่ดัชนี `i` ไปยังอาเรย์ `b` เริ่มที่ดัชนี `j` เป็นจำนวน `len` ตัว (ให้ความสามารถของเมท็อดนี้ทำได้เท่ากับของ `System.arraycopy` ที่ยอมให้อาเรย์ `a` กับ `b` เป็นตัวเดียวกัน และ `i` น้อยกว่า `j` หรือ `i` มากกว่า `j` ก็ทำได้)
- `double maxsum(double[] d)` คืนผลรวมของช่วงข้อมูลที่ติดกันใน `d` ที่มีผลรวมมากสุด เช่น `d={2,-1,3,-5,3}` จะคืน 4 เพราะ $2 - 1 + 3$ เป็นช่วงของข้อมูลที่ติดกันใน `d` ที่มีผลรวมมากสุด (ข้อแนะนำ : ให้ใช้วงวน `for` หลายวงซ้อนกัน เพื่อหาผลรวมของช่วงข้อมูลใน `d` ทุกรูปแบบ และหาช่วงมากสุดไปในตัวด้วย)

4. จงเขียนเมท็อดหาผลต่างของคู่ข้อมูลในอาเรย์ `d` ที่ค่าผลต่างมีค่ามากสุด เช่น `d={3,-1,0,-5,2}` มีคู่ที่ผลต่างมีค่ามากสุดคือ 3 กับ –5 ดังนั้น จึงคืน 8 และจงเขียนอีกเมท็อดที่หาผลในลักษณะเดียวกันแต่ใช้กับกรณีอาเรย์สองมิติ

5. จงเขียนเมท็อดเพื่อตรวจสอบเมทริกซ์ ดังต่อไปนี้
   - `boolean isZero(double[][] d)` ตรวจสอบว่า `d` เป็นเมทริกซ์ที่ข้อมูลทุกตัวเป็นศูนย์หรือไม่
   - `boolean isIdentity(double[][] d)` ตรวจสอบว่า `d` เป็นเมทริกซ์ที่ข้อมูลเป็น 0 หมดยกเว้นข้อมูลที่แนวทแยงมุมจากมุมซ้ายบนถึงขวาล่างเป็น 1 หมดหรือไม่ (เรียกว่า เมทริกซ์เอกลักษณ์)
   - `boolean isLowerTriangular(double[][] d)` ตรวจสอบว่า `d` เป็นเมทริกซ์แบบสามเหลี่ยมล่างหรือไม่ คือ เมทริกซ์ที่ข้อมูลเหนือแนวทแยงมุมเป็น 0 หมด
   - `boolean isUpperTriangular(double[][] d)` ตรวจสอบว่า `d` เป็นเมทริกซ์แบบสามเหลี่ยมบนหรือไม่ คือ เมทริกซ์ที่ข้อมูลใต้แนวทแยงมุมเป็น 0 หมด

6. จงเขียนเมท็อด `double[][] transpose(double[][] d)` ที่คืนเมทริกซ์สลับเปลี่ยน (transpose matrix) ของเมทริกซ์ `d`

7. จงเขียนเมท็อด `int[] maxmin(int[] d)` ที่คืนอาเรย์สองช่อง ที่มี
   - ช่องที่ 0 เก็บค่ามากที่สุดในอาเรย์ `d`
   - ช่องที่ 1 เก็บค่าน้อยที่สุดในอาเรย์ `d`

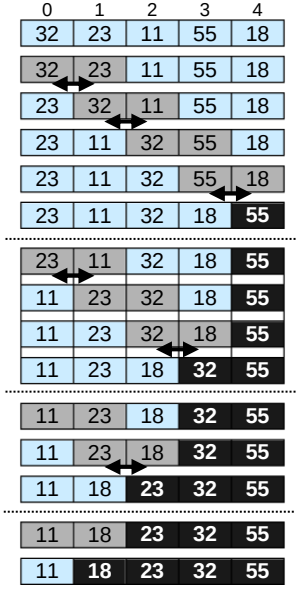
8. จงเขียนเมท็อด `int[] maxmin(int[] d, int i, int j)` คล้ายกับข้อที่แล้ว เพียงแต่ให้หาค่ามากสุดและน้อยสุดที่เก็บใน `d` ตั้งแต่ช่องที่ `i` ถึง `j` โดยให้เขียนแบบเรียกซ้ำที่มีหลักการทำงานดังนี้

- แบ่งข้อมูลในช่วงที่สนใจเป็นสองช่วงย่อย ช่วงซ้ายและช่วงขวา ไปหา `maxmin` ของช่วงซ้าย และไปหา `maxmin` ของช่วงขวา
- นำค่ามากสุดของสองช่วงย่อยมาเปรียบเทียบ ได้ค่ามากสุดของทั้งสองช่วงรวมกัน
- นำค่าน้อยสุดของสองช่วงย่อยมาเปรียบเทียบ ได้ค่าน้อยสุดของทั้งสองช่วงรวมกัน

9. จงเขียนเมท็อดวาดกราฟเส้นของข้อมูลอนุกรมเวลาคล้ายกับ รหัสที่ 7-21 ต่างกันตรงที่รับเฉพาะอาเรย์ `y` ก็พอ เพราะถือว่าค่า `x` ของแต่ละ `y` ห่างเท่า ๆ กัน

10. เมท็อด `getMovingAverage` ในรหัสที่ 7-22 หาค่าเฉลี่ยจากข้อมูลสามตัวติดกัน จงปรับให้เป็นเมท็อดที่หาค่าเฉลี่ยจากข้อมูลที่ติดกัน `k` ตัว (ให้ `k` เป็นพารามิเตอร์อีกตัว)

11. *การเรียงลำดับแบบฟอง* (bubble sort) อาศัยการเปรียบเทียบข้อมูลในอาเรย์ตัวที่ติดกันว่า กลับลำดับกันหรือไม่ ถ้าใช่ก็สลับที่กัน กระทำการเปรียบเทียบและสลับข้อมูลไล่ตั้งแต่ซ้ายไปขวา หนึ่งรอบได้ตัวมากสุดมาอยู่หลังสุด ทำอีกรอบ (โดยลดจำนวนข้อมูลลงหนึ่ง) ก็จะได้ตัวมากสุดของที่เหลือย้ายมาอยู่ช่องขวาสุดของกลุ่มที่เหลือ ทำเช่นนี้ $n - 1$ รอบจนเหลือข้อมูลตัวเดียว จะได้ข้อมูลในแถวลำดับเรียงลำดับตามต้องการ รูปทางขวาแสดงตัวอย่างการทำงาน แถวแรกแสดงข้อมูลขาเข้า มีข้อมูลอยู่ 5 ตัว, รอบที่ 1 เปรียบเทียบข้อมูลคู่ที่ติดกัน 4 ครั้ง เกิดการสลับ 3 ครั้ง ได้ 55 อยู่ช่องขวาสุด, รอบที่ 2 เปรียบเทียบคู่ที่ติดกัน 3 ครั้ง เกิดการสลับ 2 ครั้ง ได้ 32 อยู่ช่อง 3, รอบที่ 3 เปรียบเทียบคู่ที่ติดกัน 2 ครั้ง เกิดการสลับ 1 ครั้ง ได้ 23 อยู่ช่อง 2, รอบสุดท้ายเหลือคู่เดียว ไม่มีการสลับ ได้ข้อมูลเรียงลำดับเรียบร้อย จงเขียนเมท็อด `bubbleSort(int[] d)` เพื่อเรียงลำดับข้อมูลในอาเรย์ `d` ด้วยวิธีแบบฟอง



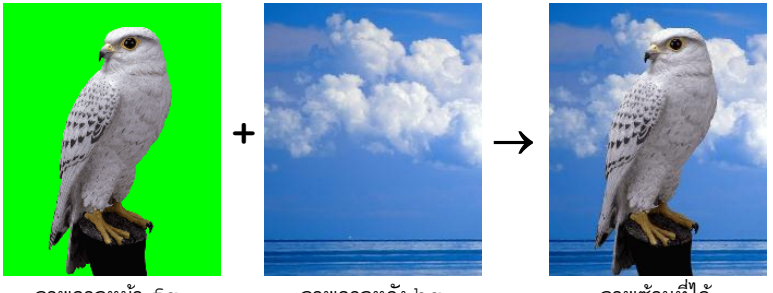
12. จงเขียนเมท็อดประมวลผลภาพดังต่อไปนี้

- `int[][] verticalFlip(int[][] p)` คืนแผนที่จุดภาพที่เป็นผลจากการพลิกภาพ `p` ตามแนวดิ่ง
- `int[][] rotateClockwise90(int[][] p)` คืนแผนที่จุดภาพที่เป็นผลจากหมุนภาพ `p` ตามเข็มไป 90 องศา
- `int[][] rotateCounterClockwise90(int[][] p)` คืนแผนที่จุดภาพที่เป็นผลจากหมุนภาพ `p` ทวนเข็มไป 90 องศา

13. จงเขียนเมท็อด `int[][] blend(int[][] p1, int[][] p2, double a)` ที่คืนแผนที่จุดภาพซึ่งได้มาจากการผสมแผนที่จุดภาพ `p1` และ `p2` เข้าด้วยกัน การผสมนี้เป็นการผสมแบบจุดต่อจุด คือ นำจุดที่มีตำแหน่งเดียวกันในแผนที่จุดภาพของทั้งสองภาพมาผสมกัน เป็นการแยกผสมแต่ละแม่สี ด้วยสูตร $ac_1 + (1-a)c_2$ โดยที่ c_1 และ c_2 คือค่าของแม่สีหนึ่งสีของสองจุด และ a คือ ค่าความเข้มของจุด c_1 ที่จะปรากฏในผลลัพธ์ (a คือพารามิเตอร์ของเมท็อดมีค่าตั้งแต่ 0 ถึง 1) รูปข้างล่างนี้แสดงผลของการผสมภาพสุนัขกับภาพแมวด้วยค่า a ที่ต่างกัน (แฟ้มภาพสุนัขและแมวในตัวอย่างนี้ชื่อ dog.jpg และ cat.jpg ที่อยู่ใน c:\java101)

$a = 0.0$ $a = 0.25$ $a = 0.50$ $a = 0.75$ $a = 1.0$

14. จงเขียนเมท็อด `int[][] greenChromaKey(int[][] fg, int[][] bg)` ที่คืนแผนที่จุดภาพของการซ้อนภาพฉากหน้า (`fg`) กับภาพฉากหลัง (`bg`) โดยกำหนดให้ภาพฉากหน้ามีสีพิเศษเรียกว่า *กุญแจสี* (หรือ*โครมาคีย์* chroma key) อยู่ ณ ตำแหน่งที่ต้องการให้ภาพฉากหลังปรากฏ ดังตัวอย่างข้างล่างนี้ ภาพซ้ายมือคือภาพฉากหน้าที่มีสีเขียวสดเป็นกุญแจสี เมื่อนำมาซ้อนกับภาพฉากหลังตรงกลาง จะได้ผลลัพธ์ดังภาพทางขวามือ (แฟ้มภาพนกและท้องฟ้าในตัวอย่างนี้ชื่อ bird.gif และ sky.jpg ที่อยู่ใน c:\java101) หมายเหตุ : ผู้สร้างภาพฉากหน้าต้องกำหนดให้กุญแจสีเป็นสีที่ไม่ปรากฏเป็นส่วนของวัตถุหลักของภาพฉากหน้า



ภาพฉากหน้า `fg` ภาพฉากหลัง `bg` ภาพซ้อนที่ได้

15. เมท็อด `greenChromaKey` ในข้อที่แล้วระบุว่า กุญแจสีต้องเป็นสีที่กำหนดไว้หนึ่งสี แต่ในทางปฏิบัติ ภาพฉากหน้าอาจมีจุดภาพหลายจุดที่ต้องเป็นกุญแจสี แต่กลับเก็บสีที่ผิดเพี้ยนไปจากกุญแจสีเพียงเล็กน้อย เช่น กำหนดให้กุญแจสีเป็น `r=0,g=255,b=0` (คือสีเขียวสด) แต่ถ้าสีในภาพเพี้ยนเป็น `r=1,g=254,b=0` จะทำงานไม่ตรงตามที่ต้องการ จงปรับปรุงเมท็อดการซ้อนภาพให้ยอมรับสีที่เพี้ยนจากกุญแจสีที่กำหนดไว้ (ไม่ให้เกินค่าอะไรบางอย่างที่เห็นสมควร) หมายเหตุ : ลองทดสอบกับแฟ้ม bird0.gif ใน c:\java101

16. *เกมปริศนา 15* (15 puzzle) เป็นแผ่นพลาสติกรูปสี่เหลี่ยมจัตุรัส ภายในประกอบด้วยแผ่นพลาสติกย่อยเล็กๆ (รูปสี่เหลี่ยมจัตุรัส) จำนวน 15 แผ่น (แต่ละแผ่นมีตัวเลขกำกับตั้งแต่ 1 ถึง 15) วางเรียงกันเป็นตาราง 4×4 มีช่องว่างหนึ่งช่องอยู่ภายใน สามารถเลื่อนแผ่นต่าง ๆ ไปมาได้ในแนวนอนแนวตั้ง จุดประสงค์ของเกมก็คือ ให้เลื่อนแผ่นสี่เหลี่ยมภายในไปมาเพื่อให้ได้แผ่นสี่เหลี่ยมเหล่านี้ เรียงเป็นระเบียบไล่ไปเรื่อย ๆ 1 ถึง 15 (จากซ้ายไปขวา จากบนลงล่าง) ดังตัวอย่างที่แสดงข้างขวานี้ เริ่มจากรูปทางบน ให้หาวิธีเลื่อนจนได้ดังรูปถัดลงมา

8	10	7	11
	5	4	14
1	13	12	6
2	9	3	15

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

ประเด็นปัญหาคือ จากแผ่นเริ่มต้นที่ให้มา (ซึ่งมีรูปแบบที่เป็นไปได้ทั้งสิ้น 15 ! = 1,307,674,368,000 แบบ) มันไม่แน่เสมอไปว่าจะมีวิธีเลื่อนกลับให้เป็นระเบียบตามที่ต้องการได้ ตัวอย่าง เช่น ถ้าให้จุดเริ่มต้นเป็นดังรูปขวานี้ จะไม่มีทางเลื่อนกลับไปเป็นสิ่งที่ต้องการได้ (ให้สังเกตว่า มันต่างกับจุดหมายที่ต้องการ เพียงแค่สลับ 14 กับ 15 เท่านั้น)

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

เราสามารถตรวจว่า จะมีวิธีเลื่อนแผ่นสี่เหลี่ยมต่าง ๆ กลับไปสู่สิ่งที่ต้องการได้หรือไม่ (โดยไม่ต้องลองเลื่อนดูจริง ๆ) ดังนี้

- นำแผ่นสี่เหลี่ยมที่เรียงเป็นตาราง มาวางเรียงกันเป็นแนวยาว ไล่จากซ้ายไปขวา บนลงล่าง ตัวอย่างเช่น ตารางทางขวานี้เรียงแล้วได้ (ไม่นำช่องว่างมาเรียง)

1	2	3	4
5	6	7	8
9	10	12	15
13	14		11

  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 13, 14, 11

- พิจารณาตัวเลขต่างๆ เป็นคู่ ๆ (ตัวเลขมี 15 ตัว จึงมีทั้งหมด (15×14)/2 คู่) แล้วนับว่ามีตัวเลขในรายการอยู่กี่คู่ที่กลับลำดับ (ตัวทางซ้ายมากกว่าตัวทางขวา) จากตัวอย่าง คู่ที่กลับลำดับคือ (12, 11), (15, 13), (15, 14), (15, 11), (13, 11) และ (14, 11) มีอยู่ 6 คู่ ให้ *L* คือจำนวนการกลับลำดับที่นับได้ในขั้นตอนนี้

- พิจารณาว่า ช่องว่างอยู่ในแถวแนวนอนที่เท่าใด (แถวบนสุดคือแถวที่ 1 ไล่ลงมา) จากตัวอย่าง ช่องว่างอยู่แถวที่ 4 ให้ *B* คือหมายเลขแถวที่ช่องว่างอยู่

- ถ้า *L*+*B* เป็นจำนวนคู่ แสดงว่าเราจะสามารถเลื่อนแผ่นสี่เหลี่ยมต่างๆ ไปสู่เป้าหมายได้ แต่ถ้า *L*+*B* เป็นจำนวนคี่ หมายความว่า ไม่มีทางจะเลื่อนได้สำเร็จ

จงเขียนเมท็อด `isLegal15Puzzle(int[][] b)` ที่ตรวจสอบว่า ตารางที่ให้มาผ่านทางอาเรย์สองมิติ `b` ขนาด 4×4 เป็นตารางที่สามารถเลื่อนไปสู่คำตอบได้หรือไม่ (แต่ละช่องใน `b` เก็บหมายเลขของแผ่น ช่องใดว่างให้เก็บค่า 0)

17. *Sudoku* (数独) เป็นปริศนาที่ได้รับความนิยมอย่างสูง แทนด้วยตารางขนาด 9×9 ภายในแบ่งเป็นตารางย่อย ๆ ขนาด 3×3 จำนวน 9 ตาราง มีตัวเลขใส่ไว้แล้วบางช่อง (ดังรูปซ้ายข้างล่างนี้) สิ่งที่ต้องทำก็คือ จงเติมตัวเลข (เลข 1 ถึง 9) ในช่องที่เหลือโดยที่
    - ไม่มีตัวซ้ำในแต่ละตารางย่อย 3×3 ใด ๆ
    - ไม่มีตัวซ้ำในแถวนอนใด ๆ
    - ไม่มีตัวซ้ำในแถวแนวตั้งใด ๆ

    ตัวอย่างในรูปข้างล่างนี้ ตารางทางขวาเป็นผลเฉลยของตารางทางซ้าย

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

    แบบฝึกหัดข้อนี้ไม่ได้ให้เขียนวิธีหาผลเฉลยของตารางปริศนา แต่ให้เขียนเมท็อด `isSudoku` ซึ่งตรวจสอบว่า ตารางที่ได้รับมา (เป็น `int[][]`) มีค่าที่เก็บตามช่องต่าง ๆ ถูกต้องตามกฎทั้งสามข้อที่เขียนไว้ข้างต้นหรือไม่ (ตารางที่ได้รับไม่จำเป็นต้องเก็บตัวเลขเต็มทุกช่อง ช่องใดที่ยังไม่เติมตัวเลข จะเก็บ 0)

18. จาวามองข้อมูลแบบ `char` เป็นจำนวนเต็มได้ (อ่านหัวข้อเพิ่มเติมในบทที่ 5) จึงใช้อักขระหนึ่งตัวแบบ `char` มาเป็นดัชนีของอาเรย์ได้ จงเขียนโปรแกรมวิเคราะห์คำศัพท์ไทย (c:/java101/riwords.txt ที่ใช้ในบทที่ 5) ว่า ตัวอักษรไทยแต่ละตัวปรากฏกี่ครั้งในแฟ้มคำศัพท์ไทยนี้ และลองวิเคราะห์แบบเดียวกัน กับแฟ้มคำศัพท์อังกฤษ (c:/java101/engwords.txt) (ข้อแนะนำ : สำหรับตัวอักษรอังกฤษใหญ่ ให้สร้างอาเรย์ขนาด 26 ช่อง สมมติว่าชื่อ `c` จะได้ว่า `c['A'-'A']`, `c['B'-'A']`, ..., `c['Z'-'A']` มีไว้เก็บจำนวนตัว A, B, ..., Z ตามลำดับ)

วัตถุสิ่งของ

จากโปรแกรมที่ได้เขียนกันมาซึ่งมุ่งเน้นขั้นตอนการทำงานของคำสั่งต่าง ๆ เพื่อแก้ไขปัญหาที่ต้องการ บทนี้นำเสนอแนวคิดการออกแบบเชิงวัตถุ ที่เน้นการออกแบบข้อมูลให้ตรงกับสภาพงานที่ต้องการประมวลผล โดยนิยามประเภทข้อมูลด้วยองค์ประกอบที่เป็นข้อมูลย่อย กับบริการที่มีให้เพื่อกระทำกับข้อมูลเหล่านั้น เราเรียกลักษณะของข้อมูลประเภทใหม่นี้ว่า *คลาส* และเรียกตัวข้อมูลที่ผลิตได้ว่า *อ็อบเจกต์* เช่น บัญชี พนักงาน สินค้า เป็นต้น ซึ่งเปรียบเสมือนวัตถุสิ่งของที่เราผลิตมาเพื่อประมวลผลภายในโปรแกรม บทนี้นำเสนอองค์ประกอบของคลาส การสร้าง และการใช้งาน โดยยกตัวอย่างเพื่อให้เห็นอย่างเป็นรูปธรรมว่า จะเขียนคลาส และใช้งานอ็อบเจกต์อย่างไร

องค์ประกอบ การสร้าง และการใช้งาน

เราได้เขียนโปรแกรมกันมาพอสมควร หากผู้อ่านลองนึกย้อนกลับไป จะพบว่า ตัวแปรที่ใช้เก็บข้อมูลหลาย ๆ ตัว มีความสัมพันธ์กัน เช่น การบรรยายตำแหน่งของลูกบอลที่เคลื่อนที่ในวินโดว์นั้น เราต้องใช้ตัวแปร `x` และตัวแปร `y` คู่กันไปเสมอ เพราะทั้งคู่นี้รวมกันแล้วจะแทนพิกัดของจุดศูนย์กลางของลูกบอลในวินโดว์ กล่าวได้ว่า จะบรรยายพิกัด ต้องใช้ค่า `x` กับ `y` เสมอ ถ้ามองในระดับลูกบอล ถามว่าจะบรรยายลูกบอล ต้องใช้ค่าอะไรบ้าง ที่เราได้เขียนมา ก็คงเป็นตำแหน่งของลูกบอล, ค่าการเปลี่ยนแปลงตำแหน่งตามแนวนอน `dx` และแนวตั้ง `dy` ของลูกบอล, และรัศมีของลูกบอล ถ้าเราจะเพิ่มสีของลูกบอล ก็คงทำได้ ข้อมูลเหล่านี้ประกอบกันเพื่อบรรยายวัตถุที่เรียกว่า ลูกบอลในโปรแกรมเรา อะไรที่มีในลูกบอลจริง ๆ เช่น น้ำหนัก หรือวัสดุที่ใช้ ที่เราไม่ได้ใช้ในโปรแกรม ก็จะไม่เก็บ ถ้ามีตัวแปรประเภทลูกบอลให้ใช้เลย ไม่ใช่มีแต่ `int`, `double`, `String` แบบพื้น ๆ ที่ใช้กันมา การเขียนโปรแกรมคงจะสะดวกขึ้น

จาวาอนุญาตให้นักเขียนโปรแกรมออกแบบประเภทข้อมูลใหม่ด้วยการเขียนคลาส ซึ่งก็คือคลาสที่เราได้เขียนกันมาตั้งแต่บทแรก คลาสมีองค์ประกอบมากกว่าเมท็อดที่เราได้เขียนมา คลาสในจาวาประกอบด้วย เมท็อดประจำคลาส ตัวแปรประจำคลาส เมท็อดประจำอ็อบเจกต์ ตัวแปรประจำอ็อบเจกต์ และตัวสร้าง การเขียนคลาสเพื่อประกาศประเภทข้อมูลใหม่จะเกี่ยวข้องกับการออกแบบตัวแปรประจำอ็อบเจกต์ เมท็อดประจำอ็อบเจกต์ และตัวสร้างอ็อบเจกต์ (ส่วนเมท็อดประจำคลาสและตัวแปรประจำคลาสเป็นเพียงส่วนเสริมให้เขียนองค์ประกอบอื่นได้ง่ายขึ้น) หัวข้อย่อยต่อไปนี้นำเสนอรายละเอียดของแต่ละองค์ประกอบ กฎเกณฑ์ในการเขียน และการใช้งาน

เมท็อดประจำคลาส

เมท็อดประจำคลาส (class method) คือ เมท็อดที่เราได้เขียนกันมาตลอดตั้งแต่บทแรก (เช่น เมท็อด `main`) คงไม่นำเสนออะไรเพิ่มเติมอีกแล้ว โดยสรุป ในมุมของผู้เรียกใช้เมท็อดประจำคลาส เราต้องเขียนชื่อคลาสนำหน้าชื่อเมท็อด คั่นด้วยเครื่องหมายจุด

ชื่อคลาส . *ชื่อเมท็อดประจำคลาส*(...)

เช่น ต้องการใช้เมท็อด `sin(x)` ของคลาส `Math` ก็เขียน `Math.sin(x)` ถ้าเมท็อดที่เรียกอยู่ในคลาสเดียวกับที่คำสั่งนั้นเรียก ก็ไม่ต้องเขียนชื่อคลาสและจุดก็ได้ สิ่งที่ทำให้เรารู้ว่า เมท็อดหนึ่งคือเมท็อดประจำคลาส อยู่ตรงที่หัวเมท็อดมีคำว่า `static` นั่นเอง (หากไม่เขียนจะเป็นอีกแบบที่จะนำเสนอต่อไป)

ตัวแปรประจำคลาส

ตัวแปรที่เราใช้กันมาทุกตัวอยู่ในเมท็อด เป็นตัวแปรเฉพาะที่ มีให้ใช้ชั่วคราวเมื่อเมท็อดถูกเรียก และหายไปเมื่อเมท็อดทำงานเสร็จ สำหรับกรณีที่เราต้องการตัวแปรที่เก็บข้อมูลไว้ตลอดการทำงานของโปรแกรม ก็เพียงแค่ประกาศตัวแปรนั้นไว้นอกเมท็อด (แต่ต้องอยู่ภายในคลาส) โดยมีกฎว่า ต้องใส่คำว่า `static` ข้างหน้าการประกาศตัวแปร และให้เลือกใส่คำว่า `public` หรือ `private` เพื่อระบุว่าเป็นตัวแปรที่ใช้ได้ทั่วไปหรือใช้ได้เฉพาะภายในคลาส เรียกตัวแปรแบบนี้ว่า *ตัวแปรประจำคลาส* (class variable) เช่น เมท็อด `getDayOfWeek` ในรหัสที่ 8-1 มีตัวแปร `dow` อ้างอิงอาเรย์ที่ถูกสร้างเพื่อเก็บชื่อวันทุกครั้งที่เรียกเมท็อด พอเมท็อดทำงานเสร็จตัวแปร `dow` ก็หายไป เมื่ออาเรย์ที่สร้างไว้ไม่มีตัวแปรใดอ้างอิง ก็กลายเป็นขยะ เมื่อเรียกเมท็อดนี้ใหม่ ก็สร้างใหม่อีก ทำเสร็จ ก็หายไปอีก ดังนั้น จึงควรทำให้ตัวแปร `dow` นี้เป็นตัวแปรประจำคลาส ดังรหัสที่ 8-2 เขียนประกาศตัวแปร `dow` และตั้งค่าเริ่มต้นไว้นอกเมท็อด กำกับด้วยคำว่า `static` ที่ด้านหน้า (ในที่นี้ให้เป็น `private` เพราะต้องการใช้ภายในคลาสเท่านั้น) ทำให้ตัวแปร `dow` ไม่หาย อาเรย์ก็ไม่หาย เรียกเมท็อด `getDayOfWeek` ครั้งใด ก็มีให้ใช้ทันที โดยทั่วไปเราใช้ตัวแปรประจำคลาสเพื่อเก็บค่าคงตัวสาธารณะที่ใช้บ่อย (นั่นคือ เขียนกำกับตัวแปรด้วย `public static`)

เช่น `Math.PI` เป็นตัวแปรประจำคลาส `Math` (เก็บค่าประมาณของ π), `DWindow.RED`, `DWindow.BLACK` เป็นตัวแปรประจำคลาส `DWindow` (เก็บค่าของสี) รวมถึง `System.in` และ `System.out` ก็เป็นตัวแปรประจำคลาส `System` เป็นต้น

```
public class DayOfWeek {
  public static String getDayOfWeek(int d) {
    if (d < 0 || d > 6) throw new IllegalArgumentException();
    String[] dow = {"เสาร์","อาทิตย์","จันทร์","อังคาร","พุธ","พฤหัสบดี","ศุกร์"};
    return dow[d];
  }
```

รหัสที่ 8-1 เมท็อด `getDayOfWeek(d)` คืนชื่อของหมายเลขวัน

```
public class DayOfWeek {
  private static String[] dow =
      {"เสาร์","อาทิตย์","จันทร์","อังคาร","พุธ","พฤหัสบดี","ศุกร์"};
  public static String getDayOfWeek(int d) {
    if (d < 0 || d > 6) throw new IllegalArgumentException();
    return dow[d];
  }
```

ตัวแปรนอกเมท็อด ใช้ได้ในทุก ๆ เมท็อดของคลาสนี้

รหัสที่ 8-2 การใช้ตัวแปรประจำคลาสกับเมท็อด `getDayOfWeek(d)`

ตัวแปรประจำอ็อบเจกต์

คลาสที่เราได้เขียนมามีแต่เมท็อดประจำคลาส แต่คลาสที่เราจะเขียนต่อจากนี้ไป เป็นคลาสที่เขียนเพื่อนิยามประเภทข้อมูลใหม่ เมท็อดประจำคลาสที่ได้เขียนกันมานั้นรับเฉพาะข้อมูลทางพารามิเตอร์มาประมวลผลเท่านั้น แต่คลาสที่เขียนเพื่อแทนประเภทข้อมูลใหม่นี้ จะมีสมาชิกที่เป็นข้อมูลย่อยของประเภทข้อมูลใหม่ กับเมท็อดที่นำข้อมูลย่อยเหล่านี้กับพารามิเตอร์ที่ได้รับมาประมวลผล ขอเริ่มด้วยวิธีการบรรยายสมาชิกที่เป็นข้อมูลกันก่อน รหัสที่ 8-3 แสดงคลาสชื่อ `Ball` ภายในบรรยายสมาชิกที่เป็นข้อมูลของลูกบอล โดยเขียนบรรยายเหมือน กับการเขียนประกาศตัวแปรประจำคลาส แต่ไม่มีคำว่า `static` คลาสนี้มีสมาชิกที่เก็บข้อมูล 5 ตัว `x` กับ `y` (แทนตำแหน่ง), `dx` กับ `dy` (แทนความเร็ว) และ `r` (แทนรัศมี) ใช้ประกอบกันเพื่อบรรยายลักษณะสมบัติของลูกบอล

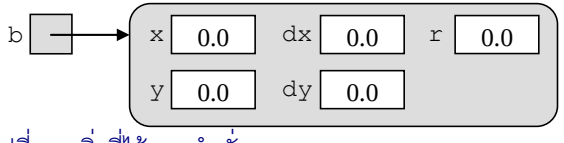
```
01 public class Ball {
02   public double x, y;
03   public double dx, dy;
04   public double r;
05 }
```

เรียกตัวแปรเหล่านี้ว่าเป็นตัวแปรประจำอ็อบเจกต์ประเภท Ball

รหัสที่ 8-3 คลาส `Ball` แสดงสมาชิกของลูกบอล

เมื่อมีคลาส `Ball` เราสามารถใช้คำสั่ง `Ball b = new Ball();` เพื่อสร้างข้อมูลประเภทใหม่มาใช้งาน ซึ่งมีรายละเอียดการทำงานดังนี้ (ดูรูปที่ 8-1)

- `new Ball()` คือ การสร้างข้อมูลประเภท `Ball` มาหนึ่งตัว เสมือนว่า ได้วัตถุสิ่งของมาใช้งานหนึ่งชิ้นที่เราจัดการได้ภายในโปรแกรมคอมพิวเตอร์ ดังนั้น เรียกสิ่งที่สร้างได้นี้ว่า *อ็อบเจกต์* (object) ซึ่งใช้เนื้อที่หน่วยความจำที่เพียงพอกับการเก็บตัวแปรต่าง ๆ ที่เป็นสมาชิก เราเรียกตัวแปรเหล่านี้ว่า *ตัวแปรประจำอ็อบเจกต์* (object variable) โดยหลังการ `new` ระบบจะตั้งค่าเริ่มต้นให้กับตัวแปรประจำอ็อบเจกต์เหมือนกับกรณีการ `new` อาเรย์ นั่นคือ ตัวแปรที่เป็นจำนวนจะได้ค่าศูนย์ ตัวแปรที่เป็น `boolean` จะได้ค่าเท็จ ในกรณีที่ต้องการตั้งค่าอื่นเป็นค่าเริ่มต้น ก็สามารถเขียนไว้ตรงบรรทัดที่ประกาศตัวแปรประจำอ็อบเจกต์ได้เลย เช่น `double r=5;`
- `Ball b` คือ การประกาศตัวแปรชื่อ `b` ที่มีไว้อ้างอิงอ็อบเจกต์ประเภท `Ball` เท่านั้น
- เครื่องหมาย = ระบุว่า ให้นำตำแหน่งที่อยู่ของอ็อบเจกต์ที่สร้างด้วย `new` ไปเก็บไว้ในตัวแปร `b` นั่นคือ ให้ `b` อ้างอิงอ็อบเจกต์ที่สร้างได้ ดังรูปที่ 8-1



รูปที่ 8-1 สิ่งที่ได้จากคำสั่ง `Ball b = new Ball();`

ผู้อ่านเห็นคำว่า “อ้างอิง” คงคิดถึงตัวแปรอ้างอิงอาเรย์ จะว่าไปแล้ว ตัวอาเรย์ก็คืออ็อบเจกต์แบบหนึ่ง จึงใช้แนวคิดเดียวกัน กล่าวคือ อ็อบเจกต์ทุกตัวต้องมีตัวแปรมาอ้างอิง จะมีตัวแปรหลายตัวอ้างอิงอ็อบเจกต์เดียวกันก็ได้ แต่ถ้าไม่มีใครอ้างอิง อ็อบเจกต์นั้นก็เป็นเสมือน “ขยะ” เนื้อที่หน่วยความจำที่เคยใช้เก็บอ็อบเจกต์นั้นจะถูกคืนกลับสู่ระบบ

การใช้สมาชิกที่เป็นตัวแปรประจำอ็อบเจกต์กระทำได้โดยเขียนชื่อตัวแปรประจำอ็อบเจกต์ตามหลังชื่อตัวแปรอ้างอิงอ็อบเจกต์ คั่นด้วยเครื่องหมายจุด ดังนี้

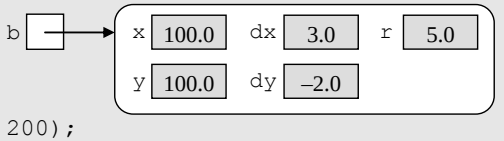
ตัวแปรอ้างอิงอ็อบเจกต์ **.** *ชื่อตัวแปรประจำอ็อบเจกต์*

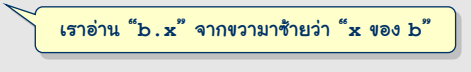
เช่น บรรทัดที่ 4 ถึง 7 ของรหัสที่ 8-4 แสดงการสร้างลูกบอลให้กับตัวแปรอ้างอิง `b` แล้วตั้งค่าให้กับตัวแปรประจำอ็อบเจกต์ที่ `b` อ้างอิงอยู่ บรรทัดที่เหลือในรหัสที่ 8-4 คือคำสั่งสร้างวินโดว์และทำให้ลูกบอลเด้งไปมา

หากเราเขียนรหัสที่ 8-4 ใหม่ โดยแยกภาระการเลื่อนลูกบอล และการวาดลูกบอลออกมาเป็นเมท็อด จะพบว่า เราส่งเพียงตัวแปรอ้างอิงอ็อบเจกต์ลูกบอลไปให้เมท็อดก็พอ ภายในเมท็อดสามารถเข้าใช้ตัวแปรประจำอ็อบเจกต์ทุกตัวได้ผ่านทางตัวแปรอ้างอิงที่ได้รับ ดังรหัสที่ 8-5 เช่น เมท็อด `draw` รับลูกบอลที่ส่งมาจากเมท็อด `main` เก็บในตัวแปร `b` สิ่งที่ส่งมาคือตัวอ้างอิงอ็อบเจกต์ลูกบอล ดังนั้น `b` จึงอ้างอิงลูกบอลลูกเดียวกับที่ใช้ใน `main` เรารับ `b` มาตัวเดียว แต่สามารถ

ใช้หรือเปลี่ยนแปลง x, y, dx, dy และ r ของ b ได้ ส่วนเมท็อด createRandomBall (หน้าถัดไป) มีการสร้างลูกบอลภายในเมท็อด ตั้งค่าเริ่มต้นให้กับตัวแปรประจำอ็อบเจกต์ต่าง ๆ แล้วก็คืนผลกลับไปให้ผู้เรียกอ้างอิงลูกบอลที่สร้างขึ้น การส่งอ็อบเจกต์ให้เมท็อด และการคืนอ็อบเจกต์จากเมท็อด ทำงานในลักษณะเดียวกับการส่งและการคืนอาเรย์ หากเราเขียนเมท็อด move และ draw แบบไม่ใช้แนวคิดของอ็อบเจกต์ ก็ต้องส่งตัวแปรถึง 5 ตัวด้วยกัน ซึ่งยุ่งยากต่อการใช้งาน

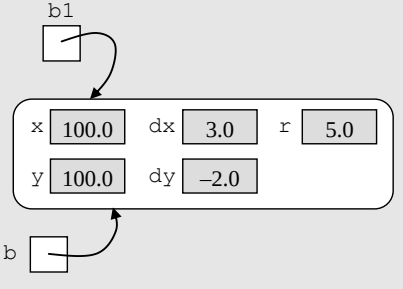
```
01 import jlab.graphics.DWindow;
02 public class TestBall {
03   public static void main(String[] args) {
04     Ball b = new Ball();
05     b.x = 100; b.y = 100;
06     b.dx = 3; b.dy = -2;
07     b.r = 5;
08     DWindow w = new DWindow(200, 200);
09     while (true) {
10       w.clearBackground();
11       b.x += b.dx; b.y += b.dy;
12       if (b.x <= b.r || b.x >= w.getWidth() - b.r) b.dx = -b.dx;
13       if (b.y <= b.r || b.y >= w.getHeight()- b.r) b.dy = -b.dy;
14       b.x = Math.min(Math.max(b.x, b.r), w.getWidth()  - b.r);
15       b.y = Math.min(Math.max(b.y, b.r), w.getHeight() - b.r);
16       w.fillEllipse(DWindow.BLACK, b.x, b.y, 2*b.r, 2*b.r);
17       w.sleep(50);
18     }
19   }
20 }
```





รหัสที่ 8-4 โปรแกรมแสดงลูกบอล แบบใช้อ็อบเจกต์ลูกบอล

```
01 import jlab.graphics.DWindow;
02 public class TestBall {
03   public static void main(String[] args) {
04     DWindow w = new DWindow(200, 200);
05     Ball b1 = createRandomBall(200,200);
06     Ball b2 = createRandomBall(200,200);
07     while (true) {
08       w.clearBackground();
09       move(w, b1); move(w, b2);
10       draw(w, b1); draw(w, b2);
11       w.sleep(50);
12     }
13   }
14   private static void draw(DWindow w, Ball b) {
15     w.fillEllipse(DWindow.BLACK, b.x, b.y, 2*b.r, 2*b.r);
16   }
17   private static double random(double a, double b) {
18     return a + (b - a) * Math.random();
19   }
```



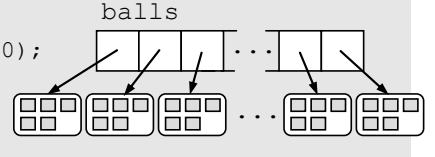
รหัสที่ 8-5 โปรแกรมแสดงลูกบอล แบบใช้เมท็อดที่รับอ็อบเจกต์ไปใช้งาน (มีต่อ)

```
20    private static void move(DWindow w, Ball b) {
21      b.x += b.dx; b.y += b.dy;
22      if (b.x <= b.r || b.x >= w.getWidth() - b.r) b.dx = -b.dx;
23      if (b.y <= b.r || b.y >= w.getHeight()- b.r) b.dy = -b.dy;
24      b.x = Math.min(Math.max(b.x, b.r), w.getWidth()  - b.r);
25      b.y = Math.min(Math.max(b.y, b.r), w.getHeight() - b.r);
26    }
27    private static Ball createRandomBall(double w, double h) {
28      Ball b = new Ball();
29      b.x = w/2; b.y = h/2;
30      b.dx = random(-5, 5);  b.dy = random(-5, 5);
31      b.r =  random(3, 8);
32      return b;
33    }
34  }
```

รหัสที่ 8-5 โปรแกรมแสดงลูกบอล แบบใช้เมท็อดที่รับอ็อบเจกต์ไปใช้งาน (ต่อ)

ขอปรับปรุงโปรแกรมต่ออีกครั้ง คราวนี้ขอปรับให้แสดงลูกบอล 50 ลูก แน่นอนว่า ก็ต้องคิดถึงอาเรย์ ดังรหัสที่ 8-6 แสดงการใช้คำสั่ง `new Ball[50]` ซึ่งเป็นการสร้างแค่อาเรย์ขนาด 50 ช่อง แต่ละช่องเตรียมไว้เก็บตัวอ้างอิงไปยังอ็อบเจกต์ลูกบอล (ซึ่งยังไม่ได้สร้าง) หลังการสร้างอาเรย์ใหม่ แต่ละช่องจะเก็บค่าคงตัว `null` (ซึ่งเป็นค่าคงตัวของภาษาจาวา) การที่ตัวแปรอ้างอิงเก็บค่า `null` จะหมายความว่า ตัวแปรนี้ “ไม่ได้อ้างอิงอ็อบเจกต์ใดๆ” จากนั้นอาศัยวงวน `for` เพื่อสร้างลูกบอล และให้แต่ละช่องในอาเรย์อ้างอิงลูกบอลที่สร้างได้ (บรรทัดที่ 6 และ 7) เมื่อเข้าวงวนไม่รู้จบ ก็ใช้วงวน `for` อีกวงสั่งให้ลูกบอล `balls[i]` ปรับตำแหน่งและความเร็วด้วยเมท็อด `move` ตามด้วยการเรียกเมท็อด `draw` เพื่อแสดงวงกลมในวินโดว์

```
01 import jlab.graphics.DWindow;
02 public class TestBall {
03   public static void main(String[] args) {
04     DWindow w = new DWindow(200, 200);
05     Ball[] balls = new Ball[50];
06     for (int i=0; i<balls.length; i++)
07       balls[i] = createRandomBall(200,200);
08     w.setRepaintDuringSleep(true);
09     while (true) {
10       w.clearBackground();
11       for (int i=0; i<balls.length; i++) {
12         move(w, balls[i]);
13         draw(w, balls[i]);
14       }
15       w.sleep(50);
16     }
17   }
..   // เมท็อด move, draw, createRandomBall, และ random เหมือนที่เขียนในรหัสที่ 8-5
```



ให้ลูกบอลทีละลูกเคลื่อนที่และแสดง

รหัสที่ 8-6 โปรแกรมแสดงลูกบอล แบบใช้อาเรย์ของอ็อบเจกต์

เมท็อดประจำอ็อบเจกต์

เมื่อใดที่เราออกแบบคลาสให้แทนประเภทข้อมูลใหม่ ต้องถามตัวเองสองคำถามนี้เสมอ คือ

1. ประเภทข้อมูลใหม่นี้ประกอบด้วยข้อมูลย่อยอะไรบ้างที่บรรยายลักษณะของอ็อบเจกต์ที่จะสร้าง หัวข้อที่แล้วได้นำเสนอตัวอย่างการเขียนคลาส `Ball` ที่ภายในมีข้อมูลประกอบด้วยตำแหน่ง ความเร็ว และรัศมี ซึ่งก็คือ ตัวแปรประจำอ็อบเจกต์
2. ควรมีบริการอะไร ที่จะให้ผู้ใช้อ็อบเจกต์ใหม่นี้เรียกใช้ให้เกิดประโยชน์ เช่น เราสร้างลูกบอลขึ้นมาทำไม จากตัวอย่างที่นำเสนอ ก็เพื่อให้มันเคลื่อนที่ได้ และแสดงได้ เป็นต้น

คำตอบของคำถามข้อที่สองจะนำไปสู่การเขียนเมท็อดให้กับอ็อบเจกต์ที่ผู้ใช้สามารถเรียกได้เพื่อขอรับบริการที่อ็อบเจกต์นั้นมีให้ จากหัวข้อที่แล้ว ผู้ใช้คลาส `Ball` (ซึ่งคือเมท็อด `main` ของคลาส `TestBall`) ต้องเขียนเมท็อด `move` และ `draw` เอง ทั้ง ๆ ที่ควรเป็นหน้าที่รับผิดชอบของผู้เขียนคลาส `Ball` ว่า จะให้ลูกบอลเคลื่อนอย่างไรและวาดอย่างไร เพราะผู้เขียนคลาส `Ball` ควรเป็นผู้ที่เข้าใจพฤติกรรมและลักษณะสมบัติของลูกบอลที่ประสงค์ให้เป็น ถ้า `Ball` มีเมท็อด `move` และ `draw` ให้ใช้ ผู้ใช้งานลูกบอลจะได้ใช้ `balls[i].move(w)` เพื่อสั่งให้ `balls[i]` เคลื่อนที่บนวินโดว์ `w` และใช้ `balls[i].draw(w)` เพื่อสั่งให้ `balls[i]` วาดตัวเองบนวินโดว์ `w` การใช้งานก็จะสะดวกมากขึ้น หากย้อนกลับไปดูการใช้งาน `DWindow` พบว่า เราใช้ `w.drawLine(...)`, `w.sleep(50)` และบริการอื่น ๆ โดยไม่รู้รายละเอียดการทำงานภายในว่า ทำอย่างไร เพราะเราในฐานะผู้ใช้ต้องการรู้เพียงว่า ใช้อย่างไร ส่วนผู้ที่เขียนคลาส `DWindow` เป็นผู้รับผิดชอบรายละเอียดการทำงานของบริการต่าง ๆ ที่วินโดว์มีให้

กลับมาที่คลาส `Ball` เราจะเพิ่มเมท็อด `move` และ `draw` เพื่อให้เรียกใช้กับอ็อบเจกต์ลูกบอลได้ ดังรหัสที่ 8-7 ขอเน้นว่า เมท็อดที่เขียนนี้ต้องเรียกใช้กับอ็อบเจกต์ คือ จะเรียกใช้ได้ ต้องมีอ็อปเจกต์กำกับการเรียก จึงเรียกว่า *เมท็อดประจำอ็อบเจกต์* (object method) ถ้าต้องการใช้ตัวแปรประจำอ็อบเจกต์ เราเขียน *ตัวแปรอ้างอิงอ็อบเจกต์* . *ชื่อตัวแปรประจำอ็อบเจกต์* เช่น `b.x` เพื่อใช้ตัวแปรชื่อ `x` ประจำอ็อบเจกต์ `b` ถ้าไม่มีอ็อบเจกต์ ก็ย่อมไม่มีตัวแปร `x` ของ `b` สำหรับกรณีของเมท็อดประจำอ็อบเจกต์ เราเขียนรูปแบบการเรียกใช้ ดังนี้

ตัวแปรอ้างอิงอ็อบเจกต์ . *ชื่อเมท็อดประจำอ็อบเจกต์*(...)

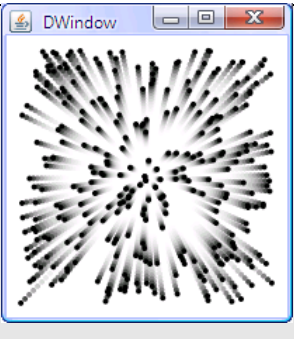
เช่น `b.move(w)` เป็นการเรียกเมท็อดชื่อ `move` ประจำอ็อบเจกต์ `b` การทำงานจะไปที่เมท็อด `move` (บรรทัดที่ 7 ของรหัสที่ 8-7) ภายในเมท็อดสามารถเขียนคำสั่ง หรือประกาศตัวแปรได้ทุกอย่างตามที่ได้ศึกษากันมา นอกจากนี้ยังสามารถใช้ตัวแปรประจำอ็อบเจกต์ที่ถูกเรียกได้ด้วย ตัวแปร `x`, `y`, `dx`, `dy`, และ `r` ที่เราใช้ในเมท็อด คือ ตัวแปรประจำอ็อบเจกต์ที่ถูกเรียก

```
01 import jlab.graphics.DWindow;
02 public class Ball {
03   double x, y;
04   double dx, dy;
05   double r;
06   //---------------------------
07   public void move(DWindow w) {
08     x += dx; y += dy;
09     if (x <= r || x >= w.getWidth() - r) dx = -dx;
10     if (y <= r || y >= w.getHeight()- r) dy = -dy;
11     x = Math.min(Math.max(x, r), w.getWidth()  - r);
12     y = Math.min(Math.max(y, r), w.getHeight() - r);
13   }
14   public void draw(DWindow w) {
15     w.fillEllipse(DWindow.BLACK, x, y, 2*r, 2*r);
16   }
17 }
```

รหัสที่ 8-7 คลาส `Ball` แสดงตัวแปรและเมท็อดประจำอ็อบเจกต์ลูกบอล

ขอสรุปความแตกต่างระหว่างเมท็อดประจำคลาสกับเมท็อดประจำอ็อบเจกต์ ในมุมของผู้เรียกใช้เมท็อด ความแตกต่างอยู่ที่การใช้ชื่อคลาส หรือใช้อ็อบเจกต์นำหน้าเมท็อด [1] สำหรับในมุมของผู้เขียนเมท็อด เมท็อดประจำคลาสมีคำว่า `static` ที่หัวเมท็อด ในขณะที่เมท็อดประจำอ็อบเจกต์ไม่มี รหัสที่ 8-8 คือโปรแกรมแสดงลูกบอลเคลื่อนที่ที่สร้างลูกบอลจากคลาส `Ball` ในรหัสที่ 8-7 และเรียกใช้เมท็อดประจำลูกบอลเพื่อสั่งให้เคลื่อนที่และแสดงผล

```
01 import jlab.graphics.DWindow;
02 public class TestBall {
03   public static void main(String[] args) {
04     DWindow w = new DWindow(200, 200);
05     Ball[] balls = new Ball[300];
06     for (int i=0; i<balls.length; i++)
07       balls[i] = createRandomBall(200,200);
08     w.setRepaintDuringSleep(true);
09     while (true) {
10       w.fade(0.3);
11       for (int i=0; i<balls.length; i++) {
12         balls[i].move(w);
13         balls[i].draw(w);
14       }
15       w.sleep(50);
16     }
17   }
..   ...
```



เรียกเมท็อด move และ draw ประจำอ็อบเจกต์ balls[i]

รหัสที่ 8-8 โปรแกรมแสดงลูกบอล แบบเรียกเมท็อดประจำอ็อบเจกต์

[1] โดยทั่วไปการใช้ตัวแปรหรือใช้เมท็อดประจำคลาส จะเขียนชื่อคลาสนำหน้าชื่อตัวแปรหรือเมท็อดตามลำดับ แต่จาวาอนุญาตให้เราใช้ชื่อตัวแปรอ้างอิงอ็อบเจกต์นำหน้าชื่อตัวแปรหรือเมท็อดประจำคลาสของอ็อบเจกต์นั้นก็ได้

ตัวสร้าง

ที่ผ่านมา เราสร้างอ็อบเจกต์ของคลาส `Ball` ด้วยคำสั่ง `new Ball()` ตามด้วยการตั้งค่าให้กับตัวแปรต่าง ๆ ประจำอ็อบเจกต์ ถ้าเราสามารถเขียน `new Ball(5)` เพื่อสร้างลูกบอลที่มีรัศมี 5 ก็คงจะสะดวกขึ้น หรือเขียน `new Ball(5,10,20)` เพื่อสร้างลูกบอลรัศมี 5 มีศูนย์กลางที่พิกัด (10,20) หรือเขียน `new Ball(5,10,20,-2,3)` เพื่อสร้างลูกบอลรัศมี 5 มีศูนย์กลางที่พิกัด (10,20), และมี `dx=-2, dy=3` ก็จะยิ่งสะดวกขึ้น คำสั่ง `new` *ชื่อคลาส*`(...)` เป็นการสั่งให้ระบบจองเนื้อที่ในหน่วยความจำสำหรับเก็บตัวแปรประจำอ็อบเจกต์ของคลาสที่เขียน ตามด้วยการเรียก*ตัวสร้าง* (contructor) ของคลาสนั้น รหัสที่ 8-9 แสดงคลาส `Ball` ที่มีตัวสร้างอยู่สองแบบ คือ แบบที่รับรัศมีและจุดศูนย์กลาง กับแบบที่รับค่าสำหรับตัวแปรประจำอ็อบเจกต์ทุกตัว ตัวสร้างมีลักษณะคล้ายเมท็อด แต่มีข้อกำหนดในการเขียนตัวสร้างดังนี้

- ***หัวของตัวสร้างของคลาสใด ต้องเขียนชื่อเหมือนชื่อของคลาสนั้น*** ตัวสร้างของคลาส `Ball` ก็ต้องชื่อ `Ball` เวลาเรียกใช้ก็ต้องใช้คำสั่ง `new Ball(...)`
- ***หัวของตัวสร้างต้องไม่มีประเภทของผลลัพธ์*** เนื่องจากตัวสร้างถูกเรียกจากคำสั่ง `new` ซึ่งระบบได้สร้างเนื้อที่ของอ็อบเจกต์ และในที่สุดจะคืนตำแหน่งที่อยู่ของอ็อบเจกต์ที่สร้างได้จากการ `new` ตัวสร้างจึงไม่ต้องกำหนดประเภทผลลัพธ์ และก็ห้ามเขียนคำว่า `void` ด้วย จะได้เห็นชัดว่า เป็นตัวสร้าง (ถ้าเติม `void` จะกลายเป็นเมท็อด)
- ***หัวของตัวสร้างต้องไม่มีคำว่า `static`*** ตัวสร้างถูกเรียกหลังจากระบบได้สร้างเนื้อที่ของอ็อบเจกต์ และล้างค่าให้กับตัวแปรต่าง ๆ ประจำอ็อบเจกต์เรียบร้อยแล้ว (นั่นคือ ให้ค่า 0 กับตัวแปรจำนวน ให้ค่า `false` กับตัวแปรตรรกะ และให้ค่า `null` กับตัวแปรที่มีไว้อ้างอิงอาเรย์หรืออ็อบเจกต์) การทำงานภายในตัวสร้างจึงสามารถใช้ตัวแปรประจำอ็อบเจกต์ได้ เสมือนเป็นเมท็อดประจำอ็อบเจกต์ จึงไม่มีคำว่า `static` เช่นกัน

```
public class Ball {
  ...
  public Ball(double r0, double x0, double y0) {
    r = r0;
    x = x0; y = y0;
    dx = random(-3, 3); dy = random(-3, 3);
  }
  public Ball(double r0,double x0,double y0,double dx0,double dy0){
    r = r0;
    x = x0; y = y0;
    dx = dx0; dy = dy0;
  }
```

ตัวสร้างแบบที่ 1

ตัวสร้างแบบที่ 2

ตัวสร้างอ็อบเจกต์ของคลาส เขียนคล้ายเมท็อด ใช้ชื่อคลาสเป็นชื่อเมท็อด แต่ไม่มีประเภทของผลลัพธ์

รหัสที่ 8-9 ตัวสร้างของคลาส `Ball`

แล้วเราต้องเขียนให้ตัวสร้างทำอะไร ? ภาระหลักของตัวสร้างก็คือ การตั้งค่าเริ่มต้นให้กับตัวแปรประจำอ็อบเจกต์ตามพารามิเตอร์ที่ได้รับ ถ้ารับไม่ครบ ก็ต้องตัดสินใจว่า จะตั้งค่าอะไรให้กับตัวแปรที่เหลือ หรือไม่ก็ปล่อยให้มีค่าตามที่ระบบตั้งให้ตอนเริ่มต้น ถ้าดูตัวสร้างของรหัสที่ 8-9 แบบแรกตั้งรัศมีและตำแหน่งตามพารามิเตอร์ที่รับมา ส่วนความเร็ว `dx, dy` นั้นให้ค่าสุ่ม ส่วนแบบที่สองตั้งตัวแปรทุกตัวตามพารามิเตอร์ จะเขียนตัวสร้างกี่แบบ แต่ละแบบจะตั้งค่าอย่างไร ก็ขึ้นกับผู้ออกแบบว่า จะให้บริการและอำนวยความสะดวกกับผู้ใช้งานในลักษณะใด

รหัสที่ 8-10 แสดงคลาส `Ball` ที่มีหลายองค์ประกอบที่ได้นำเสนอมา ประกอบด้วยตัวแปรประจำอ็อบเจกต์ ตัวสร้าง เมท็อดประจำอ็อบเจกต์ เมท็อดประจำคลาส และเมท็อด `main` ซึ่งเราเขียนไว้ในคลาสนี้เลย (จึงสามารถสั่งเพื่อทดสอบการทำงานของคลาสนี้ได้ด้วย) ให้สังเกตว่า เราเขียนเมท็อด `hitWall` เป็นเมท็อดประจำอ็อบเจกต์แบบส่วนตัวไว้ใช้เอง เขียนเมท็อด `random` ให้เป็นเมท็อดประจำคลาส เพราะเมท็อดนี้ไม่มีการใช้ตัวแปรประจำอ็อบเจกต์เลย [2] และเขียนตัวสร้างไว้สองแบบ เราใช้ตัวสร้างแบบแรกใน `main` ส่วนแบบหลังถึงแม้ไม่มีการใช้งาน แต่ผู้เขียนคลาสคาดว่า น่าจะมีการเรียกใช้ในอนาคต

ถ้าผู้อ่านลองทบทวนโปรแกรมที่เราได้เขียนกันมาตั้งแต่บทแรก จะพบว่า เราได้ใช้และสร้างอ็อบเจกต์กันมาพอสมควรเหมือนกัน (เพียงแต่บอกให้จำ ๆ ไปก่อนเท่านั้น ไม่เคยอธิบายรายละเอียด) ขอเขียนทบทวนเป็นตัวอย่างให้ดูอีกครั้ง

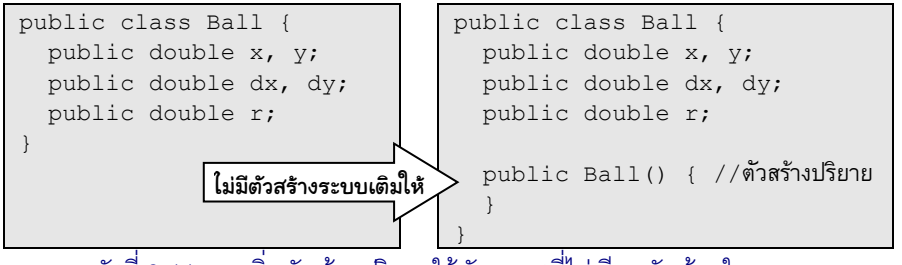
- `new Scanner(System.in)` เป็นการสร้างอ็อบเจกต์ของคลาส `Scanner` ส่ง `System.in` ไปให้ตัวสร้างตั้งค่า เพื่อให้รู้ว่าจะให้เป็นตัวอ่านจากแป้นพิมพ์
- `new Scanner(new File("c/temp/data.txt"))` มีการสร้างอ็อบเจกต์สองตัว ตัวแรกสร้างอ็อบเจกต์ของคลาส `File` ส่งสตริงที่เก็บชื่อแฟ้มไปให้ตัวสร้างของ `File` ได้อ็อบเจกต์ของ `File` กลับมา ส่งให้ตัวสร้างของ `Scanner` เพื่อเปิดแฟ้มและสร้างตัวอ่านจากแฟ้มข้อมูลนั้น
- `new PrintStream(new File("c:/temp/out.txt"))` มีการสร้างอ็อบเจกต์ทำนองเดียวกับของข้อที่แล้ว คือ สร้างอ็อบเจกต์ของ `File` แล้วส่งให้ตัวสร้างของ `PrintStream` เพื่อสร้างอ็อบเจกต์ที่เราสามารถนำไปใช้บันทึกข้อความลงแฟ้มได้
- `new DWindow(200,150)` เป็นการสร้างอ็อบเจกต์ของคลาส `DWindow` โดยส่งความกว้างและความสูงไปให้ตัวสร้างของ `DWindow`
- `new IllegalArgumentException("...")` เป็นการสร้างอ็อบเจกต์ของคลาสที่แทนสิ่งผิดปกติที่ระบบอนุญาตให้ใช้โยนจากเมท็อดที่ทำงานอยู่ โดยตัวสร้างรับสตริงที่จะนำไปใช้เป็นข้อความแจ้งให้ผู้ใช้ทราบ

[2] ถึงแม้เมท็อดหนึ่งจะไม่ใช้ตัวแปรประจำอ็อบเจกต์ในเมท็อดเลย จะเขียนให้เป็นเมท็อดประจำอ็อบเจกต์ก็ไม่ผิด

```
01 import jlab.graphics.DWindow;
02 public class Ball {
03   //------- ตัวแปรประจำอ็อบเจกต์ (object variable) --------
04   public double x, y;
05   public double dx, dy;
06   public double r;
07   //------- ตัวสร้าง (constructor) ----------------------
08   public Ball(double r0, double x0, double y0) {
09     r = r0; x = x0; y = y0;
10     dx = random(-3, 3); dy = random(-3, 3);
11   }
12   public Ball(double r0,double x0,double y0,double dx0,double dy0){
13     r = r0; x = x0; y = y0;
14     dx = dx0; dy = dy0;
15   }
16   //------- เมท็อดประจำอ็อบเจกต์ (object method) ----------
17   public void move(DWindow w) {
18     x += dx; y += dy;
19     if (hitWall(x, w.getWidth()))  dx = -dx;
20     if (hitWall(y, w.getHeight())) dy = -dy;
21     x = Math.min(Math.max(x, r), w.getWidth()  - r);
22     y = Math.min(Math.max(y, r), w.getHeight() - r);
23   }
24   public void draw(DWindow w) {
25     w.fillEllipse(DWindow.BLACK, x, y, 2*r, 2*r);
26   }
27   private boolean hitWall(double c, double length) {
28     return (c <= r || c >= length-r);
29   }
30   //------- เมท็อดประจำคลาส (class method) -------------
31   private static double random(double a, double b) {
32     return a + (b - a) * Math.random();
33   }
34   //------- เมท็อด main เพื่อทดสอบการทำงานของ Ball --------
35   public static void main(String[] args) {
36     DWindow w = new DWindow(200, 200);
37     Ball[] balls = new Ball[300];
38     for (int i=0; i<balls.length; i++)
39       balls[i] = new Ball(2, 100, 100);
40     w.setRepaintDuringSleep(true);
41     while (true) {
42       w.fade(0.3);
43       for (int i=0; i<balls.length; i++) {
44         balls[i].move(w);  balls[i].draw(w);
45       }
46       w.sleep(50);
47     }
48   }
49 }
```

รหัสที่ 8-10 คลาส Ball ที่แสดงครบทุกองค์ประกอบ

ถ้าผู้อ่านย้อนกลับไปที่โปรแกรมในรหัสที่ 8-4 ซึ่งมีการสร้างอ็อบเจกต์ของคลาส `Ball` ที่เขียนในรหัสที่ 8-3 โดยคลาส `Ball` ที่เขียนขณะนั้นไม่มีตัวสร้างเลย แล้วเราเรียก `new Ball()` ในรหัสที่ 8-4 ได้อย่างไร ภาษาจาวากำหนดไว้ว่า ในกรณีที่คลาสไม่มีตัวสร้างเลยสักตัว จาวาจะเพิ่ม*ตัวสร้างปริยาย* (default constructor) ให้หนึ่งตัวที่ไม่รับพารามิเตอร์ใด ๆ และภายในตัวสร้างนี้ไม่มีคำสั่งใด ๆ ดังแสดงในรหัสที่ 8-11 แต่ถ้าเราเขียนตัวสร้างสักตัวหรือมากกว่าให้กับคลาสแล้ว จาวาก็จะไม่เติมตัวสร้างปริยายให้ เช่น เราไม่สามารถสร้างอ็อบเจกต์ด้วย `new Ball()` กับคลาส `Ball` ในรหัสที่ 8-10 ได้ เพราะคลาสนี้ไม่มีตัวสร้าง `Ball()` แบบไม่รับพารามิเตอร์ และระบบก็ไม่ได้เติมตัวสร้างปริยายให้



รหัสที่ 8-11 การเพิ่มตัวสร้างปริยายให้กับคลาสที่ไม่เขียนตัวสร้างใด ๆ เลย

ข้อมูลส่วนบุคคล

ทุกวันนี้ คงไม่มีใครนำบัตรประจำตัวประชาชนหรือสมุดบัญชีธนาคารของตัวเองติดไว้ที่กระเป๋าเสื้อให้สาธารณชนดูข้อมูลได้ตามใจชอบเป็นแน่ เพราะเราถือว่า นี่คือข้อมูลส่วนตัว ใครต้องการดูต้องขอ (จะอนุญาตให้ดูหรือไม่นั้นเป็นอีกประเด็นที่ต้องพิจารณา) ในโลกของอ็อบเจกต์ก็เช่นกัน จะให้ตัวอ็อบเจกต์ปลอดภัย ตัวแปรประจำอ็อบเจกต์ซึ่งเก็บข้อมูลของอ็อบเจกต์ก็ควรถูกปกปิดเป็นข้อมูลส่วนตัว ใช้ได้เฉพาะภายในคลาสเท่านั้น ไม่ใช่ว่า ผู้ที่มีตัวแปรอ้างอิง `p` มายังอ็อบเจกต์ของเราแล้ว จะใช้ `p.id` มาหยิบ `id` ซึ่งคือ เลขประจำตัวของเราไปใช้ได้ หยิบไปใช้ว่าน่ากลัวแล้ว การเปลี่ยนแปลงค่า เช่น `p.id = 1234` ยิ่งน่ากลัวกว่า

เราสามารถป้องกันการเข้าใช้ตัวแปรประจำอ็อบเจกต์จากคลาสอื่นด้วยการเติมคำว่า `private` ไว้หน้าการประกาศตัวแปรประจำอ็อบเจกต์ โดยทั่วไปหลังจากได้ปกปิดเพื่อใช้ส่วนตัวแล้ว ก็มักเขียนเมท็อดให้บริการตามที่อ็อบเจกต์ควรจะมีให้ ซึ่งอาจเป็นเมท็อดขอดูหรือตั้งค่าตัวแปรเหล่านั้น หรืออาจจะใช้ในลักษณะอื่น เช่น รหัสที่ 8-12 แสดงคลาส `BankAccount` มีไว้สร้างบัญชีธนาคาร มีตัวสร้างหนึ่งตัวรับหมายเลขบัญชี มีตัวแปรประจำอ็อบเจกต์ชื่อ `id` ไว้เก็บหมายเลขบัญชี และ `balance` ไว้เก็บยอดเงินคงเหลือในบัญชีซึ่งมีค่าเริ่มต้นที่ 0 (หลังเปิดบัญชีใหม่ ๆ) มีเมท็อดให้ขอดู `id` และ `balance` ชื่อ `getID` และ `getBalance` ตามลำดับ

```
01 public class BankAccount {
02   private String id;       // หมายเลขบัญชี
03   private double balance; // ยอดเงินคงเหลือ
04
05   public BankAccount(String newID) {
06     id = newID;
07   }
08   public String getID() {
09     return id;
10   }
11   public double getBalance() {
12     return balance;
13   }
14   public void deposit(double amt) {
15     if (amt < 0) throw new IllegalArgumentException("amt<0:"+amt);
16     balance += amt;
17   }
18   public void withdraw(double amt) {
19     if (amt < 0) throw new IllegalArgumentException("amt<0:"+amt);
20     if (amt > balance) throw new IllegalArgumentException("ไม่พอ");
21     balance -= amt;
22   }
23   public void transferTo(BankAccount to, double amt) {
24     withdraw(amt);
25     to.deposit(amt);
26   }
27 }
```

รหัสที่ 8-12 ตัวอย่างการใช้ `private` เพื่อปกปิดการเข้าใช้ตัวแปรจากภายนอก

เราไม่อนุญาตให้ผู้ใช้มาเปลี่ยนตัวแปรประจำอ็อบเจกต์ของบัญชีธนาคารโดยตรง จึงให้ตัวแปรทั้งสองเป็น `private` โดยเฉพาะหมายเลขบัญชี หลังจากเปิดบัญชีแล้ว จะไม่มีการเปลี่ยนแปลงเด็ดขาด จึงไม่มีเมท็อดใดที่ทำหน้าที่เปลี่ยน `id` อ็อบเจกต์บัญชีธนาคารก็ต้องมีบริการที่เกี่ยวกับการฝากและถอนเงิน ซึ่งมีผลกับค่าของ `balance` ดังนั้น จึงมีเมท็อด `deposit` และ `withdraw` เพื่อฝากและถอนเงินตามลำดับ โดยเราสามารถควบคุมพฤติกรรมของการฝากหรือถอนได้ ดังที่แสดงในรหัสที่ 8-12 บรรทัดที่ 15 ตรวจสอบก่อนว่า ห้ามฝากเงินเป็นจำนวนติดลบ ทำนองเดียวกันกับในบรรทัดที่ 19 ก็ไม่ยอมให้ถอนเงินเป็นจำนวนติดลบเช่นกัน และบรรทัดที่ 20 ก็ห้ามไม่ให้ถอนเงินเกินยอดเงินที่มีในบัญชี (คือห้ามไม่ให้ `amt` ที่อยากถอนมากกว่า `balance` ของบัญชี) ด้วยวิธีการประกาศตัวแปรประจำอ็อบเจกต์ให้ใช้ได้ส่วนตัวเฉพาะภายในคลาสที่เขียนเท่านั้น ทำให้เราสบายใจได้ว่า ข้อมูลของอ็อบเจกต์จะถูกควบคุมให้มีค่าตามสถานะที่เป็นไปตามข้อกำหนด เช่น หากเราจะเขียนคลาสบัญชีธนาคารแบบยอมให้ถอนเงินเกินยอดคงเหลือได้ภายในวงเงินที่กำหนด ก็สามารถเปลี่ยนการตรวจสอบ `amt` และ `balance` ในเมท็อด `withdraw` ได้

`transferTo` (บรรทัดที่ 23) ซับซ้อนเล็กน้อย เมท็อดนี้โอนเงินจากบัญชีของอ็อบเจกต์ที่ถูกเรียกไปยังบัญชี `to` เป็นเงิน `amt` ต้องขอเน้นอีกครั้งว่า เมท็อดประจำอ็อบเจกต์ถูกเรียกใช้ได้ต้องมีอ็อบเจกต์นำหน้าชื่อเมท็อด เมื่อมาทำที่เมท็อดนี้ ตัวแปรประจำอ็อบเจกต์ทั้งหลาย ก็คือตัวแปรประจำอ็อบเจกต์ที่ถูกเรียกนั่นเอง ดังนั้น ในเมท็อด `transferTo` เราสามารถโอนเงินจาก `balance` (ของอ็อบเจกต์ที่ถูกเรียก) ไปยัง `to.balance` (คือ `balance` ของอ็อบเจกต์ `to`) แต่เราคงไม่ใช้คำสั่ง `balance -= amt` ตามด้วย `to.balance += amt` เพราะควรใช้บริการ `withdraw` และ `deposit` จะสะดวกกว่า จึงเขียน `withdraw(amt)` (แทนการถอนจากบัญชีตัวเอง) ตามด้วย `to.deposit(amt)` (แทนการฝากเข้าบัญชี `to`) ในบรรทัดที่ 24 และ 25 ตามลำดับ (ให้สังเกตว่า เราต้องถอนก่อนฝาก ไม่ใช่ฝากก่อนถอน)

รหัสที่ 8-13 แสดงโปรแกรมทดสอบการใช้งานคลาส `BankAccount` เริ่มด้วยการสร้างสองบัญชีให้กับตัวแปร `ba1` และ `ba2` จากนั้นฝากเงินเข้าบัญชีทั้งสองเป็นเงิน 1000 และ 500 ตามลำดับ ตามด้วยคำสั่ง `ba1.transferTo(ba2,200)` ซึ่งคือ การโอนเงินจาก `ba1` ไปให้ `ba2` เป็นจำนวน 200 ปิดท้ายด้วยการใช้ `getBalance` เพื่อขอยอดเงินในบัญชีไปแสดง (ได้ 800.0, 700.0 แสดงทางจอภาพ) ให้สังเกตว่า เราใช้ `ba1.balance` หรือ `ba2.balance` เพื่อขอใช้ยอดเงินในบัญชีไม่ได้เพราะ `balance` เป็น `private` อ่านจากนอกคลาสไม่ได้ แต่ถ้าย้าย `main` นี้ไปไว้ในคลาส `BankAccount` การเขียน `ba1.balance` หรือ `ba2.balance` ย่อมทำได้ เพราะเป็นการใช้ในคลาสเดียวกับที่ตัวแปรนี้ปรากฏอยู่

```
01 public class TestBankAccount {
02   public static void main(String[] a) {
03     BankAccount ba1 = new BankAccount("08-1287-91");
04     BankAccount ba2 = new BankAccount("08-0981-93");
05     ba1.deposit(1000);
06     ba2.deposit(500);
07     ba1.transferTo(ba2, 200);
08     System.out.println(ba1.getBalance() + "," + ba2.getBalance());
09   }
10 }
```

รหัสที่ 8-13 โปรแกรมทดสอบการใช้งาน `BankAccount`

toString กับ equals

เพื่ออำนวยความสะดวกให้กับผู้ใช้คลาสใหม่ที่เราเขียนขึ้น โดยทั่วไปจะเขียนเมท็อดประจำอ็อบเจกต์ชื่อ `toString` เพื่อคืนสตริงที่บรรยายลักษณะของอ็อบเจกต์ และเมท็อด `equals` เพื่อให้ผู้ใช้เปรียบเทียบความเท่ากันของอ็อบเจกต์ของคลาสใหม่นี้

เมท็อด toString ช่วยให้นักเขียนโปรแกรมส่งอ็อบเจกต์ไปให้ System.out.print แสดงผลทางจอภาพได้เลย เช่น เขียน System.out.println(ba1 + "," + ba2) แทนบรรทัดที่ 8 ในรหัสที่ 8-13 เพื่อแสดงรายละเอียดของบัญชี ba1 และ ba2 ได้เลย เนื่องจากเมื่อใดมีการต่อสตริงกับอ็อบเจกต์ หรือส่งอ็อบเจกต์ไปแสดงผล ระบบจะเรียกเมท็อด toString ของอ็อบเจกต์นั้นให้อัตโนมัติ ดังนั้น ในฐานะผู้เขียนคลาส ควรให้บริการ toString ซึ่งคืนสตริงที่แสดงถึงลักษณะสมบัติของอ็อบเจกต์ โดยทั่วไปมักนำค่าของตัวแปรประจำอ็อบเจกต์มาแปลงเป็นสตริงแล้วประกอบกันให้สื่อความหมาย รหัสที่ 8-14 แสดง toString ของ BankAccount ซึ่งคืนสตริงที่ได้มาจากการนำหมายเลขบัญชี (id) มาต่อกับยอดเงิน (balance)

```
public class BankAccount {
  ...
  public String toString() {
    return "[" + id + "," + balance + "]";
  }
```

รหัสที่ 8-14 เมท็อด toString ของคลาส BankAccount

toString อำนวยความสะดวกให้กับนักเขียนโปรแกรมระหว่างการพัฒนาซอฟต์แวร์ โดยเฉพาะในช่วงที่กำลังหาจุดบกพร่องในโปรแกรม นักเขียนโปรแกรมมักแสดงค่าของอ็อบเจกต์เพื่อตรวจสอบว่า อ็อบเจกต์ที่สนใจมีลักษณะสมบัติเป็นไปตามที่คาดหวังหรือไม่ระหว่างการทำงาน ดังนั้น จึงมักเขียน toString ให้คืนสตริงที่ “คน” (ซึ่งมักหมายถึงนักเขียนโปรแกรม) อ่านรู้เรื่อง

เมท็อดที่ควรเขียนให้กับคลาสที่ออกแบบใหม่อีกเมท็อดหนึ่งคือ equals หากลองย้อนกลับไปคิดถึงการเปรียบเทียบสตริง ได้เคยนำเสนอว่า ไม่ควรเขียน a == b เพื่อเปรียบเทียบว่า สตริง a มีค่าเหมือนสตริง b หรือไม่ แต่ให้ใช้ a.equals(b) แทน ทั้งนี้เพราะสตริงก็คืออ็อบเจกต์ ตัวแปรสตริงก็คือตัวแปรที่อ้างอิงไปยังอ็อบเจกต์สตริง ดังนั้น ถ้า a และ b เป็นตัวแปรสตริง, a==b จะเป็นจริง ก็เมื่อ a และ b ชี้ไปยังสตริงตัวเดียวกัน แต่เนื่องจากสตริงที่มีค่าเหมือนกันอาจมีหลายตัวได้ จึงใช้ == เปรียบเทียบความเท่ากันไม่ได้ เช่น "MA".toLowerCase()=="ma" จะได้ค่า false สรุปว่า ถ้า p และ q เป็นตัวแปรอ้างอิงอ็อบเจกต์

- ใช้ p==q เพื่อตรวจสอบว่า p และ q ชี้ไปยังอ็อบเจกต์ตัวเดียวกันหรือไม่
- ใช้ p.equals(q) เพื่อตรวจสอบว่า p และ q ชี้ไปยังอ็อบเจกต์ที่มีค่าเท่ากันหรือไม่

จึงเป็นเรื่องที่ปฏิบัติกันทั่วไปว่า ถ้าคลาสที่เราออกแบบใหม่มีการสร้างอ็อบเจกต์ที่อาจถูกนำไปเปรียบเทียบความเท่ากัน ก็ต้องเขียนเมท็อด equals จะเขียนเมท็อดนี้อย่างไร ก็ขึ้นกับตัวแปรประจำอ็อบเจกต์ที่ผู้ออกแบบคลาสได้กำหนดไว้ให้เป็นลักษณะสมบัติของอ็อบเจกต์ เช่น ในกรณีของบัญชีธนาคาร อ็อบเจกต์สองตัวจะเป็นบัญชีที่เหมือนกัน ก็ควรมีหมายเลขบัญชีและยอดเงินเท่ากัน เขียนได้ดังรหัสที่ 8-15

```
public class BankAccount {
  ...
  public boolean equals(BankAccount b) {   //   อ่านหมายเหตุข้างล่าง [3]
    return id.equals(b.id)  && balance == b.balance;
  }
```

รหัสที่ 8-15 เมท็อด equals ของคลาส BankAccount

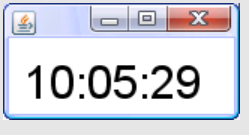
ตัวอย่าง

หัวข้อนี้นำเสนอตัวอย่างการเขียนและการนำไปใช้งาน เริ่มจากคลาสเล็ก ๆ แทนเวลา คลาสของจำนวนเชิงซ้อน คลาสวินโดว์วาดกราฟเส้น คลาสข้อมูลอนุกรมเวลา และคลาสสำหรับเกมจับคู่

คลาสเวลา

ขอเริ่มด้วยคลาสเล็ก ๆ ที่มีไว้เก็บเวลา ตั้งชื่อว่า Time (รหัสที่ 8-17) ภายในมีตัวแปร hour, minute, และ second แทนเวลา มีตัวสร้างให้สองตัว แบบที่ไม่รับพารามิเตอร์ แบบนี้จะนำเวลาปัจจุบันมาตั้งให้ [4] กับแบบที่รับพารามิเตอร์สามตัว คือ ชั่วโมง นาที และวินาที มีเมท็อดให้บริการคืนค่าของตัวแปรประจำอ็อบเจกต์, มีเมท็อดมาตรฐาน toString ที่คืนสตริงของเวลาในรูปแบบ hh:mm:ss และมีเมท็อด increment ที่เพิ่มเวลาให้อีก 1 วินาที (การเพิ่มอาจมีการทดไปยังนาที ทดไปยังชั่วโมง หรือเปลี่ยนชั่วโมงเป็น 0 เมื่อเป็น 24 นาฬิกา) เราสามารถนำคลาสนี้มาสร้างอ็อบเจกต์เพื่อเขียนโปรแกรมแสดงนาฬิกาที่แสดงค่าของเวลาทุกวินาที ดังรหัสที่ 8-16

```
01 import jlab.graphics.DWindow;
02 public class DigitalClock {
03   public static void main(String[] args) {
04     DWindow w = new DWindow(150, 50);
05     Time t = new Time();
06     while (true) {
07       w.clearBackground();
08       w.drawString(t.toString(), 30, 10, 10);
09       w.sleep(1000);
10       t.increment();
11     }
12   }
13 }
```



ขอเวลาปัจจุบัน

แสดงสตริงของเวลา t ขนาดตัวอักษร 30 ที่พิกัด (10,10) บนวินโดว์ w

หยุดทำงาน 1 วินาที แล้วเพิ่มค่าใน t อีกหนึ่งวินาที

รหัสที่ 8-16 โปรแกรมแสดงนาฬิกาตัวเลข

[3] เมท็อด equals ที่เขียนนี้ มีหัวเมท็อดที่ไม่ตรงตามมาตรฐานของจาวา ซึ่งจะนำเสนอรายละเอียดในบทถัดไป

[4] เวลาขณะโปรแกรมทำงานหาได้จากการสร้างอ็อบเจกต์ของคลาสมาตรฐานที่มีชื่อเต็มคือ java.util.Date หลังสร้างแล้ว สามารถใช้เมท็อด getHours(), getMinutes(), และ getSeconds() ดังแสดงในบรรทัดที่ 9 ของรหัสที่ 8-17

```
01 import java.util.Date;
02 public class Time {
03   private int hour;
04   private int minute;
05   private int second;
06
07   public Time() {
08     Date now = new Date();
09     setTime(now.getHours(), now.getMinutes(), now.getSeconds());
10   }
11   public Time(int h, int m, int s) {
12     setTime(h, m, s);
13   }
14   public void setTime(int h, int m, int s) {
15     hour = h; minute = m; second = s;
16   }
17   public int hour() {
18     return hour;
19   }
20   public int minute() {
21     return minute;
22   }
23   public int second() {
24     return second;
25   }
26   public void increment() {  // เพิ่ม 1 วินาที
27     second = (second + 1) % 60;
28     if (second == 0) {
29       minute = (minute + 1) % 60;
30       if (minute == 0) hour = (hour + 1) % 24;
31     }
32   }
33   public String toString() {
34     return twoDigits(hour) + ":" +
35            twoDigits(minute) + ":" + twoDigits(second);
36   }
37   private String twoDigits(int d) {
38     String s = "00" + d;
39     return s.substring(s.length()-2, s.length());
40   }
41 }
```

ใช้คลาสมาตรฐาน Date ของจาวา เพื่อขอดูเวลาปัจจุบัน

เมท็อดนี้ไม่ได้ตรวจสอบพารามิเตอร์ ควรปรับปรุง

เมท็อดคืนค่าของตัวแปรประจำอ็อบเจกต์

เมท็อดที่ควรเขียนให้บริการคืนสตริงที่แทนค่าของอ็อบเจกต์

แปลงจำนวนเต็ม d ให้เป็นสตริงของเลขโดด 2 หลัก

รหัสที่ 8-17 คลาส Time

คลาสจำนวนเชิงซ้อน

จาวามีจำนวนเต็ม int และจำนวนจริง double ให้ใช้ แต่ก็มีปัญหาว่า int เก็บค่าได้จำกัด double เก็บค่าได้ไม่แม่นยำ 100% ถ้าต้องการเก็บจำนวนเต็มขนาดไม่จำกัด ก็ต้องใช้คลาสมาตรฐานของจาวาชื่อ BigInteger หรือถ้าต้องการจำนวนจริงแม่นยำสูง ๆ ตามที่

ต้องการ ก็ต้องใช้ BigDecimal (ผู้อ่านลองอ่านรายละเอียดใน help file ของทั้งสองคลาสดู) รหัสที่ 8-18 แสดงตัวอย่างการใช้ BigInteger เพื่อคำนวณหาค่า 100! จะได้ผลดังนี้

100! = 30414093201713378043612608166064768844377641568960512000000000000

```
01 import java.math.BigInteger;
02 public class TestBigInteger {
03   public static void main(String[] args) {
04     int n = 100;
05     System.out.println(n + "! = " + fac(n));
06   }
07   public static BigInteger fac(int n) {
08     BigInteger nfac = new BigInteger("1");
09     for (int i=1; i<=n; i++) {
10       nfac = nfac.multiply(new BigInteger("" + i));
11     }
12     return nfac;
13   }
14 }
```

ตัวสร้างของ BigDecimal รับสตริง

$$n! = 1\times2\times3\times \ldots \times(n-1)\times n$$

รหัสที่ 8-18 ตัวอย่างการใช้งานคลาส BigInteger

แต่ถ้าต้องการประมวลผลด้วยจำนวนเชิงซ้อน เช่น การหารากของฟังก์ชัน $0.5x^2 - x + 1$ ซึ่งคือ $(1+\sqrt{-1})$ กับ $(1-\sqrt{-1})$ จะเขียนโปรแกรมอย่างไร จาวาไม่มีข้อมูลประเภทจำนวนเชิงซ้อนให้ใช้ ทั้งที่จำนวนเชิงซ้อนมีใช้ในงานวิเคราะห์คำนวณทางวิศวกรรมมากมาย วิธีที่ง่าย ๆ ในการประมวลผลจำนวนเชิงซ้อน คือ แทนจำนวนเชิงซ้อนหนึ่งตัวโดยอาศัยตัวแปรสองตัว real เก็บส่วนที่เป็นจำนวนจริง และ imag เก็บส่วนที่เป็นจำนวนจินตภาพ สองส่วนนี้ต้องใช้คู่กันไปเสมอ สถานการณ์ที่เราต้องใช้ข้อมูลหลายตัวประกอบกัน คู่กันไปตลอดเช่นนี้ นำเราไปสู่การเขียนคลาสใหม่ ขอตั้งชื่อคลาสใหม่ที่แทนจำนวนเชิงซ้อนว่า Complex ภายในมีตัวแปรประจำอ็อบเจกต์เป็น double สองตัวชื่อ real และ imag มีเมท็อดประจำอ็อบเจกต์ไว้ประมวลผลซึ่งคงหนีไม่พ้นการบวก ลบ คูณ หารจำนวนเชิงซ้อน ดังนี้ (ให้ $i = \sqrt{-1}$)

- การบวก : $(a + bi) + (c + di) = (a + c) + (b + d)i$
- การลบ : $(a + bi) - (c + di) = (a - c) + (b - d)i$
- การคูณ : $(a + bi)(c + di) = ac + adi + bci + bdi^2 = (ac - bd) + (bc + ad)i$
- การหาร : $\frac{(a+bi)}{(c+di)} = \left(\frac{ac+bd}{c^2+d^2}\right) + \left(\frac{bc-ad}{c^2+d^2}\right)i$ โดยที่ c และ d ต้องไม่เท่ากับ 0

ระหว่างการเขียนคลาส เขียนเมท็อดในคลาส ขอให้คิดถึงการใช้งานด้วย สมมติว่ามีจำนวนเชิงซ้อน c1 กับ c2 ต้องการนำมาบวกกัน เราคงเขียน c1 + c2 ไม่ได้ เพราะจาวาไม่อนุญาตให้เขียน c1 กับ c2 เป็นอ็อบเจกต์ก็ต้องใช้วิธีเรียกเมท็อด กำหนดให้ add เป็นชื่อเมท็อดการบวก จะเรียกใช้เมท็อดอย่างไร และหมายความว่าอย่างไร ตัวอย่างเช่น

- `c1.add(c2)`: เขียนแบบนี้เป็นเมท็อดประจำอ็อบเจกต์ เป็นการเรียกเมท็อด `add` ของ `c1` แล้วส่ง `c2` ไปให้ ก็คงต้องคิดต่อว่า จะให้ผลการบวกอยู่ที่ใด เช่น
  - `c1.add(c2)` แทนการบวก `c2` เพิ่มเข้าไปใน `c1` หรือ
  - `c1.add(c2)` แทนการบวก `c1` เพิ่มเข้าไปใน `c2` หรือ
  - `c1.add(c2)` แทนการบวก `c1` กับ `c2` ได้ผลบวกคืนกลับมาเป็นอ็อบเจกต์ใหม่ที่เก็บผลลัพธ์ โดย `c1` และ `c2` ไม่เปลี่ยนแปลง
- `Complex.add(c1,c2)` : เขียนแบบนี้แสดงว่า เป็นเมท็อดประจำคลาส ผลที่ได้สามารถทำได้สามแบบเหมือนกรณีของเมท็อดประจำอ็อบเจกต์

ขอเลือกใช้ `c1.add(c2)` ซึ่งคืนอ็อบเจกต์ใหม่ที่เก็บผลบวก โดย `c1` และ `c2` ไม่เปลี่ยนแปลง ซึ่งเป็นรูปแบบเดียวกับของ `BigInteger` การให้คืนผลกลับมาทำให้เราสามารถเขียนเรียกเมท็อดอื่นต่อ ๆ กันไปได้เรื่อย เช่น `c1.add(c2).multiply(c3).substract(c4)` เนื่องจากการเรียกเมท็อดจะทำจากซ้ายไปขวา ดังนั้น คำสั่งข้างต้นคือการนำ `c1` บวกกับ `c2` ได้ผลคืนมาคูณกับ `c3` ได้ผลคืนมาลบด้วย `c4` ซึ่งคือ `(c1+c2)×c3-c4`

นอกจากนี้ ขอเขียนคลาส `Complex` ในลักษณะที่หลังจากสร้างอ็อบเจกต์ให้มีค่าตามที่ต้องการแล้ว จะคงค่านั้นไปตลอด เปลี่ยนแปลงค่าไม่ได้อีก เรียกอ็อบเจกต์ลักษณะนี้ว่า *อ็อบเจกต์ที่เปลี่ยนค่าไม่ได้* (immutable object) คลาส `String` และ `BigInteger` ก็ผลิตอ็อบเจกต์แบบนี้ อ็อบเจกต์ที่เปลี่ยนค่าไม่ได้นั้นปลอดภัย สร้างปัญหาน้อย จะเขียนคลาสให้อ็อบเจกต์เปลี่ยนค่าไม่ได้ ต้องเขียนให้ตัวสร้างรับค่าเริ่มต้น ตัวแปรประจำอ็อบเจกต์ต้องเป็น `private` ไม่มีเมท็อดใดเปลี่ยนค่าตัวแปรเหล่านี้ และผลลัพธ์ของการกระทำกับอ็อบเจกต์ให้เก็บในอ็อบเจกต์ใหม่แล้วคืนกลับให้ผู้เรียก รหัสที่ 8-19 แสดงคลาส `Complex` มีเมท็อดคืนส่วนจำนวนจริงและจินตภาพ มีบริการบวก ลบ คูณ หาร และหาค่าสัมบูรณ์ มีตัวสร้างซึ่งรับส่วนจำนวนจริงและจินตภาพ เช่น `new Complex(2,4)` ได้จำนวนเชิงซ้อน $(2 + 4i)$ และมีเมท็อด `toString` และ `equals` ด้วย

การหารากของสมการด้วยวิธีของนิวตัน

ในเรื่องการวิเคราะห์เชิงตัวเลข วิธีของนิวตัน (Newton's method) เป็นวิธีที่รู้จักและใช้ในการหารากของฟังก์ชัน กำหนดให้ $f(z)$ คือ ฟังก์ชันที่สนใจ มี $f'(z)$ เป็นอนุพันธ์ของ $f(z)$ ถ้า z_n คือค่าประมาณที่ทำให้ $f(z_n) \approx 0$ จะได้ $z_{n+1} = z_n - f(z_n) / f'(z_n)$ เป็นค่าประมาณของรากที่ดีกว่า z_n ตัวอย่างเช่น $f(z) = z^2 - 3z + 2$ มี $f'(z) = 2z - 3$ ถ้าให้ $z_0 = 4$ จะได้

- $f(4) = 6$, $f'(4) = 5$, $z_1 = 4 - 6/5 = 2.8$
- $f(2.8) = 1.44$, $f'(2.8) = 2.6$, $z_2 = 2.8 - 1.44/2.6 = 2.246$
- $f(2.246) = 0.307$, $f'(2.246) = 1.492$, $z_3 = 2.246 - 0.307/1.492 = 2.041$
- $f(2.041) = 0.042$, $f'(2.041) = 1.081$, $z_4 = 2.041 - 0.042/1.081 = 2.002$
- $f(2.002) = 0.002$, $f'(2.002) = 1.003$, $z_5 = 2.002 - 0.002/1.003 = 2.000$
- $f(2.000) = 0.000$, $f'(2.000) = 1.000$, $z_6 = 2.000 - 0.000/1.000 = 2.000$

```
01 public class Complex {
02   private double real;  // จำนวนจริง
03   private double imag;  // จำนวนจินตภาพ
04   public Complex(double re, double im) {
05     real = re; imag = im;
06   }
07   public double real() {
08     return real;
09   }
10   public double imag() {
11     return imag;
12   }
13   public String toString() {
14     return "(" + real + ", " + imag + "i)";
15   }
16   public boolean equals(Complex z) {
17     return real == z.real && imag == z.imag;
18   }
19   public Complex add(Complex z) {
20     return new Complex(real + z.real, imag + z.imag);
21   }
22   public Complex substract(Complex z) {
23     return new Complex(real - z.real, imag - z.imag);
24   }
25   public Complex multiply(Complex z) {
26     return new Complex(real*z.real - imag*z.imag,
27                        imag*z.real + real*z.imag);
28   }
29   public Complex divide(Complex z) {
30     double d = z.real*z.real + z.imag*z.imag;
31     return new Complex((real*z.real + imag*z.imag)/d,
32                        (imag*z.real - real*z.imag)/d);
33   }
34   public double abs() {
35     return Math.sqrt(real*real + imag*imag);
36   }
37   public Complex add(double c) {
38     return new Complex(real + c, imag);
39   }
40   public Complex substract(double c) {
41     return new Complex(real - c, imag);
42   }
43   public Complex multiply(double c) {
44     return new Complex(c * real, c * imag);
45   }
46   public Complex divide(double c) {
47     return new Complex(real / c, imag / c);
48   }
49 }
```

Complex ผลิตอ็อบเจกต์ที่เปลี่ยนค่าไม่ได้ เมท็อด add, substract, multiply, และ divide จึงต้องผลิตอ็อบเจกต์ใหม่ที่เก็บผลลัพธ์

$$(a+bi)+(c+di)=(a+c)+(b+d)i$$

$$(a+bi)-(c+di)=(a-c)+(b-d)i$$

$$(a+bi)(c+di)=(ac-bd)+(bc+ad)i$$

$$\frac{(a+bi)}{(c+di)}=\left(\frac{ac+bd}{c^2+d^2}\right)+\left(\frac{bc-ad}{c^2+d^2}\right)i$$

มีเมท็อดบวก ลบ คูณ หาร กับจำนวนจริง เพื่อความสะดวกในการใช้งาน

รหัสที่ 8-19 คลาส Complex สำหรับใช้งานจำนวนเชิงซ้อน

ถ้าให้ z_0 เริ่มที่ค่าอื่น ก็อาจได้รากอีกตัว เช่น ให้ $z_0 = 0$ จะได้ $z_1 = 0.667$, $z_2 = 0.933$, $z_3 = 0.996$, $z_4 = 0.999$, $z_5 = 0.999$, $z_6 = 1.000$ ค่าใหม่จะลู่เข้าหารากใด ก็ขึ้นกับว่า จุดเริ่มต้นอยู่ใกล้ค่าใดกว่ากัน วิธีนิวตันใช้หารากที่เป็นจำนวนเชิงซ้อนก็ได้ ถ้าเราคำนวณด้วยจำนวนเชิงซ้อนให้หมด รหัสที่ 8-20 แสดงโปรแกรมหารากของฟังก์ชันด้วยวิธีของนิวตัน โดยใช้คลาส `Complex` ที่ได้นำเสนอมา

```
01 import java.util.Scanner;
02 public class NewtonMethod {
03   public static void main(String[] args) {
04     Scanner kb = new Scanner(System.in);
05     System.out.print("initial real = ");
06     double r = kb.nextDouble();
07     System.out.print("initial imag = ");
08     double i = kb.nextDouble();
09     Complex root = findRoot(new Complex(r, i));
10     System.out.println(root);
11   }
12   public static Complex findRoot(Complex z1) {
13     Complex z0;
14     do {
15       z0 = z1;
16       z1 = z0.substract(f(z0).divide(df(z0)));
17     } while (z1.substract(z0).abs() > 1e-10);
18     return z1;
19   }
20   private static Complex f(Complex z) {
21     return z.multiply(z).multiply(z).substract(1);
22   }
23   private static Complex df(Complex z) {
24     return z.multiply(z).multiply(3);
25   }
26 }
```

รับค่าเริ่มต้นจากผู้ใช้

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

วนจนได้รากที่ละเอียดพอสมควร

$f(z) = z^3 - 1$

$f'(z) = 3z^2$

รหัสที่ 8-20 โปรแกรมหารากของฟังก์ชัน $f(z) = z^3 - 1$ ด้วยวิธีของนิวตัน

คลาสวินโดว์แสดงกราฟเส้น

โปรแกรมวาดกราฟเส้นที่เคยนำเสนอในรหัสที่ 7-21 รับข้อมูลในอาเรย์หนึ่งมิติมาวาดกราฟเส้นบนวินโดว์ หากต้องการเขียนโปรแกรมที่มีการวาดกราฟเส้นในลักษณะอื่น ก็สามารถนำโปรแกรมนี้ไปปรับปรุงได้ แต่ถ้าเราปรับโปรแกรมนี้มาใช้แนวคิดการเขียนแบบคลาสและอ็อบเจกต์ ทำให้มีประเภทข้อมูลใหม่ชื่อ `LineGraph` มีเมท็อดชื่อ `plot` ให้เรียกใช้ จะอำนวยความสะดวกให้กับผู้ใช้งานทั่วไป หัวข้อนี้นำเสนอการเขียนคลาส `LineGraph` ที่นำข้อมูลจากอ็อบเจกต์ของคลาส `TimeSeries` ไปแสดงเป็นกราฟเส้นบนวินโดวส์ ตารางที่ 8-1 และตารางที่ 8-2 แสดงตัวสร้างและเมท็อดของคลาส `LineGraph` และ `TimeSeries` ตามลำดับ

ตารางที่ 8-1 ตัวสร้างและเมท็อดประจำอ็อบเจกต์ของ LineGraph

การเรียกใช้	ความหมาย
LineGraph(w,h)	ตัวสร้างรับความกว้าง w และความสูง h ของวินโดว์
g.plot(color, ts)	แสดงกราฟเส้นสี color จากข้อมูลในอนุกรมเวลา ts

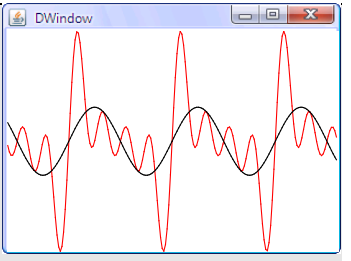
ตารางที่ 8-2 ตัวสร้างและเมท็อดประจำอ็อบเจกต์ของ TimeSeries

การเรียกใช้	ความหมาย
TimeSeries()	ตัวสร้าง
ts.add(v)	เพิ่มค่า v ต่อท้ายข้อมูลอนุกรมเวลา ts
ts.size()	คืนจำนวนข้อมูลในข้อมูลอนุกรมเวลา ts
ts.get(k)	คืนข้อมูลตัวที่ k ของข้อมูลอนุกรมเวลา ts (ตัวแรกคือตัวที่ 0)
ts.min()	คืนค่าน้อยที่สุดในข้อมูลอนุกรมเวลา ts
ts.max()	คืนค่ามากที่สุดในข้อมูลอนุกรมเวลา ts
ts.getMovingAverage()	คืนข้อมูลอนุกรมเวลาชุดใหม่ที่ได้จากการหาค่าเฉลี่ยเคลื่อนที่ของ ts

คลาส TimeSeries เป็นคลาสที่แทนข้อมูลอนุกรมเวลา เก็บข้อมูลเรียงต่อ ๆ กันไปคล้ายกับการเก็บข้อมูลในอาเรย์หนึ่งมิติ ข้อมูลที่เก็บนี้ คือ ค่า y ในการวาดกราฟ ส่วนค่า x คือ เลขลำดับหรือดัชนีของข้อมูลในอนุกรม เราไม่ต้องเก็บค่า x เพราะข้อมูลอนุกรมเวลาแต่ละตัวอยู่บนแกน x (ซึ่งแทนแกนเวลา) ที่มีระยะห่างเท่า ๆ กัน แล้วทำไมถึงไม่ใช้อาเรย์หนึ่งมิติชื่อ y แทนน่าจะง่ายกว่า เหตุผลหลักที่เราเลือกสร้างคลาสใหม่ เพราะต้องการให้คลาส TimeSeries นี้ทำตัวเหมือนอาเรย์ของ double ที่ขนาดขยายได้ตามปริมาณข้อมูลที่เพิ่มเข้าไปอย่างอัตโนมัติ ผู้ใช้มีหน้าที่นำข้อมูลใหม่เพิ่มต่อท้ายไปเรื่อย ๆ จนกว่าจะพอใจ ก็นำไปใช้งานได้ มีเมท็อด size ที่คืนจำนวนข้อมูล และ get(k) เพื่อขอค่าของข้อมูลตัวที่ดัชนี k ในอนุกรม นอกจากนี้ยังมีเมท็อดให้บริการอื่น ๆ ที่เกี่ยวข้องกับข้อมูลอนุกรมเวลา ได้แก่ การหาค่าน้อยสุด การหาค่ามากสุด และการหาค่าเฉลี่ยเคลื่อนที่

รหัสที่ 8-21 แสดงตัวอย่างการใช้งาน LineGraph และ TimeSeries บรรทัดที่ 4 สร้างอ็อบเจกต์ของ LineGraph มีขนาด 300×200 จุดภาพ แล้วใช้เมท็อด sinewave(n) เพื่อสร้างอ็อบเจกต์ของ TimeSeries ที่เก็บข้อมูลตามสูตร $y(x) = \sum_{k=1}^{n} \sin(kx)$ โดย x แปรค่าจาก -10 ถึง 10 เพิ่มค่าทีละ 0.1 บรรทัดที่ 5 สร้างข้อมูลอนุกรมเวลาชุดแรกที่มีค่า n = 4 และบรรทัดที่ 6 สร้างอีกชุดที่มี n = 1 การสร้างข้อมูลอนุกรมเวลาในเมท็อด sinewave กระทำได้ง่าย เพียงแค่เรียกตัวสร้าง (บรรทัดที่ 11) โดยไม่ต้องระบุขนาด จากนั้นเรียกใช้เมท็อด add เพื่อเพิ่มข้อมูลต่อท้ายไปเรื่อย ๆ เมื่อได้ข้อมูลอนุกรมเวลาทั้งสอง ก็สามารถเรียกใช้เมท็อด plot ของอ็อบเจกต์ LineGraph เพื่อวาดกราฟเส้นของทั้งสองอนุกรมซ้อนกันบนวินโดว์เดียวกัน (บรรทัดที่ 7 และ 8)

```
01 import jlab.graphics.DWindow;
02 public class TestLineGraph {
03   public static void main(String[] args) {
04     LineGraph g = new LineGraph(300, 200);
05     TimeSeries s4 = sinewave(4);
06     TimeSeries s1 = sinewave(1);
07     g.plot(DWindow.RED, s4);
08     g.plot(DWindow.BLACK, s1);
09   }
10   private static TimeSeries sinewave(int n) {
11     TimeSeries s = new TimeSeries();
12     for (double x = -10; x <= 10; x+=0.1) {
13       double v = 0;
14       for (int k = 1; k <= n; k++) v += Math.sin(k * x);
15       s.add(v);
16     }
17     return s;
18   }
19 }
```



$$y(x) = \sum_{k=1}^{n} \sin(kx)$$

เพิ่ม v ต่อท้ายในข้อมูลอนุกรมเวลา s

รหัสที่ 8-21 ตัวอย่างโปรแกรมวาดกราฟเส้นโดยใช้คลาส `TimeSeries` และ `LineGraph`

```
01 import jlab.graphics.DWindow;
02 public class LineGraph {
03   private DWindow window;
04   private double ymin, ymax;
05   public LineGraph(int w, int h) {
06     window = new DWindow(w, h);
07   }
08   public void plot(int color, TimeSeries ts) {
09     if (ymin == ymax) {    // ตอนเริ่มต้น ymin = ymax = 0
10       ymin = ts.min(); ymax = ts.max();
11     }
12     int n = ts.size();
13     double width = window.getWidth();
14     double height = window.getHeight();
15     for (int i = 0; i < n-1; i++) {
16       double sx0 = toScreen(i,   0, n-1, width);
17       double sx1 = toScreen(i+1, 0, n-1, width);
18       double sy0 = toScreen(ts.get(i),   ymax, ymin, height);
19       double sy1 = toScreen(ts.get(i+1), ymax, ymin, height);
20       window.drawLine(color, sx0, sy0, sx1, sy1);
21     }
22   }
23   private static double toScreen(double v, double min,
24                                  double max, double len) {
25     return (v - min) * (len-1) / (max - min);
26   }
27 }
```

เก็บ ymin และ ymax ไว้ใช้ในการ plot ครั้งต่อไป

หาค่า ymin และ ymax เฉพาะครั้งแรกที่เรียก plot

ปรับพิกัดของข้อมูลที่ได้รับ เป็นพิกัดบนวินโดว์

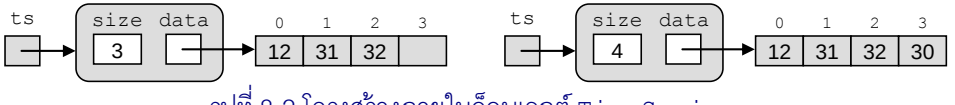
รหัสที่ 8-22 คลาส `LineGraph` สำหรับแสดงวินโดว์พร้อมกราฟเส้น

รหัสที่ 8-22 แสดงรายละเอียดของคลาส `LineGraph` มีตัวสร้างหนึ่งตัวรับความกว้างและความสูงของวินโดว์ ภาระของตัวสร้างนี้ คือ สร้างวินโดว์ (จากคลาส `DWindow`) เก็บไว้ในตัวแปร `window` ประจำอ็อบเจกต์ อ็อบเจกต์ของคลาสนี้ให้บริการเดียว คือ เมท็อด `plot` รับสีและข้อมูลอนุกรมเวลาที่จะนำมาวาดกราฟเส้น เริ่มด้วยการหาค่าน้อยสุดและมากสุดของข้อมูลในอนุกรม เก็บใส่ `ymin` และ `ymax` ตามลำดับ โดยเรียกใช้บริการ `min` และ `max` ของอ็อบเจกต์อนุกรมเวลา ในบรรทัดที่ 10 เราใช้สองค่านี้เพื่อปรับค่าของข้อมูลในอนุกรมให้ตกอยู่ในช่วงความสูงของวินโดว์ โดยเราจะหาทั้งสองค่านี้เฉพาะครั้งแรกที่เรียก `plot` (ตอนสร้างอ็อบเจกต์นี้ใหม่ ๆ `ymin` มีค่าเท่ากับ `ymax` คือ เท่ากับ 0) เนื่องจากเรารู้แล้วว่า ค่า `x` คือ เลขลำดับของข้อมูลมีค่าตั้งแต่ `0` ถึง `n-1` โดยที่ `n` คือจำนวนข้อมูลในอนุกรม ดังนั้น จึงไม่ต้องหา `xmin` และ `xmax` เหมือนที่เคยทำในรหัสที่ 7-21 คำสั่งที่เหลือในบรรทัดที่ 12 ถึง 21 ทำหน้าที่ในทำนองเดียวกับในรหัสที่ 7-21 คือวนหยิบข้อมูลในอนุกรมมาวาดทีละเส้น ๆ ต่อ ๆ กันจนได้เป็นกราฟเส้นตามต้องการ

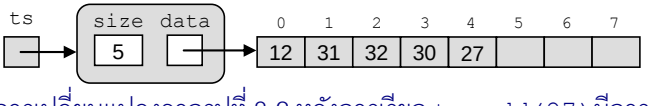
คลาสข้อมูลอนุกรมเวลา

อ็อบเจกต์ของคลาส `TimeSeries` ประกอบด้วยตัวแปร `size` ซึ่งเก็บจำนวนข้อมูลของอนุกรม และอาเรย์ `data` ซึ่งเก็บตัวข้อมูลตั้งแต่ช่องที่ `0` ถึง `size-1` รูปที่ 8-2 ซ้ายแสดงตัวอย่างของข้อมูลอนุกรมเวลา ⟨12, 31, 32⟩ ภายในอ็อบเจกต์จึงมี `size=3` และข้อมูลทั้งสามตัวนี้เก็บในช่อง 0 ถึง 2 ของอาเรย์ที่มีขนาด 4 ช่อง, เมื่อมีการเพิ่ม 30 ต่อท้าย ด้วยคำสั่ง `ts.add(30)` จะได้ผลดังรูปที่ 8-2 ขวา สรุปว่า ข้อมูลใหม่ถูกต่อท้ายที่ `data[size]` แล้วเพิ่มค่าของ `size` อีก 1

เมื่ออาเรย์ `data` เก็บข้อมูลเต็มแล้ว เช่น ในรูปที่ 8-2 ขวา (`data.length` มีค่าเท่ากับ `size`) ถ้าเรียก `ts.add(27)` ต้องขยายอาเรย์ก่อน ซึ่งอาศัยเมท็อด `doubleCapacity` (ที่เคยนำเสนอในรหัสที่ 7-16) เพื่อขยายอาเรย์ `data` อีกเท่าตัว โดยสร้างอาเรย์ใหม่มีขนาดเป็นสองเท่าของของเดิม จากนั้นทำสำเนาข้อมูลทุกช่องจากอาเรย์เดิมไปยังอาเรย์ใหม่ ปิดท้ายด้วยการให้ `data` อ้างอิงอาเรย์ใหม่ นั่นคือ ให้ `data` ทิ้งอาเรย์เดิม และไปอ้างอิงอาเรย์ใหม่ที่ใหญ่กว่าเดิม จากนั้นเพิ่มข้อมูลใหม่ `v` ต่อท้ายด้วย `data[size++] = v` (ซึ่งเพิ่มค่าของ `size` อัตโนมัติหลังใส่ `v` ในอาเรย์แล้ว) (ได้ผลดังรูปที่ 8-3)



รูปที่ 8-2 โครงสร้างภายในอ็อบเจกต์ `TimeSeries`



รูปที่ 8-3 ผลการเปลี่ยนแปลงจากรูปที่ 8-2 หลังการเรียก `ts.add(27)` มีการขยายอาเรย์

```
01 public class TimeSeries {
02   private int      size = 0;
03   private double[] data = new double[4];
04
05   public void add(double v) {
06     if (size == data.length) data = doubleCapacity(data);
07     data[size++] = v;
08   }
09   public int size() {
10     return size;
11   }
12   public double get(int k) {
13     return data[k];
14   }
15   public double min() {
16     if (size == 0) throw new IllegalStateException("size=0");
17     double min = data[0];
18     for (int i = 1; i < size; i++)
19       if (data[i] < min) min = data[i];
20     return min;
21   }
22   public double max() {
23     if (size == 0) throw new IllegalStateException("size=0");
24     double max = data[0];
25     for (int i = 1; i < size; i++)
26       if (data[i] > max) max = data[i];
27     return max;
28   }
29   public TimeSeries getMovingAverage() {
30     TimeSeries ma = new TimeSeries();
31     ma.add((data[0] + data[1]) / 2);
32     for (int i = 1; i <= size-2; i++) {
33       ma.add((data[i-1] + data[i] + data[i+1]) / 3);
34     }
35     ma.add((data[size-2] + data[size-1]) / 2);
36     return ma;
37   }
38   private static double[] doubleCapacity(double[] d) {
39     double[] a = new double[2*d.length];
40     for (int i=0; i<d.length; i++) a[i] = d[i];
41     return a;
42   }
43 }
```

เพิ่มต่อท้าย แล้วเพิ่มจำนวน

ข้อมูลเต็มอาเรย์แล้ว ให้ขยายอาเรย์เป็นสองเท่า

ไม่มีข้อมูล หาค่าน้อยสุดไม่ได้

ไม่มีข้อมูล หาค่ามากสุดไม่ได้

สร้างอ็อบเจกต์เพื่อเก็บผลลัพธ์

ค่าเฉลี่ยของตัวแรก

เฉลี่ยสามค่าที่ติดกัน

ค่าเฉลี่ยของตัวสุดท้าย

สร้างอาเรย์ใหม่

ทำสำเนาจากของเดิมไปของใหม่

คืนอาเรย์ใหม่ที่ใหญ่กว่าเดิม

รหัสที่ 8-23 คลาส `TimeSeries` เก็บข้อมูลอนุกรมเวลา

คลาส `TimeSeries` (รหัสที่ 8-23) มีเมท็อด `add` ไว้เพิ่มข้อมูลใหม่ต่อท้ายอนุกรม (ซึ่งขยายอาเรย์ให้อัตโนมัติ) มีเมท็อด `size` คืนจำนวนข้อมูล, มีเมท็อด `get(k)` คืน `data[k]`, มีเมท็อด `min` และ `max` คืนค่าน้อยสุดและมากสุดของข้อมูลในอนุกรมตามลำดับ และมีเมท็อด

`getMovingAverage` ที่สร้างและคืนอ็อบเจกต์อนุกรมเวลาตัวใหม่ซึ่งเก็บผลลัพธ์ที่ได้จากการหาค่าเฉลี่ยของอ็อบเจกต์ที่ถูกเรียก ให้สังเกตว่า การประมวลข้อมูลในอนุกรมเวลาจะไม่ได้ทำทั้งอาเรย์ แต่จะทำตั้งแต่ตัวที่ 0 ถึง `size-1` เท่านั้น เพราะนี่คือช่องที่เก็บข้อมูลของอนุกรม

อนึ่ง หากย้อนไปดูที่เมท็อด `min` และ `max` ของรหัสที่ 8-23 พบว่า ในกรณีที่ยังไม่มีข้อมูลเลย นั่นคือ `size` เป็น 0 ก็ย่อมไม่มีค่าน้อยสุดและมากสุดให้คืน จึงโยนสิ่งผิดปกติกลับไปตามที่ได้เคยกระทำมา แต่ครั้งนี้เราเลือกที่จะโยนอ็อบเจกต์ของคลาส `IllegalStateException` (บรรทัดที่ 16 และ 23) ซึ่งต่างกับการโยน `IllegalArgumentException` ที่เคยทำในอดีต เช่น การส่งอาเรย์ขนาด 0 ช่องไปให้เมท็อด `min(int[] d)` เพราะเป็นสิ่งผิดปกติที่เกิดจากค่าของพารามิเตอร์ที่ได้รับ แต่ `min` และ `max` ของ `TimeSeries` ไม่รับพารามิเตอร์ใดๆ เพราะจะหาค่าน้อยสุดและมากสุดจากข้อมูลภายในอ็อบเจกต์เอง จึงถือว่า อ็อบเจกต์อยู่ในสภาวะ (state) ที่ไม่พร้อมให้บริการ `min` กับ `max` จึงโยน `IllegalStateException` ซึ่งเป็นหนึ่งในคลาสมาตรฐานของจาวาสำหรับการแจ้งสิ่งผิดปกติ จะว่าไปแล้วเราจะโยนสิ่งผิดปกติชนิดใดเพื่อแจ้งให้ผู้ใช้ทราบก็ได้ แต่ถ้าเราเลือกโยนตัวที่สื่อความหมายกว่า จะได้ประโยชน์ในการหาสาเหตุของสิ่งผิดปกติที่เกิดขึ้นในโปรแกรมได้ดีกว่า (จะนำเสนอรายละเอียดของประเด็นนี้ในบทที่ 10)

เกมจับคู่ทดสอบความจำ

เกมนี้หลาย ๆ คนคงเคยเล่นเพื่อทดสอบความจำ มีบัตรตัวเลขจำนวนหนึ่งวางคว่ำบนโต๊ะ สิ่งที่ต้องทำคือ พลิกบัตรทีละคู่ ถ้าได้หมายเลขไม่ตรงกันก็พลิกกลับดังเดิม แต่ถ้าได้หมายเลขตรงกัน ก็เปิดไว้เช่นนั้น แล้วดำเนินการพลิกคู่อื่นต่อไป ทำจนจับคู่ได้ครบทุกคู่ ดูสิว่าใครทำเสร็จเร็วกว่ากัน เราจะเขียนโปรแกรมใช้วินโดว์แสดงบัตรและใช้เมาส์เพื่อเลือกบัตร โปรแกรมนี้ประกอบด้วยสองคลาส `Card` กับ `MemoryGame` อ็อบเจกต์ของ `Card` แทนบัตร ส่วนอ็อบเจกต์ของ `MemoryGame` แทนวินโดว์ที่แสดงบัตรและตัวควบคุมการเล่น รูปที่ 8-4 แสดงตัวอย่างการเล่นเกม

รูปที่ 8-4 ตัวอย่างการเล่นเกมจับคู่ทดสอบความจำ

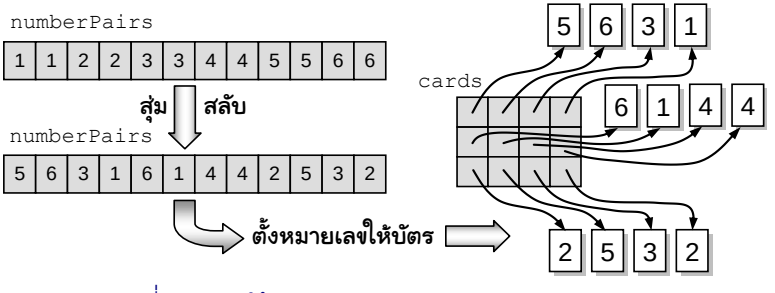
```
01 import jlab.graphics.DWindow;
02 public class Card {
03   public static final int WIDTH = 60, HEIGHT = 60;
04   private int number;              // หมายเลขของบัตรใบนี้
05   private int row, col;            // ตำแหน่ง (หมายเลขแถว หมายเลขสดมภ์)
06   private boolean matched;         // จับคู่ไปแล้วหรือยัง
07   private DWindow window;          // วินโดว์ที่แสดงบัตรใบนี้
08
09   public Card(DWindow w, int r, int c, int n) {
10     window = w;
11     row = r; col = c;
12     number = n;
13   }
14   public int number() {            // คืนหมายเลขของบัตรใบนี้
15     return number;
16   }
17   public void show() {
18     draw(DWindow.WHITE, true);   // บัตรสีขาว แสดงหมายเลขของบัตร
19   }
20   public void hide() {
21     draw(DWindow.GRAY, false);   // บัตรสีเทา ไม่แสดงหมายเลข
22   }
23   public void matched() {
24     matched = true;              // บอกให้รู้ว่าบัตรใบนี้ถูกจับคู่แล้ว
25     draw(DWindow.PINK, true);    // แสดงหมายเลข บนบัตรสีชมพู
26   }
27   public boolean isMatched() {   // คืนสภาวะว่า บัตรใบนี้จับคู่แล้วหรือยัง
28     return matched;
29   }
30   private void draw(int color, boolean showNumber) {
31     int x = col * WIDTH, y = row * HEIGHT;
32     window.fillRect(color, x, y, WIDTH, HEIGHT);
33     window.drawRect(DWindow.BLACK, x, y, WIDTH, HEIGHT);
34     if (showNumber) window.drawString(""+number, 30, x+5, y+5);
35   }
36 }
```

final บอกว่า เป็นตัวแปรที่ให้ค่าแล้วเปลี่ยนแปลงไม่ได้อีก

รหัสที่ 8-24 คลาส `Card` แทนบัตรตัวเลขของเกมจับคู่ทดสอบความจำ

รหัสที่ 8-24 แสดงคลาส `Card` ซึ่งผลิตอ็อบเจกต์ที่แสดงได้บนวินโดว์ มีเมท็อด `hide`, `show`, และ `matched` เมท็อด `hide` แสดงบัตรสีเทา, `show` แสดงบัตรสีขาวมีหมายเลข, ส่วน `matched` แสดงบัตรชมพูมีหมายเลขที่แทนสภาวะว่าจับคู่ไปแล้ว มีเมท็อด `number` ที่คืนหมายเลขของบัตร มีเมท็อด `isMatched` ให้ถามว่า บัตรใบนี้ถูกจับคู่หรือยัง ดังนั้น แต่ละอ็อบเจกต์จึงต้องเก็บหมายเลข, สภาวะว่าจับคู่หรือยัง, วินโดว์ที่แสดงบัตร, และตำแหน่งเลขแถวกับสดมภ์ของบัตรบนวินโดว์ โดยมีตัวแปรประจำคลาสเก็บความกว้างและความยาวของบัตร (ซึ่งเหมือนกันทุกใบจึงมีคำว่า `static` เพื่อเก็บที่คลาสที่เดียว) สองตัวแปรนี้มีคำว่า `final` เพื่อระบุว่า เป็นตัวแปรที่เปลี่ยนแปลงค่าไม่ได้อีกแล้ว (จะอธิบายความหมายของ `final` เพิ่มเติมในหัวข้อถัดไป)

รหัสที่ 8-25 แสดงคลาส MemoryGame ซึ่งทำหน้าที่แสดงผลและควบคุมเกม ตัวสร้างของคลาสนี้รับจำนวนแถวและสดมภ์ที่จะแสดงบัตร (จำนวนบัตรเท่ากับผลคูณของจำนวนแถวกับจำนวนสดมภ์ซึ่งต้องเป็นเลขคู่) ภาระของตัวสร้าง คือ สร้างวินโดว์ให้มีขนาดใหญ่พอซึ่งคำนวณได้จากจำนวนแถว จำนวนสดมภ์ กับความกว้างและความยาวของบัตร (บรรทัดที่ 8) จากนั้นสร้างอาเรย์สองมิติชื่อ cards มีจำนวนแถวและสดมภ์ตามที่ได้รับเพื่อเก็บบัตร (ดูรูปที่ 8-5) และเตรียมอาเรย์หนึ่งมิติชื่อ numberPairs มีขนาดเท่ากับจำนวนบัตร ตั้งค่าเริ่มต้นเป็น 1,1,2,2,3,3,... (บรรทัดที่ 11-12) แล้วสลับข้อมูลในอาเรย์นี้อย่างสุ่ม ๆ (บรรทัดที่ 13-14) เพื่อนำไปตั้งเป็นหมายเลขประจำบัตรต่าง ๆ ที่สร้างแล้วเก็บใน cards (บรรทัดที่ 15-20) และเรียกเมท็อด hide กับทุกใบที่สร้างขึ้น (จะได้ไม่แสดงหมายเลขให้ผู้ใช้เห็น) เป็นอันจบการทำงานของตัวสร้าง



รูปที่ 8-5 การใช้อาเรย์ numberPairs และ cards

คลาส MemoryGame มีเมท็อด begin อีกตัวให้เรียกเมื่อพร้อมเล่น เมท็อดนี้ตั้งค่า numUnmatched ให้เท่ากับครึ่งหนึ่งของจำนวนบัตร (บรรทัดที่ 23) ซึ่งคือจำนวนคู่ของบัตรที่ยังไม่ถูกจับคู่ ในแต่ละรอบที่ผู้ใช้เล่น ถ้าจับคู่ได้ numUnmatched จะมีค่าลดลงหนึ่ง ดังนั้น วงวนการทำงานของเกมจะทำตราบเท่าที่ยังต้องจับคู่ต่อ (บรรทัดที่ 25) การทำงานภายในวงวนเริ่มด้วยการรอให้ผู้ใช้กดคลิกที่บัตรใบหนึ่งด้วยเมท็อด waitForCardClicked (บรรทัดที่ 26) ซึ่งจะคืนบัตรที่ถูกคลิกเก็บใน c1 (รายละเอียดของเมท็อดนี้จะอธิบายในภายหลัง) เมื่อผู้ใช้คลิกแล้ว ก็ต้องสั่งให้ c1.show() เพื่อแสดงหมายเลขให้ผู้ใช้ดู จากนั้นรอให้ผู้ใช้คลิกอีกใบ (บรรทัดที่ 27) เก็บใส่ c2 แสดงหมายเลขของบัตรใบนี้ รอสักครึ่งวินาทีให้ผู้ใช้ดูและจำ (บรรทัดที่ 28) แล้วก็ตรวจสอบว่า หมายเลขของทั้งสองใบที่ถูกคลิกเหมือนกันหรือไม่ ถ้าเหมือนก็บอกให้ c1 และ c2 รู้ว่าจับคู่ได้แล้ว (ด้วยการเรียกเมท็อด matched) แล้วลดจำนวนการจับคู่ที่เหลือลงหนึ่ง แต่ถ้าหมายเลขไม่เหมือน ให้รีบซ่อนหมายเลขของทั้งสองใบทันทีด้วย hide แล้ววนกลับไปทำรอบถัดไป เมื่อใดออกจากวงวนย่อมแสดงว่า จับครบทุกคู่แล้ว ให้แสดงเวลาที่ใช้ในการเล่นเกมให้ผู้ใช้ทราบ (ซึ่งคือผลต่างของเวลาตอนออกจากวงวนกับเวลาตอนเริ่มต้น) โดยเรียกเมท็อดประจำคลาส System ชื่อ currentTimeMillis ที่คืนเวลาปัจจุบันมีหน่วยเป็นมิลลิวินาที จึงต้องหารผลต่างที่ได้ด้วย 1000 เพื่อแสดงเป็นหน่วยวินาที (บรรทัดที่ 38)

```
01 import jlab.graphics.DWindow;
02 public class MemoryGame {
03   private DWindow window;
04   private Card[][] cards;
05
06   public MemoryGame(int row, int col) {
07     if ((row * col) % 2 != 0) throw new IllegalArgumentException();
08     window = new DWindow(Card.WIDTH * col, Card.HEIGHT * row);
09     cards = new Card[row][col];
10     int[] numberPairs = new int[row * col];
11     for (int i = 0; i < numberPairs.length; i++) // 1,1,2,2,3,3,...
12       numberPairs[i] = 1 + i / 2;
13     for (int i = 0; i < numberPairs.length / 2; i++) // สุ่มสลับ
14       swap(numberPairs, i, random(i+1, numberPairs.length-1));
15     for (int i = 0, k = 0; i < row; i++) { //สร้างบัตรเก็บตามตำแหน่งต่างๆ
16       for (int j = 0; j < col; j++) {
17         cards[i][j] = new Card(window, i, j, numberPairs[k++]);
18         cards[i][j].hide();
19       }
20     }
21   }
22   public void begin() {
23     int numUnmatched = cards.length * cards[0].length / 2;
24     long startTime = System.currentTimeMillis();
25     while (numUnmatched > 0) {
26       Card c1 = waitForCardClicked(null); c1.show();
27       Card c2 = waitForCardClicked(c1);   c2.show();
28       window.sleep(500);
29       if (c1.number() == c2.number()) {
30         c1.matched(); c2.matched();
31         numUnmatched--;
32       } else {
33         c1.hide(); c2.hide(); // ไม่ตรงกัน รีบซ่อนหมายเลข
34       }
35     }
36     window.clearBackground();
37     long time = (System.currentTimeMillis() - startTime)/1000;
38     window.drawString("คุณทำได้ใน  " + time+ "  วินาที",14,5,5);
39   }
```

ตั้งหมายเลขให้บัตรตามลำดับที่สุ่มๆ

รอผู้ใช้คลิกบัตรสองใบที่ไม่ซ้ำกัน และยังไม่ถูกจับคู่

จับครบทุกคู่แล้ว ให้แสดงเวลาที่ใช้

รหัสที่ 8-25 คลาส MemoryGame ไว้เล่นเกมจับคู่ทดสอบความจำ

รหัสที่ 8-26 แสดงเมท็อด waitForCardClicked ที่เราเรียกใช้ใน begin ของรหัสที่ 8-25 เมท็อดนี้เรียกใช้เมท็อด waitForMouseClicked ของอ็อบเจกต์ DWindow เพื่อรอจนกว่าผู้ใช้จะคลิกเมาส์ในบริเวณวินโดว์ จะคืนการทำงานกลับมา ซึ่งเราสามารถอ่านพิกัดของเมาส์ได้จากเมท็อด w.getMouse().getX() และ getY() เราเคยอ่านพิกัดเมาส์แบบนี้นานมาแล้วครั้งหนึ่งในบทที่ 3 จะว่าไปแล้ว w.getMouse() คืนอ็อบเจกต์พิเศษเป็นประเภท DPoint (ดูภาคผนวก ข) ที่มีเมท็อด getX และ getY เมื่อได้พิกัดมาแล้ว ต้องคำนวณกลับไป

เป็นหมายเลขแถวและสดมภ์ จะได้รู้ว่า ตรงกับบัตรใบใดที่เก็บในอาเรย์ cards (บรรทัดที่ 46) โดยจะไม่สนใจบัตรที่ถูกจับคู่ไปแล้ว (c.isMatched() เป็นจริง) หรือเป็นบัตรเดียวกับพารามิเตอร์ c0 การเรียก waitForCardClicked เพื่อเลือกบัตรใบที่สองต้องไม่ใช่บัตรใบแรก ที่เลือกไปแล้ว ดังนั้น บรรทัดที่ 27 ของรหัสที่ 8-25 จึงส่ง c1 ลงมาให้ c0 เพื่อป้องกันไม่ให้เลือกซ้ำ ส่วนการเลือกบัตรครั้งแรกเราส่ง null มา (บรรทัดที่ 26) เพราะ null เป็นค่าพิเศษที่ไม่เหมือนตัวอ้างอิงของอ็อบเจกต์ใด ๆ ดังนั้น จึงใช้วงวน do-while วนรอผู้ใช้คลิกไปเรื่อย ๆ จนได้ใบใหม่ที่ยังไม่ถูกจับคู่

```
40   private Card waitForCardClicked(Card c0) {
41     Card c;
42     do {
43       window.waitForMouseClicked();      // รอผู้ใช้คลิกบนวินโดว์
44       int x = window.getMouse().getX(); // อ่านพิกัด x ของเมาส์
45       int y = window.getMouse().getY(); // อ่านพิกัด y ของเมาส์
46       c = cards[y / Card.HEIGHT][x / Card.WIDTH];
47     } while (c == c0 || c.isMatched());
48     return c;
49   }
50   private int random(int a, int b) {
51     return a + (int)((b - a + 1) * Math.random());
52   }
53   private void swap(int[] d, int i, int j) {
54     int t = d[i]; d[i] = d[j]; d[j] = t;
55   }
56   //----------------------------------------
57   public static void main(String[] args) {
58     new MemoryGame(3, 4).begin();
59   }
60   //----------------------------------------
61 }
```

เปลี่ยนพิกัดเมาส์เป็นหมายเลขแถวและสดมภ์

สร้างเกมขนาด 3 แถว 4 สดมภ์ แล้วเริ่มเล่น

รหัสที่ 8-26 คลาส MemoryGame ไว้เล่นเกมจับคู่ทดสอบความจำ (ต่อ)

รหัสที่ 8-26 มีเมท็อดเสริมช่วยในการสุ่มจำนวนเต็ม และสลับค่าในอาเรย์ และมีเมท็อด main เพิ่มให้ผู้ใช้สั่งโปรแกรมทำงาน ภายในเมท็อดก็เพียงแค่สร้างอ็อบเจกต์ MemoryGame โดยระบุจำนวนแถวและสดมภ์ของบัตรต่าง ๆ ตามด้วยการเรียก begin ให้เริ่มเล่นเกมทันที

เกร็ดอ็อบเจกต์

ยังมีเรื่องจำเป็นที่ควรรู้ในการเขียนโปรแกรมเชิงวัตถุด้วยจาวา หัวข้อย่อยต่อไปนี้นำเสนอการใช้คำสำคัญ this ในการอ้างอิงอ็อบเจกต์ที่ถูกเรียก กับ this(...) ในการเรียกตัวสร้าง, การเขียนตัวสร้างสำเนาที่ใช้บ่อย และการใช้ final เพื่อกำกับตัวแปรที่ตั้งค่าได้ครั้งเดียว

this และ this(...)

จาวามีคำสำคัญคำหนึ่ง คือ `this` คำนี้ใช้งานได้สองแบบ แบบที่หนึ่งเขียน `this` เสมือนเป็นตัวแปรอ้างอิง แบบที่สองเขียน `this(...)` เสมือนเป็นการเรียกตัวสร้าง มีรายละเอียดและการใช้งานดังต่อไปนี้

ขอทวนอีกครั้งว่า การเรียกเมท็อดประจำอ็อบเจกต์ ต้องเขียนในรูปแบบที่มีตัวอ้างอิงอ็อบเจกต์นำหน้าชื่อเมท็อด เมื่อการทำงานมาอยู่ในเมท็อดแล้ว เราก็สามารถใช้ตัวแปรประจำอ็อบเจกต์ที่ถูกเรียกได้ ในกรณีที่เราต้องการใช้ตัวอ้างอิงอ็อบเจกต์ที่ถูกเรียกนี้ในเมท็อด ให้ใช้คำว่า `this` ดังตัวอย่างในรหัสที่ 8-27

- ใช้ `this.balance` (แทนการเขียน `balance` ซึ่งมีความหมายเดียวกัน) อ่านได้ว่า ต้องการยอดเงินคงเหลือของบัญชี "นี้" ที่ถูกเรียกใช้บริการ
- ใช้ `this.withdraw(amt)` (แทนการเขียน `withdraw(amt)` ซึ่งมีความหมายเดียวกัน) อ่านได้ว่า ต้องการถอนเงินจำนวน `amt` จากบัญชี "นี้" ที่ถูกเรียกใช้บริการ

```
public class BankAccount {
  private String id;      // หมายเลขบัญชี
  private double balance; // ยอดเงินคงเหลือ
  ...
  public double getBalance() {
    return this.balance;  // คืนยอดเงินของบัญชี "นี้"
  }
  public void transferTo(BankAccount to, double amt) {
    this.withdraw(amt);   // ถอนเงินจากบัญชี "นี้"
    to.deposit(amt);      // ฝากเข้าบัญชี to
  }
```

รหัสที่ 8-27 ตัวอย่างการใช้ `this` แทนตัวอ็อบเจกต์ที่ถูกเรียก

รหัสที่ 8-28 แสดงคลาส `Point` มีไว้แทนจุดในระนาบสองมิติ ภายในมีตัวแปร `x` และ `y` ประจำอ็อบเจกต์เพื่อเก็บพิกัดของจุด มีตัวสร้างที่รับพิกัด `x` และ `y` รหัสทางซ้ายแสดงวิธีเขียนที่ผิด เพราะเราตั้งชื่อพารามิเตอร์เหมือนกับชื่อตัวแปรประจำอ็อบเจกต์ การตั้งชื่อซ้ำนี้ไม่ผิดกฎ แต่ตัวแปลโปรแกรมจะถือว่า `x` และ `y` ที่เขียนในตัวสร้างนั้นคือ `x` และ `y` ที่เป็นพารามิเตอร์ เราจึงต้องตั้งชื่อพารามิเตอร์เป็น `x0` และ `y0` ให้แตกต่างดังรหัสทางขวา ทำให้ `x` และ `y` ในตัวสร้างทางขวาคือตัวแปรประจำอ็อบเจกต์

```
public class Point {
  int x, y;
  public Point(int x, int y) {
    x = x; y = y;
  }
```

```
public class Point {
  int x, y;
  public Point(int x0, int y0) {
    x = x0; y = y0;
  }
```

รหัสที่ 8-28 ตัวสร้างทางซ้ายทำงานผิด ทางขวาทำงานถูก

ถ้ารู้สึกหงุดหงิดไม่ต้องการตั้งชื่อพารามิเตอร์ใหม่ ต้องการตั้งให้เหมือนกับตัวแปรประจำอ็อบเจกต์ ก็ต้องเขียนในตัวสร้างให้ชัดเจนด้วยการเติม `this.` ให้ครบ ดังตัวอย่างในรหัสที่ 8-29

```
public class Point {
  int x, y;
  public Point(int x, int y) {
    this.x = x; this.y = y;
  }
```

นำ x และ y (ที่เป็นพารามิเตอร์) ไปเก็บใน x และ y ของอ็อบเจกต์ "นี้"

รหัสที่ 8-29 ตัวอย่างการใช้ `this` เพื่อสร้างความแตกต่างกับพารามิเตอร์

คราวนี้มาดูการใช้ `this` ในอีกรูปแบบหนึ่ง ถ้าย้อนกลับไปดูรหัสที่ 8-10 จะเห็นว่า เราน่าจะเขียนให้ตัวสร้างตัวแรกไปเรียกใช้ตัวสร้างตัวที่สอง เพราะมีการทำงานที่คล้ายคลึงกัน ดังแสดงในรหัสที่ 8-30 แต่จาวาไม่อนุญาตให้เรียก `Ball(...)` โดยตรง การเขียน `Ball(...)` กระทำได้เฉพาะหลังคำสั่ง `new` เท่านั้น ถ้าต้องการเรียกตัวสร้าง สามารถใช้ `this(...)` แทน ดังแสดงในรหัสที่ 8-31

```
public class Ball {
  ...
  public Ball(double r0, double x0, double y0) {
ผิ  Ball(r0, x0, y0, random(-3,3), random(-3,3));
  }
  public Ball(double r0,double x0,double y0,double dx0,double dy0){
    r = r0;  x = x0; y = y0;  dx = dx0; dy = dy0;
  }
```

รหัสที่ 8-30 การเขียนคำสั่งในตัวสร้างที่มีการเรียกตัวสร้างอื่น โดยใช้ชื่อคลาสซึ่งผิด

```
public class Ball {
  ...
  public Ball(double r0, double x0, double y0) {
ถูก this(r0, x0, y0, random(-3,3), random(-3,3));
  }
  public Ball(double r0,double x0,double y0,double dx0,double dy0){
    r = r0;  x = x0; y = y0;  dx = dx0; dy = dy0;
  }
```

รหัสที่ 8-31 การเขียนคำสั่งในตัวสร้างที่มีการเรียกตัวสร้างอื่น โดยใช้ `this` ซึ่งถูกต้อง

การใช้ `this(...)` เพื่อเรียกตัวสร้างนั้นมีกฎจุกจิกเพิ่มเติมเล็กน้อย คือ ต้องเขียนเป็นบรรทัดแรกภายในตัวสร้างเท่านั้น ดังนั้น จึงเขียน `this(...)` ในเมท็อดทั่วไปไม่ได้ เมื่อเรียกใช้ในตัวสร้าง ก็ต้องเขียนเป็นบรรทัดแรก เช่น รหัสที่ 8-32 แทนส่วนของคลาส `Time` ที่เคยเขียนในรหัสที่ 8-17 แต่เขียนตัวสร้างอีกแบบ (ซึ่งผิด) ตัวสร้างตัวบนเขียน `this(...)` เพราะต้องการเรียกใช้ตัวสร้างตัวล่าง แต่เขียนเป็นบรรทัดที่ 2 ในตัวสร้างซึ่งผิด ในกรณีเช่นนี้ ควรย้ายคำสั่งในตัวสร้างตัวล่าง แยกไปเขียนเป็นเมท็อด แล้วใช้วิธีเรียกเมท็อดแทน ดังที่ได้เขียนในคลาส `Time` ในรหัสที่ 8-17

```
import java.util.Date;
public class Time {
  private int hour, minute, second;
  public Time() {
    Date now = new Date();
    this(now.getHours(), now.getMinutes(), now.getSeconds());
  }
  public Time(int h, int m, int s) {
    hour = h; minute = m; second = s;
  }
```

ผิด เพราะไม่ได้เขียน this(...) เป็นบรรทัดแรกของตัวสร้าง

รหัสที่ 8-32 ตัวอย่างการเรียกตัวสร้างที่ผิด เพราะไม่ได้เรียกที่บรรทัดแรก

ตัวสร้างสำเนา

ตัวสร้างที่นักเขียนโปรแกรมมักเขียนกันเมื่อเขียนคลาสใหม่คือ ตัวสร้างแบบไม่รับพารามิเตอร์ที่ตั้งค่าปริยายให้กับตัวแปรต่าง ๆ ประจำอ็อบเจกต์ นอกจากนี้นักเขียนโปรแกรมมักเขียนตัวสร้างอีกแบบเรียกว่า *ตัวสร้างสำเนา* (copy constructor) รับพารามิเตอร์หนึ่งตัวเป็นอ็อบเจกต์ของคลาสเดียวกับของตัวสร้าง เช่น รหัสที่ 8-33 แสดงตัวสร้างของจุด `Point(Point p)` ซึ่งรับ `p` ที่เป็นจุดเช่นกัน ตัวสร้างนี้มีหน้าที่ทำสำเนาค่าของตัวแปรประจำอ็อบเจกต์ของพารามิเตอร์มาให้กับตัวแปรประจำอ็อบเจกต์ของตนเอง

```
public class Point {
  private int x, y;
  public Point(Point p) {
    this.x = p.x;
    this.y = p.y;
  }
  public void setXY(int x, int y) {
    this.x = x; this.y = y;
  }
  public int getX() {
    return x;
  }
  ...
```

ตัวสร้างสำเนาตั้งค่าให้ตัวแปรประจำอ็อบเจกต์เหมือนกับของพารามิเตอร์

รหัสที่ 8-33 ตัวสร้างสำเนาของคลาส `Point`

มาดูตัวอย่างการใช้งานตัวสร้างสำเนากัน รหัสที่ 8-34 แสดงคลาส `Circle` ซึ่งภายในเก็บอ็อบเจกต์ `Point` แทนจุดศูนย์กลางของวงกลม มีเมท็อด `getCenter` ให้บริการคืนจุดศูนย์กลางของวงกลมที่ใช้คำสั่ง `return center` คืนให้ผู้เรียก การทำเช่นนี้ต้องระวัง เพราะสิ่งที่คืนกลับไป คือ ตัวอ้างอิงอ็อบเจกต์ของจุดศูนย์กลาง ผู้ที่ได้รับตัวอ้างอิงนี้ไป ย่อมสามารถเรียกเมท็อด `setXY` ของ `Point` เปลี่ยนแปลงตำแหน่งของจุดศูนย์กลางได้ เช่น ส่วนของโปรแกรมในรหัสที่ 8-35 สร้างวงกลม `q` มีจุดศูนย์กลางที่พิกัด (0,0) จากนั้นใช้ `q.getCenter()` นำจุด

ศูนย์กลางของ q เก็บใส่ p ตามด้วยคำสั่ง p.setXY(...) เพื่อเปลี่ยนตำแหน่งของ p ซึ่งก็คือการเปลี่ยนจุดศูนย์กลางของวงกลม q ด้วย

```
public class Circle {
  private Point center;
  private int radius;
  public Circle(Point c, int r) {
    center = c;  radius = r;
  }
  public Point getCenter() {
    return center;
  }
```

ต้องการจุดศูนย์กลาง ก็คืนตัวอ้างอิงไปยังอ็อบเจกต์ center ที่เก็บไว้

รหัสที่ 8-34 คลาส Circle ที่ใช้อ็อบเจกต์ของ Point เก็บจุดศูนย์กลาง

```
    Circle q = new Circle(new Point(0,0), 10);
    Point p = q.getCenter();
    p.setXY(-999, 999);
    System.out.println(q.getCenter().getX());
```

ขอจุดศูนย์กลางของวงกลม q มาเปลี่ยนค่า โดยวงกลม q ไม่รู้ตัว

รหัสที่ 8-35 ส่วนของโปรแกรมทดสอบการใช้ Circle และ Point

คำถามก็คือว่า เราต้องการให้วงกลมมีพฤติกรรมเช่นนี้หรือไม่ นั่นคือ ยอมให้ผู้อื่นมาเปลี่ยนจุดศูนย์กลางของวงกลม โดยที่ตัววงกลมเองไม่รู้เรื่องเลยหรือไม่ หากเราไม่ยอม ใครต้องการเปลี่ยนตำแหน่งต้องเรียกเมท็อด setCenter ของวงกลมเท่านั้น เราต้องไม่คืน center ซึ่งคือตัวอ้างอิงจุดศูนย์กลางกลับให้ผู้ขอ แต่ควรทำสำเนาจุดศูนย์กลาง แล้วคืนผลที่ได้กลับไปให้แทน จึงต้องเขียนเมท็อด getCenter ให้คืนสำเนาโดยใช้ตัวสร้างสำเนา ดังรหัสที่ 8-36 สำหรับกรณีที่เรามั่นใจว่า สิ่งที่คืนกลับไปเป็นอ็อบเจกต์ที่เปลี่ยนค่าไม่ได้แน่ ๆ ก็ไม่ต้องห่วง ดังนั้น นักเขียนโปรแกรมควรเขียนตัวสร้างสำเนาให้กับคลาสที่ผลิตอ็อบเจกต์ที่เปลี่ยนค่าได้ เพื่อให้บริการทำสำเนาได้ง่าย ๆ

```
public class Circle {
  ...
  public Point getCenter() {
    return new Point(center);
  }
```

ไม่ต้องการให้ผู้อื่นมาเปลี่ยนจุดศูนย์กลาง โดยไม่บอก จึงใช้วิธีคืนสำเนาของจุดศูนย์กลางแทน

รหัสที่ 8-36 ตัวอย่างการใช้ตัวสร้างสำเนาเพื่อคืนสำเนา ไม่คืนตัวจริง

ตัวแปรที่ตั้งค่าได้ครั้งเดียว

การสร้างความมั่นใจว่า ตัวแปรตัวหนึ่งหลังจากได้รับค่าเริ่มต้นแล้ว จะไม่เปลี่ยนแปลงอีก ทำให้เราเขียนโปรแกรมในหลายสถานการณ์ได้สะดวกมากขึ้น ความต้องการเช่นนี้ทำได้โดยเพิ่มคำว่า final ข้างหน้าการประกาศตัวแปร ซึ่งกระทำได้ทั้งกับตัวแปรเฉพาะที่ในเมท็อด, พารามิเตอร์ของเมท็อด, ตัวแปรประจำอ็อบเจกต์, และตัวแปรประจำคลาส ดังรหัสที่ 8-37 เราใช้ final กำกับพารามิเตอร์ เพราะนักเขียนโปรแกรมจะได้มั่นใจว่า ค่าที่ได้รับมาให้กับพารามิเตอร์มีค่าเดิมตลอด

ทั้งเมท็อด เราเองที่เป็นคนเขียนโปรแกรมจะได้ไม่เผลอไปเปลี่ยน (ถ้ามีการเปลี่ยนในเมท็อด ตัวแปลโปรแกรมจะฟ้อง) การใช้ `final` กำกับตัวแปรประจำคลาส มักใช้คู่กับ `public` ด้วย เพื่อให้ผู้อื่นมาหยิบใช้ได้ แต่เปลี่ยนค่าไม่ได้ เช่น เราให้ความกว้างและความยาวของ `Card` เป็น 50 เพราะตั้งค่าเป็นเช่นนี้แล้ว จะเป็นเช่นนี้ตลอดไป ดังนั้น เราใช้ `final` กำกับค่าคงตัว เช่น `Math.PI` เป็นตัวแปรประจำคลาส `Math` ชื่อ `PI` ที่เป็น `final` เก็บค่าประมาณของ π ในจาวา

```
public class Card {
  public static final int WIDTH = 50, HEIGHT = 50;
  ...
  private void draw(final int color, final boolean showNumber) {
    ...
  }
```

รหัสที่ 8-37 ตัวอย่างการใช้ `final` กับพารามิเตอร์และตัวแปรประจำคลาส

ในกรณีที่เราเขียนคลาสเพื่อผลิตอ็อบเจกต์ที่เปลี่ยนค่าไม่ได้ ก็ควรใส่ `final` กำกับตัวแปรประจำอ็อบเจกต์ แล้วให้ค่าเริ่มต้นของตัวแปรนี้ในตัวสร้าง เช่น คลาส `Complex` ในรหัสที่ 8-38 แสดงการให้ค่ากับตัวแปรประจำอ็อบเจกต์ที่เป็น `final` ภายในตัวสร้าง (และให้ค่าได้ครั้งเดียวด้วย) เราไม่สามารถเขียนการให้ค่าในเมท็อดได้ เพราะเมท็อดของอ็อบเจกต์ถูกเรียกได้หลายครั้ง แต่ตัวสร้างถูกเรียกเพียงครั้งเดียวตอนที่อ็อบเจกต์เกิดเท่านั้น

```
public class Complex {
  private final double real, imag;
  public Complex(double re, double im) {
    real = re; image = im;
  }
```

รหัสที่ 8-38 ตัวอย่างการใช้ `final` กับตัวแปรประจำอ็อบเจกต์

มีความเข้าใจผิดที่เกิดขึ้นบ่อยมากเกี่ยวกับการเขียน `final` กำกับตัวแปรอ้างอิง (ทั้งแบบอ้างอิงอาเรย์ และอ้างอิงอ็อบเจกต์) ซึ่งป้องกันไม่ให้เราเปลี่ยนแปลงค่าของตัวแปร แต่ไม่ได้ป้องกันตัวอาเรย์และข้อมูลในอ็อบเจกต์ที่ถูกอ้างอิง เช่น รหัสที่ 8-39 แสดงตัวแปร `a` ที่เป็น `final` อ้างอิงอาเรย์ 4 ช่อง เราจึงเปลี่ยนให้ `a` ไปอ้างอิงอาเรย์อื่นไม่ได้ แต่ยังคงเปลี่ยนแปลงค่าภายในอาเรย์ได้ ทำนองเดียวกับ `b` ที่อ้างอิงอ็อบเจกต์ซึ่งเรายังเปลี่ยนค่าภายในอ็อบเจกต์ได้

```
  final int[] a = {1,2,3,4};
  a[0] = 24;                          // ทำได้
  a = new int[]{4,3,2,1};             // ทำไม่ได้
  final Point p = new Point(3,4);
  p.setXY(1,2);                       // ทำได้
  p = new Point(0,0);                 // ทำไม่ได้
```

รหัสที่ 8-39 ตัวแปรอ้างอิงแบบ `final` เปลี่ยนค่าไม่ได้ แต่เปลี่ยนข้อมูลของอ็อบแจกต์ที่ถูกอ้างอิงได้

เพิ่มเติม

ยังมีเรื่องจุกจิกอีกมากมายที่เกี่ยวข้องกับการเขียนคลาส หัวข้อย่อยต่อไปนี้นำเสนอประเด็นเพิ่มเติมในจาวาที่น่าสนใจสำหรับผู้เริ่มต้นเขียนโปรแกรมเชิงวัตถุ

คลาสที่ห้ามไม่ให้ new อ็อบเจกต์

มีหลายโอกาสที่เราต้องการเขียนคลาสที่ไม่ให้ผู้อื่นมา `new` อ็อบเจกต์ของคลาสนี้ เช่น เขียนคลาสที่เป็นแหล่งรวมเมท็อดที่ให้บริการหลากหลาย (เรียกคลาสแบบนี้ว่า *คลาสอรรถประโยชน์* - utility class) โดยเขียนเมท็อดเหล่านี้เป็นแบบประจำคลาส จึงไม่ต้องให้สร้างเป็นอ็อบเจกต์ หรือในกรณีที่เราต้องการควบคุมการผลิตอ็อบเจกต์ จึงไม่อนุญาตให้ `new` แต่มีเมท็อดให้บริการสร้างอ็อบเจกต์ซึ่งมีการควบคุมแทน เป็นต้น เพียงแค่ใส่คำว่า `private` กำกับไว้ที่หัวของตัวสร้างทุกตัวที่มีในคลาส ตัวแปลโปรแกรมจะไม่อนุญาตให้ใช้คำสั่ง `new` คลาสเราจากคลาสอื่น

ตัวอย่างเช่น คลาส `Value` ในรหัสที่ 8-40 มีตัวสร้างรับพารามิเตอร์ `v` ที่มีค่าได้ตั้งแต่ 0 ถึง 3 เท่านั้น ผู้ออกแบบคลาสนี้เขียนให้อ็อบเจกต์เป็นแบบเปลี่ยนค่าไม่ได้ มีค่าแตกต่างกันแค่ 4 ค่าเท่านั้น ดังนั้น ทำไมไม่สร้างอ็อบเจกต์ไว้ล่วงหน้า 4 ตัว 4 ค่า เพื่อให้ผู้ใช้ที่ต้องการค่าที่เหมือนกันใช้ร่วมกันได้ จึงเขียนคลาสนี้ใหม่ดังรหัสที่ 8-41 ที่สร้างอ็อบเจกต์เตรียมไว้ก่อนเก็บใส่อาเรย์ชื่อ `value` ใส่ `private` กำกับตัวสร้างไม่ให้ผู้อื่นใช้ และเขียนเมท็อด `getInstance(v)` ให้บริการคืนอ็อบเจกต์ที่มีค่า `v` โดยหยิบจากอาเรย์ `value` ที่ได้สร้างเตรียมไว้คืนกลับไป เพียงเท่านี้ก็สามารถคุมกำเนิดอ็อบเจกต์ของคลาสได้

```
public class Value {
  private final int v;
  public Value(int v) {
    if (v<0 || v>3) throw new IllegalArgumentException("v=" + v);
    this.v = v;
  }
```

ออกแบบให้คลาสนี้ผลิตอ็อบเจกต์เพียง 4 ค่า และเป็นอ็อบเจกต์ที่เปลี่ยนค่าไม่ได้

รหัสที่ 8-40 คลาส `Value` ควบคุมการผลิตอ็อบเจกต์

```
public class Value {
  private static Value[] value = {new Value(0), new Value(1),
                                  new Value(2), new Value(3)};
  private final int v;
  private Value(int v) { this.v = v; }

  public static Value getInstance(int v) {
    if (v<0 || v>3) throw new IllegalArgumentException("v=" + v);
    return value[v];
  }
```

เตรียมอ็อบเจกต์ 4 ตัว 4 ค่าเก็บใส่อาเรย์

คืนอ็อบเจกต์ที่เตรียมไว้ให้ผู้ใช้

รหัสที่ 8-41 คลาส `Value` ควบคุมการผลิตอ็อบเจกต์

การจัดกลุ่มคลาสเป็นชุด

จาวารุ่นที่ 6 มีคลาสมาตรฐานในระบบเกือบสี่พันคลาส นอกจากนี้ยังมีคลาสอีกมากมายในวงการที่นักเขียนโปรแกรมมักนำมาใช้งาน ดังนั้น คลาสที่มีชื่อซ้ำกันจึงมีโอกาสเป็นไปได้ แต่ถ้าเราจัดกลุ่มของคลาสต่าง ๆ ให้เป็นชุดเรียกว่า *แพ็กเกจ* (package) อย่างมีระเบียบก็จะไม่มีปัญหาเรื่องชื่อซ้ำ เช่น คลาสมาตรฐาน `Scanner` ถูกจัดอยู่ในแพ็กเกจชื่อ `java.util` คลาสนี้จึงมีชื่อเต็มว่า `java.util.Scanner` การใช้คลาสใดในโปรแกรมจึงต้องระบุชื่อเต็มของคลาสนั้น แต่การเขียนชื่อเต็มนั้นค่อนข้างรุ่มร่าม ดังนั้น เมื่อใดที่เราจะใช้งานคลาสที่อยู่ในแพ็กเกจอื่น จึงมักใช้คำสั่ง

```
import  ชื่อเต็มของคลาส ;
```

เพื่อให้ตัวแปลโปรแกรมรับทราบก่อนว่า คลาสที่เขียนในโปรแกรมมีชื่อเต็มว่าอะไร [5] รหัสที่ 8-42 แสดงตัวอย่างการใช้คำสั่ง `import` โดยเขียนชื่อเต็มของคลาสที่จะใช้หลังคำว่า `import` เช่น `import java.util.Scanner` เพื่อบอกว่า `Scanner` มีชื่อเต็มดังที่เขียน หรืออาจเขียนแบบเหมารวม เช่น `import java.io.*` เพื่อให้ตัวแปลรู้จักทุกคลาสในชุด `java.io`

```
import java.util.Scanner;
import java.io.*;          import ทุกคลาสในแพ็กเกจ java.io

public class Test {
  public static void main(String[] args) throws IOException {
    Scanner in = new Scanner(new File("data.txt"));
    ...
```

รหัสที่ 8-42 ตัวอย่างการใช้ `import`

ดังนั้น เมื่อเขียนคลาสใหม่ จึงมักระบุด้วยว่า คลาสใหม่นี้อยู่ในแพ็กเกจอะไร โดยใช้คำสั่ง

```
package  ชื่อแพ็กเกจ ;
```

ชื่อแพ็กเกจนี้มักเขียนด้วยตัวอักษรอังกฤษตัวเล็ก คั่นด้วยจุด เช่น `jlab.graphics` บางครั้งก็เขียนในลักษณะที่กลับลำดับกับชื่อโดเมนขององค์กรต้นสังกัด เช่น ถ้าทำอยู่ที่ `sun.com` ก็ใช้ชื่อ `com.sun` รหัสที่ 8-43 แสดงการเขียน `package` ในรหัสต้นฉบับ ซึ่งต้องเขียนเป็นบรรทัดแรก (ไม่นับบรรทัดที่เป็นหมายเหตุ) ตามด้วยคำสั่ง `import` ถ้ามี แล้วค่อยเริ่มการประกาศ `class`

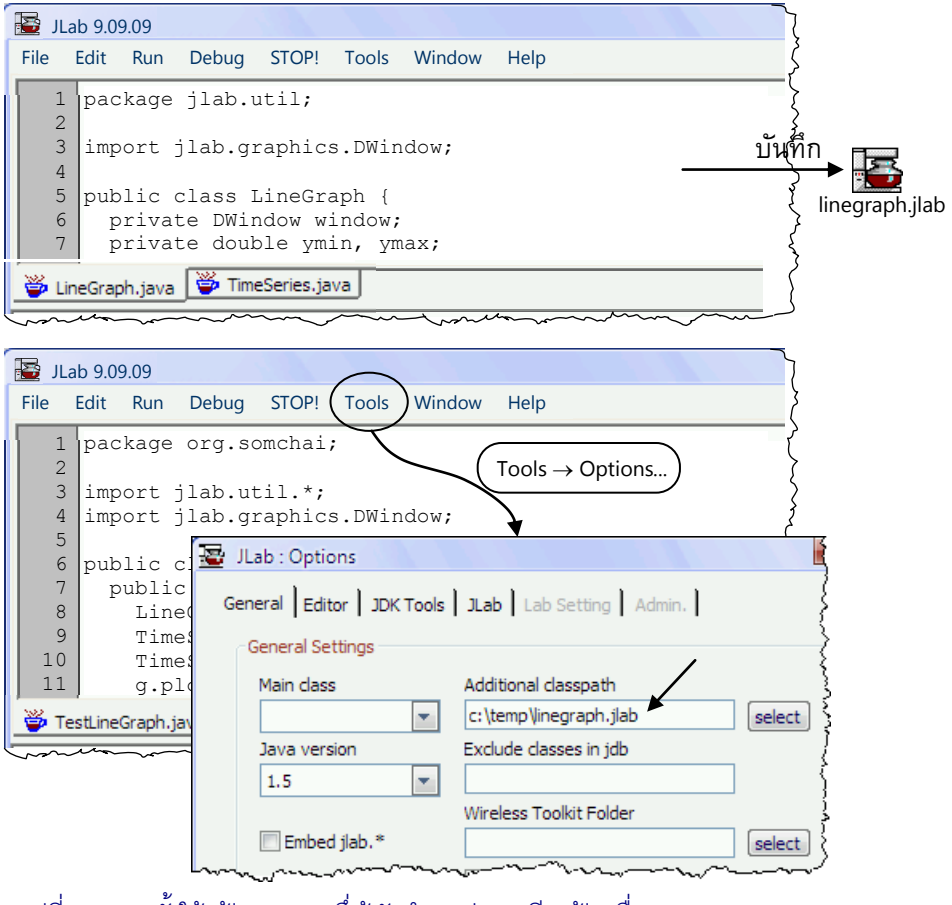
```
package jlab.graphics;      ต้องเขียน package ก่อน
import java.awt.Color;

public class DWindow {      คลาส DWindow อยู่ในแพ็กเกจ jlab.graphics
  ...
```

รหัสที่ 8-43 ส่วนของรหัสต้นฉบับแสดงคำสั่ง `package` ของคลาส `DWindow`

[5] จาวาอนุญาตให้ไม่ต้อง `import` คลาสในแพ็กเกจ `java.lang` เพราะใช้บ่อย เช่น `System`, `Math` เป็นต้น

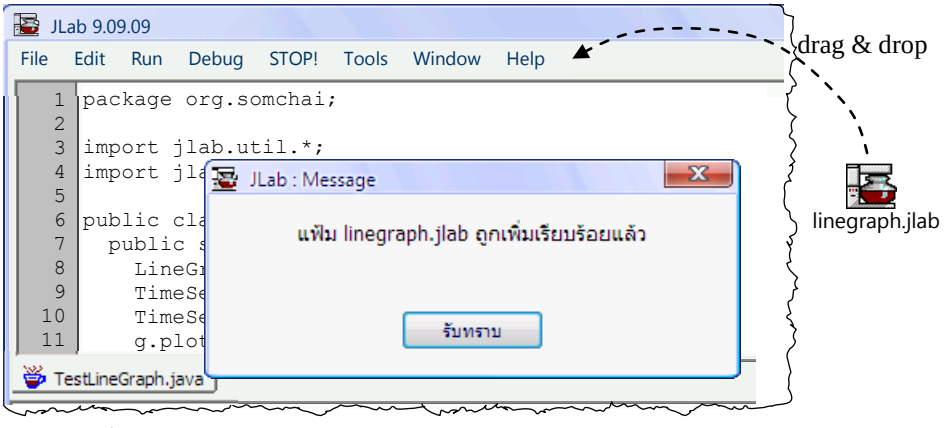
คลาสที่เขียนกันมาตั้งแต่บทแรก เราไม่ได้ระบุชื่อแพ็กเกจเลย สำหรับกรณีเช่นนี้ จาวาจะจัดให้คลาสเหล่านี้อยู่ในแพ็กเกจพิเศษเรียกว่า *แพ็กเกจปริยาย* (default package) แต่ถ้าต้องการระบุชื่อแพ็กเกจเมื่อเขียนคลาสด้วย JLab ต้องเลือกเมนู File → New → Package ระบบจะถามชื่อแพ็กเกจก่อน แล้วค่อยตามด้วยชื่อคลาสที่จะสร้าง หลังจากนั้นทุกคลาสที่สร้างจะอยู่ในแพ็กเกจเดียวกันนี้ เมื่อเขียน แปล ลองใช้งานคลาสได้ผลถูกต้อง และบันทึกลงแฟ้มเรียบร้อย ก็สามารถนำแฟ้มที่ได้ไปเรียกใช้ได้ในคลาสอื่น ๆ ในอนาคต

มาดูตัวอย่างให้เห็นเป็นรูปธรรม สมมติว่าคลาส `LineGraph` และ `TimeSeries` อยู่ในแพ็กเกจ `jlab.util` และได้บันทึกลงที่แฟ้ม c:\temp\linegraph.jlab ดังรูปที่ 8-6 บน และได้เขียนอีกคลาสชื่อ `TestLineGraph` เพื่อใช้งาน `LineGraph` และ `TimeSeries` นี้ โดยเขียนเก็บไว้ในแฟ้ม JLab อีกแฟ้มหนึ่ง จึงต้อง `import` คลาสทั้งสอง และต้องตั้งค่าให้ JLab รู้ตำแหน่งของแฟ้ม linegraph.jlab ด้วย โดยเลือกเมนู Tools → Options... แล้วกรอกตำแหน่งของแฟ้ม ดังรูปที่ 8-6 ล่าง กดปุ่ม OK ก็สามารถแปลและสั่งทำงานได้



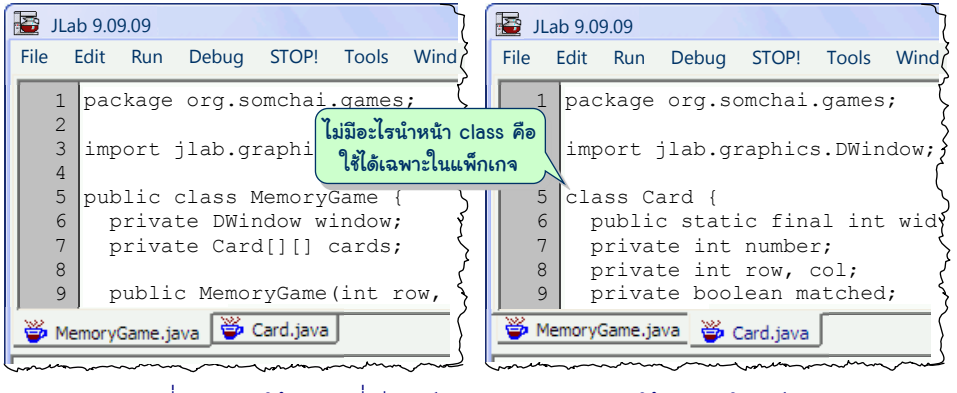
รูปที่ 8-6 การตั้งให้แฟ้ม JLab หนึ่งรู้จักตำแหน่งของอีกแฟ้มเพื่อการ `import`

หรือจะใช้วิธีนำแฟ้ม linegraph.jlab มารวมเข้าในชุดของโปรแกรมที่กำลังเขียนเลยก็ได้ โดยใช้เมาส์คลิกเพื่อเลือกแฟ้ม linegraph.jlab แล้วลากมาวางบนบริเวณเมนูด้านบนของวินโดว์ของ JLab ดังรูปที่ 8-7 ก็สามารถ import แปล และสั่งทำงานโปรแกรมได้ทันที หากมีการเปลี่ยนแปลงแฟ้ม linegraph.jlab ก็อย่าลืมลากมาวางใหม่ในวินโดว์ด้วย



รูปที่ 8-7 การลากแฟ้ม jlab มาวางในวินโดว์ระหว่างการเขียนคลาส

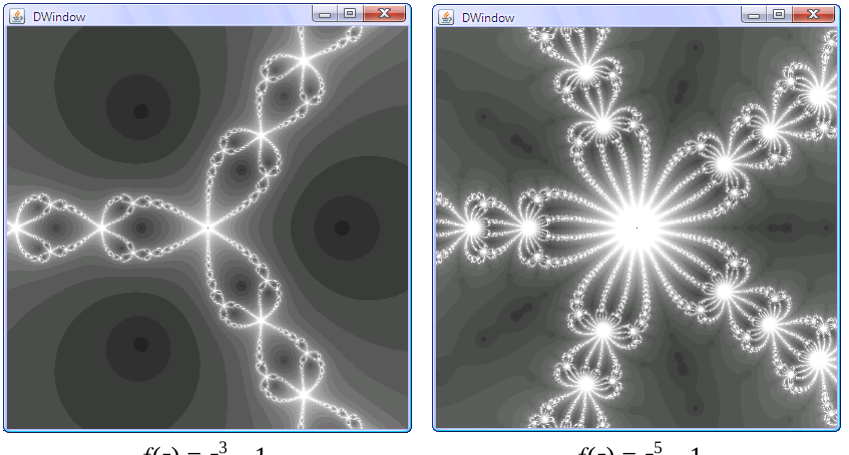
อนึ่งเราเขียน public class ที่หัวคลาส เพื่อระบุว่า คลาสที่เขียนขึ้นสามารถใช้ได้ทั่วไป (เป็นแบบ public) หากเราต้องการให้คลาสที่เขียนใช้ได้เฉพาะภายในแพ็กเกจที่เขียน ก็ไม่ต้องเขียนคำว่า public ตัวอย่างเช่น โปรแกรมเกมจับคู่ทดสอบความจำในตัวอย่างที่ได้นำเสนอไปนั้นประกอบด้วยคลาส MemoryGame และ Card ในมุมมองของการให้บริการ เราต้องการให้ผู้ใช้ใช้เพียง MemoryGame ไม่จำเป็นต้องรู้อะไรเลยเกี่ยว Card หากเขียนให้ทั้งสองคลาสอยู่ในแพ็กเกจ org.somchai.games ก็ควรให้ MemoryGame เป็น public และ Card เป็นแบบใช้ได้เฉพาะในแพ็กเกจก็เพียงพอ ดังรูปที่ 8-8



รูปที่ 8-8 การให้คลาสที่เขียนเป็นแบบสาธารณะกับใช้เฉพาะในแพ็กเกจ

สาทิสรูปของวิธีนิวตัน

ขอปิดท้ายด้วยเรื่องที่ไม่เกี่ยวกับอ็อบเจกต์ แต่น่าสนใจดี ย้อนกลับไปเรื่องของการหารากของฟังก์ชันด้วยวิธีของนิวตันนั้น จำนวนรอบที่ใช้ในการคำนวณรากเป็นข้อมูลที่น่าสนใจ จำนวนรอบที่จะใช้นี้ขึ้นกับค่าเริ่มต้น z_0 หากเราลองตั้งค่าเริ่มต้น z_0 ให้ต่างกันในระนาบของจำนวนเชิงซ้อน เช่น ในช่วงตั้งแต่ $(-1.5 - 1.5i)$ ไปจนถึง $(1.5+1.5i)$ นำค่าของแต่ละ z_0 ไปหารากด้วยวิธีของนิวตัน แล้วนับจำนวนรอบที่ใช้ นำจำนวนรอบที่ใช้มาแสดงเป็นสีกำกับจุดที่แทนค่าเริ่มต้นในระนาบจำนวนเชิงซ้อน ซึ่งแทนได้ด้วยระนาบสองมิติ x-y โดยให้แกน x แทนจำนวนจริง และแกน y แทนจำนวนจินตภาพ จะได้ภาพสวยงามมีลักษณะเป็นแฟรกทัล ดังรูปที่ 8-9 แสดงผลการใช้วิธีของนิวตันหารากของสองฟังก์ชัน $f(z) = z^3 - 1$ และ $f(z) = z^5 - 1$ โดยความสว่างของสีแปรตามจำนวนรอบที่วนทำ ดังโปรแกรมในรหัสที่ 8-44 เมท็อด `numLoops` ทำงานเหมือนกับเมท็อด `findRoot` ในรหัสที่ 8-20 แต่ปรับให้นับและคืนจำนวนรอบเป็นผลลัพธ์ โดยรับพารามิเตอร์ `maxLoops` เป็นตัวจำกัดจำนวนรอบมากสุดที่ให้วน สาเหตุที่ต้องจำกัดจำนวนรอบเพราะว่า การวนหารากอาจทำงานไม่รู้จบสำหรับค่าเริ่มต้นบางค่า (เช่น ให้ $z_0 = 0$ กับการหารากของ $f(z) = z^3 - 2z + 2$) นอกจากนี้เราจำกัดจำนวนรอบเพื่อให้ได้ความแตกต่างของระดับสีของจุดภาพที่ชัดเจน (ผู้เขียนตั้งค่านี้ไว้ที่ 30 ในโปรแกรม) บรรทัดที่ 12 คำนวณระดับความเข้มสีเทา `g` จากจำนวนรอบ `k` ที่ใช้ จากสูตร `g = 255*k/maxLoops` จะได้จุดขาวเมื่อ `k = maxLoops` และได้จุดดำเมื่อ `k` เป็น 0 ผู้อ่านที่สนใจควรลองเปลี่ยนฟังก์ชันเป็นแบบอื่น เปลี่ยนช่วงของค่าเริ่มต้นดู เปลี่ยน `maxLoops` เปลี่ยนการให้สีกับจุดภาพ ว่าจะได้ภาพที่เปลี่ยนแปลงอย่างไร



$f(z) = z^3 - 1$ $f(z) = z^5 - 1$

รูปที่ 8-9 การจินตทัศน์จำนวนรอบที่ใช้ในวิธีของนิวตัน (ความสว่างแปรตามจำนวนรอบ)

```
01 import jlab.graphics.DWindow;
02 public class NewtonFractal {
03   public static void main(String[] args) {
04     DWindow w = new DWindow(400, 400);
05     double width = w.getWidth(), height = w.getHeight();
06     int[][] p = w.getPixmap();
07     int maxLoops = 30;
08     for (int x = 0; x < width; x++) {
09       for (int y = 0; y < height; y++) {
10         Complex z0 = new Complex(-1.5+3*x/width, 1.5-3*y/height);
11         int k = numLoops(z0, maxLoops);
12         int g = k * 255 / maxLoops;
13         p[x][y] = DWindow.mixRGB(g, g, g);
14       }
15     }
16     w.setPixmap(p);
17   }
18   public static int numLoops(Complex z1, int maxLoops) {
19     Complex z0;
20     int k = 0;
21     do {
22       z0 = z1;
23       z1 = z0.substract(f(z0).divide(df(z0)));
24     } while (++k < maxLoops && z1.substract(z0).abs() > 1e-10);
25     return k;
26   }
27   private static Complex f(Complex z) {
28     return z.multiply(z).multiply(z).substract(1);
29   }
30   private static Complex df(Complex z) {
31     return z.multiply(z).multiply(3);
32   }
33 }
```

รหัสที่ 8-44 โปรแกรมแสดงรูปแฟร็กทัลที่ได้จากการหารากของนิวตัน

แบบฝึกหัด

1. จำนวนเต็มแบบ `int` เก็บค่าได้จำกัดประมาณบวกลบสองพันล้าน ถ้าต้องการคำนวณจำนวนฟิโบนักชีตัวที่ 200 ที่มีค่า 280571172992510140037611932413038677189525 ย่อมทำไม่ได้ด้วย `int` รหัสที่ 8-18 นำเสนอตัวอย่างการใช้อ็อบเจกต์ของคลาส `BigInteger` เพื่อคำนวณค่าของ 100! จงเขียนโปรแกรมที่ใช้อ็อบเจกต์เพื่อแสดงจำนวนฟิโบนักชีใด ๆ ที่ผู้ใช้กำหนด (บทที่ 6 นำเสนอนิยามและเมท็อดคำนวณจำนวนฟิโบนักชีจากนิยาม $f_n = f_{n-1} + f_{n-2}$ สำหรับ $n > 1, f_0 = 0, f_1 = 1$)

2. จงปรับปรุงคลาส `Ball` ในรหัสที่ 8-10 ให้มีสีประจำลูกบอล

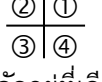
3. จงปรับปรุงคลาส `Time` ในรหัสที่ 8-17 ให้เป็นเวลาแบบ 12 ชั่วโมง สองช่วง AM และ PM

4. เมท็อด `getMovingAverage` ในรหัสที่ 8-23 จะทำงานผิดพลาดถ้าข้อมูลอนุกรมเวลาที่เก็บอยู่มีข้อมูล 0 หรือ 1 ตัว จงปรับปรุงให้รองรับสถานการณ์ดังกล่าว

5. จงย้ายเมท็อด `plot` จากคลาส `LineGraph` ในรหัสที่ 8-22 ไปอยู่ในคลาส `TimeSeries` ทำให้ผู้ใช้เรียกบริการวาดกราฟเส้นของข้อมูลอนุกรมเวลาได้โดยตรง (จึงไม่จำเป็นต้องมีคลาส `LineGraph`)

6. จงปรับปรุงคลาส `Ball` (รหัสที่ 8-10) ให้มีเมท็อดประจำคลาส `getNumberOfBalls()` ซึ่งคืนจำนวนลูกบอลทั้งหมดที่ได้ผลิตออกไปใช้งาน (ข้อแนะนำ : ใช้ตัวแปรประจำคลาสเพื่อนับจำนวนอ็อบเจกต์ของ `Ball` ตามที่โจทย์ต้องการ โดยจะเพิ่มค่าทุกครั้งในตัวสร้าง)

7. ความดันโลหิตคือความดันที่เกิดจากการบีบและคลายตัวของหัวใจ ค่าของความดันโลหิตประกอบด้วยสองค่า เช่น 120/80 ค่าแรกคือ ความดันช่วงหัวใจบีบตัว (เรียกว่า systolic) และค่าที่สองคือ ค่าความดันช่วงหัวใจคลายตัว (diastolic) ค่าทั้งสองมีหน่วยเป็นมิลลิเมตรปรอท เมื่อใดมีค่าเกิน 140/90 ถือได้ว่า ความดันโลหิตสูง จงเขียนคลาส `BloodPressure` เพื่อผลิตอ็อบเจกต์ที่แทนความดันโลหิต โดยมีเมท็อด `isHigh` ให้บริการตรวจสอบว่า ความดันโลหิตสูงหรือไม่ ให้กำหนดตัวแปรประจำอ็อบเจกต์และตัวสร้างที่เหมาะสม

8. จงเขียนคลาสชื่อ `Point` มีไว้แทนพิกัดของจุดในระนาบสองมิติ ซึ่งมีตัวสร้างและเมท็อดให้บริการดังนี้
   - `Point()` : ตัวสร้างจุดให้มีพิกัดที่ `(0,0)`
   - `Point(double x, double y)` : ตัวสร้างจุดให้มีพิกัดที่ `(x,y)`
   - `Point(Point p)` : ตัวสร้างสำเนาให้มีพิกัดเหมือนกับจุด `p`
   - `double getX()` : คืนพิกัด `x` ของจุดนี้
   - `double getY()` : คืนพิกัด `y` ของจุดนี้
   - `void setX(double x)` : ตั้งพิกัด `x` ใหม่ให้จุดนี้
   - `void setY(double y)` : ตั้งพิกัด `y` ใหม่ให้จุดนี้
   - `double distanceTo(Point p)` : คืนระยะทางสั้นสุดบนระนาบสองมิติจากจุดนี้ถึงจุด `p`
   - `int getQuadrant()` คือหมายเลขจตุภาค (quadrant) ของจุดนี้ 

   - `boolean equals(Point p)` : ตรวจสอบว่าจุดนี้กับจุด `p` มีพิกัดอยู่ที่เดียวกันหรือไม่
   - `String toString()` : คืนสตริงที่แสดงค่าพิกัดของจุดนี้

9. จงเขียนคลาสชื่อ `Line` มีไว้แทนเส้นตรงบนระนาบสองมิติ ซึ่งมีตัวสร้างและเมท็อดให้บริการดังนี้ (เราบรรยายเส้นตรงด้วยฟังก์ชัน $y = mx + b$ โดยที่ m คือความชัน และ b คือจุดตัดระหว่างเส้นตรงนี้กับเส้นแกน y)
   - `Line(double m, double b)` : ตัวสร้างเส้นตรง `y = mx + b`
   - `Line(Line line)` : ตัวสร้างสำเนาเส้นตรงนี้ให้เหมือนกับเส้น `line`
   - `double getY(double x)` : คืนพิกัด `y` ของเส้นตรงนี้ที่พิกัด `x` ที่ได้รับ
   - `double getX(double y)` : คืนพิกัด `x` ของเส้นตรงนี้ที่พิกัด `y` ที่ได้รับ
   - `boolean contains(Point p)` : ตรวจสอบว่าจุด `p` อยู่บนเส้นตรงนี้หรือไม่
   - `boolean isParallelTo(Line line)` : ตรวจสอบเส้นนี้ขนานกับ `line` หรือไม่
   - `boolean isPerpendicularTo(Line line)` : ตรวจสอบเส้นนี้ตั้งฉากกับ `line` หรือไม่
   - `Point intersection(Line line)` : คืนจุดตัดของเส้นนี้กับเส้น `line`
   - `boolean equals(Line line)` : ตรวจสอบว่าเส้นตรงนี้เป็นเส้นที่เหมือนกับ `line` หรือไม่
   - `String toString()` : คืนสตริงที่แสดงฟังก์ชันของเส้นตรงนี้

10. จงเขียนคลาสชื่อ `Rectangle` มีไว้แทนสี่เหลี่ยมผืนผ้า ที่มีตัวสร้างและเมท็อดให้บริการดังนี้
    - `Rectangle(double x, double y, double w, double h)` : ตัวสร้างสี่เหลี่ยมผืนผ้ากว้าง `w` สูง `h` มีมุมซ้ายบนที่พิกัด `(x, y)`
    - `double getArea()` : คืนพื้นที่ของสี่เหลี่ยมนี้
    - `double getPerimeter()` : คืนเส้นรอบรูปของสี่เหลี่ยมนี้
    - `boolean contains(Point p)` : ตรวจว่าจุด `p` อยู่ในสี่เหลี่ยมนี้หรือไม่

11. กำหนดให้กระปุกออมสินเก็บได้เฉพาะเหรียญ 1 บาท, 2 บาท, 5 บาท และ 10 บาท จงเขียนคลาส `PiggyBank` ซึ่งเป็นคลาสผลิตกระปุกออมสิน ที่มีตัวสร้างและเมท็อดให้บริการดังนี้
    - `PiggyBank()` : ตัวสร้างกระบุกออมสินเปล่า ๆ
    - `void add1(int c)` : แทนการหยอดเหรียญ 1 บาทจำนวน `c` เหรียญลงในกระปุก
    - `void add2(int c)` : แทนการหยอดเหรียญ 2 บาทจำนวน `c` เหรียญลงในกระปุก
    - `void add5(int c)` : แทนการหยอดเหรียญ 5 บาทจำนวน `c` เหรียญลงในกระปุก
    - `void add10(int c)` : แทนการหยอดเหรียญ 10 บาท `c` เหรียญลงในกระปุก
    - `void clear()`: แทนการเทเหรียญทั้งหมดออกจากกระปุก
    - `int getTotal()` : คืนจำนวนเงินทั้งหมดที่เก็บในกระปุก
    - `String toString()` : คืนสตริงที่แสดงจำนวนเหรียญแต่ละประเภทในกระปุกนี้

12. จำนวนตรรกยะ (rational number) คือ จำนวนที่เขียนเป็นอัตราส่วนของจำนวนเต็มสองจำนวน ที่เรียกว่า *ตัวเศษ* (numerator) และ*ตัวส่วน* (denominator) เช่น 1/3, 10/1 เป็นต้น จงเขียนคลาส `Rational` มีไว้ผลิตอ็อบเจกต์ของจำนวนตรรกยะ ที่มีตัวสร้างและเมท็อดให้บริการดังนี้
    - `Rational()` : ตัวสร้างจำนวนตรรกยะ `0/1`
    - `Rational(int n)` : ตัวสร้างจำนวนตรรกยะ `n/1`
    - `Rational(int n, int d)` : ตัวสร้างจำนวนตรรกยะ `n/d`
    - `Rational add(Rational r)` : คืนผลบวกของจำนวนนี้กับ `r`
    - `Rational sub(Rational r)` : คืนผลต่างของจำนวนนี้กับ `r`
    - `Rational mul(Rational r)` : คืนผลคูณของจำนวนนี้กับ `r`
    - `Rational div(Rational r)` : คืนผลหารของจำนวนนี้ด้วย `r`
    - `Rational inv()` : คืนส่วนกลับของจำนวนนี้
    - `boolean equals(Rational r)` : ตรวจสอบว่าจำนวนนี้มีค่าเท่ากับของ `r` หรือไม่ (ข้อสังเกต : 1/2, 2/4, 4/8 มีค่าเท่ากันหมด)
    - `String toString()` : คืนสตริงที่แสดงจำนวนตรรกยะนี้

    คลาสนี้เป็นคลาสที่ผลิตอ็อบเจกต์แบบที่เปลี่ยนค่าไม่ได้ (ให้สังเกตว่า เมท็อดข้างบนนี้ไม่เปลี่ยนค่าของอ็อบเจกต์ที่ถูกเรียกแต่อย่างใด) ดังนั้น จึงควรประกาศค่าคงตัวประจำคลาส (`public static final`) ชื่อ `ONE` กับ `ZERO` ที่แทนค่า 1 กับ 0 เพื่อผู้ใช้จะได้เรียกใช้ได้เลย โดยไม่ต้องสร้างอ็อบเจกต์

    ให้สังเกตว่า ในขณะที่การบวกจำนวนจริงแบบ `double` ที่มีค่า 0.1 สิบครั้งได้ผลไม่เท่ากับ 1.0 (ดังที่เคยนำเสนอในบทที่ 2) แต่การนำจำนวนตรรกยะของคลาส `Rational` ที่มีค่า 1/10 มาบวกกันสิบครั้งต้องได้ 1

13. จงเขียนคลาสชื่อ `IntSet` มีไว้ผลิตอ็อบเจกต์ที่ทำหน้าที่คล้ายเซต (ที่คุ้นเคยในคณิตศาสตร์) แต่ในที่นี้จะเป็นเซตของจำนวนเต็ม และเป็นจำนวนเต็มที่มีค่าตั้งแต่ `0` ถึง `n-1` เท่านั้น (`n` เป็นค่าที่ผู้ใช้กำหนด) มีตัวสร้างและเมท็อดให้บริการดังนี้
    - `IntSet(int n)` : ตัวสร้างเซตที่เก็บข้อมูลได้เฉพาะ `0` ถึง `n-1` หลังสร้างได้เซตว่าง
    - `void add(int x)` : เพิ่ม `x` เข้าไปในเซตนี้
    - `boolean contains(int x)` : ตรวจสอบว่าเซตนี้มี `x` อยู่หรือไม่
    - `boolean isSubsetOf(IntSet s)` : ตรวจสอบว่าเซตนี้เป็นเซตย่อยของเซต `s` หรือไม่
    - `IntSet union(IntSet s)` : คืนเซตใหม่ที่เป็นผลมาจากการยูเนียนเซตนี้กับ `s`

- `IntSet intersection(IntSet s)` : คืนเซตใหม่ที่เป็นผลมาจากการอินเตอร์เซกชันเซตนี้กับ `s`
- `boolean equals(IntSet s)` : ตรวจสอบว่าเซตนี้เหมือนกับเซต `s` หรือไม่
- `String toString()` : คืนสตริงที่แสดงสมาชิกทุกตัวของเซตนี้

ข้อแนะนำ : วิธีเก็บข้อมูลในเซตแบบง่าย คือ การสร้างอาเรย์ของ `boolean` ชื่อ `set` ขนาด `n` ช่อง ถ้าเซตนี้เก็บจำนวนเต็ม `k` ก็ให้ `set[k]` เก็บ `true` ถ้าไม่มีค่า `k` ในเซตนี้ ก็ให้ `set[k]` เก็บ `false`

14. จงเขียนคลาสชื่อ `DataBin` มีไว้ผลิตอ็อบเจกต์ที่ทำหน้าที่เสมือนถังเก็บข้อมูลที่เป็นจำนวนจริง เพื่อให้บริการการคำนวณค่าทางสถิตศาสตร์ มีตัวสร้างและเมท็อดให้บริการดังนี้

DataBin

- ~~`DataCollection`~~`()` : ตัวสร้างถังเก็บข้อมูล หลังสร้างได้ถังว่าง ๆ
- `void add(double x)` : เพิ่มข้อมูล `x` ในถัง
- `int size()` : คืนจำนวนข้อมูลที่เก็บในถัง
- `double getSum()` : คืนผลรวมของข้อมูลที่เก็บในถัง
- `double getMean()` : คืนค่าเฉลี่ยของข้อมูลที่เก็บในถัง
- `double getSD()` : คืนค่าเบี่ยงเบนมาตรฐานของข้อมูลที่เก็บในถัง
- `double[] getData()` : คืนข้อมูลทั้งหมดในถังออกมาในรูปของอาเรย์

15. ซอฟต์แวร์ที่ใช้งานกับแฟ้มข้อมูลทั่วไปมักมีการจำว่าได้เคยใช้งานกับแฟ้มใดบ้าง โดยจะจำชื่อแฟ้มล่าสุดจำนวนหนึ่งไว้ให้ผู้ใช้เลือกได้เพื่อความสะดวก เช่น ไมโครซอฟต์เวิร์ดรุ่น XP จะแสดงชื่อแฟ้มเอกสาร 4 แฟ้มล่าสุดที่ผู้ใช้เปิดใช้งานไว้ที่เมนู จงเขียนคลาส `RecentlyUsed` มีไว้ผลิตที่เก็บรายการชื่อแฟ้มที่ใช้ล่าสุดจำนวน `n` ชื่อ โดยชื่อที่ต้นรายการแทนชื่อที่เพิ่งใช้ล่าสุด มีตัวสร้างและเมท็อดให้บริการดังนี้

- `RecentlyUsed(int n)` : ตัวสร้างรายการชื่อล่าสุดขนาด `n` ช่อง เสมือนเป็นอาเรย์ของสตริงขนาด `n` ช่อง ที่เก็บสตริงเรียงลำดับตามเวลาการอ้างอิงชื่อนั้น
- `RecentlyUsed(String[] a)` : ตัวสร้างรายการชื่อล่าสุดที่มีจำนวนช่องและค่าเริ่มต้นตามที่ปรากฏในอาเรย์ `a` ที่ได้รับ
- `void add(String s)` : เพิ่มชื่อ `s` ในรายการให้เป็นชื่อล่าสุด ถ้า `s` มีอยู่แล้วในรายการ ก็เพียงแค่ย้าย `s` มาอยู่ที่ต้นรายการ แต่ถ้า `s` ไม่มีอยู่ในรายการ ก็จะแทรก `s` ที่ต้นรายการ ในกรณีที่รายการมีข้อมูลครบทุกช่องอยู่แล้ว ชื่อที่อยู่หลังสุดของรายการ ก็จะถูกลบออกจากรายการ
- `String get(int k)` : คืนชื่อที่ `k` ในรายการ
- `String[] toArray()` คืออาเรย์ที่เก็บทุกชื่อในรายการตามลำดับ
- `String toString()` : คืนสตริงที่แสดงทุกชื่อที่เก็บในรายการตามลำดับ

ดูตัวอย่างการใช้งานข้างล่างนี้เพื่อให้เข้าใจลักษณะการใช้งาน และการเปลี่ยนแปลงข้อมูลในรายการ

```
RecentlyUsed r = new RecentlyUsed(3);
r.add("A");   // <A>          เพิ่ม A ที่ต้นรายการ
r.add("B");   // <B, A>       เพิ่ม B ที่ต้นรายการ
r.add("C");   // <C, B, A>    เพิ่ม C ที่ต้นรายการ
r.add("B");   // <B, C, A>    ย้าย B มาอยู่ต้นรายการ
r.add("B");   // <B, C, A>    B อยู่ต้นรายการอยู่แล้ว ไม่เปลี่ยนแปลง
r.add("D");   // <D, B, C>    เพิ่ม D ที่ต้นรายการ  A ถูกเตะออก
```

สร้างใหม่จากเก่า

หลักปฏิบัติสำคัญในการคิดประดิษฐ์สิ่งใหม่ คือ หลีกเลี่ยงการออกแบบสิ่งประดิษฐ์ที่ซ้ำกับของที่มีอยู่ โดยต้องถามตัวเองเสมอว่า ของที่เราต้องการออกแบบใหม่นั้นต่างจากของที่เคยมีอย่างไร และจะนำสิ่งที่มีอยู่เดิมมาต่อยอดให้เกิดประโยชน์ได้อย่างไรให้ตรงกับวัตถุประสงค์ที่ตั้งใจไว้ วิธีการออกแบบคลาสใหม่ในบทที่แล้วอาศัยแนวคิดที่เรียกว่า *การประกอบ* (composition) คือ การเขียนคลาสที่ภายในประกอบด้วยข้อมูลย่อยที่อาจเป็นข้อมูลประเภทพื้นฐาน เช่น `double`, `int` หรือเป็นอ็อบเจกต์ของคลาสเดิมที่มีอยู่ บทนี้นำเสนอการออกแบบคลาสใหม่ด้วยแนวคิดอีกแบบเรียกว่า *การรับทอด* (inheritance) ที่ทำให้คลาสใหม่รับลักษณะสมบัติต่าง ๆ ของคลาสเดิมที่มีอยู่มาใช้ และยังสามารถขยายและเปลี่ยนลักษณะสมบัติจากคลาสเดิมนั้นได้อีกด้วย โดยจะนำเสนอแนวคิด ตัวอย่างการเขียนโปรแกรม ตัวอย่างการใช้งาน และหลักปฏิบัติในการออกแบบคลาสเพื่อให้ได้คลาสที่สั้น สวย และปรับปรุงเปลี่ยนแปลงได้ง่าย

การรับทอด

การออกแบบคลาสใหม่ที่อาศัยลักษณะสมบัติบางประการของคลาสเดิมกระทำได้ง่าย ๆ ด้วยการนำรหัสต้นฉบับส่วนที่ต้องการของคลาสเดิมมาเขียนในคลาสใหม่ หากในภายหลังพบปัญหาการทำงานบางประการในคลาสเดิมที่นำไปใช้ในคลาสอื่น ก็ต้องตามแก้ไขเปลี่ยนแปลงรหัสต้นฉบับของทุก ๆ คลาสที่นำไปใช้ด้วย นับเป็นภาระที่ยุ่งยากมากในการบำรุงรักษาซอฟต์แวร์

แต่ถ้าเราสร้างคลาสใหม่ที่ใช้รหัสเครื่อง (หรือที่เรียกว่ารหัสไบต์) ของคลาสเดิม การเปลี่ยนแปลงรหัสเครื่องของคลาสเดิม จะมีผลกระทบต่อคลาสอื่นที่ใช้คลาสที่มีการเปลี่ยนแปลงนี้

โดยอัตโนมัติ ตัวอย่างเช่น เราได้เขียนกันมาหลายโปรแกรมที่ใช้คลาส DWindow หากผู้เขียนคลาส DWindow ปรับปรุงให้ทำงานเร็วขึ้น กำจัดข้อผิดพลาดที่พบ หรือใช้หน่วยความจำในการประมวลผลให้น้อยลง เมื่อเรานำรหัสเครื่องของ DWindow รุ่นใหม่มาติดตั้งในระบบ ก็ไม่จำเป็นต้องแก้ไขรหัสต้นฉบับของโปรแกรมที่ใช้ DWindow โปรแกรมเหล่านี้สามารถใช้ลักษณะสมบัติใหม่ๆ ของ DWindow รุ่นใหม่ได้ทันที

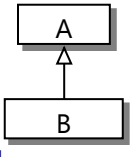
การรับทอดเป็นแนวคิดของการสร้างคลาสใหม่จากรหัสเครื่องของคลาสเดิมที่มีอยู่ ขอเริ่มนำเสนอกรณีง่าย ๆ ก่อน รหัสที่ 9-1 แสดงตัวอย่างการเขียนคลาสใหม่ชื่อ B ซึ่งผลิตอ็อบเจกต์ที่มีลักษณะสมบัติเหมือนของคลาส A กล่าวได้ว่า

- คลาส B รับทอดลักษณะสมบัติ (เสมือนรับมรดกหรือสืบทอดพันธุกรรม) จากคลาส A
- คลาส A เป็น*ซูเปอร์คลาส* (superclass) และคลาส B เป็น*ซับคลาส* (subclass)
- คลาส A เป็น*คลาสแม่* คลาส B เป็น*คลาสลูก*

ถ้ารหัสเครื่องของคลาสแม่มีการเปลี่ยนแปลง จะส่งผลมาให้กับคลาสลูกโดยอัตโนมัติ ความสัมพันธ์ในการรับทอดลักษณะสมบัติจากคลาสแม่สู่คลาสลูกนี้ เขียนได้เป็นแผนภาพที่แสดงในรหัสที่ 9-1 คลาสแทนด้วยสี่เหลี่ยมมุมฉากมีชื่อคลาสอยู่ภายใน มีเส้นเชื่อมพุ่งจากคลาสลูกไปยังคลาสแม่

```
01 public class B extends A {
02
03 }
```

B รับทอดจาก A

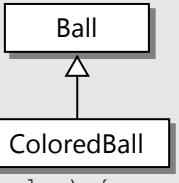


รหัสที่ 9-1 คลาส B มีตัวแปรและเมท็อดประจำอ็อบเจกต์เหมือนของคลาส A

การรับทอดคงไม่ได้มีไว้ให้สร้างคลาสใหม่แบบง่าย ๆ ดังรหัสที่ 9-1 แต่มีไว้ให้สร้างคลาสลูกที่ขยายลักษณะสมบัติเพิ่มเติมจากคลาสแม่ (จาวาจึงใช้คำว่า extends ที่หัวคลาสเพื่อแทนความสัมพันธ์ดังกล่าว) คลาสลูกสามารถเพิ่มตัวแปรและเมท็อดประจำอ็อบเจกต์ ตัวแปรและเมท็อดประจำคลาส และตัวสร้างได้เหมือนการเขียนคลาสทั่วไป ขอเน้นว่า คลาสลูกรับทอดทุกอย่างที่ไม่เป็น private ของคลาสแม่ ยกเว้นตัวสร้างของคลาสแม่ คลาสลูกต้องเขียนตัวสร้างของตัวเอง ทั้งนี้เพราะหน้าที่ของตัวสร้างคือการตั้งค่าเริ่มต้นให้กับตัวแปรประจำอ็อบเจกต์ของตัวเอง ตัวสร้างของคลาสแม่จึงตั้งค่าเริ่มต้นให้ตัวแปรประจำอ็อบเจกต์ของคลาสแม่ ตัวสร้างของคลาสลูกก็ตั้งค่าเริ่มต้นของตัวแปรประจำอ็อบเจกต์ของคลาสลูก ต่างคนต่างทำ ไม่เกี่ยวกัน จึงไม่รับทอดตัวสร้าง (โดยคลาสลูกต้องเรียกใช้ตัวสร้างของคลาสแม่ ซึ่งจะกล่าวต่อไป)

มาดูสักตัวอย่างให้เห็นเป็นรูปธรรม จากคลาส Ball (รหัส 8-10) ที่ได้เขียนมา เป็นคลาสที่มีไว้ผลิตลูกบอลสีดำเด้งไปมาในวินโดว์ หากต้องการให้ตั้งสีลูกบอลได้ ก็เขียนเป็นคลาสใหม่ชื่อ ColoredBall ที่ขยายลักษณะสมบัติจากคลาส Ball ดังแสดงในรหัสที่ 9-2 มีการเขียนคำสั่งเพิ่มเติมดังนี้

```
01 import jlab.graphics.DWindow;
02 public class ColoredBall extends Ball {
03   private int color;
04   public ColoredBall(double r, double x, double y) {
05     this(r, x, y, DWindow.BLACK);
06   }
07   public ColoredBall(double r, double x, double y, int color) {
08     super(r, x, y);
09     this.color = color;
10   }
11   public void setColor(int c) {
12     this.color = c;
13   }
14   public int getColor() {
15     return this.color;
16   }
17   public void draw(DWindow w) {
18     w.fillEllipse(this.color, super.x,super.y,2*super.r,2*super.r);
19   }
20 }
```



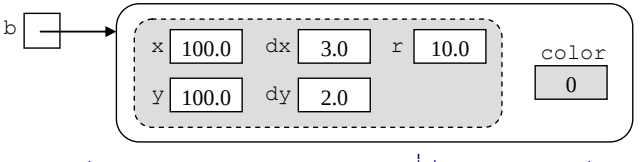
เรียกตัวสร้างของคลาส Ball

หัวเมท็อดเหมือนของคลาสแม่ เป็นการแทนเมท็อดที่ได้รับทอด

ใช้ตัวแปรของคลาสแม่

รหัสที่ 9-2 คลาส `ColoredBall` เขียนจากคลาส `Ball` ให้ตั้งสีได้

- เพิ่มตัวแปรประจำอ็อบเจกต์ชื่อ `color` ไว้เก็บสีของลูกบอล (บรรทัดที่ 3) ทำให้อ็อบเจกต์ของ `ColoredBall` มีตัวแปรประจำอ็อบเจกต์เพิ่มอีก 1 ตัว จากของคลาสแม่ที่มีอยู่แล้ว 5 ตัว ดังตัวอย่างที่แสดงในรูปที่ 9-1



รูปที่ 9-1 ตัวอย่างอ็อบเจกต์ของ `ColoredBall` ที่มีตัวแปรประจำอ็อบเจกต์ 6 ตัว

- เพิ่มเมท็อดประจำอ็อบเจกต์ `setColor` และ `getColor` ไว้ตั้งสีและคืนสีของลูกบอลตามลำดับ (บรรทัดที่ 11 ถึง 16)
- เขียนตัวสร้างที่รับรัศมีและตำแหน่งของลูกบอล (บรรทัดที่ 4 ถึง 6) ภายในตัวสร้างนี้เรียกใช้ตัวสร้างอีกตัวด้วยคำสั่ง `this(...)` โดยตั้งให้ลูกบอลนี้เป็นสีดำ (อย่าลืมว่าตัวสร้างของคลาสแม่ไม่ได้ส่งทอดมาให้คลาสลูก)
- เขียนตัวสร้างอีกตัวที่นอกจากจะรับรัศมีและตำแหน่งแล้ว ยังรับสีของลูกบอลด้วย (บรรทัดที่ 7 ถึง 9) ให้สังเกต คำสั่ง `super(r,x,y)` ที่บรรทัดแรก ของตัวสร้างนี้ คือการเรียกตัวสร้างของซูเปอร์คลาสหรือคลาสแม่ เนื่องจากหน้าที่ของตัวสร้าง คือ ตั้งค่าเริ่มต้นให้กับตัวแปรประจำอ็อบเจกต์ ดังนั้น จึงควรเรียกตัวสร้างของคลาสแม่เพื่อให้คลาสแม่ตั้งค่าเริ่มต้นให้กับตัวแปรประจำอ็อบเจกต์ที่คลาสแม่รับผิดชอบให้เสร็จเสียก่อน

(ในตัวอย่างคือ รัศมี ตำแหน่ง และความเร็วของลูกบอล) แล้วค่อยกลับมาตั้งค่าเริ่มต้นให้กับตัวแปรที่เป็นของคลาสลูก (ในตัวอย่างคือ สีของลูกบอล) [1]

- เขียนเมท็อด `draw` ใหม่ (บรรทัดที่ 17 ถึง 19) เพื่อแทนเมท็อด `draw` ที่รับทอดมาจากคลาสแม่ เพราะต้องวาดลูกบอลให้มีสีด้วย ระบบรู้ว่า เมท็อดที่เขียนจะแทนของที่รับทอดมา ก็เมื่อเมท็อดนั้นมีหัวเหมือนกับของคลาสแม่นั่นเอง ให้สังเกตว่า เราสามารถใช้ตัวแปรประจำอ็อบเจกต์ (และใช้เมท็อดประจำอ็อบเจกต์) ของคลาสแม่ได้ด้วยการนำหน้าตัวแปร (และเมท็อด) ด้วย `super.` (เทียบได้กับ `this.` เมื่อใช้ของอ็อบเจกต์เราเอง) เช่น `super.x` แทนการใช้ `x` ของคลาสแม่, `super.draw(w)` แทนการเรียกเมท็อด `draw` ของคลาสแม่ (ถ้าเขียน `this.draw(w)` ก็เป็นการเรียกเมท็อด `draw` ของคลาสเราเอง)

เราจะละไม่เขียน `this.` และ `super.` ก็ได้ หากไม่กำกวม (ไม่มีตัวแปรเฉพาะที่ในเมท็อดที่มีชื่อซ้ำกับชื่อตัวแปรประจำอ็อบเจกต์) เพราะถือว่า “ของของแม่ก็เหมือนของของลูก” ดังรหัสที่ 9-3

```
01 import jlab.graphics.DWindow;
02 public class ColoredBall extends Ball {
03   private int color;
04   public ColoredBall(double r, double x, double y) {
05     this(r, x, y, DWindow.BLACK);
06   }
07   public ColoredBall(double r, double x, double y, int color) {
08     super(r, x, y);
09     this.color = color;
10   }
11   public void setColor(int c) {
12     color = c;
13   }
14   public int getColor() {
15     return color;
16   }
17   public void draw(DWindow w) {
18     w.fillEllipse(color, x, y, 2*r, 2*r);
19   }
20 }
```

บรรทัดนี้ห้ามตัด this. เพราะมีชื่อซ้ำ

ใช้ x, y, และ r ของซูเปอร์คลาสได้เลย

รหัสที่ 9-3 คลาส `ColoredBall` (แบบไม่มี `this. super.`) เพราะของแม่ก็เหมือนของลูก

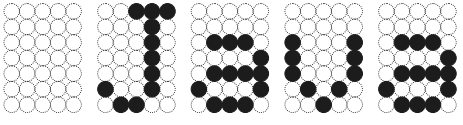
[1] การใช้ `super(...)` เพื่อเรียกตัวสร้างของคลาสแม่ต้องปรากฏที่บรรทัดแรกของตัวสร้าง จากที่เคยนำเสนอว่าถ้าต้องการเรียกตัวสร้างตัวอื่นในคลาสเดียวกันให้ใช้ `this(...)` และต้องเขียนที่บรรทัดแรกของตัวสร้างเช่นกัน ดังนั้น จะเขียนทั้ง `super(...)` และ `this(...)` ในตัวสร้างเดียวกันไม่ได้ สำหรับตัวสร้างใดในคลาสที่บรรทัดแรกไม่ใช่ทั้ง `super(...)` และ `this(...)` ระบบจะเติมคำสั่ง `super()` ให้อัตโนมัติ ซึ่งคือการเรียกตัวสร้างของคลาสแม่ที่ไม่รับพารามิเตอร์ จึงมักถือเป็นแนวปฏิบัติว่า คลาสใดที่เขียนขึ้นมาแล้วคาดว่าจะมีคลาสอื่นมา `extends` ก็ควรเขียนตัวสร้างแบบไม่รับพารามิเตอร์ เผื่อให้คลาสลูกเรียกไว้ด้วย

ตัวอย่าง

หัวข้อนี้นำเสนอตัวอย่างการสร้างคลาสใหม่ด้วยการรับทอด เริ่มจากคลาสที่มีไว้สร้างแผงแอลอีดีแบบจุด ที่นำไปสร้างคลาสลูกสำหรับแสดงตัวเลข และสร้างแผงแสดงจำนวนเต็ม ตามด้วยตัวอย่างการสร้างวินโดว์ `DWindow3D` ที่มีความลึก และคลาสลูกบอลที่ใช้ได้ในห้องสามมิติ

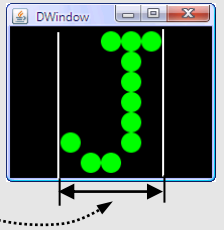
แอลอีดีแบบจุด

ผู้อ่านคงเคยเห็นแผงไฟซึ่งประกอบไปด้วยหลอดไฟดวงเล็ก ๆ วางเรียงกันเป็นแถว ๆ ที่สามารถเปล่งแสงได้หลายสีเพื่อแสดงข้อความและรูปภาพ ในปัจจุบันหลอดไฟแต่ละดวงนี้ทำจากไดโอดเปล่งแสง (light-emitting diode) เรียกสั้น ๆ ว่า *แอลอีดี* (LED) และเรียกกลุ่มของแอลอีดีที่วางเรียงกันเป็นแถว ๆ แบบนี้ว่า *แอลอีดีแบบจุด* (dot-matrix LED) เราจะมาเขียนคลาสชื่อ `DotMatrixLED` เพื่อให้สร้างอ็อบเจกต์ที่ใช้จำลองการใช้งานแอลอีดีแบบจุดบนจอภาพ รูปที่ 9-2 แสดงตัวอย่างแอลอีดีแบบจุดขนาด 5×7 (กว้าง×สูง) และตัวอย่างตัวอักษร



รูปที่ 9-2 ตัวอย่างการนำแอลอีดีแบบจุดขนาด 5×7 มาเรียงเป็นแผง

```
01 import jlab.graphics.DWindow;
02 public class TestDotMatrixLED {
03   public static void main(String[] args) {
04     boolean[][] J = {{ false, false,  true, true, true },
05                      { false, false, false, true, false },
06                      { false, false, false, true, false },
07                      { false, false, false, true, false },
08                      { false, false, false, true, false },
09                      { true,  false, false, true, false },
10                      { false,  true,  true, false, false } };
11     DotMatrixLED led = new DotMatrixLED(5, 7);
12     led.setColor(DWindow.GREEN);
13     led.setWidth(100);
14     led.setDots(J);
15     DWindow w = new DWindow(200, 150);
16     w.setBackground(DWindow.BLACK);
17     led.draw(w, 50, 5);
18   }
19 }
```



แสดง led มุมซ้ายบนอยู่ที่ (50,5)

รหัสที่ 9-4 ตัวอย่างการใช้งานคลาส `DotMatrixLED`

ก่อนจะเขียนตัวคลาส มาลองทำความเข้าใจการใช้งานก่อน รหัสที่ 9-4 แสดงตัวอย่างการสร้างอ็อบเจกต์ของ `DotMatrixLED` ในบรรทัดที่ 11 โดยต้องระบุขนาดของแอลอีดีให้กับตัวสร้าง

จากนั้นใช้เมท็อด setColor เพื่อตั้งสีของแอลอีดีตอนสว่าง ใช้เมท็อด setWidth เพื่อตั้งความกว้าง (หน่วยเป็นจุดภาพ) แล้วใช้เมท็อด setDots ซึ่งรับอาเรย์สองมิติแบบ boolean ที่ระบุว่าให้หลอดดวงใดสว่างดวงใดมืด ตามด้วยการสร้างวินโดว์แล้วเรียก draw เพื่อแสดงแอลอีดีบนวินโดว์ โดย draw รับพิกัดบนของวินโดว์ที่ให้แสดงมุมซ้ายบนของตัวแอลอีดี

```
01 import jlab.graphics.DWindow;
02
03 public class DotMatrixLED {
04   private boolean[][] dots;
05   private double width;
06   private int color;
07
08   public DotMatrixLED(int w, int h) {
09     dots = new boolean[h][w];
10     color = DWindow.GREEN;
11     width = 50;
12   }
13   public void setDots(boolean[][] s) {
14     if (s.length == dots.length && s[0].length == dots[0].length) {
15       for (int i = 0; i < s.length; i++)
16         for (int j = 0; j < s[0].length; j++)
17           dots[i][j] = s[i][j];
18     }
19   }
20   public void setWidth(double w) {
21     if (w >= dots[0].length) width = w;
22   }
23   public int getWidth() {
24     return (int) Math.round(width);
25   }
26   public void setColor(int c) {
27     color = c;
28   }
29   public int getColor() {
30     return color;
31   }
32   public void draw(DWindow w, double x, double y) {
33     double wd = width / dots[0].length;
34     double xc = x + wd / 2;
35     double yc = y + wd / 2;
36     for (int i = 0; i < dots.length; i++)
37       for (int j = 0; j < dots[0].length; j++)
38         if (dots[i][j])
39           w.fillEllipse(color, xc + j*wd, yc + i*wd, wd, wd);
40   }
41 }
```

ทำสำเนาค่าในอาเรย์ที่ได้รับมาเก็บในอาเรย์ของเรา

ความกว้าง w ต้องไม่น้อยกว่าจำนวนจุดในหนึ่งแถว เพราะ LED หนึ่งดวง ต้องใช้อย่างน้อย 1 จุดภาพ

แสดงแอลอีดีให้มุมซ้ายบนอยู่ที่พิกัด (x,y) ของวินโดว์ w

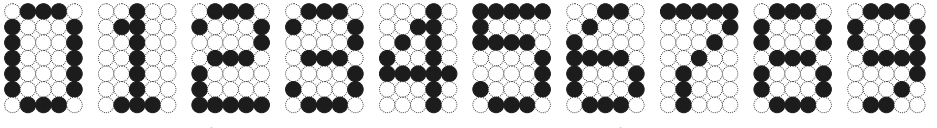
แสดงเฉพาะจุดที่ dots[i][j] มีค่าเป็นจริง

รหัสที่ 9-5 คลาส DotMatrixLED

ภายในคลาส DotMatrixLED ต้องมีอาเรย์สองมิติชื่อ dots แบบ boolean แทนแอลอีดีแต่ละจุด มีตัวแปร color เก็บสี และตัวแปร width เก็บความกว้างของแผงแอลอีดี (ไม่ต้องเก็บความสูงเพราะคำนวณได้จากขนาดของแต่ละจุดของแอลอีดีและจำนวนแถวของแผงแอลอีดี) เขียนเป็นคลาสได้ดังรหัสที่ 9-5 ให้สังเกตว่า setDots ใช้วงวนสองวงซ้อนกันเพื่อทำสำเนาค่า boolean ในแต่ละช่องของอาเรย์สองมิติที่รับมาเก็บในอาเรย์ dots ที่ได้สร้างเตรียมไว้ในตัวสร้าง โดย setDots จะไม่ใช้แค่คำสั่ง dots = s; ทั้งนี้เพราะคำสั่ง dots = s; คือการให้ตัวแปร dots อ้างอิงอาเรย์เดียวกับของผู้เรียก ซึ่งหากผู้เรียกเปลี่ยนค่าอาเรย์ของตัวเองหลังเรียก setDots จะส่งผลกระทบมายัง dots ของเราด้วย เพื่อป้องกันผลข้างเคียงดังกล่าว จึงเลือกใช้วิธีทำสำเนาแทน สำหรับเมท็อด draw จะไล่วาดทีละแถว ๆ คำนวณจุดศูนย์กลางของแอลอีดีแต่ละจุด แล้วใช้เมท็อด fillEllipse ของ DWindow วาด

แอลอีดีแสดงตัวเลข

เราสามารถใช้ DotMatrixLED เพื่อแสดงตัวอักษร ตัวเลข หรือสัญลักษณ์อะไรก็ได้ ตามค่าของอาเรย์ที่บังคับในแอลอีดีแต่ละจุดสว่างหรือดับได้ตามต้องการ แต่ถ้าเราต้องการใช้แอลอีดีแบบจุดนี้เพื่อแสดงเฉพาะตัวเลขอยู่บ่อย ๆ ก็ควรมีคลาสใหม่ที่อำนวยความสะดวกในการใช้งานจะขอเขียนคลาสใหม่ให้เป็นคลาสลูกของ DotMatrixLED ชื่อ DecimalLED ที่แสดงได้เฉพาะเลขโดด 0 ถึง 9 โดยผู้ใช้ระบุค่าที่ต้องการแสดงด้วย int ได้เลย ไม่ต้องให้ค่าเป็นอาเรย์สองมิติ รูปที่ 9-3 แสดงลักษณะของเลขโดดต่าง ๆ ที่แสดงได้จากอ็อบเจกต์ของคลาสนี้



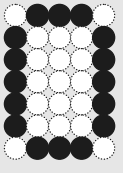
รูปที่ 9-3 การใช้แอลอีดีแบบจุดขนาด 5×7 เพื่อแสดงตัวเลข

รหัสที่ 9-6 แสดงรายละเอียดของคลาส DecimalLED มีเมท็อดหลักคือ setDigit ให้บริการตั้งเลขโดดที่ต้องการแสดง มีตัวแปรประจำคลาสซึ่งเป็นอาเรย์สองมิติอยู่ 10 ตัว เพื่อเก็บลักษณะการแสดงเลขโดด 0 ถึง 9 (รหัสที่ 9-6 แสดงเพียงตัวเดียวคือของเลข 0) จากนั้นนำอาเรย์ทั้งสิบตัวนี้ไปเก็บใส่ตัวแปรประจำคลาสซึ่งเป็นอาเรย์ชื่อ dots (อาเรย์นี้จึงเป็นอาเรย์สามมิติ) [2] ดังนั้น setDigit(n) จึงทำเพียงแค่เรียก super.setDots(dots[n]) ของคลาสแม่ให้ตั้งการสว่างและดับของแอลอีดีแต่ละจุดตามค่าใน dots[n] นอกจากนี้มีตัวแปรประจำอ็อบเจกต์ชื่อ digit เพื่อเก็บค่าของเลขโดดที่ถูกสั่งให้แสดงใน setDigit ด้วย เพื่อให้ผู้ใช้สอบถามค่าของเลขโดดที่แสดงอยู่ได้ด้วยเมท็อด getDigit

[2] การที่เราให้ตัวแปร zero, one, ..., nine และ dots เป็นตัวแปรประจำคลาส ทำให้มีตัวแปรเหล่านี้อยู่เพียงชุดเดียวประจำคลาส สามารถใช้ร่วมกันได้ระหว่างอ็อบเจกต์ทั้งหลายของ DecimalLED

ด้วยการเป็นคลาสลูกของ DotMatrixLED ทำให้สามารถให้บริการเมท็อดที่รับทอดมาซึ่งคือ setWidth, getWidth, setColor, getColor, draw, และ setDots ได้ด้วย แต่ถ้าพิจารณาให้ดี จะพบว่า DecimalLED ไม่ควรให้บริการ setDots เพราะจะทำให้เสียบทบาทของการแสดงเฉพาะเลขโดดตามข้อกำหนด แต่คลาสลูกต้องรับมรดกทั้งหลายที่คลาสแม่ให้ จะปฏิเสธไม่ได้ แล้วจะทำอย่างไร ? วิธีหนึ่งที่ทำได้คือ การเขียนกำชับในข้อกำหนดการใช้คลาสว่าห้ามเรียก setDots จากนั้นเขียนเมท็อด setDots ในคลาสลูกนี้ให้โยนสิ่งผิดปกติทันที เมื่อมีใครมาเรียกใช้ให้ทำงาน (ก็เขียนเตือนไว้แล้วว่าอย่าเรียก ยังมาเรียกอีก !) โดยทั่วไป มักโยนสิ่งผิดปกติที่ชื่อว่า UnsupportedOperationException ดังแสดงในรหัสที่ 9-6 อนึ่งให้สังเกตว่า การเรียก setDots ใน setDigit ต้องเขียนคำว่า super. กำกับ เพื่อระบุว่าจะเรียกของคลาสแม่ หากเขียน setDots เฉย ๆ หมายถึง this.setDots จะเกิดสิ่งผิดปกติทันที

```
import jlab.graphics.DWindow;
public class DecimalLED extends DotMatrixLED {
  private static final boolean[][] zero =
      { { false, true, true, true, false },
        { true, false, false, false, true },
        { true, false, false, false, true },
        { true, false, false, false, true },
        { true, false, false, false, true },
        { true, false, false, false, true },
        { false, true, true, true, false } };
  //
  // ... ขอละไม่แสดงรายละเอียดของเลข  one, two, ..., nine
  //
  private static final boolean[][][] dots =
    { zero, one, two, three, four, five, six, seven, eight, nine };
  private int digit;

  public DecimalLED() {
    super(5,7);
    setDigit(0);
  }
  public void setDigit(int n) {
    if (n < 0 || n > 9) throw new IllegalArgumentException("" + n);
    digit = n;
    super.setDots(dots[digit]);
  }
  public int getDigit() {
    return digit;
  }
  public void setDots(boolean[][] s) {
    throw new UnsupportedOperationException();
  }
}
```



ต้องเขียน super. เพราะต้องการเรียกของคลาส DotMatrixLED

DecimalLED ไม่ต้องการให้บริการ setDots ถ้าฝืนเรียก จะเกิดสิ่งผิดปกติ

รหัสที่ 9-6 คลาส DecimalLED

แผงแอลอีดีแสดงจำนวนเต็ม

รหัสที่ 9-7 แสดงคลาส `IntegerLEDWindow` เพื่อผลิตวินโดว์ที่แทนแผงแอลอีดีสำหรับแสดงจำนวนเต็ม โดยสร้างให้คลาส `IntegerLEDWindow` รับทอดมาจากคลาส `DWindow` ภายในอาศัยการนำอ็อบเจกต์ของคลาส `DecimalLED` หลาย ๆ หลักมาประกอบกันเพื่อแสดงจำนวนเต็มในวินโดว์

เราเพิ่มเมท็อด `setValue` และ `getValue` เพื่อตั้งจำนวนเต็มที่ต้องการแสดง และคืนจำนวนเต็มที่กำลังแสดง มีตัวสร้างหนึ่งตัวที่รับจำนวนหลัก และความกว้างของเลขโดดหนึ่งหลักในวินโดว์ บรรทัดแรกของตัวสร้างเรียกตัวสร้างของ `DWindow` เพื่อระบุความกว้างและความสูงของวินโดว์ ซึ่งคำนวณได้จากจำนวนหลักและความกว้างของเลขหนึ่งหลัก (บรรทัดที่ 9) จากนั้นสร้างอาเรย์มีขนาดเท่ากับจำนวนหลัก แล้วสร้างแอลอีดีของเลขแต่ละหลักเก็บใส่อาเรย์แต่ละช่อง (บรรทัดที่ 13) สำหรับเมท็อด `setValue` ก็เพียงใช้วงวนดึงเลขทีละหลัก (เริ่มที่หลักหน่วย) ของค่าที่ได้รับมา เพื่อตั้งค่าให้กับแอลอีดีทีละตัวตามด้วยการสั่งแสดงผล (บรรทัดที่ 24 และ 25)

```
01 import jlab.graphics.DWindow;
02
03 public class IntegerLEDWindow extends DWindow {
04   private DecimalLED[] digits;
05   private int value;
06   private double digitWidth;
07
08   public IntegerLEDWindow(int numDigits, double digitWidth) {
09     super(numDigits * digitWidth, digitWidth * 7 / 5);
10     super.setBackground(super.BLACK);
11     digits = new DecimalLED[numDigits];
12     for (int i = 0; i < digits.length; i++) {
13       digits[i] = new DecimalLED();
14       digits[i].setWidth(digitWidth * 0.8);
15     }
16     this.digitWidth = digitWidth;
17   }
18   public int getValue() {
19     return value;
20   }
21   public void setValue(int d) {
22     value = d;
23     for (int i = digits.length - 1; i >= 0; i--) {
24       digits[i].setDigit(d % 10);
25       digits[i].draw(this, (i + 0.1) * digitWidth, 10);
26       d = d / 10;
27     }
28   }
29 }
```

เรียกตัวสร้างของ DWindow

คูณ 0.8 เพราะต้องการให้ขนาดของแอลอีดีเล็กกว่าที่ต้องการ เพื่อแสดงช่องว่างซ้ายขวาข้างละ 10%

รหัสที่ 9-7 คลาส `IntegerLEDWindow`

รหัสที่ 9-8 แสดงตัวอย่างการสร้างอ็อบเจกต์ของ IntegerLEDWindow เพื่อแสดงแผงตัวเลขขนาด 4 หลักที่นับค่าเพิ่มขึ้นเรื่อย ๆ ทุก ๆ 100 มิลลิวินาที คล้าย ๆ กับเป็นนาฬิกาจับเวลา


```
01 import jlab.graphics.DWindow;
02
03 public class Counter {
04   public static void main(String[] args) {
05     IntegerLEDWindow c = new IntegerLEDWindow(4, 50);
06     c.setValue(0);
07     c.setRepaintDuringSleep(true);
08     while (true) {
09       c.clearBackground();
10       c.setValue((c.getValue()+1)%1000);
11       c.sleep(100);
12     }
13   }
14 }
```

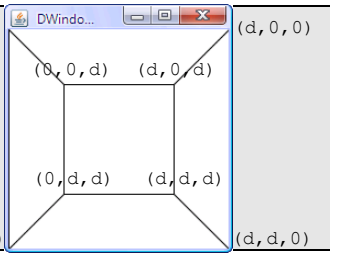
รหัสที่ 9-8 โปรแกรมแสดงตัวนับที่ใช้อ็อบเจกต์ของ IntegerLEDWindow

DWindow3D ที่มีความลึก

DWindow เป็นคลาสที่มีไว้ผลิตวินโดว์เพื่อใช้แสดงภาพในระนาบสองมิติ การสั่งให้ลากเส้น วาดวงรี วาดสี่เหลี่ยม บนวินโดว์ต้องกำหนดพิกัด (x, y) บนระนาบ หัวข้อนี้นำเสนอการสร้างคลาสใหม่ชื่อ DWindow3D ที่เพิ่มความลึก เสมือนเป็นห้องทรงสี่เหลี่ยมมุมฉาก เพื่อให้ผู้ใช้วาดรูปเรขาคณิตสองมิติอย่างง่าย ได้แก่ การลากเส้น วาดสี่เหลี่ยมและวงรี ในห้องสามมิติที่กำหนดพิกัดเป็น (x, y, z)

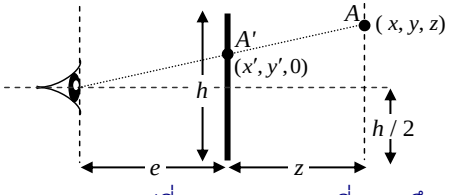
ความลึกของห้องทรงสี่เหลี่ยมมุมฉากเป็นเรื่องที่เราจินตนการอยากให้มี เมื่อต้องแสดงวัตถุในห้องนี้บนจอภาพก็ต้องแปลงให้อยู่บนระนาบสองมิติ ดังนั้น ภาระที่ต้องคิดก่อนเขียนคลาส คือ จุด (x, y, z) ในห้องทรงสี่เหลี่ยมมุมฉาก จะปรากฏเป็นจุดที่ตำแหน่งใดบนระนาบสี่เหลี่ยมมุมฉาก (ซึ่งคือวินโดว์ที่เรามองเห็น) รหัสที่ 9-9 แสดงตัวอย่างการใช้งานคลาส DWindow3D เริ่มด้วยการสร้างห้องที่ต้องกำหนดทั้งความกว้าง ความสูง และความลึก ในที่นี้เสมือนสร้างห้องทรงลูกบาศก์ขนาด dxdxd จากนั้นสั่งให้วาดสี่เหลี่ยมผืนผ้าขนานกับจอภาพขนาด dxd อยู่ลึกเข้าไป d เสมือนเป็นผนังหลังห้องด้วยคำสั่ง drawRect ที่รับพิกัดของมุมฉากสองมุมที่อยู่ทแยงกันของสี่เหลี่ยมผืนผ้าที่ต้องการวาด ในตัวอย่างคือ จุด (0,0,d) กับ (d,d,d) เพื่อให้ผู้มองรู้สึกว่าผนังอยู่ลึก คือ อยู่ไกลจากผู้มอง จะใช้วิธีแสดงภาพให้เล็กลงเป็นสัดส่วนกับความลึก จากนั้นวาดสี่เหลี่ยมอีกสองรูปตั้งฉากกับจอภาพ เสมือนเป็นผนังทางซ้ายและขวาของห้อง เนื่องจากเราตั้งสมมติฐานว่า ตาผู้มองอยู่ตรงกับจุดศูนย์กลางของจอภาพ และเพื่อให้แลดูเป็นรูปสามมิติจึงให้ผนังนั้นเล็กลงและลู่เข้าหาจุดศูนย์กลางของภาพ จึงแสดงผนังทั้งสองนี้เป็นรูปสี่เหลี่ยมคางหมู

```
int d = 200;
DWindow3D w3 = new DWindow3D(d, d, d);
w3.drawRect(0, 0, d, d, d, d);
w3.drawRect(0, 0, 0, 0, d, d);
w3.drawRect(d, 0, d, d, d, 0);
```



รหัสที่ 9-9 ตัวอย่างการใช้งานคลาส DWindow3D

รูปที่ 9-4 แสดงตาของผู้ใช้ที่มองตรงเข้าหาจอคอมพิวเตอร์ เส้นทึบตรงกลางรูปแทนจอภาพ กำหนดให้ตาอยู่ห่างจากจอภาพเป็นระยะทาง e ให้วัตถุที่อยู่ติดขอบจอภาพอยู่ที่ความลึกเป็น 0 (นั่นคือ $z = 0$) และให้วินโดว์มีความสูง h และกว้าง w จะได้ว่า จุด A ที่มีพิกัด (x, y, z) คือ อยู่ลึกเข้าไป z จะปรากฏบนจอภาพที่พิกัด (x', y') ตามสูตรที่แสดงทางขวามือ ในกรณีที่วาดห้องทรงสี่เหลี่ยมมุมฉาก มุมของผนังทางซ้ายและขวาจะลู่เข้ามากหรือน้อย ก็ขึ้นกับค่า e ที่เราปรับได้



$$x' = \frac{w}{2} + \left(x - \frac{w}{2}\right)\frac{e}{e+z}$$

$$y' = \frac{h}{2} + \left(y - \frac{h}{2}\right)\frac{e}{e+z}$$

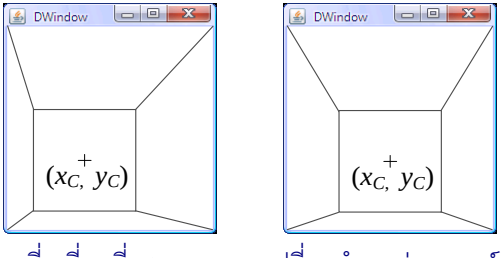
รูปที่ 9-4 การฉายจุดที่ความลึก z มาที่จอภาพ

ถ้าให้ตาของผู้มองเปลี่ยนตำแหน่ง (แต่ยังคงมองตรงไปยังจอภาพ) เช่นมองตรงไปที่ (x_C, y_C) บนวินโดว์ จะได้สูตรการเปลี่ยนจุด (x, y, z) ในห้องกล่องสี่เหลี่ยมมุมฉาก มาเป็นจุดที่พิกัด (x', y') บนจอภาพในวินโดว์ ดังนี้

$$x' = x_C + \left(x - x_C\right)\frac{e}{e+z}$$

$$y' = y_C + \left(y - y_C\right)\frac{e}{e+z}$$

ถ้าเราเปลี่ยนจุดศูนย์กลางของการมองนี้ไปมา ภาพที่ปรากฏบนวินโดว์ก็จะเปลี่ยนไปมา ทำให้ผู้มองรับรู้ถึงความลึกของภาพได้ดีขึ้น ดังตัวอย่างในรูปที่ 9-5



รูปที่ 9-5 รูปห้องทรงสี่เหลี่ยมที่แสดงจากการเปลี่ยนตำแหน่งจุดศูนย์กลางของการมอง

```
01 import jlab.graphics.*;
02
03 public class DWindow3D extends DWindow {
04   private double depth;
05   private double xc, yc;
06
07   public DWindow3D(double w, double h, double d) {
08     super(w, h);
09     depth = d;
10     xc = w/2; yc = h/2;
11   }
12   public double getDepth() {
13     return depth;
14   }
15   public void drawLine(double x0, double y0, double z0,
16                        double x1, double y1, double z1) {
17     super.drawLine(toSX(x0, z0), toSY(y0, z0),
18                    toSX(x1, z1), toSY(y1, z1));
19   }
20   public void drawRect(double x0, double y0, double z0,
21                        double x1, double y1, double z1) {
22     drawLine(x0, y0, z0, x0, y1, z0);
23     drawLine(x0, y0, z0, x1, y0, z1);
24     drawLine(x1, y1, z1, x1, y0, z1);
25     drawLine(x1, y1, z1, x0, y1, z0);
26   }
27   public void fillEllipse(int c, double x, double y, double z,
28                           double a, double b) {
29     double sx = toSX(x, z);
30     double sy = toSY(y, z);
31     super.fillEllipse(c, sx, sy, scale(a, z), scale(b, z));
32   }
33   public void drawFrame(double step) {
34     double w = getWidth(), h = getHeight();
35     for (int z = 0; z <= depth; z += step)
36       drawRect(0, 0, z, w, h, z);
37     drawRect(0, 0, 0, 0, h, depth);
38     drawRect(w, 0, 0, w, h, depth);
39   }
40   private double toSX(double x, double z) {
41     return xc + scale(x - xc, z);
42   }
43   private double toSY(double y, double z) {
44     return yc + scale(y - yc, z);
45   }
46   private double scale(double v, double z) {
47     return v * 200 / (200 + z);  // ให้ e = 200
48   }
```

ปรับขนาดตามความลึก

$$x' = x_C + (x - x_C)\frac{e}{e+z}$$

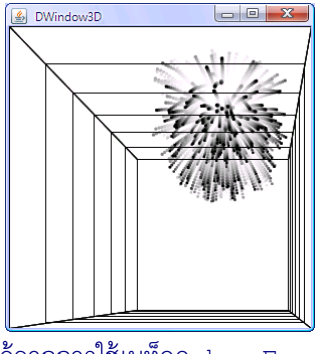
$$y' = y_C + (y - y_C)\frac{e}{e+z}$$

รหัสที่ 9-10 คลาส DWindow3D ขยายความสามารถจากคลาส DWindow

รหัสที่ 9-10 แสดงคลาส `DWindow3D` ซึ่งขยายลักษณะมาจากคลาส `DWindow` ที่เราใช้กันมา โดยเราไม่ต้องมีรหัสต้นฉบับของ `DWindow` และไม่รู้เลยด้วยว่า `DWindow` มีรายละเอียดการทำงานภายในคลาสอย่างไร รู้เพียงแต่ว่า คลาสนี้มีอะไรให้เราใช้บ้างก็พอ การสร้างคลาสใหม่ให้เป็นคลาสลูกของ `DWindow` ทำให้คลาสใหม่นี้สามารถให้บริการต่างๆ ที่ `DWindow` มีให้ได้ทั้งหมด สิ่งที่ต้องเขียนเพิ่มคือภาระการบริการที่เกี่ยวข้องกับการวาดรูปในห้องสามมิติเท่านั้น

ตัวแปรประจำอ็อบเจกต์ที่ต้องเพิ่มในคลาสใหม่นี้ คือ `depth` แทนความลึก และ `xc` กับ `yc` แทนจุดของวินโดว์ที่ตรงกับตาผู้มอง (บรรทัดที่ 4 และ 5) เราเขียนตัวสร้างไว้แบบเดียวที่รับความกว้าง ความสูง และความลึกของห้องทรงสี่เหลี่ยมมุมฉาก เริ่มด้วยการเรียกตัวสร้างของคลาสแม่โดยส่งเฉพาะความกว้างและความสูงไป (บรรทัดที่ 8) ส่วนความลึกเก็บไว้ที่ตัวเอง (บรรทัดที่ 9) ปิดท้ายด้วยการตั้ง `xc` และ `yc` ที่ตำแหน่งตรงกลางวินโดว์

เราเพิ่มบริการลากเส้นตรง (`drawLine`) วาดสี่เหลี่ยมผืนผ้า (`drawRect`) และวาดวงรี (`fillEllipse`) ให้กับคลาสใหม่นี้ ซึ่งต้องแปลงพิกัด (x, y, z) ของรูปที่จะวาดเป็นพิกัด (x', y') บนวินโดว์ จึงเขียนเป็นเมท็อดใช้ส่วนตัวชื่อ `toSX(x,z)` และ `toSY(y,z)` (บรรทัดที่ 40 ถึง 45) ที่ทำตามสูตรที่ได้เขียนมาก่อนหน้านี้ โดยเขียนเมท็อดเสริมชื่อ `scale` ที่ถูกใช้ใน `toSX` และ `toSY` นอกจากนี้ยังสามารถใช้ `scale` ในการปรับขนาดของวัตถุตามความลึกได้อีกด้วย เช่น เราเห็นวงกลมที่อยู่ลึก (ไกล) มีขนาด (รัศมี) เล็กลง เป็นต้น `toSX` และ `toSY` ช่วยให้เขียนเมท็อดลากเส้น วาดรูปได้ง่ายขึ้น เมท็อด `drawLine` เพียงแค่เปลี่ยนพิกัดของจุดปลายทั้งสองในห้อง ไปเป็นจุดปลายบนระนาบ แล้วใช้ `drawLine` ของคลาสแม่ลากเส้นบนระนาบ (บรรทัดที่ 17 และ 18) เมท็อด `drawRect` เรียกใช้ `drawLine` แบบสามมิติเพื่อลากเส้นขอบของสี่เหลี่ยมทั้งสี่เส้น (บรรทัดที่ 22 ถึง 25) ส่วนเมท็อด `fillEllipse` แปลงพิกัดของจุดศูนย์กลาง ปรับขนาดกว้างและสูงของวงรีที่เห็นบนระนาบ แล้วสั่งให้ `fillEllipse` ของคลาสแม่วาดวงรีให้ (บรรทัดที่ 29 ถึง 31) นอกจากนี้ยังแถมเมท็อด `drawFrame` ให้อีกหนึ่งเมท็อดที่แสดงผนังห้องทรงสี่เหลี่ยมที่สร้าง และเส้นกรอบตามผนัง เพื่อให้ผู้มองรับรู้ความลึกได้ง่ายขึ้น ดังตัวอย่างในรูปที่ 9-6



รูปที่ 9-6 เส้นขอบตามผนังที่ได้จากการใช้เมท็อด `drawFrame` เพื่อให้รับรู้ความลึกได้ดีขึ้น

ผู้อ่านอาจสงสัยว่า เราจะเปลี่ยน xc และ yc เพื่อเปลี่ยนมุมมองได้อย่างไร ขอนำเสนอวิธีง่าย ๆ ด้วยการนำตำแหน่งของตัวชี้เมาส์มาเป็นค่าของ xc และ yc โดยจะเปลี่ยนเมื่อมีการกดปุ่มแล้วลากเมาส์เท่านั้น DWindow มีอีกบริการที่ไม่เคยนำเสนอมาก่อน คือ ตัวอ็อบเจกต์วินโดว์นี้จะเรียกเมท็อดที่ชื่อว่า onMouseDragged อย่างอัตโนมัติเมื่อได้รับสัญญาณการลากเมาส์จากระบบ โดยเมท็อดนี้มีเขียนในคลาส DWindow ไว้แล้ว แต่ไม่ได้ทำอะไร ดังนั้น ถ้าเราต้องการให้ทำอะไร ก็แค่เขียนเมท็อดชื่อเดียวกันนี้ที่คลาสเราเพื่อแทนเมท็อดของคลาสแม่ ดังแสดงในรหัสที่ 9-11 (ที่เขียนต่อจากรหัสที่ 9-10) เมท็อดนี้รับพารามิเตอร์ซึ่งเป็นอ็อบเจกต์ของคลาส DPoint ที่มีเมท็อด getX() และ getY() เพื่อขอพิกัดของจุดที่ได้รับ ซึ่งแทนตำแหน่งของตัวชี้เมาส์บนวินโดว์ มาตั้งให้กับ xc และ yc เพื่อเปลี่ยนมุมมองของผู้ใช้

```
03 public class DWindow3D extends DWindow {
..   ...
49   public void onMouseDragged(DPoint p) {
50     this.xc = p.getX();
51     this.yc = p.getY();
52   }
```

เมื่อมีการลากเมาส์บนวินโดว์ ระบบจะขัดจังหวะโปรแกรมที่ทำอยู่ และกระโดดมาทำที่เมท็อดนี้ ทำเสร็จก็กลับไปทำงานที่ถูกขัดจังหวะตอนต้นต่อ

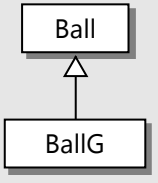
รหัสที่ 9-11 การเพิ่มเมท็อดที่ปรับจุดศูนย์กลางการมองของผู้ใช้ใน DWindow3D

ลูกบอลเด้งในห้อง

เพื่อให้เห็นการใช้งาน DWindow3D จะขอเขียนคลาสที่ผลิตลูกบอลอีกแบบที่เด้งไปมาได้ในห้องสามมิตินี้ ให้ชื่อว่า Ball3D แต่ก่อนจะเขียนคลาสนี้ ขอเขียนคลาส BallG ซึ่งผลิตลูกบอลเคลื่อนบนระนาบสองมิติที่มีพฤติกรรมเคลื่อนตามแนวดิ่งแบบมีความเร่งเสมือนได้รับแรงโน้มถ่วงของโลก รหัสที่ 9-12 แสดงรายละเอียดของคลาส BallG ซึ่งเป็นคลาสลูกของ Ball มีตัวแปร gravity เก็บความเร่ง มีเมท็อดตั้งค่ากับคืนค่า gravity และที่สำคัญคือ เขียนเมท็อด move ใหม่ให้แทนของคลาสแม่ บรรทัดที่ 16 สั่งปรับค่าความเร็วตาม gravity เสมือนมีความเร่งนั่นเอง (หรืออาจเป็นการหน่วงก็ได้ถ้า dy ติดลบ เมื่อกำลังเคลื่อนที่ขึ้น) จากนั้นจึงสั่งให้ลูกบอลเลื่อน โดยเรียกเมท็อด move ของคลาสแม่ เพียงเท่านี้ก็ได้ลูกบอลเด้งแบบมีแรงโน้มถ่วง ผู้อ่านควรลองเขียนโปรแกรมสร้างลูกบอลแบบ BallG แล้วสั่งเคลื่อนไปมาดูว่า มีพฤติกรรมเป็นดังคาดหรือไม่

คราวนี้มาเขียนคลาส Ball3D ให้เป็นคลาสลูกของ BallG (เป็นหลานของ Ball โดยอัตโนมัติ) ดังรหัสที่ 9-13 เนื่องจากเป็นลูกบอลในห้องสามมิติ ก็ต้องเพิ่มตัวแปรเก็บพิกัด z และ dz เก็บความเร็วในแนวลึก เพิ่มเมท็อด draw และ move (รับวินโดว์แบบ DWindow3D) เมท็อด draw สั่งให้วินโดว์วาดวงกลมในสามมิติ (บรรทัดที่ 12) ส่วน move สั่งให้ลูกบอลเคลื่อนในสองมิติก่อน โดยเรียก move ของคลาสแม่ (บรรทัดที่ 15) แล้วจึงปรับค่าของ z (บรรทัด 16 ถึง 18) ซึ่งทำในทำนองเดียวกับที่ปรับพิกัด x และ y ในคลาส Ball ให้สังเกตว่า เมท็อด move และ draw ที่เขียนนี้ไม่ได้แทนเมท็อดของคลาสแม่ เพราะมีหัวเมท็อดไม่เหมือนกัน (ของคลาสแม่นั้นรับพารา-

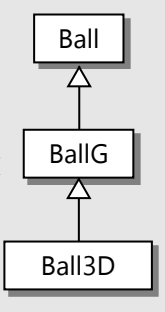
มิเตอร์เป็น DWindow ไม่ใช่ DWindow3D) จึงถือว่า เป็นการเพิ่มอีกสองเมท็อด (ที่มีชื่อเหมือนกับของคลาสแม่) รหัสที่ 9-14 และรูปที่ 9-7 แสดงตัวอย่างการใช้ Ball3D เพื่อสร้างลูกบอลเคลื่อนที่ใน DWindow3D




```
01 import jlab.graphics.DWindow;
02
03 public class BallG extends Ball {
04   private double gravity = 2;
05
06   public BallG(double r, double x, double y) {
07     super(r, x, y);
08   }
09   public double getGravity() {
10     return gravity;
11   }
12   public void setGravity(double g) {
13     gravity = g;
14   }
15   public void move(DWindow w) {
16     dy += gravity;
17     super.move(w);
18   }
19 }
```

ปรับความเร็ว ก็คือมีความเร่ง

รหัสที่ 9-12 คลาส BallG คือ Ball ที่เด้งเสมือนมีแรงโน้มถ่วง




```
01 import jlab.graphics.DWindow;
02
03 public class Ball3D extends BallG {
04   public double z, dz;
05
06   public Ball3D(double r, double x, double y, double z) {
07     super(r, x, y);
08     this.z = z;
09     dz = -5 + (10 * Math.random());
10   }
11   public void draw(DWindow3D w3) {
12     w3.fillEllipse(DWindow.BLACK, x, y, z, 2*r, 2*r);
13   }
14   public void move(DWindow3D w3) {
15     super.move(w3);
16     z += dz;
17     if (z <= 0 || z >= w3.getDepth()) dz = -dz;
18     z = Math.min(Math.max(z, 0), w3.getDepth());
19   }
20 }
```

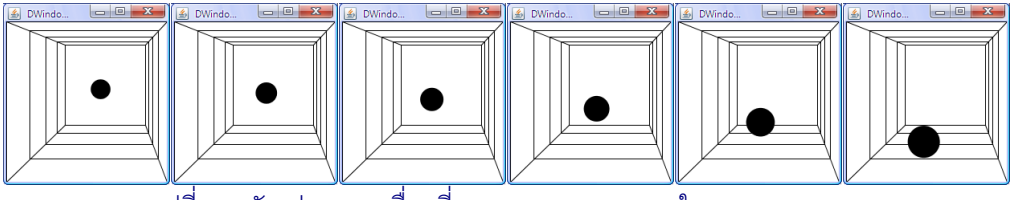
แสดงวงกลมในห้องสามมิติ w3

ตรวจสอบการชนผนังหน้าและหลัง

รหัสที่ 9-13 คลาส Ball3D คือ BallG ที่เด้งในห้อง

```
01 public class TestBall3D {
02   public static void main(String[] args) {
03     DWindow3D w3 = new DWindow3D(200, 200, 200);
04     Ball3D[] balls = new Ball3D[1];  // ผู้อ่านลองสร้างหลายลูกดูเอง
05     for (int i = 0; i < balls.length; i++) {
06       balls[i] = new Ball3D(20, 100, 100, 100);
07       balls[i].setGravity(3*Math.random());
08     }
09     w3.setRepaintDuringSleep(true);
10     while (true) {
11       w3.clearBackground();
12       w3.drawFrame(50);
13       for (int i = 0; i < balls.length; i++) {
14         balls[i].move(w3);
15         balls[i].draw(w3);
16       }
17       w3.sleep(50);
18     }
19   }
20 }
```

รหัสที่ 9-14 ตัวอย่างโปรแกรมการใช้ Ball3D กับ DWindow3D



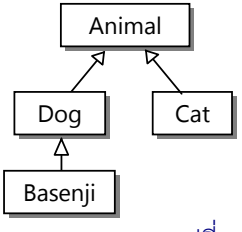
รูปที่ 9-7 ตัวอย่างการเคลื่อนที่ของลูกบอล Ball3D ใน DWindow3D

หนึ่งอ็อบเจกต์ หลายบทบาท

การสร้างคลาสด้วยการรับทอดก่อให้เกิดความสัมพันธ์ฉันแม่ลูกระหว่างคลาส และยังทำให้มองอ็อบเจกต์ของคลาสลูกเป็นประเภทข้อมูลแบบคลาสแม่ได้ รูปที่ 9-8 แสดงความสัมพันธ์ของตัวอย่างคลาสสี่คลาส แมวเป็นสัตว์ประเภทหนึ่ง สุนัขเป็นสัตว์อีกประเภทหนึ่ง และบาเซนจิเป็นสุนัขพันธุ์หนึ่ง เห็นได้ว่า คลาสแม่เป็นคลาสที่บรรยายประเภทข้อมูลที่เป็นกรณีทั่วไปกว่าคลาสลูก และคลาสลูกเป็นประเภทข้อมูลกรณีพิเศษของคลาสแม่ จึงสามารถใช้ตัวแปรอ้างอิงของคลาสแม่ไปอ้างอิงอ็อบเจกต์ของคลาสลูกได้ ตัวแปรแบบ Dog อ้างอิงอ็อบเจกต์ได้ทั้งแบบ Dog และ Basenji ในขณะที่ตัวแปรแบบ Animal อ้างอิงสัตว์ทุกชนิดในแผนภาพได้ ดังตัวอย่างรหัสที่แสดงข้างรูป สามารถเขียน Animal a1 = new Dog(); ได้ เพราะสุนัขทุกตัวเป็นสัตว์ แต่จะเขียน Dog d = new Animal(); ไม่ได้ เพราะสัตว์ทุกตัวไม่ได้เป็นสุนัข เทียบได้กับการ

เขียน `double x = 12;` ซึ่งทำได้เพราะ 12 ที่เป็นจำนวนเต็มถือว่า เป็นจำนวนจริงได้ แต่ได้ทางกลับกัน ไม่สามารถเขียน `int k = 12.99;` เพราะจำนวนจริงไม่เป็นจำนวนเต็ม

เนื่องจากการเรียกใช้บริการของเมท็อดประจำอ็อบเจกต์ต้องกระทำผ่านตัวอ้างอิงอ็อบเจกต์ ตัวแปลโปรแกรมจะอนุญาตให้เรียกใช้เมท็อดที่มีในคลาสของตัวอ้างอิงนั้นเท่านั้น เช่น ให้ `Dog` มีเมท็อด `bark` ทำให้ `Basenji` ซึ่งเป็นคลาสลูกมี `bark` ด้วย และให้ `Basenji` มีเมท็อด `hunt` ที่ `Dog` ไม่มี ดังนั้น ถ้ามีคำสั่ง `Dog d = new Basenji();` ก็เรียกใช้ `d.bark();` ได้ แต่จะเรียก `d.hunt();` ไม่ได้ (ตัวแปลโปรแกรมไม่ยอม) เพราะ `d` เป็นตัวแปรแบบ `Dog` ไม่มีบริการ `hunt` (ถึงแม้เราจะเห็นชัด ๆ ว่า `d` กำลังอ้างอิงสุนัขพันธุ์บาเซนจิอยู่ก็ตาม)




```
Cat     c1 = new Cat();
Basenji b1 = new Basenji();
Dog     d1 = new Dog();
Dog     d2 = new Basenji();
Animal  a1 = new Dog();
Animal  a2 = new Basenji();
Animal  a3 = new Cat();
```

รูปที่ 9-8 แผนภาพความสัมพันธ์ของคลาส

การเปลี่ยนบทบาท

จากที่ได้นำเสนอว่า การเขียน `Dog d = new Animal();` กระทำไม่ได้ แล้วถ้าให้ `Animal a = new Dog();` ตามด้วย `Dog d = a;` จะได้หรือไม่ เพราะเห็นชัด ๆ อยู่แล้วจากคำสั่งก่อนหน้า ที่ให้ `a` อ้างอิงอ็อบเจกต์ของสุนัข คำตอบก็คือ ไม่ได้ ตัวแปลโปรแกรมไม่ยอม เพราะถึงแม้ว่าเรา (คน) จะเห็นชัด ๆ แต่ตัวแปลเขามองไม่เห็น ในมุมของตัวแปลโปรแกรม ตัวแปร `a` อ้างอิงอ็อบเจกต์ที่เป็นสัตว์ แต่สัตว์ทุกตัวไม่ได้สุนัข ดังนั้น จะให้ `d` ที่เป็นตัวอ้างอิงสุนัขไปอ้างอิงสัตว์ที่ `a` อ้างอิงย่อมทำไม่ได้

อย่างไรก็ตาม ถ้าเราต้องการทำ `Dog d = a;` จริง ๆ เพราะมั่นใจมาก ๆ ว่าในโปรแกรมที่เขียนอยู่ สัตว์ที่ `a` อ้างอิงอยู่นั้นเป็นสุนัขจริง ๆ ก็สามารถทำได้คล้าย ๆ กับกรณีที่เราต้องการนำค่าในตัวแปร `double` มาใส่ในตัวแปร `int` ก็ต้องเขียน `(int)` นำหน้า นี่ก็เช่นกัน ต้องเขียนเป็น

```
Dog d = (Dog) a;
```

จะทำให้ผ่านด่านการตรวจสอบของตัวแปลโปรแกรม แต่เมื่อถึงตอนทำคำสั่งนี้ ระบบจะตรวจสอบขณะทำงานว่า `a` กำลังอ้างอิงอ็อบเจกต์ที่เป็นสุนัขหรือไม่ ถ้าใช่ก็ไม่มีปัญหา แต่ถ้าปรากฏว่าไม่ใช่ ระบบจะโยนสิ่งผิดปกติประเภท `ClassCastException` เช่น คำสั่งข้างล่างนี้

```
Animal a = new Cat();
Dog d = (Dog) a;
```

จะเกิดการโยนสิ่งผิดปกติดังกล่าว เพราะระบบตรวจสอบขณะทำงานพบว่า อ็อบเจกต์ที่ a อ้างอิงเป็นแมว ไม่ใช่สุนัข (ต้องขอบอกว่า เหตุการณ์เช่นนี้จะไม่เกิด ในกรณีของการเปลี่ยน double เป็น int เพราะระบบจะเปลี่ยนได้เสมอ เนื่องจากได้ตกลงกันแล้วว่า จะตัดค่าหลังจุดนิยมทิ้งเพื่อทำให้เป็นจำนวนเต็ม)

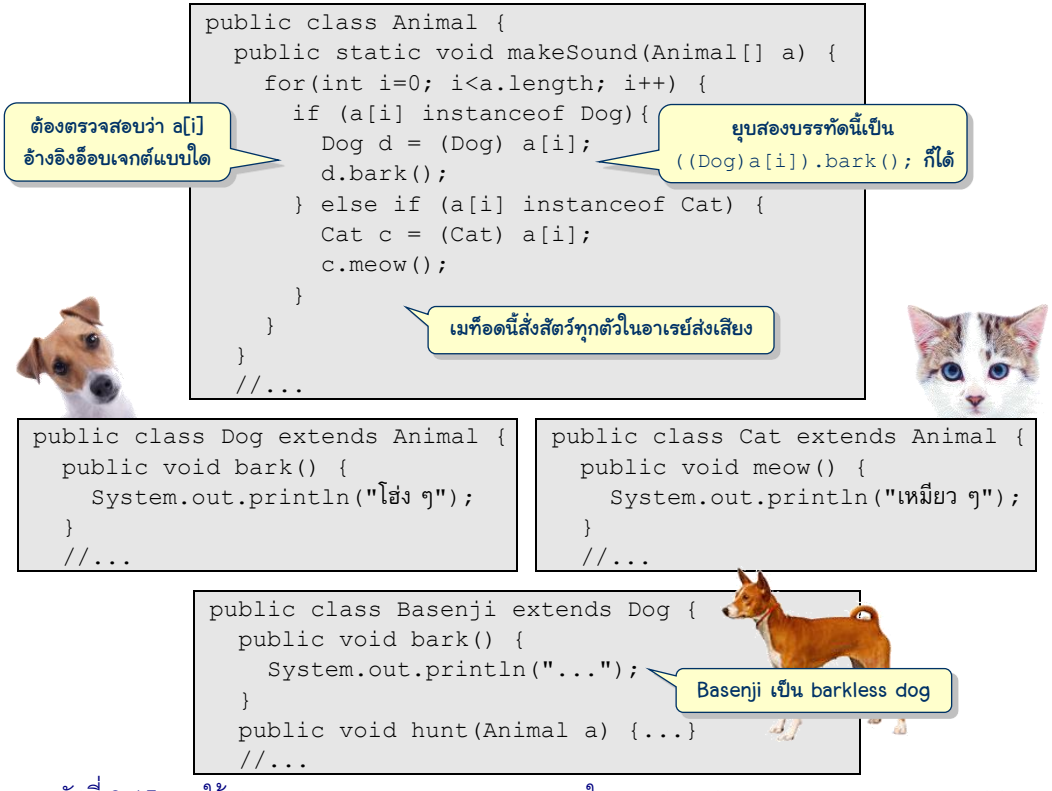
กล่าวโดยสรุป การเปลี่ยนบทบาทตัวอ้างอิงนี้ มีสองทิศทาง คือจากคลาสระดับล่างในลำดับชั้นการรับทอด ขึ้นไปเป็นคลาสระดับบน เรียกว่า upcast (เช่น จากคลาสลูกเป็นคลาสแม่) และจากคลาสระดับบนลงมาเป็นคลาสระดับล่าง เรียกว่า downcast (เช่นจากคลาสแม่เป็นคลาสลูก) แบบแรกนั้นกระทำได้เสมอ เช่น `Animal a = new Dog();` เพราะถือว่า อ็อบเจกต์ของคลาสลูกทุกตัวเปลี่ยนบทบาทไปเป็นคลาสแม่ได้หมด ในขณะที่แบบหลังจะกระทำได้ต้องกำหนดการเปลี่ยนบทบาท โดยใส่ชื่อคลาสระดับล่างไว้ภายในวงเล็บนำหน้าตัวอ้างอิงที่จะเปลี่ยน เช่น `Dog d = (Dog) a;` ซึ่งทำให้แปลผ่าน แต่ระบบจะตรวจสอบอีกครั้งเมื่อทำงานจริง

instanceof

โดยทั่วไป ก่อนที่จะเปลี่ยนบทบาทแบบ downcast มักตรวจสอบให้แน่ชัดก่อนว่า อ็อบเจกต์ที่อ้างอิงอยู่เป็นประเภทที่คาดไว้หรือไม่ ซึ่งสามารถทำได้ด้วยตัวดำเนินการ instanceof คำสั่ง `a instanceof A` คืนจริงก็ต่อเมื่ออ็อบเจกต์ที่ a อ้างอิงอยู่เป็นอ็อบเจกต์ของคลาส A หรือเป็นของคลาสที่รับทอดจาก A

มาดูตัวอย่างการใช้งาน รหัสที่ 9-15 แสดงบางส่วนของคลาส Animal, Dog, Basenji, และ Cat แมวร้อง “เหมียว ๆ” Cat จึงมีเมท็อด meow, สุนัขเห่า “โฮ่ง ๆ” Dog จึงมีเมท็อด bark, สำหรับสุนัขพันธุ์บาเซนจินั้นไม่ค่อยส่งเสียงเห่า ผู้เขียน Basenji จึงเขียน bark เองเพื่อแทนที่ของได้รับทอดจาก Dog โดยไม่ให้ส่งเสียง ประเด็นที่น่าสนใจอยู่ที่เมท็อด makeSound ในคลาส Animal ซึ่งรับพารามิเตอร์เป็นอาเรย์ของ Animal ทำให้แต่ละช่องของอาเรย์ a[i] สามารถเก็บสัตว์ที่เขียนได้ทั้งสี่แบบ ดังนั้น makeSound ต้องตรวจสอบว่า ถ้า a[i] เป็นสุนัขต้องเรียก bark ให้เห่า ถ้าเป็นแมว ต้องเรียก meow ให้ร้อง แต่เราจะเขียน `a[i].bark()` หรือ `a[i].meow()` ไม่ได้ เพราะ a[i] เป็น Animal จึงต้องเปลี่ยนเป็นแบบ Dog และแบบ Cat ก่อนเรียก bark และ meow ตามลำดับ ดังแสดงในรหัสที่ 9-15

ถึงแม้ว่า เมท็อด makeSound ของคลาส Animal ในรหัสที่ 9-15 จะทำงานได้ตามคาด แต่ไม่ค่อยดีเท่าไหร่ ลองคิดดู หาก Animal มีคลาสลูกใหม่ เช่น Tiger มีเมท็อด roar ส่งเสียงคำราม เราก็ต้องไปแก้ไขรหัสต้นฉบับของ makeSound ใน Animal ให้ตรวจสอบว่า ถ้าเป็นเสือให้คำราม เราในฐานะที่เป็นเจ้าของ Animal จะไปทราบได้อย่างไรว่า มีใครนำคลาสเราไปใช้สร้างคลาสลูก แล้วจะเขียนคลาสกันอย่างไร จึงทำให้คลาสแม่มีความคล่องตัว สามารถปรับตัวได้ตามคลาสลูกใหม่ที่เกิดขึ้นได้โดยอัตโนมัติ



รหัสที่ 9-15 การใช้ `instanceof` และ `downcast` ในคลาส `Animal`, `Dog`, และ `Basenji`

การเรียกเมท็อดประจำอ็อบเจกต์

เราสามารถผสมผสานลักษณะสมบัติของการโปรแกรมเชิงวัตถุดังต่อไปนี้ เพื่อนำเสนอแนวทางการเขียนคลาสที่กะทัดรัด และคล่องตัว

- การมองอ็อบเจกต์ของคลาสลูกว่า เป็นประเภทข้อมูลของคลาสแม่ได้ นั่นคือ สามารถใช้ตัวแปรอ้างอิงของคลาสแม่ไปอ้างอิงอ็อบเจกต์ของคลาสลูกหลานได้
- การเขียนเมท็อดของคลาสลูก เพื่อแทนเมท็อดที่รับทอดมาจากคลาสแม่
- การที่ระบบพิจารณาขณะทำงานว่า อ็อบเจกต์ที่ถูกเรียกเมท็อดเป็นอ็อบเจกต์ของคลาสใด ก็จะเรียกเมท็อดประจำอ็อบเจกต์ของคลาสนั้น

เราได้นำเสนอสองประเด็นแรกกันมาแล้ว แต่ประเด็นสุดท้ายยังไม่เคยลงในรายละเอียดมากนัก ดูตัวอย่างในรหัสที่ 9-16 ประกอบ (คล้ายกับรหัสที่ 9-15) คราวนี้เราเขียน `makeSound` อีกหนึ่งเมท็อดที่ไม่รับพารามิเตอร์ใน `Animal` จากนั้นเขียน `makeSound` ใน `Cat`, `Dog`, และ `Basenji` เพื่อแทนของที่รับทอดมา แล้วเปลี่ยน `bark` และ `meow` ให้มาเรียก `makeSound` ในคลาสของตัวเองแทน ประเด็นที่น่าสนใจอยู่ที่เมท็อด `makeSound` ที่รับอาเรย์ของสัตว์ในคลาส

Animal ซึ่งเขียนได้สั้นมาก เขียนแค่ a[i].makeSound() ก็พอ ไม่ต้องใช้ instanceof และเปลี่ยนบทบาทของตัวอ้างอิงแต่อย่างใด เราเขียนเช่นนี้ได้ เพราะ a[i] ซึ่งเป็น Animal มีเมท็อด makeSound() ตัวแปลโปรแกรมก็ถือว่าใช้ได้ไม่ฟ้องอะไร การเขียนเช่นนี้มองว่า a[i] เป็น Animal โดยไม่ต้องรู้ว่า เป็นสัตว์ชนิดใด อาจเป็นสุนัข หรือแมว หรืออื่นๆ ก็ได้ที่จะเขียนเป็นคลาสลูก ๆ ของ Animal ในอนาคต ก็จะใช้ได้ในเมท็อดนี้ทั้งสิ้น

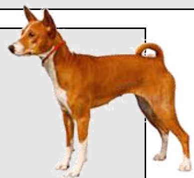
```
public class Animal {
  public void makeSound() { }
  public static void makeSound(Animal[] a) {
    for(int i=0; i<a.length; i++) {
      a[i].makeSound();
    }
  }
  //...
```

จะเรียก makeSound ตัวใด ขึ้นกับว่า a[i] คืออ็อบเจกต์ของคลาสใด

```
public class Dog extends Animal {
  public void bark() {
    makeSound();
  }
  public void makeSound() {
    System.out.println("โฮ่ง ๆ");
  }
  //...
```

```
public class Cat extends Animal {
  public void meow() {
    makeSound();
  }
  public void makeSound() {
    System.out.println("เหมียว ๆ");
  }
  //...
```

```
public class Basenji extends Dog {
  public void makeSound() {
    System.out.println("...");
  }
  public void hunt(Animal a) {...}
  //...
```



รหัสที่ 9-16 คลาส Animal, Dog, และ Basenji

ผู้อ่านคงสงสัยว่า ในเมื่อ a[i] เป็น Animal คำสั่ง a[i].makeSound() ก็น่าจะเรียก makeSound() ของคลาส Animal แล้วสุนัขที่ถูกเก็บอยู่ในอาเรย์ a จะร้อง "โฮ่ง ๆ" ได้อย่างไร คำตอบก็คือ ระบบจะไม่ทำตรงไปตรงมาอย่างที่กล่าว แต่ระบบจะพิจารณาขณะทำงานว่า a[i] กำลังอ้างอิงอ็อบเจกต์ของคลาสใด ก็จะไปเรียกใช้เมท็อด makeSound ประจำอ็อบเจกต์ของคลาสนั้น เช่น ถ้า a[i] อ้างอิงสุนัข a[i].makeSound() ก็จะร้อง "โฮ่ง ๆ" ถ้า a[i] อ้างอิงแมว a[i].makeSound() ก็จะร้อง "เหมียว ๆ" ผู้อ่านลองพิจารณารหัสที่ 9-17 แล้วตอบว่า จะแสดงอะไรทางจอภาพ [3]

[3] c1.makeSound() ได้ เหมียว ๆ, b1.makeSound() ได้ ..., d1.makeSound() ได้ โฮ่ง ๆ, d2.makeSound() ได้ ..., a1.makeSound() ได้ โฮ่ง ๆ, a2.makeSound() ได้ ..., a3.makeSound() ได้ เหมียว ๆ

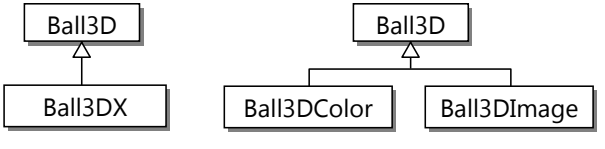
หากเขียนคลาส Tiger ให้เป็นคลาสลูกของ Animal ก็เขียนเมท็อด makeSound เพื่อแทนของที่รับทอดจาก Animal ให้ส่งเสียงคำราม เพียงเท่านี้ เสือที่ส่งไปอยู่ในอาเรย์ a แล้วเรียก makeSound(a) ของ Animal ย่อมร้องคำรามเมื่อเรียก a[i] ที่อ้างอิงอ็อบเจกต์ที่เป็นเสือ เห็นได้ว่า การเขียนคลาสด้วยแนวทางที่นำเสนอช่วยให้เขียนคลาสได้กะทัดรัด ง่ายต่อการเพิ่มเติมคลาสใหม่ ๆ ที่ยังคงใช้ได้กับคลาสเดิมที่ได้เคยเขียนไว้

```
01 public class TestAnimal {
02   public static void main(String[] args) {
03     Cat     c1 = new Cat();
04     Basenji b1 = new Basenji();
05     Dog     d1 = new Dog();
06     Dog     d2 = new Basenji();
07     Animal  a1 = new Dog();
08     Animal  a2 = new Basenji();
09     Animal  a3 = new Cat();
10     c1.makeSound();
11     b1.makeSound();
12     d1.makeSound();
13     d2.makeSound();
14     a1.makeSound();
15     a2.makeSound();
16     a3.makeSound();
17   }
18 }
```

รหัสที่ 9-17 ตัวอย่างการเรียก makeSound กับอ็อบเจกต์หลายประเภท

ลูกบอลเด้งได้ (ต่อ)

กลับมาเขียนลูกบอลกันต่อ เรามีลูกบอลเด้งได้ในห้องแล้ว คราวนี้ต้องการลูกบอลเด้งได้ที่มีเงาที่พื้น (สมมติว่ามีหลอดไฟขนาดใหญ่ให้แสงสว่างตรงดิ่งลงมาที่พื้น) ตั้งสีได้ หรือไม่ก็ใช้ภาพที่อ่านจากแฟ้มภาพแทนการวาดวงกลม โดยเราจะเขียนเป็นคลาสในสองลักษณะ แบบแรกเขียนคลาสเดียวที่มีลักษณะสมบัติทุกอย่างที่ต้องการ กับอีกแบบที่เขียนแยกเป็นสองคลาสดังรูปที่ 9-9



รูปที่ 9-9 ตัวอย่างลูกบอล และลำดับชั้นการรับทอดของคลาสที่จะเขียนในหัวข้อนี้

แต่ก่อนอื่นขอเขียนเพิ่มในคลาส DWindow3D ให้สามารถวาดรูป (ที่อ่านจากแฟ้มภาพ) มาแสดงในห้อง ดังรหัสที่ 9-18 แสดงเมท็อด drawImage ที่รับ img ซึ่งเป็นอ็อบเจกต์ของคลาส DImage [4] มาแสดงที่พิกัด (x,y,z) ในห้อง การแสดงภาพในวินโดว์อาศัยเมท็อด draw ของ DWindow (ซึ่งเป็นคลาสแม่) ในบรรทัดที่ 55 ซึ่งต้องส่งอ็อบเจกต์ DImage และตำแหน่งที่เปลี่ยนจากพิกัดในห้องมาเป็นพิกัดบนระนาบ (ด้วยเมท็อด toSX และ toSY) แต่ก็ต้องอย่าลืมปรับขนาดของภาพตามความลึกของภาพที่จะแสดงในห้องด้วย scaleTo ซึ่งเป็นเมท็อดประจำอ็อบเจกต์ของ DImage (บรรทัดที่ 54) เพียงเท่านี้ ก็มีบริการเพิ่มเพื่อแสดงภาพในห้องตามที่ต้องการ

```
03 public class DWindow3D extends DWindow {
..   ...
53   public void drawImage(DImage img, double x, double y, double z) {
54     img.scaleTo(scale(1, z));
55     super.draw(img, toSX(x, z), toSY(y, z));
56   }
57 }
```

รหัสที่ 9-18 การเพิ่มเมท็อด drawImage ในคลาส DWindow3D

คลาสที่รับผิดชอบมากไป

เริ่มที่คลาส Ball3DX (รหัสที่ 9-19) เราต้องการให้คลาสนี้สร้างลูกบอลที่มีเงา ตั้งสีหรือตั้งภาพแทนลูกบอลได้ จึงให้เป็นคลาสลูกของ Ball3D มีตัวแปร color เก็บสี และตัวแปร image ไว้อ้างอิงอ็อบเจกต์ของ DImage ที่มีค่าเริ่มต้นเป็น null (บรรทัดที่ 5) หมายความว่า ยังไม่ได้อ้างอิงภาพใด ๆ มีเมท็อด setColor และ setImage เพื่อตั้งสีและตั้งภาพตามลำดับ แฟ้มภาพที่รับมาแสดงแทนลูกบอลนี้ควรมีขนาดเป็นสี่เหลี่ยมจัตุรัส เพราะหลังจากสร้างอ็อบเจกต์ของ DImage แล้ว จะตั้งรัศมีของลูกบอลให้เป็นครึ่งหนึ่งของความกว้างภาพ (บรรทัดที่ 15) สำหรับตัวสร้าง เราเขียนให้เหมือนกับของ Ball3D ภาระสำคัญของคลาสนี้อยู่ที่เมท็อด draw ซึ่งต้องตรวจสอบว่า จะวาดรูปลูกบอลจากภาพหรือจะวาดรูปวงกลมมีสี ให้ถือว่า ถ้าผู้ใช้เรียก setImage ย่อมหมายความว่า ต้องการวาดลูกบอลจากแฟ้มภาพ ดังนั้น ถ้า image มีค่าไม่ใช่ null แสดงว่าได้รับการตั้งค่าให้อ้างอิงรูปภาพ จึงใช้บริการ drawImage ของ DWindow3D เพื่อวาดภาพในห้องสามมิติ (บรรทัดที่ 20) แต่ถ้าเป็น null ก็เลือกวาดลูกบอลเป็นวงกลมโดยใช้เมท็อด fillEllipse (บรรทัดที่ 22) แต่ต้องอย่าลืมว่า ลูกบอลต้องมีเงาด้วย ดังนั้น จึงต้องวาดเงาก่อนวาดลูกบอล (บรรทัดที่ 18) เพราะถ้าลูกบอลติดพื้น ลูกบอลจะได้ทับเงา ขอแสดงเงาแบบเหมือนจริงเล็กน้อย (นั่นคือไม่สมจริง) โดยวาดเงาเป็นรูปวงรีที่พื้น มีพิกัด x และ z เหมือนของลูกบอล แต่

[4] คลาสนี้มีชื่อเต็มว่า jlab.graphics.DImage (เป็นคลาสพิเศษของ JLab ที่เขียนขึ้นเพื่ออำนวยความสะดวกในการศึกษาการเขียนโปรแกรม เราสร้างอ็อบเจกต์ของ DImage ได้ง่าย ๆ โดยส่งชื่อแฟ้มภาพให้ตัวสร้างของคลาส เช่น new DImage("c:/java101/fball.gif") ภาพที่อ่านควรเป็นภาพรูปแบบ gif หรือ png ที่สามารถตั้งสีพื้นแบบโปร่งใสได้ ทำให้เห็นภาพแลดูเป็นธรรมชาติกลมกลืนกับฉากหลัง

พิกัด y ติดพื้นเสมอ คือ มีค่าเท่ากับความสูงของห้อง ส่วนความกว้างของเงาให้แปรตามค่า y ของลูกบอล นั่นคือ ลูกบอลยิ่งใกล้พื้น เงายิ่งมีขนาดกว้างขึ้น ๆ จนกว้างเท่าลูกบอลเมื่อลูกบอลแตะพื้น

```
01 import jlab.graphics.*;
02
03 public class Ball3DX extends Ball3D {
04   private int color = DWindow.BLACK;
05   private DImage image = null;
06
07   public Ball3DX(double r, double x, double y, double z) {
08     super(r, x, y, z);
09   }
10   public void setColor(int c) {
11     color = c;
12   }
13   public void setImage(String imageFile) {
14     image = new DImage(imageFile);
15     r = image.getWidth()/2;
16   }
17   public void draw(DWindow3D w3) {
18     drawShadow(w3);
19     if (image != null) {
20       w3.drawImage(image, x, y, z);
21     } else {
22       w3.fillEllipse(color, x, y, z, 2*r, 2*r);
23     }
24   }
25   private void drawShadow(DWindow3D w3) {
26     double s = y / w3.getHeight();
27     w3.fillEllipse(DWindow.BLACK, x, w3.getHeight(), z,
28                    2*r*s, r/2*s);
29   }
30 }
```

x และ z ของเงาเหมือนของเดิม แต่ y ให้อยู่ติดพื้น

รหัสที่ 9-19 คลาส Ball3DX เป็นคลาสลูกของ Ball3D ที่มีเงาและตั้งสีหรือตั้งภาพลูกบอลได้

คลาสที่รับผิดชอบเท่าที่จำเป็น

มาดูการเขียนคลาสเพื่อสร้างลูกบอลที่มีเงา ตั้งสีหรือตั้งภาพแทนลูกบอลได้ในอีกลักษณะคราวนี้เราจะเขียนแยกเป็นสองคลาส คลาสหนึ่งชื่อ Ball3DColor ไว้สร้างลูกบอลที่มีสี (รหัสที่ 9-20) กับอีกคลาสชื่อ Ball3DImage ไว้สร้างลูกบอลที่แสดงภาพลูกบอลจากแฟ้มภาพ (รหัสที่ 9-21) แต่ละคลาสรับผิดชอบเพียงเรื่องเดียว ไม่นำมารวมกันแบบคลาส Ball3DX เป็นการลดความซับซ้อน ไม่ต้องตรวจสอบว่า ถ้าเป็นแบบสีทำอย่าง เป็นแบบภาพทำอย่าง

```
01 import jlab.graphics.*;
02
03 public class Ball3DColor extends Ball3D {
04   private int color;
05   public Ball3DColor(int c, double r, double x, double y,double z){
06     super(r, x, y, z);
07     color = c;
08   }
09   public void draw(DWindow3D w3) {
10     drawShadow(w3);
11     w3.fillEllipse(color, x, y, z, 2*r, 2*r);
12   }
13   private void drawShadow(DWindow3D w3) {
14     double s = y / w3.getHeight();
15     w3.fillEllipse(DWindow.BLACK, x, w3.getHeight(), z,
16                    2*r*s, r/2*s);
17   }
18 }
```

รับสีของลูกบอล

รหัสที่ 9-20 คลาส Ball3DColor วาดลูกบอลเป็นวงกลมมีสี

```
01 import jlab.graphics.*;
02
03 public class Ball3DImage extends Ball3D {
04   private DImage image;;
05   public Ball3DImage(String fn, double x, double y, double z) {
06     super(10, x, y, z);
07     image = new DImage(fn);
08     r = image.getWidth() / 2;
09   }
10   public void draw(DWindow3D w3) {
11     drawShadow(w3);
12     w3.drawImage(image, x, y, z);
13   }
14   private void drawShadow(DWindow3D w3) {
15     double s = y / w3.getHeight();
16     w3.fillEllipse(DWindow.BLACK, x, w3.getHeight(), z,
17                    2*r*s, r/2*s);
18   }
19 }
```

รับชื่อแฟ้มภาพของลูกบอล

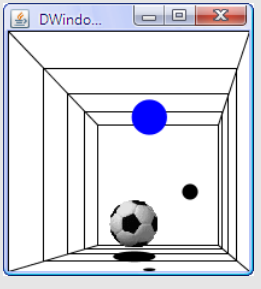
เราสามารถจัดความซ้ำซ้อนของเมท็อด drawShadow ที่คลาสนี้กับที่คลาส Ball3DColor โดยการสร้างคลาสนามธรรม ที่อธิบายในหัวข้อเพิ่มเติม

รหัสที่ 9-21 คลาส Ball3DImage วาดภาพลูกบอลจากแฟ้มภาพ

ลองคิดดูว่า ถ้าเราต้องการลูกบอลที่แสดงภาพเคลื่อนไหวขณะลูกบอลเคลื่อนที่ หรือลูกบอลที่มีรูปร่างไม่คงตัว หรือลูกบอลเปล่งแสง หรืออื่น ๆ ก็อาจเลือกเพิ่มความสามารถเหล่านี้ให้กับคลาส Ball3DX ให้รับผิดชอบการแสดงลูกบอลหลากหลายลักษณะนี้ ผู้อ่านคงจินตนาการได้ว่าเมท็อด draw ต้องมีคำสั่งตรวจสอบเพื่อแยกกรณีการแสดงลูกบอลในหลากหลายลักษณะ หรือเราจะแยกเขียนเป็นหลาย ๆ คลาส แต่ละคลาสมีเมท็อด draw ที่รับผิดชอบบทบาทการแสดงลูกบอล

ตามข้อกำหนด ซึ่งน่าจะมีความซับซ้อนของคลาสน้อยกว่าแบบแรก เพราะระบบจะเป็นผู้พิจารณาและตรวจสอบให้เองว่า คำสั่ง `b.draw(w)` จะไปเรียกเมท็อด `draw` ใด ขึ้นกับว่า `b` อ้างอิงอ็อบเจกต์ของคลาสใดขณะทำงาน (ผู้อ่านควรลองสั่งงานโปรแกรมในรหัสที่ 9-22 ที่สร้างลูกบอลหลายแบบในวินโดว์)

```
01 import jlab.graphics.*;
02
03 public class TestBall3D {
04   public static void main(String[] args) {
05     int h = 200, h2 = h/2;
06     DWindow3D w = new DWindow3D(h, h, h);
07     Ball3D[] balls = new Ball3D[] {
08            new Ball3DColor(DWindow.BLACK, 10, h2, h2, h2),
09            new Ball3DColor(DWindow.BLUE, 15, h2, h2, h2),
10            new Ball3DImage("c:/java101/fball.gif", h2, h2, h2) };
11     w.setRepaintDuringSleep(true);
12     while (true) {
13       w.clearBackground();
14       w.drawFrame(50);
15       for (int i = 0; i < balls.length; i++) {
16         balls[i].move(w);
17         balls[i].draw(w);
18       }
19       w.sleep(50);
20     }
21   }
22 }
```



รหัสที่ 9-22 ตัวอย่างโปรแกรมใช้งาน `Ball3D`, `Ball3DColor`, และ `Ball3DImage`

เกร็ดการรับทอด

หัวข้อนี้นำเสนอเนื้อหาน่ารู้ที่เกี่ยวกับการรับทอด อันได้แก่ คลาสชื่อ `Object` ที่เป็นคลาสบรรพบุรุษของทุกคลาสในจาวา การใช้ `@Override` เพื่อดักจับข้อผิดพลาดในการเขียนเมท็อดของคลาสลูกเพื่อแทนของคลาสแม่ และการใช้ `final` เพื่อป้องกันการแทนเมท็อดและการรับทอด

Object : บรรพบุรุษของทุกคลาส

บทนี้นำเสนอการเขียนคลาสลูกที่รับทอดลักษณะสมบัติจากคลาสแม่ จาวาเป็นภาษาที่อนุญาตให้สร้างคลาสใหม่ที่รับทอดจากคลาสแม่เพียงคลาสเดียวเท่านั้น เรียกว่า *การรับทอดทางเดียว* (single inheritance) หากพิจารณาคลาสต่าง ๆ ที่เขียนกันมาตั้งแต่บทแรก ล้วนเป็นคลาสที่ไม่ได้กำหนดว่า `extends` จากคลาสใดเลย นั่นคือ ไม่ได้ระบุว่าเป็นลูกของใคร จาวากำหนดให้คลาสแบบนี้เป็นลูกของคลาสพิเศษคลาสหนึ่งมีชื่อว่า `Object` (นี่เป็นชื่อคลาส อย่าสับสน) จึงถือ

ได้ว่า Object เป็นคลาสบรรพบุรุษ หรือบางทีเรียกว่า เป็น*คลาสราก* (root class) ของทุกคลาสในจาวา ดังนั้น การเขียนประกาศที่หัวคลาสว่า `public class MyClass` ก็เหมือนกับการเขียน `public class MyClass extends Object` นั่นเอง [5]

เมื่อคลาส Object เป็นบรรพบุรุษของทุกคลาส ดังนั้น อ็อบเจกต์ของ Animal, Dog, Cat, Ball, DWindow, และอื่น ๆ ล้วนเป็น Object ทั้งสิ้น หากเราประกาศตัวแปรให้เป็นแบบ Object ย่อมสามารถอ้างอิงอ็อบเจกต์ได้ทุกชนิดในจาวา เช่น `Object a;` เป็นการประกาศตัวแปร a มีไว้อ้างอิงอ็อบเจกต์ที่เป็น Object ดังนั้น จะให้ `a = new Dog();` หรือ `a = new Ball();` ย่อมได้ทั้งนั้น หรือในกรณีที่เราเขียนเมท็อดที่ต้องการรับอ็อบเจกต์ทุกชนิด ก็สามารถประกาศพารามิเตอร์เป็นแบบ Object ได้เช่นกัน แต่อย่าลืมว่า เมื่อ a เป็นแบบ Object ตัวแปลโปรแกรมจะให้เรียกเมท็อดเฉพาะที่อยู่ในคลาส Object เท่านั้น จะเขียน `a.bark()` คงไม่ได้

คลาส Object มีเมท็อดประจำอ็อบเจกต์อยู่ 11 เมท็อด (นั่นหมายความว่า ทุกอ็อบเจกต์ในจาวามีทั้ง 11 เมท็อดเหล่านี้จากการรับทอดให้เรียกใช้) ในจำนวนนี้มีอยู่สองเมท็อดคือ equals และ toString ซึ่งเราเคยนำเสนอในบทที่แล้วว่า ควรเขียนให้กับคลาสใหม่เสมอ หากเราเขียนคลาสใหม่แล้วไม่ได้เขียนสองเมท็อดนี้ ก็จะรับทอดสองเมท็อดนี้จากคลาสแม่ ถ้าคลาสแม่ คลาสยาย ... ไล่ขึ้นไปเรื่อยในลำดับชั้นการรับทอดไม่ได้เขียนสองเมท็อดนี้เลย สุดท้ายจะถึงคลาส Object ที่มีสองเมท็อดนี้ คลาสเราก็ย่อมได้รับทอดสองเมท็อดนี้จากคลาส Object รหัสที่ 9-23 แสดงรายละเอียดของทั้งสองเมท็อดนี้ในคลาส Object

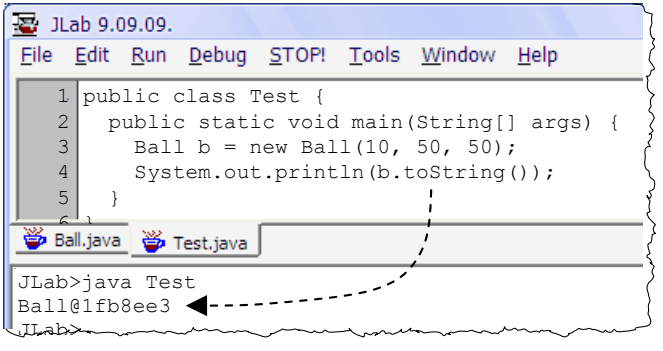
```
package java.lang;

public class Object {
  ...
  // คืนผลการเปรียบเทียบว่า อ็อบเจกต์ obj "เท่ากับ" อ็อบเจกต์ this หรือไม่
  public boolean equals(Object obj) {
    return this == obj;
  }
  // คืนสตริงที่แทนค่าของอ็อบเจกต์ this
  public String toString() {
    return this.getClass().getName() + "@" +
           Integer.toHexString(hashCode());
  }
  ...
}
```

equals ของ Object บอกว่า เท่ากัน ก็ต่อเมื่อเป็นอ็อบเจกต์ตัวเดียวกัน

รหัสที่ 9-23 เมท็อด equals และ toString ที่ปรากฏในคลาส Object

[5] เวลาอ่านเนื้อหาในหัวข้อนี้ ต้องคิดเสมอว่า คำว่า "Object" เขียนโอตัวใหญ่ เป็นชื่อคลาส เป็นคลาสบรรพบุรุษของทุกคลาสในจาวา ต่างจากคำว่า "อ็อบเจกต์" ซึ่งหมายถึงข้อมูลที่เราสร้างขึ้นมาเพื่อการประมวลผล

เริ่มที่ toString ก่อน ผู้เขียนคลาสมาตรฐาน Object เขียนให้เมท็อด toString คืนสตริงที่ประกอบด้วยชื่อคลาส (ได้จากการเรียก getClass().getName() ที่ไม่ขออธิบาย แต่ได้ผลคือชื่อคลาสของอ็อบเจกต์ this) ตามด้วยเครื่องหมาย @ และจำนวนเต็มในรูปของจำนวนฐานสิบหก (ที่ได้มาจากการเรียกอีกเมท็อดของอ็อบเจกต์ที่ชื่อว่า hashCode ที่ขอไม่อธิบายเช่นกัน) เราไม่ได้เขียนให้คลาส Ball เป็นคลาสลูกของใคร จึงถือว่าเป็นลูกของคลาส Object และเราก็ไม่ได้เขียน toString ในคลาส Ball ดังนั้น b.toString() โดยที่ b อ้างอิงลูกบอล ย่อมไปทำที่ toString ของคลาส Object จะได้ผลดังตัวอย่างรูปที่ 9-10 (หากผู้อ่านสั่งโปรแกรมทำงานอีกครั้ง อาจได้จำนวนฐานสิบหกที่ต่างจากที่แสดงก็ได้) หากเราไม่ต้องการให้การเรียก b.toString() ได้ผลแบบนี้ ก็ควรเขียน toString ที่คลาส Ball เพื่อแทนของคลาสแม่ที่รับทอดมา



รูปที่ 9-10 ตัวอย่างผลที่ได้จากเมท็อด toString ของคลาส Object

สำหรับเมท็อด equals ที่คลาส Object ทำงานง่าย ๆ ด้วยการคืนผลของการเปรียบเทียบ this == obj ซึ่งจะคืนจริงก็ต่อเมื่อ this และ obj มีค่าเท่ากัน นั่นคือ เมื่อ this และ obj อ้างอิงอ็อบเจกต์ "ตัวเดียวกัน" ถ้าเรากำลังเขียนคลาสใหม่ และคิดว่าการเปรียบเทียบความ "เท่ากัน" ของอ็อบเจกต์ของคลาสใหม่นี้ คือต้องเป็น "ตัวเดียวกัน" เช่น กรณีของ Ball เราถือว่าลูกบอลสองลูกเท่ากันก็ต่อเมื่อเป็นลูกบอลลูกเดียวกันเท่านั้น (นั่นคือลูกบอลสองลูกที่มีขนาดเท่ากัน อยู่ตำแหน่งเดียวกัน และเคลื่อนด้วยความเร็วเท่ากัน ก็ถือว่าไม่เท่ากัน เป็นคนละลูก) ก็ไม่ต้องเขียนเมท็อด equals เอง ใช้ของที่รับทอดจากบรรพบุรุษได้เลย แต่ถ้าเราคิดว่า อ็อบเจกต์ของคลาสเราอาจมีมากกว่าหนึ่งตัวที่มีค่าเท่ากัน ก็ต้องเขียน equals ใหม่เอง เช่น อ็อบเจกต์จำนวนเชิงซ้อนของคลาส Complex มีได้หลายตัวที่มีค่าเท่ากัน เราจึงเขียน equals เองดังรหัสที่ 9-24 (ซึ่งเหมือนกับที่เคยเขียนในรหัสที่ 8-19) ให้สังเกตว่า equals นี้รับพารามิเตอร์เป็น Complex จึงไม่ได้เป็นเมท็อดของลูกที่ไปแทนของแม่ ถือว่าคลาส Complex มีเมท็อด equals สองตัว ตัวหนึ่งรับพารามิเตอร์แบบ Complex อีกตัวได้รับทอดจากบรรพบุรุษซึ่งรับพารามิเตอร์แบบ Object ลองพิจารณาส่วนของโปรแกรมในรหัสที่ 9-25 มีการสร้างจำนวนเชิงซ้อนสอง

จำนวน z1 และ z2 และมีตัวแปรอ้างอิง z3 (ซึ่งเป็นแบบ Object) ถูกตั้งค่าให้เท่ากับ z2 มีผลการทำงานของสามคำสั่งถัดไปดังนี้

- z1.equals(z2) เป็นการเรียก equals ที่รับพารามิเตอร์แบบ Complex เพราะ z2 เป็น Complex ได้ผลเป็น true เพราะทั้งสองอ็อบเจกต์มีค่า real และ imag เท่ากัน
- z2.equals(z3) เป็นการเรียก equals ที่รับพารามิเตอร์แบบ Object เพราะ z3 เป็น Object ได้ผลเป็น true เพราะ z2 และ z3 อ้างอิงอ็อบเจกต์ตัวเดียวกัน
- z1.equals(z3) เป็นการเรียก equals ที่รับพารามิเตอร์แบบ Object เพราะ z3 เป็น Object ได้ผลเป็น false เพราะ z1 และ z3 อ้างอิงอ็อบเจกต์คนละตัว (ถึงแม้ว่าจะมีค่าเท่ากันก็ตาม)

```
public class Complex {
  private double real, imag;
  ...
  public boolean equals(Complex z) {
    return real == z.real && imag == z.imag;
  }
```

เมท็อดนี้ไม่ได้แทนของคลาสแม่ เพราะหัวเมท็อดไม่เหมือนกัน

รหัสที่ 9-24 เมท็อด equals ของคลาส Complex ที่เคยเขียนในรหัส 8-19

```
  Complex z1 = new Complex(1,1);
  Complex z2 = new Complex(1,1);
  Object  z3 = z2;
  System.out.println(z1.equals(z2));  // equals ของ Complex ได้ true
  System.out.println(z2.equals(z3));  // equals ของ Object  ได้ true
  System.out.println(z1.equals(z3));  // equals ของ Object  ได้ false
```

รหัสที่ 9-25 ตัวอย่างสาธิตข้อผิดพลาดของ equals ในรหัสที่ 9-24

```
public class Complex {
  private double real, imag;
  ...
  public boolean equals(Object obj) {
    if (!(obj instanceof Complex)) return false;
    Complex z = (Complex) obj;
    return real == z.real && imag == z.imag;
  }
```

เมท็อดนี้แทนของคลาส Object

รหัสที่ 9-26 เมท็อด equals ของคลาส Complex ที่แทนของคลาส Object

นั่นหมายความว่า ไม่ควรเขียน equals ดังที่ได้ทำมาในรหัสที่ 9-24 ควรเขียนให้แทนของบรรพบุรุษไปเลย ดังรหัสที่ 9-26 เริ่มด้วยการเขียนหัวเมท็อดให้ตรงกับของบรรพบุรุษซึ่งรับพารามิเตอร์เป็น Object เริ่มทำงานด้วยการตรวจสอบว่า obj ที่รับมานั้นเป็นจำนวนเชิงซ้อนหรือเปล่า ถ้าไม่ใช่ ให้คืน false ว่าไม่เท่าทันที (เหตุที่ต้องตรวจสอบนี้ เพราะเราให้พารามิเตอร์

เป็น `Object` ผู้เรียกก็สามารถส่งอ็อบเจกต์ของ `Cat` มาได้โดยตัวแปลโปรแกรมไม่ฟ้อง) ถ้าผ่านการตรวจสอบ ก็ downcast แปลงเป็น `Complex` เพื่อเปรียบเทียบค่า `real` และ `imag` ต่อไป

@Override

สิ่งที่มักผิดบ่อย ๆ ในการเขียนเมท็อดที่เราคิดตอนเขียนว่า จะเขียนเพื่อแทนของที่รับทอดมาจากคลาสแม่ แต่ก็สะกดชื่อเมท็อดผิด หรือเขียนพารามิเตอร์คนละแบบ หรือจำนวนพารามิเตอร์ไม่เท่ากัน (นั่นคือ เขียนหัวเมท็อดไม่เหมือนกับของคลาสแม่) โดยสิ่งที่เขียนนี้ผ่านการพิจารณาของตัวแปลโปรแกรม ได้เป็นเมท็อดใหม่ในคลาส ผู้เขียนคลาสก็คิดว่าถูก แต่ทำงานเข้าจริง ๆ กลับพบพฤติกรรมที่ไม่ตรงตามที่คาดหวังไว้ (เพราะเมท็อดที่เขียนไม่ได้ถูกเรียก ระบบไปเรียกเมท็อดของคลาสแม่แทน) ผู้อ่านคิดว่ารหัสที่ 9-27 มีอะไรไม่ถูกต้อง [6]

```
public class Complex {
  ...
  public boolean equal(Object obj) {
    if (!(obj instanceof Complex)) return false;
    Complex z = (Complex) obj;
    return real == z.real && imag == z.imag;
  }
  public String tostring() {
    return "(" + real + ", " + imag + "i)";
  }
```

รหัสที่ 9-27 เมท็อด `equal` และ `tostring` ที่ไม่ได้ไปแทนของคลาส `Object`

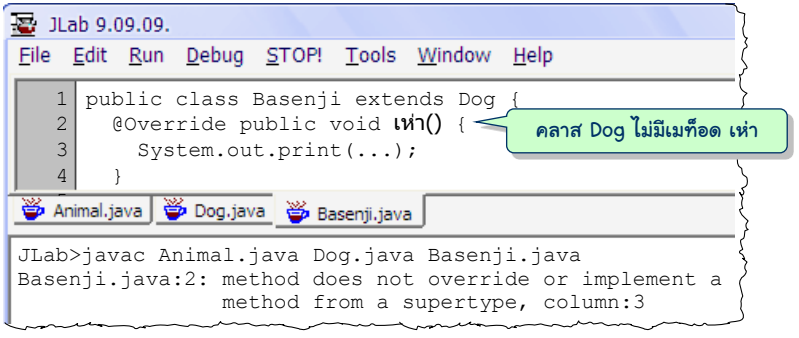
วิธีป้องกันความผิดพลาดดังกล่าวกระทำได้โดยการเขียนคำว่า `@Override` ไว้ข้างหน้าเมท็อดที่เราตั้งใจให้แทนของที่ได้รับทอดจากคลาสแม่ ดังตัวอย่างในรหัสที่ 9-28 หากตัวแปลโปรแกรมพบว่า ไม่มีเมท็อดใดที่ได้รับทอดมาที่มีหัวเมท็อดเหมือนบรรทัดที่เขียนเลย ก็จะฟ้องให้ทราบ ดังตัวอย่างในรูปที่ 9-11

```
public class BallG extends Ball {
  ...
  @Override
  public void move(DWindow w) {
    dy += gravity;
    super.move(w);
  }
```

@Override บอกว่าเมท็อดนี้เขียนขึ้นเพื่อแทนของที่ได้รับทอด

รหัสที่ 9-28 การใช้ `@Override` กำกับหน้าเมท็อด

6 ผิดเพราะ equal ไม่มี s และ tostring ใช้ s ตัวเล็ก ถือว่าได้เขียนเมท็อดใหม่ ไม่ได้เขียนทับเมท็อดของคลาสแม่



รูปที่ 9-11 ข้อความแจ้งความผิดพลาดเรื่องการแทนเมท็อดที่ไม่มีในคลาสแม่

คลาสและเมท็อดแบบ final

`final` เป็นคำสำคัญในจาวา มีไว้กำกับการประกาศตัวแปร เพื่อให้ตัวแปรนั้นรับค่าได้ครั้งเดียว แล้วค่าของตัวแปรจะไม่เปลี่ยนอีกแล้ว นอกจากนี้เราสามารถใช้ `final` กำกับหน้าเมท็อดเพื่อระบุว่า คลาสลูกห้ามเขียนเมท็อดของตนมาแทนเมท็อดนี้ และสามารถเขียน `final` กำกับหน้าคำว่า `class` เพื่อหมายความว่า ห้ามคลาสใดมาเป็นลูกของคลาสนี้

เรามักใช้ `final` กำกับคลาสที่ผลิตอ็อบเจกต์แบบเปลี่ยนค่าไม่ได้ ดังตัวอย่างในรหัสที่ 9-29 แสดงคลาส `NameTag` ซึ่งถูกออกแบบมาใช้ผลิตป้ายชื่อที่เปลี่ยนค่าไม่ได้ (แต่ในตัวอย่างเราลืมใส่ `final` กำกับคลาส) คลาสนี้มีตัวแปรประจำอ็อบเจกต์เป็นแบบ `final` ได้รับการตั้งค่าครั้งแรกครั้งเดียวที่ตัวสร้าง ไม่มีเมท็อดใดเปลี่ยนค่าได้ สามารถผลิตอ็อบเจกต์แบบเปลี่ยนค่าไม่ได้ตามต้องการ ใครที่เขียนเมท็อดรับ ส่ง และประมวลผลอ็อบเจกต์แบบ `NameTag` ก็สบายใจได้ว่าอ็อบเจกต์จะคงค่าเดิมเสมอไปไม่เปลี่ยนแปลง แต่ความจริงไม่เป็นเช่นนั้น เช่น รหัสที่ 9-30 แสดงคลาส `MutableNameTag` เขียนเป็นคลาสลูกของ `NameTag` ภายในเขียนทุกอย่างเองเลียนแบบ `NameTag` แถมบริการที่เปลี่ยนค่าภายในได้ (เมท็อด `setID` และ `setName`) ด้วยความที่เป็นคลาสลูกของ `NameTag` อ็อบเจกต์ของ `MutableNameTag` จึงสามารถถูกส่งไปยังเมท็อดที่รับพารามิเตอร์เป็น `NameTag` หรือคืนกลับมาจากเมท็อดที่คืนผลลัพธ์เป็น `NameTag` ได้ตามปกติ (เพราะอ็อบเจกต์ของคลาสลูกถือได้ว่าเป็นข้อมูลประเภทคลาสแม่) ทำให้แนวคิดการออกแบบของคลาส `NameTag` ที่ไม่ต้องการให้อ็อบเจกต์เปลี่ยนค่าได้ตามที่ตั้งใจไว้ผิดไป แต่ถ้าเราเขียนให้คลาส `NameTag` เป็น `final` ก็สบายใจได้ว่า จะไม่เกิดเหตุการณ์ดังกล่าว คลาส `String` ที่เราใช้กันมาตั้งแต่บทแรกเป็นอีกตัวอย่างของคลาสที่เป็น `final` มีลูกไม่ได้

สำหรับเมท็อดที่เป็น `final` (ซึ่งคลาสลูกเขียนแทนไม่ได้) มักใช้กับเมท็อดที่ต้องการประกันว่ามีพฤติกรรมดังที่เขียน เมื่อเรียกใช้กับทุกอ็อบเจกต์ของคลาสนี้ และของคลาสลูกหลาน เช่น รหัสที่ 9-31 แสดงคลาสที่ถ้าใครมา `new` แล้วตัวสร้างทำงาน (ซึ่งอาจมาจากคลาสลูกก็ได้) จะ

แสดงข้อความลิขสิทธิ์ทางจอภาพ เพื่อไม่ให้ใครเปลี่ยนพฤติกรรมนี้ จึงเขียนให้เมท็อดที่แสดงข้อความลิขสิทธิ์นี้เป็น `final`

```
public class NameTag {
  private final int id;
  private final String name;
  public NameTag(int id, String name) {
    this.id = id; this.name = name;
  }
  public int getID() {
    return id;
  }
  public String getName() {
    return name;
  }
}
```

ถ้าเติม final ให้คลาสนี้ จะทำให้มีลูกไม่ได้

รหัสที่ 9-29 คลาส `NameTag` ถูกออกแบบให้ผลิตอ็อบเจกต์แบบเปลี่ยนค่าไม่ได้

```
public class MutableNameTag extends NameTag {
  private int mutableID;
  private String mutableName;
  public MutableNameTag(int id, String name) {
    super(id, name);
    this.mutableID = id; this.mutableName = name;
  }
  public int getID() {
    return mutableID;
  }
  public String getName() {
    return mutableName;
  }
  public void setID(int id) {
    this.mutableID = id;
  }
  public void setName(String name) {
    this.mutableName = name;
  }
}
```

เปลี่ยนชื่อ และหมายเลขประจำตัวได้

รหัสที่ 9-30 คลาส `MutableNameTag` ขยายจาก `NameTag` แต่ให้บริการเปลี่ยนแปลงค่าภายใน

```
public class MyMagicWindow {
  public MyMagicWindow(int width, int height) {
    //...
    showLicense();
  }
  public final void showLicense() {
    System.out.println("ลิขสิทธิ์ของนายสมชาย ให้ใช้ฟรีเพื่อการศึกษาเท่านั้น");
  }
```

รหัสที่ 9-31 คลาส `MyMagicWindow` มีเมท็อด `showLicense` ที่ไม่ให้ใครแก้ไข

เพิ่มเติม

หัวข้อนี้นำเสนอเนื้อหาเพิ่มเติมที่เกี่ยวกับการโปรแกรมเชิงวัตถุและภาษาจาวา อันได้แก่ การกำหนดทัศนวิสัยขององค์ประกอบต่าง ๆ ของคลาส การเขียนและการใช้งานคลาสนามธรรม และอินเทอร์เฟซ ปิดท้ายการอธิบายความหมายของศัพท์เทคนิคบางคำที่ใช้ในวงการ

ทัศนวิสัย

ผู้ออกแบบคลาสควรกำหนดทัศนวิสัยของคลาส เมท็อด ตัวสร้าง และตัวแปรต่าง ๆ ซึ่งคือระดับการเผยแพร่ให้คลาสอื่น ๆ เข้าใช้สิ่งต่าง ๆ เหล่านี้ ซึ่งมีทั้งหมดสี่ระดับดังนี้

- ใช้ได้ส่วนตัว : ให้ใส่คำว่า `private` กำกับ
- ใช้ได้ทั่วไป : ให้ใส่คำว่า `public` กำกับ
- ใช้ได้เฉพาะเพื่อน : ไม่ต้องใส่คำใดกำกับ (เพื่อนคือคลาสที่อยู่ในแพ็กเกจเดียวกัน)
- ใช้ได้เฉพาะลูก : ให้ใส่คำว่า `protected` กำกับ (จาวาอนุญาตให้คลาสเพื่อนใช้ของที่เป็นแบบ `protected` ได้ด้วย)

อนึ่ง สำหรับกรณีของคลาสนั้น เรากำหนดได้เฉพาะแบบใช้ได้ทั่วไป (`public`) และแบบให้เพื่อนใช้เท่านั้น สำหรับคำว่า `protected` เป็นแบบใหม่ที่ยังไม่เคยนำเสนอมาก่อน หากเราคิดว่าคลาสที่กำลังเขียนอยู่มีตัวแปรประจำอ็อบเจกต์หรือเมท็อดใดที่ไม่ต้องการเปิดเผยให้ใช้ทั่วไป แต่คาดว่า น่าจะเป็นประโยชน์กับคลาสลูก ก็ให้ใส่ `protected` กำกับ ดังตัวอย่างต่อไปนี้

- คลาส `Ball` มีตัวแปร `x`, `y`, `dx`, `dy` และ `r` แบบ `public` แต่ถ้าคิดให้ดี เราไม่น่าเปิดให้ใครก็ได้มาใช้หรือแก้ไขได้โดยตรง เพราะการเปลี่ยนแปลงตำแหน่งและความเร็วของลูกบอลอยู่ในเมท็อด `move` แต่ที่ผ่านมาเราต้องเขียนเป็น `public` เพราะต้องการให้คลาสลูกใช้ ดังนั้น จึงควรกำหนดให้ตัวแปรเหล่านี้เป็น `protected` จะเหมาะสมกว่า

- คลาส `DWindow3D` มีเมท็อด `toSX`, `toSY` และ `scale` ซึ่งเขียนเป็นแบบใช้ส่วนตัวเพื่อแปลงพิกัดในสามมิติมาอยู่บนระนาบหน้าจอ สมมติว่า เราจะสร้างคลาสลูกซึ่งเป็นวินโดว์สามมิติที่ผู้ชมต้องสวมแว่นตาแดงน้ำเงิน (ดูแบบฝึกหัด) เพื่อให้รับรู้ความลึกได้สมจริง โดยจะแสดงภาพสองภาพสำหรับตาแต่ละข้าง ข้างหนึ่งภาพสีแดง อีกข้างภาพสีน้ำเงิน คลาสลูกนี้ก็คงต้องใช้เมท็อด `toSX`, `toSY` และ `scale` อย่างแน่นอน จึงควรเขียนให้เมท็อดทั้งสามใน `DWindow3D` ให้เป็นแบบ `protected`

คลาสนามธรรม

หากลองย้อนกลับไปดูลูกบอลของคลาส `Ball3DColor` และ `Ball3DImage` (รหัสที่ 9-20 และ 21) ที่เขียนมา จะพบความซ้ำซ้อนของเมท็อด `drawShadow` ที่รับผิดชอบการแสดงเงา

ของลูกบอลที่พื้น ถ้าเราเพิ่มลูกบอลประเภทอื่นให้เป็นคลาสลูกของ `Ball3D` และหวังให้มีการแสดงเงา ก็คงต้องเขียน `drawShadow` กำกับไว้เช่นกัน ทำไมเราไม่ย้ายเมท็อดนี้ไปไว้ที่คลาส `Ball3D` น่าจะเหมาะสมกว่า และจะว่าไปแล้ว เรามี `Ball3DColor` แล้ว ก็ไม่มีความจำเป็นต้องมี `Ball3D` ที่แสดงเฉพาะลูกบอลสีดำเท่านั้น ครั้นจะตัด `Ball3D` ออกจากสารบบ ก็ไม่ดีเท่าไหร่ เพราะเป็นแหล่งรวมของเมท็อดที่คลาสลูกใช้ร่วมกันได้ (เช่น `drawShadow` และ `move`) และยังทำให้สามารถมองอ็อบเจกต์ของทั้ง `Ball3DColor` และ `Ball3DImage` เป็นแบบ `Ball3D` ได้ด้วย ปัญหาคือจะทำอย่างไรจึงไม่อนุญาตให้ผู้ใช้มาสร้าง `Ball3D` แต่ยังคงมี `Ball3D` ให้คลาสลูกใช้ได้ตามที่นำเสนอ

เราสามารถใช้แนวคิดของ *คลาสนามธรรม* (abstract class) กับปัญหาข้างต้นได้ คลาสนามธรรมเป็นคลาสในจาวาที่มีคำว่า `abstract` กำกับที่หัวคลาส ระบบจะไม่อนุญาตให้สร้างอ็อบเจกต์ของคลาสประเภทนี้ แต่มีคลาสนี้ไว้เพื่อเป็นแม่ของคลาสอื่น เป็นที่เก็บตัวแปรและเมท็อดร่วมของคลาสลูกทั้งหลาย และทำให้เรามองคลาสลูกทุกคลาสเป็นประเภทเดียวกันได้ จุดเด่นอีกประการหนึ่งของคลาสนามธรรมอยู่ตรงที่สามารถเขียน*เมท็อดนามธรรม* (abstract method) ในคลาสได้ เมท็อดนามธรรมมีคำว่า `abstract` กำกับที่หัว เป็นเมท็อดที่เขียนเฉพาะหัวเมท็อด ไม่มีรายละเอียดการทำงานของตัวเมท็อด เป็นการประกาศว่า คลาสใดที่จะมาเป็นลูกต้องเขียนเมท็อดที่มีหัวตรงกันพร้อมรายละเอียดการทำงานไว้ด้วย เมท็อดนามธรรมจึงบอกแค่ว่า คลาสนี้จะให้บริการอะไร (แต่ไม่บอกว่าทำอย่างไร) แล้วผลักภาระให้คลาสลูกบรรยายรายละเอียดการทำงาน (จึงมักเรียกคลาสที่ไม่เป็นนามธรรมว่า *คลาสรูปธรรม* - concrete class)

รหัสที่ 9-32 เขียนคลาส `Ball3D` ให้เป็นคลาสนามธรรม มีตัวแปร `z` กับ `dz`, มีเมท็อด `drawShadow` และมีตัวสร้างหนึ่งตัวเป็นแบบ `protected` ให้เฉพาะลูกใช้ ผู้อ่านอาจสงสัยว่าในเมื่อเราสร้างอ็อบเจกต์ของคลาสนามธรรมไม่ได้ แล้วจะเขียนตัวสร้างไว้ทำไม เหตุผลก็คือ เราต้องเขียนตัวสร้างของคลาสนามธรรมให้คลาสลูกเรียกใช้ด้วย `super(...)` ในตัวสร้างของคลาสลูก เพื่อจะได้ตั้งค่าเริ่มต้นให้กับตัวแปรประจำอ็อบเจกต์ของคลาสนามธรรม (ซึ่งในตัวอย่างนี้คือ `z` กับ `dz`) นอกจากนี้ยังมีเมท็อด `move` ที่เมื่อคลาสลูกรับทอดไปแล้ว ก็เป็นแบบ `public` ให้ใช้ได้ทั่วไป และจุดที่สำคัญคือ คลาสนี้มี `draw` เป็นเมท็อดนามธรรม (ให้สังเกตเครื่องหมายอัฒภาค `;` ปิดท้ายหัวเมท็อดทันที) ทำให้คลาสรูปธรรมที่จะมาเป็นคลาสลูกต้องเขียนเมท็อด `draw` พร้อมรายละเอียดด้วย (ดังรหัสที่ 9-33)

ผู้อ่านอาจสงสัยอีกว่า ถ้าเราเขียนเมท็อด `draw` ให้สมบูรณ์ที่คลาสลูกๆ แต่ไม่มีเมท็อดนามธรรม `draw` ที่คลาส `Ball3D` จะได้ไหม คำตอบคือ ได้ ถือว่า `draw` ที่คลาสลูกเป็นเมท็อดใหม่ที่เพิ่มเข้าไป ไม่มีอะไรผิด สามารถใช้งานลูกบอลได้ตามปกติ แต่คุณสมบัติที่หายไป (ซึ่งเป็นคุณสมบัติเด่นของการรับทอด) คือ เมื่อมีตัวแปรแบบ `Ball3D` มาอ้างอิงอ็อบเจกต์ลูกบอลที่เป็น

แบบคลาสลูก ๆ จะไม่สามารถเรียกเมท็อด draw กับตัวแปรนั้นได้ เพราะ Ball3D ไม่มี draw ให้เรียก จึงไม่สามารถเขียนโปรแกรมแบบรหัสที่ 9-22 ได้

กล่าวโดยสรุป เราเขียนคลาสนามธรรมเพื่อบรรยายลักษณะสมบัติของประเภทข้อมูลตามข้อกำหนด โดยสามารถประกาศตัวแปร และเขียนเมท็อดที่คลาสนามธรรมเพื่อให้คลาสลูก ๆ ได้ใช้ร่วมกันไว้ที่คลาสนามธรรมที่เป็นแม่นี้ได้ สำหรับเมท็อดประจำอ็อบเจกต์ใดที่เป็นหนึ่งในลักษณะสมบัติตามข้อกำหนด แต่ไม่สามารถเขียนได้ที่คลาสแม่ ก็ให้เขียนเป็นเมท็อดนามธรรม เพื่อบังคับให้คลาสลูกเขียนรายละเอียดการทำงาน (หากไม่เขียน ตัวแปลโปรแกรมจะฟ้อง)

```
01 import jlab.graphics.*;
02
03 public abstract class Ball3D extends BallG {
04   protected double z, dz;
05   protected Ball3D(double r, double x, double y, double z) {
06     super(r, x, y);
07     this.z = z;
08     dz = -5 + (10 * Math.random());
09   }
10   public void move(DWindow3D w3) {
11     super.move(w3);
12     z += dz;
13     if (z <= 0 || z >= w3.getDepth()) dz = -dz;
14     z = Math.min(Math.max(z, 0), w3.getDepth());
15   }
16   protected void drawShadow(DWindow3D w3) {
17     double s = y / w3.getHeight();
18     w3.fillEllipse(DWindow.BLACK, x, w3.getHeight(), z,
19                    2*r*s, r/2*s);
20   }
21   public abstract void draw(DWindow3D w3);
22 }
```

ตัวสร้างนี้มีไว้ในลูกเรียก

Ball3D ต้องมีเมท็อด draw แต่จะวาดลูกบอลอย่างไร ปล่อยให้คลาสลูกจัดการ

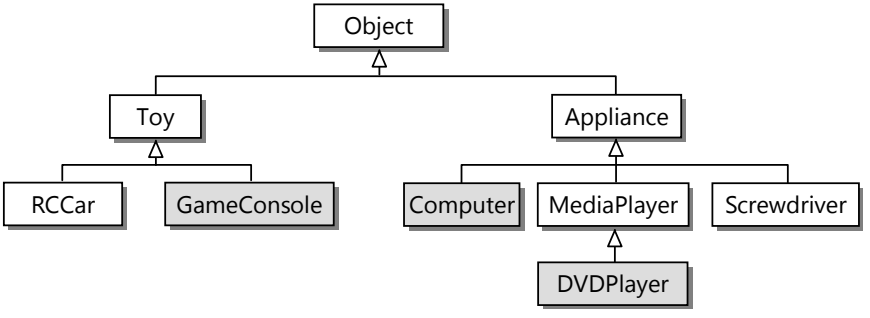
รหัสที่ 9-32 คลาส Ball3D เขียนเป็นคลาสนามธรรม

```
01 import jlab.graphics.*;
02
03 public class Ball3DColor extends Ball3D {
04   private int color;
05   public Ball3DColor(int c, double r, double x, double y,double z){
06     super(r, x, y, z);
07     color = c;
08   }
09   @Override public void draw(DWindow3D w3) {
10     drawShadow(w3);
11     w3.fillEllipse(color, x, y, z, 2*r, 2*r);
12   }
13 }
```

รหัสที่ 9-33 คลาส Ball3DColor เป็นคลาสลูกของ Ball3D ในรหัสที่ 9-32

อินเทอร์เฟซ

รูปที่ 9-12 แสดงตัวอย่างการรับทอดของคลาสจำนวนหนึ่ง ทางด้านซ้ายเป็นของเล่น ทางขวาเป็นอุปกรณ์ใช้ในบ้านสารพัดชนิด มีคลาส GameConsole, Computer และ DVDPlayer แทนอุปกรณ์ที่ใช้พลังงานไฟฟ้า กำหนดให้คลาสเหล่านี้มีเมท็อด sleep เพื่อให้การทำงานของอุปกรณ์เข้าสู่ภาวะการประหยัดพลังงาน



รูปที่ 9-12 แผนภาพการรับทอดของคลาสจำนวนหนึ่ง

หากเราต้องการเขียนอีกคลาสเพื่อผลิตตัวควบคุมการเปิดปิดอุปกรณ์ไฟฟ้าในบ้าน มีเมท็อด savePower ที่รับอาเรย์ของอุปกรณ์ไฟฟ้ามาสั่ง sleep ให้หมด เนื่องจากคลาสอุปกรณ์ไฟฟ้าข้างต้นเป็นเครือญาติห่าง ๆ กัน จึงต้องเขียนเมท็อด savePower รับพารามิเตอร์เป็นอาเรย์ของ Object เพื่อให้สามารถรับอ็อบเจกต์ของคลาสทั้งสามนี้ได้ดังรหัสที่ 9-34

```
public class HomeMonitor {
  ...
  public void savePower(Object[] obj) {
    for (int i=0; i<obj.length; i++) {
      if (obj[i] instanceof Computer)
        ((Computer)obj[i]).sleep();
      else if (obj[i] instanceof DVDPlayer)
        ((DVDPlayer)obj[i]).sleep();
      else if (obj[i] instanceof GameConsole)
        ((GameConsole)obj[i]).sleep();
    }
  }
```

ต้องตรวจสอบว่าเป็นประเภทใด จะได้ downcast ได้ถูกประเภท

รหัสที่ 9-34 savePower รับอาเรย์ของ Object เพื่อให้รับอุปกรณ์ไฟฟ้าหลากหลายชนิด

ถ้าจาวายอมให้มีการรับทอดจากหลาย ๆ คลาสได้ ก็เพียงแค่เขียนคลาสนามธรรมชื่อ PowerSaving ภายในมีเมท็อดนามธรรมชื่อ sleep จากนั้นเขียนให้คลาส GameConsole, Computer และ DVDPlayer มี PowerSaving นี้เป็นคลาสพ่ออีกคลาส (จากเดิมที่มีคลาสแม่) เพียงเท่านั้นก็ถือว่า ทั้งสามคลาสนี้เป็น PowerSaving ทำให้เขียนเมท็อด savePower

ให้รับอาเรย์ของ PowerSaving ได้ ดังรหัสที่ 9-35 ที่สั้นกะทัดรัด ไม่จำกัดอุปกรณ์ไฟฟ้าแค่สามชนิด รับได้ทุกชนิดที่เป็น PowerSaving แต่น่าเสียดายที่จาวาเป็นภาษาที่รับทอดทางเดียว (มีคลาสแม่ได้คลาสเดียว) จึงทำตามที่เสนอมาไม่ได้

```
public class HomeMonitor {
  ...
  public void savePower(PowerSaving[] dev) {
    for (int i=0; i<dev.length; i++)
      dev[i].sleep();
  }
```

รหัสที่ 9-35 savePower รับอาเรย์ของ PowerSaving เพื่อให้รับอุปกรณ์ที่มีเมท็อด sleep

อย่างไรก็ตาม จาวามี*อินเทอร์เฟซ* (interface) ที่ใช้แก้ปัญหาข้างต้นนี้ได้ ขอแสดงตัวอย่างเพื่อตอบคำถามว่า อินเทอร์เฟซคืออะไร รหัสที่ 9-36 แสดงอินเทอร์เฟซชื่อ PowerSaving สังเกตได้ว่า

- การเขียนอินเทอร์เฟซคล้ายการเขียนคลาส แต่ใช้คำว่า interface แทน class
- ภายในอินเทอร์เฟซมีได้แต่ค่าคงตัวและเมท็อดนามธรรมแบบ public

```
01 public interface PowerSaving {
02   public static final int MIN = 1;
03   public static final int MAX = 5;
04
05   public abstract void sleep();
06   public abstract void wakeup();
07   public abstract boolean isSleeping();
08 }
```

ค่าคงตัว

เมท็อดนามธรรม

รหัสที่ 9-36 อินเทอร์เฟซ PowerSaving

เนื่องจากเมท็อดในอินเทอร์เฟซต้องเป็น public abstract และตัวแปรก็ต้องเป็น public static final เท่านั้น ดังนั้น จาวาจึงอนุญาตให้เราเขียนแบบย่อได้ดังรหัสที่ 9-37 แต่ต้องเข้าใจด้วยว่า มีความหมายเช่นเดียวกับรหัสที่ 9-36 [7]

```
01 public interface PowerSaving {
02   int MIN = 1;
03   int MAX = 5;
04
05   void sleep();
06   void wakeup();
07   boolean isSleeping();
08 }
```

ค่าคงตัว เป็นแบบ public static final

เมท็อดนามธรรม เป็นแบบ public abstract

รหัสที่ 9-37 อินเทอร์เฟซ PowerSaving (เขียนแบบย่อ)

[7] เราสร้างอินเทอร์เฟซใน JLab ด้วยวิธีเดียวกับการสร้างคลาส แล้วค่อยเปลี่ยนคำว่า class เป็น interface

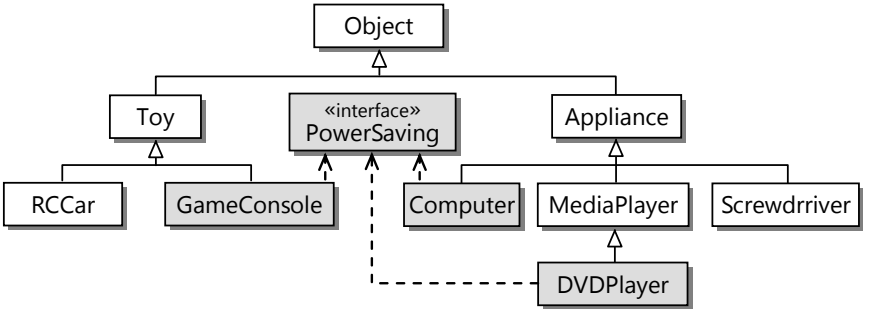
กลับมาที่ตัวควบคุมอุปกรณ์ไฟฟ้า เมื่อมีอินเทอร์เฟซ `PowerSaving` เราสามารถเขียนให้คลาสอุปกรณ์ไฟฟ้าทั้งหลายเป็นข้อมูลประเภท `PowerSaving` ได้ด้วยการเขียนต่อท้ายที่หัวคลาสว่า `implements PowerSaving` (ดังรหัสที่ 9-38) เพื่อประกาศว่า คลาสนี้ได้เขียนรายละเอียดของทุกเมท็อดที่ปรากฏเป็นเมท็อดนามธรรมในอินเตอร์เฟซ `PowerSaving` (ถ้าตัวแปลโปรแกรมไม่พบเมท็อดดังกล่าวในคลาส ก็จะแจ้งข้อผิดพลาด) เราเขียนความสัมพันธ์ระหว่างคลาสกับอินเทอร์เฟซด้วยเส้นประหัวลูกศรเป็นขีดดังแสดงในรูปที่ 9-13

```
public class GameConsole extends Toy implements PowerSaving {
  ...
}
```

```
public class Computer extends Appliance implements PowerSaving {
  ...
}
```

```
public class DVDPlayer extends MediaPlayer implements PowerSaving {
  ...
}
```

รหัสที่ 9-38 หัวคลาสแสดงการเขียนให้คลาสเป็น `PowerSaving`



รูปที่ 9-13 `GameConsole`, `Computer`, และ `DVDPlayer` เป็น `PowerSaving`

Comparable

`Comparable` เป็นอินเทอร์เฟซหนึ่งของจาวา ดังรหัสที่ 9-39 อินเทอร์เฟซนี้มีเมท็อดเดียวชื่อ `compareTo` ถ้า `a` เป็นอ็อบเจกต์แบบ `Comparable` การใช้ `a.compareTo(b)` จะคืนจำนวนเต็มที่แทนผลการเปรียบเทียบอ็อบเจกต์ `a` กับอ็อบเจกต์ `b` ดังนี้

- ถ้า `a.compareTo(b) < 0` หมายความว่า `a < b`
- ถ้า `a.compareTo(b) > 0` หมายความว่า `a > b`
- ถ้า `a.compareTo(b) = 0` หมายความว่า `a = b`

บทที่ 5 ได้นำเสนอการใช้ `compareTo` กับสตริง ก็เพราะ `String` ก็เป็นคลาสที่ปฏิบัติตาม `Comparable`

```
01 package java.lang;
02 public interface Comparable {
03   public int compareTo(Object obj);
04 }
```

รหัสที่ 9-39 อินเทอร์เฟซ Comparable

การที่คลาส String มีเมท็อด compareTo ก็น่าจะเพียงพอที่จะสามารถเรียกใช้เมท็อดนี้กับสตริงได้ โดยไม่จำเป็นต้องเขียนให้ String implements Comparable เลย แต่เหตุผลที่เขียนประกาศให้ String ปฏิบัติตาม Comparable ก็เพราะมีบริการมาตรฐานหลายตัวที่เขาจะรับข้อมูลแบบ Comparable เท่านั้นไปประมวลผล (เพราะจะได้มั่นใจว่า สามารถใช้เมท็อด compareTo ในระหว่างการประมวลผลได้) เช่น คลาสของจาวาชื่อ java.util.Arrays มีเมท็อด `public void sort(Object[] a)` ที่รับอาเรย์ของ Object โดยแต่ละช่องต้องเก็บอ็อบเจกต์แบบ Comparable [8] รหัสที่ 9-40 แสดงตัวอย่างการส่งอาเรย์ของสตริงไปเรียงลำดับในบรรทัดที่ 5 ส่วนบรรทัดที่ 6 เรียกบริการ toString ที่เป็นเมท็อดประจำคลาสของ Arrays ที่รับอาเรย์ไปแปลงเป็นสตริงเพื่อใช้แสดง ได้ผล [C#, Java, Ruby, Scala] แสดงทางจอภาพ

```
01 import java.util.Arrays;
02 public class SortString {
03   public static void main(String[] args) {
04     String[] s = {"Java", "C#", "Scala", "Ruby"};
05     Arrays.sort(s);
06     System.out.println(Arrays.toString(s));
07   }
08 }
```

รหัสที่ 9-40 ตัวอย่างการใช้งาน Arrays.sort กับสตริงที่เป็น Comparable

หากเราส่งอาเรย์ของลูกบอลให้ sort ย่อมเกิดปัญหา เพราะลูกบอลที่เราเขียนไม่ได้ปฏิบัติตามอินเทอร์เฟซ Comparable โดย sort จะโยนสิ่งผิดปกติ ClassCastException เพราะเขาไม่สามารถ downcast อ็อบเจกต์ที่รับให้เป็น Comparable ได้ระหว่างการทำงาน มาถึงตรงนี้ นึกถึงเรื่องลูกบอล ก็ขอยกอีกสักตัวอย่างเกี่ยวกับคลาส Ball3D หากผู้อ่านได้ลองสั่งลูกบอลสามมิติเด้งไปมาในห้องสักสามสี่ลูก จะเห็นว่า การวาดลูกบอลนั้นในบางจังหวะแลดูแปลก คือมีลูกบอลที่อยู่หลังห้องไกลจากคนดูมาบังลูกบอลที่อยู่ใกล้คนดู ซึ่งไม่เหมือนธรรมชาติที่ของใกล้ตัวบังของไกลออกไป อาการนี้มีสาเหตุมาจากวงวนแสดงลูกบอลในอาเรย์ที่เราหยิบลูกบอลทีละลูกมาแสดงไล่จากช่องที่ 0 ไปจนถึงช่องสุดท้าย ถ้าช่องทางซ้ายเก็บลูกที่อยู่ใกล้กว่าช่องทางขวา จะ

[8] ผู้อ่านคงสงสัยว่า ทำไมเมท็อด sort นี้ถึงไม่รับอาเรย์ของ Comparable เลยก็สิ้นเรื่อง ประเด็นนี้เป็นเรื่องของการออกแบบคลังคลาส ที่ผู้ออกแบบเห็นว่า มีอยู่อีกหลายบริการที่คืนอาเรย์ของ Object กลับมา ถ้าต้องการส่งไป sort ก็จะได้ส่งไปให้เลยโดยไม่ต้องทำ casting ให้ยุ่งยาก

เกิดปัญหาดังกล่าว ถ้าจะให้สมจริงมากขึ้น ควรหยิบลูกบอลมาวาดตามลำดับระยะทางที่ห่างเข้าไปในห้อง ลูกไกลวาดก่อนลูกใกล้ ลูกใกล้จะได้บังลูกไกล สรุปคือ เราต้องเรียงลำดับลูกบอลในอาเรย์ให้เรียงลำดับตามค่า z ของลูกบอลในอาเรย์ แล้วจึงนำลูกบอลแต่ละลูกไปวาดตามลำดับค่า z จากมากไปน้อย รหัสที่ 9-41 แสดงคลาส Ball3D ที่ปรับปรุงให้ปฏิบัติตาม Comparable และรหัสที่ 9-42 แสดงการวาดลูกบอลตามลำดับไกลมาใกล้

```
import jlab.graphics.*;

public abstract class Ball3D extends BallG implements Comparable {
  protected double z, dz;

  public int compareTo(Object obj) {
    Ball3D that = (Ball3D) obj;
    if (this.z == that.z) return 0;
    if (this.z < that.z) return -1;
    return 1;
  }
  protected Ball3D(double r, double x, double y, double z) { ... }
  public void move(DWindow3D w3) { ... }
  protected void drawShadow(DWindow3D w3) { ... }
  public abstract void draw(DWindow3D w3);
}
```

สัญญาว่าจะปฏิบัติตามอินเทอร์เฟซ Comparable

รับมาเป็น Object ต้อง downcast เป็น Ball3D ก่อนใช้งาน

ใช้ z เป็นเกณฑ์ในการเปรียบเทียบ

รหัสที่ 9-41 คลาส Ball3D เขียนให้ปฏิบัติตามอินเทอร์เฟซ Comparable

```
01 import jlab.graphics.*;
02 import java.util.Arrays;
03 public class TestBall3D {
04   public static void main(String[] args) {
05     int h = 200, h2 = h/2;
06     DWindow3D w = new DWindow3D(h, h, h);
07     Ball3D[] balls = new Ball3D[] {
08             new Ball3DColor(DWindow.BLUE, 15, h2, h2, h2),
09             new Ball3DImage("c:/java101/fball.gif", h2, h2, h2) };
10     w.setRepaintDuringSleep(true);
11     while (true) {
12       w.clearBackground();
13       w.drawFrame(50);
14       Arrays.sort(balls);
15       for (int i = balls.length-1; i >= 0; i--) {
16         balls[i].draw(w);
17         balls[i].move(w);
18       }
19       w.sleep(50);
20     }
21   }
22 }
```

เรียงลำดับลูกบอลตามค่า z จากน้อยไปมาก

หยิบลูกบอลมาวาด ลูกไกลก่อนลูกใกล้

รหัสที่ 9-42 ตัวอย่างโปรแกรมแสดงลูกบอลที่มีการวาดลูกไกลก่อนลูกใกล้

ข้อดีของอินเทอร์เฟซประการหนึ่ง คือ เราเขียนให้คลาสหนึ่งปฏิบัติตามอินเทอร์เฟซหลายตัวได้ เช่น เขียนหัวคลาสของ GameConsole เป็น

```
class GameConsole extends Toy implements PowerSaving, Comparable
```

ทำให้มองอ็อบเจกต์ของคลาสนี้เป็นประเภท PowerSaving ก็ได้ เป็น Comparable ก็ได้ (นอกจากจะเป็น GameConsole เป็น Toy และเป็น Object ตามสายพันธุ์การรับทอด) เมื่อประกาศว่าจะเป็น Comparable ก็ต้องเขียนเมท็อด compareTo เพิ่มในคลาสให้สมบูรณ์ด้วย [9] (โดยจะใช้กำลังไฟของอุปกรณ์เป็นตัวเปรียบเทียบ หรือจะใช้ราคา หรือใช้อายุการใช้งาน ก็คงขึ้นกับปัญหาที่สนใจ)

ศัพท์ OO

เรื่องของคลาส อ็อบเจ็กต์ และการรับทอดที่ได้นำเสนอมาเป็นส่วนหนึ่งของแนวคิดการเขียนโปรแกรมเชิงวัตถุ (object-oriented programming – OOP) ที่ผ่านมาไม่ได้นำเสนอเนื้อหาด้วยศัพท์เทคนิค จึงขอสรุปศัพท์ต่าง ๆ ที่ใช้กันในวงการพอสังเขปดังนี้

- *Encapsulation* : (แปลแบบตรงไปตรงมาว่า การจับใส่แคบซูล) หมายถึงแนวคิดที่เราปกปิดรายละเอียดการเก็บข้อมูลและวิธีการทำงานของเมท็อดภายในอ็อบเจกต์ ไม่ให้ผู้อื่นนอกคลาสเห็น เปิดเผยเฉพาะหน้าที่ที่อ็อบเจกต์ให้บริการ วิธีเรียกใช้ (ซึ่งคือเฉพาะหัวเมท็อด) ก็พอ ทำให้เรามีอิสระที่จะเปลี่ยนการทำงานและโครงสร้างการจัดเก็บข้อมูลภายในอ็อบเจกต์ในภายหลังได้ โดยไม่กระทบต่อผู้ใช้บริการ
- *Polymorphism* : (แปลตามพจนานุกรมราชบัณฑิตฯ ว่า *ภาวะพหุสัณฐาน* poly – พหุ แปลว่า หลาย, morph – สัณฐาน แปลว่า รูปร่าง ลักษณะ) หมายถึง การมองอ็อบเจกต์ในหลากหลายลักษณะได้ เช่น เรามองอ็อบเจกต์ของ Basenji เป็น Basenji, เป็น Dog, เป็น Animal หรือเป็น Object ได้ทั้งสิ้นตามลำดับชั้นการรับทอด ทั้งนี้จะมองเป็นอะไรก็ขึ้นกับว่า เรานำตัวอ้างอิงแบบใดมาอ้างอิง แต่ต้องอย่าลืมว่า เมื่อใช้ตัวอ้างอิงแบบใด ก็เรียกใช้เมท็อดได้เฉพาะที่ปรากฏในคลาสของตัวอ้างอิงนั้น เช่น ถ้าใช้แบบ Animal ก็เรียกใช้เมท็อดเฉพาะที่ Animal มี (เพราะได้สวมบทเป็น Animal อยู่)
- *Overriding* : หมายถึงการเปลี่ยนพฤติกรรมที่คลาสลูกต้องการให้ทำต่างจากของที่รับทอดมา แต่ยังคงมีวิธีเรียกใช้เมท็อดเหมือนกัน นั่นคือ การเขียนเมท็อดที่คลาสลูก เพื่อแทนเมท็อดที่รับทอดมาจากคลาสแม่

[9] ต้องขอย้ำตรงนี้อีกครั้งว่า การเพิ่มเพียงเมท็อด public int compareTo(Object a) ให้กับคลาสหนึ่งไม่ได้ทำให้คลาสนั้นเป็น Comparable สิ่งที่ต้องทำคือ ต้องเขียน implements Comparable ต่อท้ายที่หัวคลาส (เพื่อประกาศว่าจะปฏิบัติตาม) และเขียนเมท็อด compareTo ให้สมบูรณ์

- *Late binding* : หมายถึง การเลือกเมท็อดประจำอ็อบเจกต์ที่จะเรียกใช้งาน จะกระทำขณะโปรแกรมทำงานถึงคำสั่งที่จะเรียกเมท็อดนั้น เช่น ถ้าถามว่า คำสั่ง `a.do()` จะไปทำที่เมท็อดใด แน่นอนว่า ต้องไปทำที่เมท็อด `do` แต่ก็อาจมี `do` อยู่หลายที่ แล้วจะเลือก `do` ใดทำงาน การตัดสินใจนี้ จะไปคิดตอนทำงานจริงว่า `a` กำลังอ้างอิงอ็อบเจกต์คลาสใด ก็ไปทำเมท็อด `do` ของคลาสนั้น (โดยไม่พิจารณาจากคลาสของตัวอ้างอิง `a`) เพราะด้วยแนวคิดของ polymorphism ทำให้ตัวอ้างอิงหนึ่งสามารถอ้างอิงอ็อบเจกต์ได้หลากหลายชนิด ผนวกกับแนวคิดของ overriding ทำให้เมท็อดของคลาสแม่ถูกแทนที่ได้ด้วยเมท็อดใหม่ของคลาสลูกได้ แสดงว่าเมท็อด `do` ที่เขียนนั้นอาจมีหลายพฤติกรรมขึ้นกับว่าอ็อบเจกต์ที่ถูกอ้างอิงเป็นอะไรกันแน่ late binding จึงกำหนดว่าให้ตัดสินใจเลือกเมท็อดขณะทำงาน จึงจะรู้ประเภทของอ็อบเจกต์ ซึ่งต่างจาก early binding เช่น คำสั่ง `Math.sin(x)` ที่ตัวแปลโปรแกรมกำหนดได้เลยขณะแปลโปรแกรมว่า ต้องไปทำที่เมท็อด `sin` ในคลาส `Math` คำว่า bind จึงสะท้อนถึงการผูกมัดชื่อเมท็อดกับตัวเมท็อด ส่วน late หมายถึงรอไปคิดตอนทำงาน ส่วน early หมายถึงรู้ได้ตั้งแต่ตอนแปลโปรแกรม (บางคนเรียน late binding ว่า dynamic binding และเรียก early binding ว่า static binding)
- *Inheritance* : แปลว่า การรับทอด ซึ่งเป็นหลักการการสร้างคลาสลูกที่รับทอดลักษณะสมบัติจากคลาสแม่ที่เราได้นำเสนอกันมาในบทนี้

แบบฝึกหัด

1. จงวาดแผนผังคลาสเพื่อแสดงระดับชั้นการรับทอดที่ควรเป็นระหว่างคลาสต่าง ๆ ในแต่ละข้อต่อไปนี้

   - (สำหรับผู้สนใจรูปทรงเรขาคณิต) `Shape`, `Oval`, `Polygon`, `PolyLine`, `Circle`, `Ellipse`, `Rectangle`, `Pentagon`, `Triangle`, `Square`, `Quadrilateral`, `Trapesoid`, และ `Parallelogram`
   - (สำหรับผู้สนใจยานพาหนะ) `Vehicle`, `Car`, `Truck`, `Sedan`, `Coupe`, `Minivan`, `PickupTruck`, `Motocycle`, `Bicycle` และ `SportUtilityVehicle`
   - (สำหรับผู้สนใจสัตว์โบราณ) `Archosaurs`, `Thecodonts`, `Saurischians`, `Pterosaurs`, `Dinosaurs`, `Crocodilians` และ `Ornithischians`

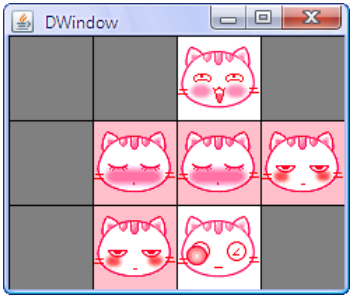
2. จงเขียนเมท็อด `equals` แทนของที่รับทอดจาก `Object` ให้กับคลาส `BloodPressure`, `PiggyBank` และ `Time` ที่ได้เขียนในแบบฝึกหัดของบทที่ 8 (ดูรหัสที่ 9-26 เป็นแนวทาง)

3. คลาส `Point`, `Line`, `Rational`, และ `IntSet` ในแบบฝึกหัดของบทที่ 8 มีเมท็อด `equals` ที่เขียนหัวเมท็อดไม่ตรงกับของที่ได้รับจากคลาส `Object` จึงเป็นการเขียนเมท็อดเพิ่ม จงเขียนเมท็อด `equals` ให้กับคลาสต่าง ๆ ข้างต้น เพื่อแทนของที่ได้รับจาก `Object`

4. จงเขียนคลาสใหม่ชื่อ `SavingAccount` ให้เป็นคลาสลูกของ `BankAccount` (รหัส 8-12) คลาสใหม่นี้แทนบัญชีออมทรัพย์ซึ่งมีการให้ดอกเบี้ย จึงต้องมีเมท็อดเพื่อตั้งดอกเบี้ยต่อปี และเมท็อดคำนวณดอกเบี้ยที่ได้รับ

5. จงเขียนคลาสใหม่ชื่อ `OverdraftedAccount` ให้เป็นคลาสลูกของ `BankAccount` (รหัส 8-12) คลาสใหม่นี้แทนบัญชีธนาคารที่เจ้าของบัญชีได้ทำข้อตกลงกับธนาคารว่าให้สามารถถอนเงินเกินยอดในบัญชีได้ (โดยจำกัดวงเงินไว้ตามที่ตกลงกันไว้)

6. รหัสที่ 8-24 และ 8-25 ในบทที่ 8 นำเสนอคลาส `Card` และ `MemoryGame` ประกอบกันเป็นโปรแกรมเกมจับคู่ทดสอบความจำ คลาส `Card` มีไว้ผลิตอ็อบเจกต์แทนบัตรตัวเลขให้จำเพื่อจับคู่ แบบฝึกหัดข้อนี้ต้องการให้เขียนคลาสใหม่ชื่อ `ImageCard` สำหรับผลิตบัตรที่แสดงรูปแทนตัวเลข ดังตัวอย่างในรูปทางขวา มีรายละเอียดดังนี้



   - เขียน `ImageCard` ให้เป็นคลาสลูกของ `Card`
   - เขียนตัวสร้างให้เหมือนกับของ `Card` ภายในนี้มีการอ่านแฟ้มภาพเตรียมไว้ก่อน โดยใช้คลาส `DImage` (ที่ได้นำเสนอตัวอย่างการใช้งานใน รหัสที่ 9-21) รูปหัวแมวต่างๆ ที่แสดงในตัวอย่างได้จากแฟ้ม 1.gif ถึง 10.gif ใน c:\java101 โดยบัตรหมายเลข 1, 2, ... แสดงภาพจากแฟ้ม 1.gif, 2.gif, ... ตามลำดับ
   - เขียนเมท็อด `draw` ใหม่ใน `ImageCard` เพื่อแทนของที่ได้รับจาก `Card` และใน `draw` นี่เองให้แสดงภาพแทนตัวเลข
   - เนื่องจากคลาส `Card` ที่เขียนไว้ก่อนหน้านี้ ไม่ได้เผื่อไว้ให้ใครมาเป็นคลาสลูก จึงกำหนดลักษณะของตัวแปรประจำอ็อบเจกต์ และเมท็อด `draw` ไว้แบบ `private` ให้ปรับปรุงคลาส `Card` เพื่อให้เหมาะกับการเขียน `ImageCard` ด้วย
   - สำหรับการทดสอบ สามารถเปลี่ยนแค่บรรทัดที่สร้าง `Card` ใน `MemoryGame` ให้เป็นสร้าง `ImageCard` เท่านั้นก็เพียงพอ (เพียงหนึ่งบรรทัดเท่านั้น ลองพิจารณาดูว่าควรเป็นบรรทัดใดในรหัสที่ 8-25)

7. จากคลาส A และ B ข้างล่างนี้ ถ้าสั่ง `new B();` กับ `new B("x");` จะได้ผลลัพธ์อะไร

```
public class A {
  public A() {

System.out.println("A");
  }
  public A(String s) {
    this();
    System.out.println(s);
  }
}
```

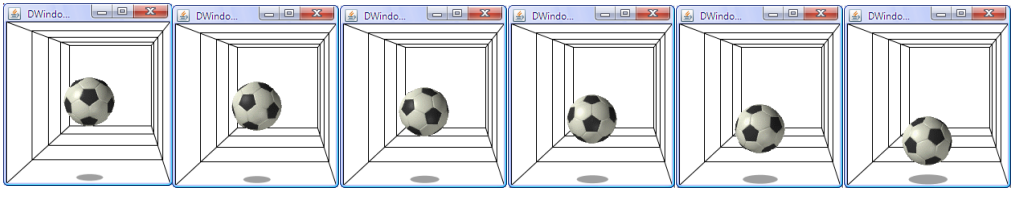
```
public class B extends A {
  public B() {
    super("z");
    System.out.println("B");
  }
  public B(String s) {
    System.out.println(s);
  }
}
```

8. การเขียนคลาส A และ B ข้างล่างนี้ มีที่ผิดหรือไม่ จงอธิบาย

```
public class A {
  public A(int a) {}
  public void a() {}
}
```

```
public class B extends A {
  public B(int b) {}
  public void a() {}
}
```

9. จงเขียนคลาส `Ball3DAnimatedImages` ให้เป็นคลาสลูกของ `Ball3D` เพื่อแสดงลูกบอลเคลื่อนที่ในห้องสามมิติ โดยภาพลูกบอลที่แสดงจะเปลี่ยนแปลงขณะที่เปลี่ยนตำแหน่งด้วย ดังตัวอย่างข้างล่างนี้แสดงลำดับภาพลูกบอลที่หมุนขณะเคลื่อนที่

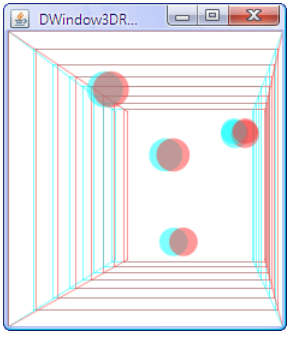


การแสดงภาพลูกบอลหมุนในตัวอย่างทำได้โดยการอ่านภาพการเปลี่ยนแปลงของลูกบอลไว้ล่วงหน้าในตัวสร้าง จากนั้นเปลี่ยนภาพที่ใช้แสดงทุกครั้งในเมท็อด `draw` จากตัวอย่าง เราอ่านภาพจากแฟ้ม ball1.gif, ball2.gif, ..., ball6.gif  (ใน c:\java101) มาเก็บเป็นอ็อบเจกต์ของ `DImage` ไว้ก่อน (จำนวน 6 ภาพ) แล้ววนนำภาพทั้งหกมาแสดงตามลำดับ (ข้อแนะนำ : คงต้องนึกถึง `DImage[]`)

10. จงปรับปรุงเมท็อด `drawFrame` ในคลาส `DWindow3D` ให้แสดงเส้นพุ่งเข้าตามกำแพง พื้น และเพดาน ดังแสดงในรูปทางขวานี้

11. เมท็อด `drawShadow` ในคลาส `Ball3DImage` (และคลาสอื่นๆ) แสดงเงาของลูกบอลด้วยวงรี ที่มีขนาดแปรตามค่า `y` ของลูกบอล (ขนาดยิ่งใหญ่เมื่อยิ่งเข้าใกล้พื้นห้อง) เพื่อให้สมจริงมากขึ้น จงปรับปรุงให้วาดวงรีด้วยสีเทาที่มีระดับความเข้มแปรตามค่า `y` ด้วย (ยิ่งใกล้พื้นสียิ่งดำ)

12. จงเขียนคลาสใหม่ชื่อ `DWindow3DAnaglyph` เป็นคลาสลูกของ `DWindow3D` ที่สามารถแสดงภาพสามมิติซึ่งประกอบด้วยภาพสำหรับตาซ้ายซ้อนกับภาพสำหรับตาขวา โดยภาพที่ให้ตาซ้ายดูแสดงด้วยสีฟ้า ภาพที่ให้ตาขวาดูแสดงด้วยสีแดง เมื่อผู้มองสวมแว่นตาที่มีกระดาษแก้วสีแดงที่ตาซ้ายและสีฟ้าที่ตาขวา กระดาษแก้วของตาแต่ละข้างก็จะกรองภาพของตาอีกฝั่งออก ตาสองตาจึงมองภาพคนละภาพ หากภาพของตาแต่ละข้างแสดงได้อย่างเหมาะสม ผู้มองก็จะรับรู้ถึงความลึกของวัตถุในจอภาพ



รูปทางขวาแสดงตัวอย่างของ `DWindow3DAnaglyph` (แต่น่าเสียดายที่เป็นภาพสีเทา ☹ แค่ต้องการให้เห็นว่าถ้ามองแบบไม่ใส่แว่น จะเห็นภาพซ้อน ๆ กัน ผู้อ่านควรต้องเขียนคลาสนี้ให้สำเร็จ จึงจะได้เห็นความลึกของห้องจริง)

`DWindow3D` มีตัวแปร x_C และ y_C ซึ่งเป็นพิกัดบนจอที่ตามองตรงไป ค่าทั้งสองถูกใช้ในการแปลงพิกัด (x, y, z) ให้ห้อง ไปเป็นพิกัดของระนาบบนจอภาพ สำหรับคลาสใหม่นี้เรามีสองตา แต่ละตาอยู่คนละตำแหน่ง จึงต้องมี x_C และ y_C แยกกันสำหรับตาทั้งสอง แต่เนื่องจากตาอยู่ในระดับความสูง (y_C) เดียวกัน ความแตกต่างจึงอยู่แค่ค่าของ x_C ดังนั้น วิธีทำแบบง่ายคือ จากค่า x_C และ y_C ที่มีกำกับ `DWindow3D` อยู่แล้ว เราใช้ $(x_C - \Delta)$ กับ y_C เพื่อคำนวณพิกัดให้ตาซ้าย และใช้ $(x_C + \Delta)$ กับ y_C เพื่อคำนวณพิกัดให้ตาขวา (โดยที่ Δ คือครึ่งหนึ่งของระยะห่างระหว่างตาทั้งสอง) สรุปสิ่งที่ต้องทำมีดังนี้

- เขียนคลาส `DWindow3DAnaglyph` ให้เป็นคลาสลูกของ `DWindow3D`
- เขียนตัวสร้าง
- เขียนเมท็อด `drawLine` ใหม่แทนของที่รับจากคลาสแม่ โดยต้องลากสองเส้น เส้นสำหรับตาซ้าย อีกเส้นสำหรับตาขวา
- เขียนเมท็อด `fillEllipse` ใหม่แทนของที่รับจากคลาสแม่ โดยต้องวาดวงรีสองวง วงหนึ่งสำหรับตาซ้าย อีกวงสำหรับตาขวา
- ตั้งค่าคงตัวประจำคลาสสองค่าสำหรับสีแดงและฟ้าเพื่อใช้ในการวาด ดังนี้

```
static final int RED  = DWindow.mixARGB(128, 255, 50, 50);
static final int CYAN = DWindow.mixARGB(128, 0, 255, 255);
```

(การผสมสีทั้งสองข้างต้นนี้มีการตั้งค่าความทึบของสีด้วย โดยเราตั้งให้ความทึบเป็น 128 จาก 255 จึงหมายความว่า ทึบประมาณ 50% หรืออีกนัยหนึ่งคือโปร่งประมาณ 50% ที่ต้องทำเช่นนี้ก็เพราะว่า ต้องการให้สีของเส้นหรือวงกลมที่วาดซ้อนทับกัน ผสมกัน ให้สังเกตว่า เราไม่ได้ใช้สีน้ำเงินสด แต่เป็นการผสมสีน้ำเงินสดกับเขียวสด เพราะได้สีที่แสดงบนจอภาพที่ใช้ได้ดีกับกระดาษแก้วสีฟ้าที่ผู้เขียนใช้ อย่างไรก็ตาม การตั้งสีนี้อาจต้องปรับค่าให้เหมาะกับกระดาษแก้วสีที่ใช้กับแว่นตา)

13. จงเขียนคลาส `BinaryLEDWindow` ให้เป็นคลาสลูกของ `IntegerLEDWindow` (รหัสที่ 9-7) โดยเขียนเมท็อด `setValue` เพื่อแทนของคลาสแม่ ให้คลาสใหม่นี้มีไว้ผลิตอ็อบเจกต์ที่แสดงจำนวนเต็มเป็นตัวเลขฐานสอง

14. จงเขียนให้คลาส `Rational`, `Time`, `PiggyBank`, `BloodPressure` เป็นประเภท `Comparable` (คือเขียนให้คลาสเหล่านี้ `implements Comparable` โดยเขียนเมท็อด `compareTo` เพื่อเปรียบเทียบความน้อยกว่า มากกว่า และเท่ากันของอ็อบเจกต์)

15. จาวามีคลาสมาตรฐานชื่อ `ArrayList` อยู่ในแพ็กเกจ `java.util` มีไว้ผลิตอ็อบเจกต์ซึ่งทำตัวเสมือนเป็นที่เก็บข้อมูลแบบอาเรย์ แต่เป็นอาเรย์ที่มีขนาดปรับได้ตามข้อมูลที่เพิ่มเข้าไป รหัสข้างล่างนี้แสดงตัวอย่างการใช้งาน สรุปเมท็อดคร่าว ๆ ดังนี้

    - `add(x)` : เพิ่ม `x` ต่อท้าย
    - `add(i,x)` : เพิ่ม `x` ที่ดัชนี `i` (ข้อมูลเดิมตั้งแต่ `i` เป็นต้นไป ถูกดันไปทางขวา)
    - `remove(i)` : ลบข้อมูลที่ดัชนี `i` (ข้อมูลตั้งแต่ `i+1` เป็นต้นไป ถูกดันมาทางซ้าย)
    - `size()` : คืนจำนวนข้อมูลที่เก็บอยู่
    - `set(i,x)` : เปลี่ยนข้อมูลที่ดัชนี `i` ให้เป็น `x`
    - `get(i)` : คืนข้อมูลตัวที่ดัชนี `i`

```
ArrayList a = new ArrayList();  // []
a.add("A");     // ["A"]
a.add("B");     // ["A", "B"]
a.add(0,"X");   // ["X", "A", "B"]
a.add(0,"Y");   // ["Y", "X", "A", "B"]
a.remove(1);    // ["Y", "A", "B"]
a.set(1,"Z");   // ["Y", "Z", "B"]
String x = (String) a.get(a.size()-1); // x = "B"
```

อาจสงสัยว่าทำไมตอนใช้ `get` ต้องมี `(String)` ด้านหน้าด้วย ต้องเข้าใจว่า `ArrayList` เป็นคลาสที่ผู้ออกแบบตั้งใจให้เก็บข้อมูลที่เป็นอ็อบเจกต์แบบใดก็ได้ในจาวา ดังนั้น ผู้ออกแบบจึงเขียน `add` ให้มีหัวเมท็อดเป็น `add(Object x)` เพราะ `Object` เป็นคลาสบรรพบุรุษของทุกคลาสในจาวา อ็อบเจกต์ใด ๆ ในจาวาจึงถือว่าเป็นประเภท `Object` ด้วย ทำให้ต้องเขียนเมท็อด `get(i)` ให้มีหัวเป็น `Object get(int i)` จะได้คืนข้อมูลได้ทุกประเภท แล้วผลักภาระให้นักเขียนโปรแกรมเปลี่ยนประเภทกลับไปสู่ประเภทที่ตนเองใช้ จากรหัสข้าง-ต้น เราเก็บสตริงใน `a` เวลา `get` กลับมา จึงต้อง downcast กลับมาเป็น `String`

จงเขียนคลาส `ArrayList` ของตัวเองแบบง่าย ๆ ที่ให้บริการแค่ 6 เมท็อดข้างต้นกับตัวสร้างแบบไม่รับพารามิเตอร์อีก 1 ตัว (ข้อแนะนำ : ภายในใช้อาเรย์ของ `Object` เป็นที่เก็บข้อมูล และมีตัวแปรแบบ `int` อีกตัวไว้เก็บจำนวนข้อมูล โดยให้ขยายอาเรย์เมื่อจำเป็นใน `add`)

9 สิ่งผิดปกติ

ซอฟต์แวร์ต้องทำงานได้ครบถ้วนถูกต้องตามข้อกำหนด และต้องมีความทนทานต่อสิ่งผิดปกติทั้งหลายที่ตรวจสอบไม่พบขณะพัฒนาโปรแกรม แต่มีโอกาสเกิดขึ้นระหว่างการทำงาน เช่น ไม่พบแฟ้มที่ต้องการประมวลผล ไม่สามารถเชื่อมต่อกับอินเทอร์เน็ตเพื่อรับส่งข้อมูล หรือรับข้อมูลจากผู้ใช้ที่มีรูปแบบไม่ตรงตามข้อกำหนด เป็นต้น หากไม่มีการจัดการสิ่งผิดปกติเหล่านี้ โปรแกรมอาจทำงานผิดพลาดหรือหยุดทำงานอย่างกะทันหัน ดังนั้น โปรแกรมจึงต้องมีภาระเสริมสำคัญในการแจ้งสิ่งผิดปกติทันที่ที่พบ หรือจัดการสิ่งผิดปกติที่อยู่ในขอบเขตรับผิดชอบของตน เพื่อไม่ให้สิ่งผิดปกติเหล่านี้หลุดรอดออกไปกลายเป็นข้อผิดพลาดของระบบ บทนี้นำเสนอประเภทของสิ่งผิดปกติ การสร้างสิ่งผิดปกติประเภทใหม่ ๆ คำสั่งในการสร้าง การโยน และการจัดการกับสิ่งผิดปกติ พร้อมด้วยตัวอย่างที่แสดงให้ถึงการโยนและการจัดการสิ่งผิดปกติเพื่อเพิ่มความทนทานให้กับโปรแกรม

สิ่งผิดปกติเกิดขึ้นได้เสมอ

โปรแกรมที่เราได้เขียนและลองสั่งทำงานกันมานั้น อาจแสดงพฤติกรรมแปลก ๆ หรือหยุดทำงานทันที่เมื่อใช้งานไม่ตรงตามเงื่อนไขที่กำหนดไว้ เราเรียกอาการไม่ปกติทั้งหลายที่เกิดขึ้นระหว่างการทำงานของโปรแกรมว่า *สิ่งผิดปกติ* (exception) ซึ่งอาจมีสาเหตุมาจาก

- จุดบกพร่องของตัวโปรแกรม เช่น มีการสร้างอาเรย์เป็นจำนวนลบ
- ข้อจำกัดของระบบ เช่น หน่วยความจำของเครื่องที่ทำงานมีน้อยเกินไป

- สภาพแวดล้อมระหว่างการทำงานไม่เป็นไปตามที่คาดหวังไว้ เช่น ต้องการอ่านเว็บเพจจากเครื่องแม่ข่ายในอินเทอร์เน็ต แต่มีปัญหาการสื่อสารผ่านเครือข่าย

เราในฐานะนักเขียนโปรแกรมคงป้องกันอะไรไม่ได้กับสิ่งผิดปกติที่มีสาเหตุมาจากข้อจำกัดของระบบ แต่นักเขียนโปรแกรมคงต้องรับผิดเต็ม ๆ กับสิ่งผิดปกติที่มีสาเหตุมาจากจุดบกพร่องของโปรแกรม นักเขียนโปรแกรมต้องระมัดระวังอย่างที่สุดไม่เขียนโปรแกรมผิด แต่คงปฏิเสธไม่ได้ว่า ซอฟต์แวร์ที่พัฒนากันมานั้นล้วนมีข้อผิดพลาด ปัญหาไม่ได้อยู่ที่ว่า ไม่รู้จะแก้ไขข้อผิดพลาดอย่างไร แต่ปัญหาอยู่ที่ว่า ไม่รู้ว่าที่ใดในโปรแกรมที่ทำงานผิดพลาด บางโปรแกรมทำงานผิด ณ จุดหนึ่ง แต่ไม่รู้ตัว หรือละเลยไม่ตรวจสอบ ยังคงทำงานต่อไปเรื่อย ๆ แล้วค่อยมีอาการแสดงออกให้ผู้ใช้ทราบในภายหลังว่า เกิดข้อผิดพลาดในการประมวลผล ซึ่งอาจสายเกินไปที่จะรู้ว่าผิดที่ใด และผิดเพราะสาเหตุใด สิ่งที่นักเขียนโปรแกรมพึงกระทำ คือ ตรวจสอบสถานะต่าง ๆ ระหว่างการทำงานเสมอว่า ตรงตามข้อกำหนดของการทำงานหรือไม่ ถ้าไม่ตรง ต้องรีบแจ้งความผิดปกติที่พบทันที (นักเขียนโปรแกรมมักมั่นใจว่า เหตุการณ์เช่นนั้นเช่นนี้ไม่เกิดแน่ ๆ ก็เลยไม่ตรวจสอบ แต่ก็มาพบในภายหลังว่า ความมั่นใจนั้นไม่จริง) ในจาวา เราสามารถแจ้งให้ระบบทราบถึงความผิดปกติด้วยการสร้างและโยนสิ่งผิดปกติที่มีรูปแบบที่จะช่วยในการวิเคราะห์หาสาเหตุของความผิดพลาดว่า ผิดอะไร ผิดอย่างไร และผิดที่ใด (เราจะได้นำเสนอในรายละเอียดกันในหัวข้อถัดๆ ไป)

สำหรับสิ่งผิดปกติที่มีสาเหตุจากปัจจัยที่อยู่นอกการควบคุมของโปรแกรมและของระบบ สิ่งที่ต้องทำคือ การเขียนชุดคำสั่งรองรับสถานการณ์ที่ผิดปกติเหล่านี้ ซึ่งประกอบด้วยการดักจับสิ่งผิดปกติที่อาจเกิดขึ้น และหาทางจัดการกับเหตุการณ์ดังกล่าว เพื่อให้โปรแกรมทนทานต่อสิ่งผิดปกติที่มีโอกาสเกิดขึ้นได้จากปัจจัยต่าง ๆ ถึงแม้ว่าจะต้องจบการทำงานของโปรแกรม ก็ต้องจบอย่าง “สง่างาม” คือจบอย่างที่ผู้ใช้รู้สาเหตุ และลดภาระการกู้คืนระบบให้มากที่สุดเท่าที่จะมากได้ เช่น โปรแกรมหาค่าเฉลี่ย (รหัส 3-12) ที่วนรับข้อมูลด้วยการเรียกเมท็อด `nextDouble` ของตัวอ่านแป้นพิมพ์แบบ `Scanner` หากผู้ใช้ป้อนข้อมูลไปได้ 20 จำนวน พอจะป้อนจำนวนถัดไปเผลอใส่ผิดเป็นตัวอักษร เมท็อด `nextDouble` จะโยนสิ่งผิดปกติ ถ้าโปรแกรมไม่มีการจัดการกับสิ่งผิดปกติแบบนี้ โปรแกรมจะหยุดทำงานทันที ข้อมูลที่อุตส่าห์ป้อนเข้าไปก็หายหมด เราในฐานะนักพัฒนาซอฟต์แวร์จะบอกว่า เป็นความผิดของผู้ใช้ที่ป้อนตัวอักษรซึ่งไม่ใช่จำนวน ก็ดูจะไม่ค่อยรับผิดชอบเท่าไร หากปล่อยให้เป็นเช่นนี้ ก็จะไม่มีใครมาใช้โปรแกรมนี้เป็นแน่ ดังนั้น นอกจากจะต้องเขียนโปรแกรมให้ทำงานตามข้อกำหนดแล้ว (เช่น หาค่าเฉลี่ยของชุดข้อมูลที่ได้รับ) นักเขียนโปรแกรมต้องคาดการณ์ด้วยว่า จะเกิดสิ่งผิดปกติระหว่างการทำงานที่จุดใดบ้างในโปรแกรม จะได้ตรวจสอบและจัดการกับเหตุการณ์ที่เกิดขึ้น เพื่อให้การใช้งานราบรื่นมากที่สุด ในกรณีของโปรแกรมข้างต้น เมื่อพบว่ามีการป้อนข้อมูลผิดรูปแบบ ก็ควรแจ้งให้ผู้ใช้ทราบถึงข้อผิดพลาดของผู้ใช้ และเปิดโอกาสให้ผู้ใช้ป้อนใหม่ เพื่อประมวลผลข้อมูลต่อไป ย่อมได้โปรแกรมที่มีความทนทานต่อเหตุการณ์ผิดปกติที่อาจเกิดขึ้น

try - catch

มาดูสักตัวอย่างให้เห็นเป็นรูปธรรมว่า จะรู้ได้อย่างไรว่าเกิดสิ่งผิดปกติ และจะจัดการกับสิ่งผิดปกติที่เกิดขึ้นได้อย่างไร พิจารณาส่วนของโปรแกรมหาค่าเฉลี่ยที่กล่าวในหัวข้อที่แล้ว มีการสร้างตัวอ่านแป้นพิมพ์ และวงวนการอ่านจำนวนจริงจากผู้ใช้ เพื่อหาผลรวม ก่อนไปหาค่าเฉลี่ยดังนี้

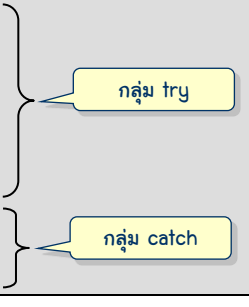
```
Scanner kb = new Scanner(System.in);
double sum = 0;
for (int i=0; i<n; i++) {
  System.out.print("data = ");
  sum += kb.nextDouble();
}
System.out.println("avg. = " + (sum/n));
```

ถ้าระหว่างการรอรับจำนวน ผู้ใช้ป้อนค่าที่ไม่ใช่จำนวน เช่น ABC เมท็อด nextDouble จะโยนสิ่งผิดปกติประเภทที่เรียกว่า InputMismatchException เนื่องจากโปรแกรมข้างบนนี้ไม่มีรหัสที่เตรียมไว้จัดการสิ่งผิดปกติ สิ่งผิดปกติที่ถูกโยนมาจะถูกโยนต่อกลับสู่ระบบทำให้โปรแกรมหยุดทำงาน และแสดงข้อความระบุความผิดพลาด (ที่ผู้ใช้อาจไม่เข้าใจ) ข้างล่างนี้ทางจอภาพ

```
Exception in thread "main" java.util.InputMismatchException
  at java.util.Scanner.throwFor(Scanner.java:840)
  at java.util.Scanner.next(Scanner.java:1461)
  at java.util.Scanner.nextDouble(Scanner.java:2387)
  at Average.main(Average.java:18)
```

เมื่อมีการ “โยน” ก็ต้องมีการ “รับ” เมท็อด nextDouble อุตส่าห์แจ้งความผิดปกติออกมา เราจึงต้องป้องกันอย่าให้สิ่งผิดปกตินี้ถูกโยนต่อไปยังระบบ ซึ่งทำได้ด้วยการประกาศรับสิ่งผิดปกติที่เกิดขึ้น ด้วยคำสั่ง try-catch ข้างล่างนี้

```
try {
  Scanner kb = new Scanner(System.in);
  double sum = 0;
  for (int i=0; i<n; i++) {
    System.out.print("data = ");
    sum += kb.nextDouble();
  }
  System.out.println("avg. = " + (sum/n));
} catch (InputMismatchException e) {
  System.out.println("ต้องใส่จำนวนจึงจะทำงานถูกต้อง");
}
```



จะเห็นได้ว่า try-catch ประกอบด้วยสองกลุ่มคำสั่ง คือ กลุ่ม try และกลุ่ม catch เรานำชุดคำสั่งที่ทำหน้าที่ประมวลผลตามปกติมาไว้ในกลุ่ม try และเขียนชุดคำสั่งที่จัดการกับสิ่งผิดปกติมาไว้ในกลุ่ม catch สามารถอ่านตัวคำสั่งได้ว่า

> " จงทำกลุ่ม try ถ้าทำได้ตามปกติ ก็ไม่ต้องทำกลุ่ม catch แต่ถ้าเกิดสิ่งผิดปกติระหว่างทำกลุ่ม try ให้กระโดดไปเริ่มทำในกลุ่ม catch ที่ตรงกับสิ่งผิดปกติที่เกิดขึ้น "

จะว่าไปแล้ว ส่วนของโปรแกรมข้างต้นยังไม่ค่อยสวยนัก เพราะถ้าผู้ใช้ป้อนผิด, เกิดสิ่งผิดปกติ, การทำงานกระโดดมาที่ `catch`, แสดงข้อความเตือน, และทำงานต่อคำสั่งถัดไป ข้อมูลที่ป้อนมาก็ไม่ถูกใช้ ระบบไม่ได้หาค่าเฉลี่ยให้ ดังนั้น ถ้าป้อนผิด ควรเปิดโอกาสให้ผู้ใช้ป้อนข้อมูลใหม่ ดังนี้

```
Scanner kb = new Scanner(System.in);
double sum = 0;
for (int i=0; i<n; i++)
  sum += readDouble(kb, "data = ");
System.out.println("avg. = " + (sum/n));
```

เรียกใช้ readDouble ในรหัสที่ 10-1

รหัสข้างต้นเรียกใช้เมท็อด `readDouble` (รหัสที่ 10-1) ซึ่งมีหน้าที่อ่านจำนวนจากตัวอ่านที่ได้รับ หากอ่านแล้วไม่ใช่จำนวน จะข้ามข้อมูลนั้น แจ้งเตือน แล้วอ่านข้อมูลถัดไป จนกว่าจะอ่านได้เป็นจำนวน เราใช้ `try-catch` ครอบ `nextDouble` มีตัวแปร `success` เก็บสถานะว่าทำกลุ่ม `try` สำเร็จหรือไม่ แล้วใช้วงวน `do-while` วนตราบเท่าที่ `success` ยังเป็นเท็จ `success` จะเป็นจริงตรงคำสั่งสุดท้ายในกลุ่ม `try` นั่นคือ จะเป็นจริงเมื่อทำทุกคำสั่งในกลุ่ม `try` ได้สำเร็จ พอออกมาตรวจสอบที่ `while` ก็จะออกจากวงวนได้ ถ้า `nextDouble` มีปัญหา จะกระโดดมาทำในกลุ่ม `catch` ตัวแปร `success` ยังมีค่า `false` พอตรวจสอบที่ `while` จะวนกลับขึ้นไปทำ `try` ใหม่ จึงหมายความว่า จะวนรับข้อมูลจากผู้ใช้จนกว่าจะป้อนข้อมูลถูกต้อง

```
public static double readDouble(Scanner sc, String msg) {
  double v = 0;
  boolean success = false;
  do {
    try {
      if (!"".equals(msg)) System.out.print(msg);
      v = sc.nextDouble();
      success = true;  // ทำสำเร็จแล้ว
    } catch (InputMismatchException e) {
      if (!"".equals(msg))
        System.out.println("ต้องใส่จำนวนจึงจะทำงานถูกต้อง กรุณาป้อนใหม่");
      sc.next();        // อ่านข้อมูลที่ผิดนั้นทิ้งไป
    }
  } while( !success ); // ยังไม่สำเร็จ วนกลับไปทำใหม่
  return v;
}
```

ถ้าผู้ใช้ส่งตัวอ่านแป้นพิมพ์ให้ sc ก็ควรส่งข้อความที่แสดงประกอบให้กับ msg ด้วย แต่ถ้าผู้ใช้ส่งตัวอ่านแฟ้มให้ sc ก็ควรส่ง "" ให้ msg จะได้ไม่แสดงอะไรทางจอภาพ

รหัสที่ 10-1 เมท็อดอ่านจำนวนถัดไปจากตัวอ่าน `sc` (จะอ่านข้ามข้อมูลที่ไม่ใช่จำนวน)

ผู้อ่านคงมีคำถามหลายประเด็นเกี่ยวกับตัวอย่างที่ได้นำเสนอมา เช่น ถ้าเกิดสิ่งผิดปกติประเภทอื่น ๆ ที่ไม่ได้เขียนในกลุ่ม `catch` จะเกิดอะไรขึ้น ตัวอักษร `e` ที่ปรากฏในวงเล็บของกลุ่ม `catch` คืออะไร สิ่งผิดปกติมีกี่แบบ สร้างแบบใหม่ ๆ ได้อย่างไร เป็นต้น จะได้อธิบายกันต่อไป

ประเภทของสิ่งผิดปกติ

สิ่งผิดปกติที่เกิดขึ้นระหว่างที่โปรแกรมทำงาน พอจำแนกออกได้เป็น 3 ประเภทด้วยกันดังนี้

1. สิ่งผิดปกติที่เกิดจากจุดบกพร่องของการเขียนโปรแกรม โดยทั่วไปเราจะไม่จัดการกับสิ่งผิดปกติประเภทนี้ เพราะสิ่งที่เกิดขึ้นสะท้อนให้เห็นว่า โปรแกรมมีจุดบกพร่องจึงทำงานผิด นักเขียนโปรแกรมควรต้องหาสาเหตุและรีบแก้ไขคำสั่งในรหัสต้นฉบับทันที เช่น

   - `IndexOutOfBoundsException` คือ การอ้างอิงดัชนีที่ไม่อยู่ในช่วง
   - `NegativeArraySizeException` คือ การสร้างอาเรย์ขนาดที่เป็นจำนวนลบ
   - `ArithmeticException` คือ การหารจำนวนเต็มด้วยศูนย์
   - `NullPointerException` คือ การใช้ตัวแปรหรือเมท็อดของอ็อบเจกต์ที่เป็น `null`
   - `IllegalArgumentException` คือ การส่งค่าให้กับเมท็อดที่ไม่ตรงข้อกำหนด

2. สิ่งผิดปกติที่เกิดจากจุดบกพร่องของตัวระบบซึ่งอยู่นอกขอบเขตการรับผิดชอบของตัวโปรแกรม ไม่สามารถจัดการได้ขณะทำงาน เช่น

   - `IOError` คือ อุปกรณ์รับส่งข้อมูลไม่สามารถทำงานได้
   - `VirtualMachineError` คือ ตัวทำงานของระบบจาวาทำงานไม่ได้
   - `NoClassDefFoundError` คือ ไม่พบคลาสที่ต้องการ
   - `OutOfMemoryError` คือ หน่วยความจำของระบบถูกใช้จนไม่พอให้ทำงาน
   - `StackOverflowError` คือ หน่วยความจำที่ใช้ระหว่างการเรียกเมท็อดมีไม่พอ

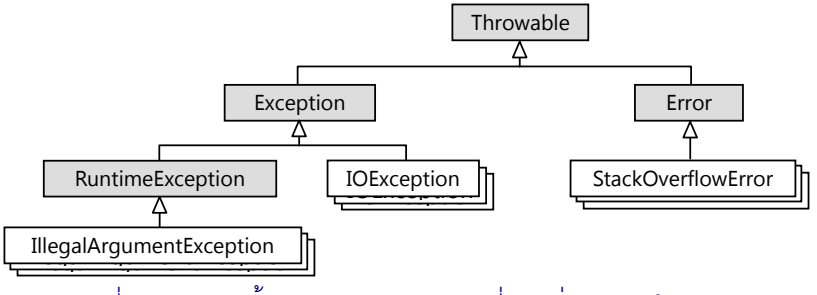
   อย่างไรก็ตามข้อผิดพลาดบางประการในประเภทนี้อาจมีสาเหตุจากจุดบกพร่องของการเขียนโปรแกรมก็เป็นได้ เช่น การสร้างอ็อบเจกต์หรือสร้างอาเรย์ที่มีจำนวนและขนาดมากเกินความจำเป็นจนเกิด `OutOfMemoryError` หรือการเรียกเมท็อดตัวเองซ้ำ ๆ มากเกินไป จนเกิด `StackOverflowError` ถือเป็นเหตุการณ์ที่นักเขียนโปรแกรมเขียนผิดกันบ่อยมาก ดังนั้น จึงต้องค้นหาสาเหตุที่ตัวโปรแกรมให้รอบคอบ

3. สิ่งผิดปกติที่ระบบหรือโปรแกรมผลิต และหวังว่า ผู้เรียกใช้บริการจะสามารถจัดการกับสิ่งผิดปกติประเภทนี้ได้ โดยไม่จำเป็นต้องหยุดการทำงานของโปรแกรม เช่น
   - `FileNotFoundException` คือ การหาแฟ้มที่กำหนดให้ไม่พบ
   - `MalformedURLException` คือ URL ที่ได้รับผิดรูปแบบมาตรฐาน
   - `UnknownHostException` คือ ไม่รู้จักชื่อเครื่องที่กำหนดให้ในเครือข่าย

สิ่งผิดปกติที่ถูกโยนในโปรแกรม ก็คืออ็อบเจกต์ของคลาสที่มีชื่อแทนประเภทของสิ่งผิดปกตินั่นเอง เช่น สิ่งผิดปกติที่แทนการหาแฟ้มไม่พบที่ชื่อ `FileNotFoundException` ก็คืออ็อบ-

เจกต์ของคลาส `FileNotFoundException` เป็นต้น จาวากำหนดว่า อ็อบเจกต์ที่จะถูกโยนได้ ต้องเป็นอ็อบเจกต์แบบ `Throwable` (ซึ่งคืออ็อบเจกต์ของคลาส `Throwable` หรือลูกหลาน) จาวานิยามคลาสมาตรฐานชื่อ `Error`, `Exception` และ `RuntimeException` ตามการรับทอดดังแสดงในรูปที่ 10-1 จากนั้นนิยามคลาสของสิ่งผิดปกติอื่น ๆ ให้เป็นคลาสลูกหลานของทั้งสามคลาสนี้อีกทอดหนึ่ง ซึ่งแบ่งคลาสของสิ่งผิดปกติออกเป็นสามประเภทตามการใช้งานดังนี้

- คลาส `RuntimeException` (และลูกหลาน) แทนสิ่งผิดปกติประเภทที่หนึ่ง (ที่มักมีสาเหตุจากจุดบกพร่องของการเขียนโปรแกรม) จาวาไม่บังคับให้โปรแกรมต้องจัดการกับสิ่งผิดปกติประเภทนี้
- คลาส `Error` (และลูกหลาน) แทนสิ่งผิดปกติประเภทที่สอง (มักมีสาเหตุที่เกิดจากข้อผิดพลาดของตัวระบบ) โปรแกรมไม่ควรจัดการกับสิ่งผิดปกติประเภทนี้
- คลาส `Exception` (และลูกหลาน) ที่ไม่ใช่ `RuntimeException` แทนสิ่งผิดปกติประเภทที่สาม ซึ่งเป็นสิ่งผิดปกติที่จาวาบังคับว่า ใครที่เรียกใช้เมท็อดที่อาจก่อให้เกิดสิ่งผิดปกติประเภทนี้ ผู้นั้นต้องเขียนชุดคำสั่งจัดการสิ่งผิดปกตินี้ มิฉะนั้นตัวแปลโปรแกรมจะฟ้อง จึงเรียกสิ่งผิดปกติประเภทนี้ว่า *สิ่งผิดปกติที่ถูกตรวจ* (*checked exception*) และเรียกสองแบบข้างต้นว่า *สิ่งผิดปกติที่ไม่ถูกตรวจ* (*unchecked exception*) [1]



รูปที่ 10-1 ลำดับชั้นการรับทอดของคลาสที่แทนสิ่งผิดปกติในจาวา

การสร้างและโยนสิ่งผิดปกติ

สิ่งผิดปกติที่เกิดขึ้นระหว่างการทำงานอาจเกิดจากตัวระบบเองที่มีการโยนอ็อบเจกต์จากภายใน [2] หรือเกิดจากการใช้คำสั่ง `throw` เพื่อโยนอ็อบเจกต์ที่แทนสิ่งผิดปกติ ในรูปแบบดังนี้

[1] ตรวจหรือไม่ตรวจในที่นี้หมายความว่า ตัวแปลโปรแกรมจะตรวจ (หรือไม่ตรวจ) ว่าต้องจัดการกับสิ่งผิดปกติ

[2] สิ่งผิดปกติที่ระบบโยน เช่น `ArrayIndexOutOfBoundsException`, `ArithmeticException`, `ClassCastException`

`throw new E()` หรือ `throw new E("เหตุผล")`

โดยที่ `E` คือคลาสของสิ่งผิดปกติ โดยทั่วไปมักใช้ตัวสร้างสิ่งผิดปกติแบบรับสตริงที่เก็บคำอธิบายเหตุผล เพราะมีประโยชน์ต่อการหาสาเหตุของความผิดปกติ

นักเขียนโปรแกรมมีหน้าที่สร้างและโยนสิ่งผิดปกติเมื่อพบสถานการณ์ที่ไม่เป็นไปตามที่คาดหวังไว้ และไม่สามารถจัดการกับความผิดปกติที่เกิดขึ้น เช่น คาดว่าเปิดแฟ้มแล้วต้องสำเร็จ แต่ระบบแจ้งว่าไม่พบแฟ้มนั้น, คาดว่าผู้ใช้ต้องป้อนจำนวน แต่ผู้ใช้กลับป้อนข้อความ, คาดว่าดอกเบี้ยที่ได้รับต้องเป็นจำนวนบวก แต่กลับได้จำนวนลบ เป็นต้น นักเขียนโปรแกรมควรเลือกคลาสของสิ่งผิดปกติที่ใกล้เคียงกับสถานการณ์ที่พบ, สร้างอ็อบเจกต์, พร้อมข้อความอธิบายเหตุผล, แล้วโยนไปให้ผู้ที่เรียกเมท็อด ให้เขารับผิดชอบกับเหตุการณ์ที่เกิดขึ้น หากเราไม่ทำเช่นนี้ ปล่อยเลยตามเลย หรือทำอะไรบางอย่างที่ผู้เรียกไม่รับทราบ อาจก่อให้เกิดผลเสียในภายหลังและหาสาเหตุได้ยาก การโยนสิ่งผิดปกติทันทีที่พบความผิดปกติ จะช่วยให้แก้ไขจุดบกพร่องได้ง่ายขึ้น

เมท็อด `deposit` ในรหัสที่ 10-2 แสดงการสร้างและโยนสิ่งผิดปกติทันทีที่พบว่า จำนวนเงินที่ฝากติดลบ (เพราะเราไม่ควรให้ฝาก เดี๋ยวเงินจะลด) สิ่งผิดปกติที่สร้างคืออ็อบเจกต์ของคลาส `IllegalArgumentException` (เป็นคลาสลูกของ `RuntimeException`) มีชื่อที่หมายความว่า พารามิเตอร์รับค่าที่ไม่ถูกต้อง เพื่อสะท้อนว่า ผู้ที่เรียกเมท็อดนี้คงจะทำงานผิดพลาด

```
public class BankAccount {
  ...
  public void deposit(double amt) {
    if (amt < 0) throw new IllegalArgumentException("amt = " + amt);
    balance += amt;
  }
```

รหัสที่ 10-2 ตัวอย่างการสร้างและโยนสิ่งผิดปกติ

การประกาศว่าจะโยนสิ่งผิดปกติ

จาวามีกฎในการเขียนหัวเมท็อดที่เกี่ยวกับการโยนสิ่งผิดปกติว่า เมท็อดใดที่อาจโยนสิ่งผิดปกติแบบที่ถูกตรวจ จะต้องเขียนชื่อคลาสของสิ่งผิดปกติเหล่านั้น ต่อท้ายคำว่า `throws` [3] ที่หัวเมท็อด เช่น ดูที่หัวเมท็อด `goo()` ในรหัสที่ 10-3 สรุปได้ว่า การทำงานของเมท็อด `goo()` อาจมีการโยนสิ่งผิดปกติของคลาส `E1`, `E2`, หรือ `E3` ดังนั้น ที่ใดที่ใช้ `goo` คงต้องเตรียมการรองรับสิ่งผิดปกติทั้งสามแบบนี้ หรือไม่ก็ต้องประกาศที่หัวเมท็อดว่าจะโยนต่อ สำหรับสิ่งผิดปกติแบบที่ไม่ถูกตรวจที่อาจเกิดขึ้นในเมท็อดนั้น จาวาไม่บังคับให้ต้องเขียนชื่อคลาส ที่หัวเมท็อดแต่อย่างใด (จะเขียนก็ได้ ไม่ว่ากัน)

[3] ให้สังเกตว่า เราเขียนคำว่า `throws` (มี `s`) ที่หัวเมท็อด แต่เมื่อต้องการโยนสิ่งผิดปกติ จะใช้ `throw` (ไม่มี `s`)

```
public void goo(double a) throws E1, E2, E3 {
  ...
}
```

ในเมท็อดนี้อาจมีการโยนแบบ E1, E2, หรือ E3

รหัสที่ 10-3 ตัวอย่างการเขียนประกาศที่หัวเมท็อดว่าอาจโยน E1, E2 หรือ E3

มาดูอีกสักตัวอย่าง รหัสที่ 10-4 แสดงเมท็อดที่รับชื่อแฟ้มมาเพื่อหาจำนวนที่มีค่ามากสุดที่เก็บอยู่ในแฟ้มนี้ การทำงานเริ่มด้วยการสร้างตัวอ่านจากแฟ้ม จากนั้นตรวจสอบว่า มีข้อมูลให้อ่านสักตัวหรือไม่ ถ้าไม่มีก็ย่อมหาจำนวนมากสุดไม่ได้ (เพราะไม่มีข้อมูลเลย) ให้โยนสิ่งผิดปกติของคลาส `NoSuchElementException` ถ้ามีก็ให้อ่านจำนวนแรกมาตั้งเป็นค่ามากสุดตอนเริ่มต้น แล้วเข้าทำงานในวงวนเพื่ออ่านจำนวนถัดไปมาเปรียบเทียบเพื่อปรับค่ามากสุดที่จำไว้ วนอ่านจนหมด ก็คืนค่ามากสุดที่เก็บไว้เป็นผลลัพธ์ของเมท็อด ภายในวงวน เราใช้เมท็อด `readDouble` ในรหัสที่ 10-1 เพื่ออ่านจำนวนจากตัวอ่าน โดยจะอ่านข้ามข้อมูลที่ไม่ใช่จำนวน

```
public static double max(String fn)
            throws FileNotFoundException, NoSuchElementException {
  Scanner in = new Scanner(new File(fn));

  if (!in.hasNext()) throw new NoSuchElementException(fn);
  double max = readDouble(in, "");
  while (in.hasNext()) {
    double v = readDouble(in, "");
    if (max < v) max = v;
  }
  return max;
}
```

อาจโยน FileNotFoundExcetion

เป็น unchecked exception ไม่จำเป็นต้องเขียนที่หัวเมท็อด แต่เขียนก็ได้

รหัสที่ 10-4 เมท็อด `max` หาจำนวนที่มีค่ามากสุดในแฟ้ม

เมื่อดูที่หัวเมท็อดสรุปได้ว่า เมท็อดนี้มีคำสั่งที่อาจโยนสิ่งผิดปกติ 2 ประเภท ซึ่งมีสาเหตุดังนี้

- คำสั่ง `new Scanner(...)` มีโอกาสโยน `FileNotFoundException` เมื่อไม่พบแฟ้มที่ต้องการ เนื่องจากเราไม่ต้องการจัดการสิ่งผิดปกติประเภทนี้ (ไม่มี `try-catch` ครอบการทำงาน) แสดงว่า ต้องการโยนสิ่งผิดปกตินี้ต่อให้ผู้เรียก `max` และเนื่องจากสิ่งผิดปกติแบบนี้เป็นแบบที่ถูกตรวจ ดังนั้น จึงต้องเขียนชื่อคลาสไว้หลัง `throws` ที่หัวเมท็อดด้วย

- คำสั่ง `throw new NoSuchElementException(fn)` แสดงอย่างชัดเจนว่าต้องการโยนสิ่งผิดปกติ ซึ่งก็ไม่ได้เขียนให้ถูกจัดการใน `try-catch` เช่นกัน แต่เนื่องจากสิ่งผิดปกตินี้เป็นแบบที่ไม่ถูกตรวจ จึงไม่จำเป็นต้องเขียนชื่อคลาสที่หัวเมท็อด แต่ในกรณีนี้เราเลือกจะเขียนประกาศที่หัวเมท็อด เพื่อแสดงให้ผู้ใช้เมท็อดทราบว่า ถ้าสามารถจัดการได้ก็จะดี เพราะเหตุการณ์ที่แฟ้มไม่มีข้อมูลมีโอกาสเกิดขึ้น โดยผู้ใช้อาจไม่รู้ก่อนเรียก

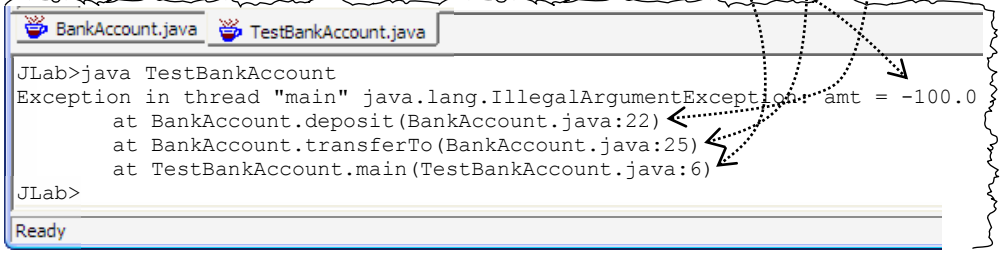
ผิดอะไร ผิดอย่างไร ผิดที่ไหน

การโยนสิ่งผิดปกติไม่ได้มีจุดประสงค์เพียงแค่บอกว่าโปรแกรมมีปัญหา เพราะนั่นไม่ได้ช่วยอะไรเลยในการแก้ไขปัญหา แต่สิ่งผิดปกติที่โยนไปนี้มีข้อมูลที่ช่วยแก้ไขปัญหาได้ด้วย คือ จะช่วยวิเคราะห์ปัญหาที่เกิดขึ้นว่า ผิดอะไร ผิดอย่างไร และ ผิดที่ไหน

ผิดอะไร คลาสของสิ่งผิดปกติที่ถูกโยนมามีชื่อที่สื่อความหมายให้เราทราบประเภทของสิ่งผิดปกติ ดังนั้น หากเราจะเป็นผู้โยนสิ่งผิดปกติ จึงต้องคิดเสมอว่า อย่าโยนสิ่งผิดปกติของคลาสที่มีความหมายกว้างเกินไป เช่น กรณีพารามิเตอร์ที่รับมาเป็นดัชนีที่อยู่นอกช่วงอาเรย์ จะเลือกโยน `IllegalArgumentException` หรือจะโยน `ArrayIndexOutOfBoundsException` ก็สื่อความหมายกับสาเหตุที่เกิด แต่แบบหลังจะตรงประเด็นมากกว่า อ่านชื่อคลาสก็รู้ว่าเกิดอะไรขึ้นทันที สิ่งที่ไม่ควรทำอย่างยิ่งคือโยนอ็อบเจกต์ของ `Exception` หรือ `RuntimeException` ที่มีความหมายกว้างเหลือเกิน ไม่ก่อให้เกิดประโยชน์ใดๆ เลยในการหาสาเหตุ

ผิดอย่างไร สตริงที่เราส่งให้ตัวสร้างสิ่งผิดปกติมีไว้บอกสาเหตุของความผิดปกติ เช่น เมื่อมีการใช้อาเรย์ในช่องที่อยู่นอกช่วง ระบบจาวาจะสร้างสิ่งผิดปกติโดยส่งค่าของดัชนีตัวปัญหานั้นไปให้ด้วย ถ้าระบุว่ามีค่า –1 ผู้ตรวจหาจุดบกพร่องจะได้รู้สาเหตุว่าเป็นจำนวนลบ แต่ถ้าบอกว่ามีค่า 2000 ก็จะมองในอีกมุมว่า ค่าของดัชนีเกินช่วงในอีกลักษณะ ดังนั้น เมื่อใดที่คิดจะสร้างและโยนสิ่งผิดปกติ ควรส่งสตริงที่บอกสาเหตุไปให้ตัวสร้างด้วย จะได้เป็นประโยชน์ต่อผู้แก้ไขจุดบกพร่อง

ผิดที่ไหน เมื่อมีการโยนสิ่งผิดปกติ ระบบจะบันทึกสิ่งที่เรียกว่า *stack trace* ซึ่งคือลำดับของการเรียกเมท็อดตั้งแต่ตำแหน่งที่โปรแกรมเริ่มทำงาน (เมท็อด `main`) ไปจนถึงตำแหน่งที่เกิดการโยนสิ่งผิดปกติ แต่ละตำแหน่งประกอบด้วยชื่อคลาส ชื่อเมท็อด และหมายเลขบรรทัดในรหัสต้นฉบับ รูปที่ 10-2 แสดงลำดับการเรียกเมท็อดของสิ่งผิดปกติ `IllegalArgumentException` ที่ถูกโยนจากเมท็อด `deposit` (บรรทัดที่ 22 ของแฟ้ม `BankAccount.java`) ซึ่งถูกเรียกจากเมท็อด `transferTo` (บรรทัดที่ 25 ของแฟ้ม `BankAccount.java`) และถูกเรียกจากเมท็อด `main` (บรรทัดที่ 6 ของ `TestBankAccount.java`) โดยมีสาเหตุมาจากการที่พารามิเตอร์ชื่อ `amt` มีค่า `-100.0` ที่ถือว่าเป็นค่าที่ไม่ถูกต้อง



รูปที่ 10-2 ข้อความแสดงลำดับการเรียกเมท็อดเมื่อโยนสิ่งผิดปกติ

การสร้างสิ่งผิดปกติแบบใหม่

เมื่อต้องการโยนสิ่งผิดปกติ เราต้องตัดสินใจว่าจะโยนของคลาสใดดี จึงจะเหมาะกับสถานการณ์ที่เกิดขึ้น โดยทั่วไปเรามักเลือกใช้คลาสมาตรฐานที่จาวามีให้ เพราะนักเขียนโปรแกรมจะคุ้นเคยและเข้าใจความหมายของคลาสเหล่านั้น เช่น ใช้ `IllegalArgumentException` เพื่อบอกว่าพารามิเตอร์รับค่าที่ไม่ถูกต้อง เป็นต้น แต่บางครั้งนักเขียนโปรแกรมอาจเลือกที่จะสร้างคลาสใหม่เพื่อแทนความผิดปกติที่ชัดเจนกว่าที่มีอยู่ เช่น ในกรณีของคลาส `BankAccount` เราอาจสร้างคลาสชื่อ `NegativeMoneyException` เพื่อแทนกรณีที่รับจำนวนเงินเป็นลบ หรือสร้างคลาสชื่อ `InsufficientBalanceException` เพื่อแทนกรณีที่เงินในบัญชีไม่พอถอน

สิ่งที่ต้องพิจารณาคือ จะให้สิ่งผิดปกติแบบใหม่นี้เป็นแบบที่ถูกตรวจ หรือไม่ถูกตรวจ หากเราคาดว่า เป็นสิ่งผิดปกติที่ต้องมีผู้รับไปจัดการ ห้ามละเลย ก็ต้องให้เป็นแบบที่ถูกตรวจ (ตัวแปลโปรแกรมจะได้ตรวจให้) ถ้าไม่เป็นเช่นนั้น ก็ให้เป็นสิ่งผิดปกติที่ไม่ถูกตรวจ จากนั้นพิจารณาว่าจะให้เป็นคลาสลูกของคลาสเดิมคลาสใดดี ถ้าหาไม่ได้ ก็ให้เป็นคลาสลูกของ `Exception` ถ้าเป็นสิ่งผิดปกติแบบที่ถูกตรวจ และให้เป็นคลาสลูกของ `RuntimeException` ถ้าเป็นแบบที่ไม่ถูกตรวจ

การเขียนคลาสสิ่งผิดปกติแบบใหม่นั้นกระทำได้ง่ายมาก เพียงแค่เขียนให้ขยายจากคลาสแม่ที่เป็นสิ่งผิดปกติ และเขียนตัวสร้างให้สองแบบ รหัสที่ 10-5 และรหัสที่ 10-6 แสดงตัวอย่างของคลาส `WebPageException` และ `OutOfRangeException` แทนสิ่งผิดปกติแบบที่ถูกตรวจและไม่ถูกตรวจตามลำดับ ทั้งสองคลาสเขียนคล้ายกัน (แตกต่างตรงที่แม่) มีตัวสร้างแบบแรกไม่ทำอะไรภายในเลย (ระบบเติม `super()` แทนการเรียกตัวสร้างแบบไม่รับพารามิเตอร์ใด ๆ ของคลาสแม่) และมีตัวสร้างที่รับสตริง ภายในเรียกตัวสร้างแบบรับสตริงของคลาสแม่นั่นเอง

```
01 import java.io.IOException;
02 public class WebPageException extends IOException {
03   public WebPageException() {
04   }
05   public WebPageException(String s) {
06     super(s);
07   }
08 }
```

รหัสที่ 10-5 คลาส `WebPageException` แทนสิ่งผิดปกติที่ถูกตรวจแบบใหม่

```
01 public class OutOfRangeException extends IllegalArgumentException {
02   public OutOfRangeException() {
03   }
04   public OutOfRangeException(String s) {
05     super(s);
06   }
07 }
```

รหัสที่ 10-6 คลาส `OutOfRangeException` แทนสิ่งผิดปกติที่ไม่ถูกตรวจแบบใหม่

ขั้นตอนการโยน-รับสิ่งผิดปกติ

หัวข้อนี้จะนำเสนอการ “โยน” และการ “รับ” สิ่งผิดปกติให้ละเอียดกว่าที่นำเสนอในตอนต้น เมื่อใดที่เงื่อนไขการทำงานของคำสั่ง [4] ไม่เป็นไปตามข้อกำหนด อาจก่อให้เกิดสิ่งผิดปกติขึ้นได้ โดยตัวคำสั่งจะโยนสิ่งผิดปกติออกมาให้เมท็อดที่ใช้คำสั่งนี้รับสิ่งผิดปกตินั้นไปจัดการว่า จะทำอย่างไร เมท็อดที่ใช้คำสั่งนี้อาจเลือกไม่รับ ซึ่งหมายถึงการขอโยนสิ่งผิดปกตินี้ต่อขึ้นไปให้ผู้เรียกเมท็อดก็ได้ รหัสที่ 10-7 แสดงตัวอย่างของลำดับการเรียกเมท็อดเริ่มจาก `main` แล้วไปเกิดสิ่งผิดปกติในเมท็อด `m3` มาดูกันว่า มีขั้นตอนการโยนและการรับสิ่งผิดปกติอย่างไร ดังนี้

```
  public static void main(String[] a) throws IOException {
❶   m1("data.txt");
    ...
  }
  private static void m1(String f) throws IOException {
    try {
❷     m2(f);
      ...
    } catch (FileNotFoundException e) {
❸     ...
    } catch (MalformedURLException e) {
❹     ...
    }
❺   ...
  }
  private static void m2(String f) throws FileNotFoundException,
                           MalformedURLException, IOException {
❻   m3(f);
    ...
  }
  private static void m3(String f) throws FileNotFoundException,
                                    MalformedURLException {
❼   Scanner in = new Scanner(new File(f));
    ...
  }
```

รหัสที่ 10-7 ตัวอย่างแสดงการโยน โยนต่อ และการรับสิ่งผิดปกติในลำดับการเรียกเมท็อด

- เริ่มที่เมท็อด `main` คำสั่ง ❶ เรียกเมท็อด `m1`
- การทำงานย้ายมาที่เมท็อด `m1` ที่นี่มีการใช้คำสั่ง `try-catch` จึงไปทำในกลุ่ม `try` เริ่มที่ ❷
  - ถ้าทำคำสั่งในกลุ่มทั้งหมดแล้วไม่มีปัญหาใด ก็ไปทำต่อที่ ❺

[4] “คำสั่ง” ในที่นี้รวมถึงการเรียกใช้เมท็อดด้วย

- แต่ถ้าเกิดสิ่งผิดปกติ ระบบจะย้ายไปทำในกลุ่ม `catch` ที่รองรับสิ่งผิดปกติประเภทเดียวกับที่ถูกโยนมา ในที่นี้นักเขียนโปรแกรมเขียนกลุ่ม `catch` ไว้สองกลุ่มเพื่อจัดการกับสิ่งผิดปกติสองแบบคือ `FileNotFoundException` และแบบ `MalformedURLException` ไม่ว่าการทำงานจะย้ายไปทำตัวจัดการที่กลุ่มใด เมื่อทำเสร็จ ก็จะออกมาทำต่อที่ ❺

- ขอกลับมาติดตามการทำงานกันต่อ คำสั่งที่ ❷ เรียกเมท็อด `m2`
- การทำงานย้ายมาที่เมท็อด `m2` คำสั่งที่ ❻ เรียกเมท็อด `m3`
- การทำงานย้ายมาที่เมท็อด `m3` คำสั่งที่ ❼ เปิดแฟ้ม สมมติว่าไม่พบแฟ้ม `f` ตัวสร้าง `Scanner` จะโยนสิ่งผิดปกติ `FileNotFoundException`

- เมื่อมีการโยนสิ่งผิดปกติ ระบบจะเริ่มค้นหาในลำดับการเรียกเมท็อดที่ผ่านมาจนถึงจุดที่เกิดการโยนสิ่งผิดปกติว่า จุดล่าสุดใดในลำดับการเรียกเมท็อดมีตัวจัดการสิ่งผิดปกติประเภทเดียวกับที่ถูกโยน ก็จะย้ายการทำงานไปทำที่ตัวจัดการนั้น จากตัวอย่าง ลำดับการเรียกเมท็อด คือ `main` → `m1` → `m2` → `m3` เมื่อพิจารณาย้อนกลับ `m3`, `m2`, `m1`, และ `main` พบว่า `m3` ไม่มีตัวจัดการ, `m2` ก็ไม่มี, แต่ `m1` มีตัวจัดการสิ่งผิดปกติประเภทเดียวกับที่ถูกโยนที่จุด ❸ การทำงานจึงถอยมาทำที่ ❸ เมื่อทำเสร็จ จะทำที่ ❺ ต่อไป

ให้สังเกตจากตัวอย่างว่า ที่หัวเมท็อด `m3` ในรหัสที่ 10-7 ประกาศว่าอาจโยนสิ่งผิดปกติสองแบบ คือ `FileNotFoundException` และ `MalformedURLException` แบบหลังคงถูกโยนจากคำสั่งที่ต่อจากจุด❼ ที่หัวเมท็อด `m2` ประกาศว่าอาจโยนสามแบบ สองแบบแรกคงมาจากการโยนสิ่งผิดปกติที่ได้รับจาก `m3` กับอีกแบบที่คงมาจากคำสั่งส่วนที่เหลือใน `m3` ที่หัวเมท็อดของ `m1` ประกาศว่ามีการโยนเพียงหนึ่งแบบคือ `IOException` เพราะสองแบบที่ได้รับจาก `m2` นั้นถูกจัดการใน `try-catch` ภายในเมท็อดแล้ว แบบ `IOException` คงไม่ได้รับการจัดการใน `m2` เลยใช้วิธีโยนต่อ หัวเมท็อด `main` ก็เช่นกัน ไม่จัดการแต่ขอโยน `IOException`

การใช้อ็อบเจกต์สิ่งผิดปกติที่ได้รับ

ผู้อ่านคงสงสัยว่า `e` ที่อยู่ภายใน `catch (FileNotFoundException e)` คืออะไร ตัว `e` นี้ก็คือตัวแปรที่เตรียมไว้อ้างอิงอ็อบเจกต์สิ่งผิดปกติที่ถูกโยนมา แล้ว `catch` รับไว้ได้ เราอาจมองว่า `catch(FileNotFoundException e)` เป็นเสมือนหัวเมท็อด (ชื่อ `catch`) ที่ระบบเรียก พร้อมกับส่งสิ่งผิดปกติมาให้กับพารามิเตอร์ `e` โดยจะเรียกกลุ่ม `catch` ที่มีพารามิเตอร์เป็นประเภทเดียวกับสิ่งผิดปกติที่เกิดขึ้น ในตัวอย่างเมื่อเกิดสิ่งผิดปกติแบบไม่พบแฟ้ม จึงเลือกไปทำในกลุ่ม `catch` ที่รับอ็อบเจกต์ `FileNotFoundException` ดังนั้น ภายในกลุ่มคำสั่งหลัง `catch` จึงสามารถใช้ตัวแปร `e` ได้ อนึ่งเนื่องจาก `e` เป็นตัวแปร จึงสามารถตั้งชื่อใดก็ได้ที่ไม่ผิดกฎการตั้งชื่อตัวแปร

แล้วจะนำตัวแปร e ไปใช้ประโยชน์อะไร ทุก ๆ คลาสที่แทนสิ่งผิดปกติเป็นคลาสลูกหลานของ Throwable คลาสนี้มีเมท็อดประจำอ็อบเจกต์ที่เรียกใช้ได้จำนวนหนึ่ง ตารางที่ 10-1 แสดงเมท็อดที่เรียกใช้ได้กับสิ่งผิดปกติเพื่อหาสาเหตุของความผิดปกติ อันได้แก่ getMessage() คืนข้อความที่ผู้โยนต้องการบอกสาเหตุ, printStackTrace() แสดงลำดับการเรียกเมท็อดจากต้นจนถึงขณะโยนสิ่งผิดปกติ และ toString() คืนข้อความบรรยายความผิดปกติที่เกิดขึ้น

ตารางที่ 10-1 เมท็อดประจำอ็อบเจกต์สิ่งผิดปกติที่ใช้บ่อย

การเรียกใช้	ความหมาย
e.getMessage()	คืนสตริงที่รับไว้ตอนสร้างสิ่งผิดปกติ e ที่แทนสาเหตุของความผิดปกติ
e.printStackTrace()	แสดง stack trace ตอนโยนสิ่งผิดปกติ e (ทางจอภาพ)
e.toString()	คืนสตริงบรรยายสิ่งผิดปกติ e

รหัสที่ 10-8 แสดงตัวอย่างการจัดการสิ่งผิดปกติที่อาจเกิดขึ้นจากการเรียกใช้เมท็อด max ในรหัสที่ 10-4 ขอทวนอีกครั้งว่า เมท็อด max รับชื่อแฟ้มเพื่อหาจำนวนที่มีค่ามากสุดในแฟ้ม max อาจโยน FileNotFoundException ซึ่งเป็นสิ่งผิดปกติแบบที่ถูกตรวจ ผู้ที่จะใช้ max จึงต้องตัดสินใจว่า ถ้าเกิดสิ่งปกติขึ้นมาจริง ๆ จะจัดการหรือไม่ หรือว่าจะเขียนที่หัวเมท็อดตัวเองว่าจะโยนต่อ นอกจากนี้เมท็อด max ยังอาจโยน NoSuchElementException ด้วย ถึงแม้ว่าจะเป็นสิ่งผิดปกติแบบที่ไม่ถูกตรวจ แต่ถ้าผู้ใช้ max จะรับไปจัดการก็กระทำได้

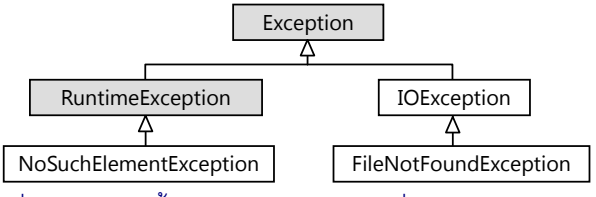
สิ่งที่เมท็อด main ในรหัสที่ 10-8 ทำคือ อ่านชื่อแฟ้มจากผู้ใช้ สั่งให้ max หาจำนวนมากสุดเพื่อนำผลมาแสดง ถ้าหาแฟ้มไม่พบ หรือพบแฟ้มแต่ไม่มีข้อมูล จะแสดงข้อความแจ้งความผิดพลาดพร้อมกับข้อความจาก e.getMessage() แล้วหยุดทำงาน ผู้เขียนโปรแกรมนี้เลือกแสดงข้อความเอง เพราะดีกว่าปล่อยให้สิ่งผิดปกติถูกโยนกลับไปให้ระบบ แล้วแสดงข้อความที่อ่านแล้วตีความยากกว่า

```
public static void main(String[] args) {
  Scanner kb = new Scanner(System.in);
  System.out.print("ชื่อแฟ้มข้อมูล = ");
  String fileName = kb.nextLine();
  try {
    double max = max(fileName);
    System.out.println("max = " + max);
  } catch (FileNotFoundException e) {
    System.out.println("ไม่พบแฟ้ม " + e.getMessage());
  } catch (NoSuchElementException e) {
    System.out.println("แฟ้มนี้ไม่มีข้อมูลเลย " + e.getMessage());
  }
}
```

รหัสที่ 10-8 ตัวอย่างการจัดการสิ่งผิดปกติที่เกิดจากเมท็อด max ในรหัสที่ 10-4

ลำดับการ catch

ในกรณีที่ try-catch มีกลุ่ม catch หลายกลุ่ม เราสามารถสลับลำดับของกลุ่ม catch ต่าง ๆ ได้ ถ้าแต่ละกลุ่ม catch รับสิ่งผิดปกติของคลาสที่เป็นเพียงญาติห่าง ๆ กัน ไม่มีความสัมพันธ์ในการรับทอด ดังตัวอย่างในรหัสที่ 10-8 มีกลุ่ม catch สองกลุ่มเตรียมไว้รับสิ่งผิดปกติของคลาส FileNotFoundException และ NoSuchElementExcecption เนื่องจากทั้งสองคลาสนี้ไม่มีใครรับทอดจากใครเลยดังแสดงในรูปที่ 10-3 จึงสามารถสลับการเขียนสองกลุ่ม catch นี้ได้



รูปที่ 10-3 ลำดับชั้นการรับทอดของคลาสสิ่งผิดปกติจำนวนหนึ่ง

สำหรับกรณีที่มีกลุ่ม catch หลายกลุ่มที่รับสิ่งผิดปกติของคลาสที่มีการรับทอดกัน ต้องรับสิ่งผิดปกติประเภทพิเศษก่อนประเภททั่วไป นั่นคือ ต้องเขียนกลุ่ม catch ที่รับซับคลาสก่อนกลุ่ม catch ที่รับซูเปอร์คลาส เช่น กำหนดให้มีกลุ่ม catch สี่กลุ่มรับ IOExcepion, Exception, FileNotFoundException และ NoSuchElementException จึงต้องเขียนลำดับของกลุ่ม catch ดังรหัสที่ 10-9 เนื่องจากเมื่อมีสิ่งผิดปกติโยนออกมาจากกลุ่ม try ระบบจะเริ่มหาว่าควรไปทำงานที่กลุ่มใด โดยไล่หาจากกลุ่มบนลงไปเรื่อย ๆ ทีละกลุ่มจนกว่าจะพบกลุ่มที่สามารถรับสิ่งผิดปกตินั้นได้ ถ้ากลุ่ม catch แรกเขียนให้รับแบบ Exception (ซึ่งเป็นคลาสบรรพบุรุษของสิ่งผิดปกติมากมาย) ดังนั้น สิ่งผิดปกติใด ๆ (ยกเว้นที่เป็นแบบ Error) ที่ถูกโยนจะไปทำที่กลุ่มนี้ ตัวแปลโปรแกรมจะฟ้องว่า กลุ่ม catch อื่นที่ตามมาไม่มีสิทธิ์ได้รับสิ่งผิดปกติใด ๆ เลย ดังนั้น เมื่อใดที่เขียนกลุ่ม catch ที่รับสิ่งผิดปกติของคลาสหนึ่งนั่นหมายความว่า จะรับสิ่งผิดปกติของลูกหลานของคลาสนั้นด้วย ถ้าต้องการจัดการคลาสลูกหลานใด ต้องเขียนดักก่อนไว้ด้านบน

```
try {
  ...
} catch (NoSuchElementException e) {
  ...
} catch (FileNotFoundException e) {
  ...
} catch (IOException e) {
  ...
} catch (Exception e) {
  ...
}
```

รหัสที่ 10-9 จาวากำหนดให้ต้อง catch ซับคลาสก่อนซูเปอร์คลาส

ตัวอย่าง

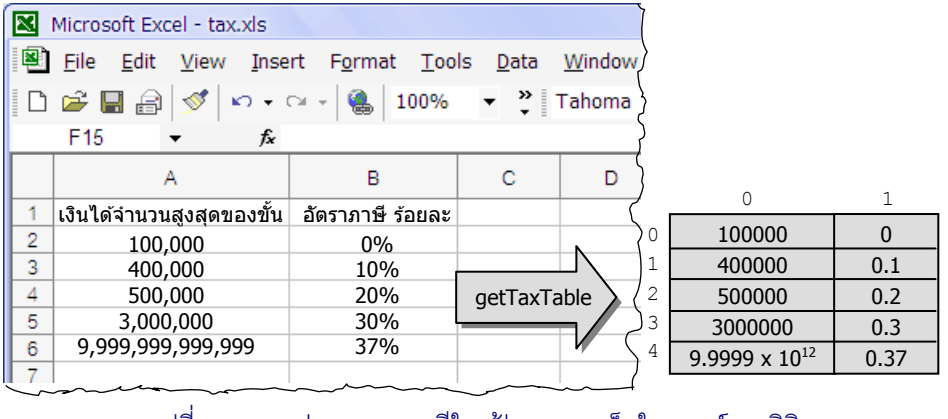
หัวข้อนี้นำเสนอตัวอย่างการจัดการสิ่งผิดปกติเมื่อมีการใช้เมท็อดในคลังคำสั่งหลากหลายรูปแบบเพื่อให้เห็นว่า เราต้องสนใจสิ่งผิดปกติที่อาจเกิดขึ้นจากการเรียกใช้บริการจากผู้อื่น ขอนำเสนอตัวอย่างการคำนวณภาษีที่ต้องอ่านตารางภาษีจากแฟ้ม Excel โดยใช้คลังคำสั่งที่มีชื่อว่า HSSF, การเพิ่มภาพฉากหลังในเอกสารรูปแบบ PDF ด้วยการใช้คลังคำสั่ง iText, และการอ่านข้อมูลจากเว็บเพจในอินเทอร์เน็ตด้วยคลังคำสั่งในชุด `java.net` (การนำเสนอตัวอย่างเหล่านี้ไม่มีจุดประสงค์จะอธิบายวิธีการใช้งานคลาสต่าง ๆ ที่เกี่ยวข้อง เพียงแต่ต้องการนำเสนอให้เห็นการใช้งานคลาสต่าง ๆ ที่ได้รับความนิยมในวงการประกอบเป็นตัวอย่างเท่านั้น)

โปรแกรมคำนวณภาษีเงินได้

เราต้องการเขียนโปรแกรมคำนวณภาษีเงินได้บุคคลธรรมดา ตัวโปรแกรมรับเงินได้สุทธิจากผู้ใช้ แล้วนำไปคำนวณภาษีตามตารางภาษีที่เก็บไว้ในแฟ้ม Excel (รูปที่ 10-4) ตัวอย่างเช่น รายได้สุทธิ 6,000,000 บาท มีขั้นตอนการคำนวณภาษีดังนี้

- 100,000 บาทแรกไม่เสียภาษี
- 400,000 บาทถัดมาเสีย 10% เป็นเงิน 40,000 บาท
- 500,000 บาทถัดมาเสีย 20% เป็นเงิน 100,000 บาท
- 3,000,000 บาทถัดมาเสีย 30% เป็นเงิน 900,000 บาท
- 2,000,000 บาทถัดมาเสีย 37% เป็นเงิน 740,000 บาท

รวมแล้วต้องเสียภาษี 1,780,000 บาท



รูปที่ 10-4 การอ่านตารางภาษีในแฟ้ม Excel เก็บในอาเรย์สองมิติ

รหัสที่ 10-10 แสดงโปรแกรมคำนวณภาษีที่ต้องการ เมท็อด `main` เริ่มด้วยการเรียก `readDouble` ที่เราเคยเขียนในรหัสที่ 10-1 เพื่ออ่านเงินได้สุทธิในบรรทัดที่ 7 จากนั้นอ่านตาราง

ภาษีจากแฟ้ม Excel ด้วยเมท็อด `getTaxTable` ที่รับชื่อแฟ้มในบรรทัดที่ 9 สิ่งที่คืนกลับมาคืออาเรย์สองมิติของจำนวนจริง เก็บข้อมูลที่เห็นเป็นตารางภาษีในรูปที่ 10-4 บรรทัดที่ 10 ส่งตารางนี้กับเงินได้สุทธิให้เมท็อด `calcTax` คำนวณภาษีเงินได้ กลับคืนมาเพื่อนำไปแสดงผล

ขั้นตอนการทำงานข้างต้นนี้เป็นขั้นตอนการทำงานกรณีที่ดำเนินไปตามปกติ แต่ก็ย่อมมีกรณีไม่ปกติ ดังนั้น จึงต้องใช้ `try-catch` โดยจะมีกรณีที่หาไม่พบแฟ้มตารางภาษี และกรณีเกิดข้อผิดพลาดระหว่างการอ่านแฟ้ม ขอใช้วิธีการจัดการอย่างง่ายสุด คือ การแสดงข้อความให้ผู้ใช้รับทราบความผิดปกติที่เกิดขึ้น แล้วก็จบการทำงานของโปรแกรม

```
01 import java.util.*;
02 import java.io.*;
03 import org.apache.poi.hssf.usermodel.*;
04
05 public class TaxCalc {
06   public static void main(String[] args) {
07     double income = readDouble(new Scanner(System.in),"เงินได้สุทธิ : ");
08     try {
09       double[][] taxTable = getTaxTable("c:/java101/tax.xls");
10       double tax = calcTax(taxTable, income);
11       System.out.println("ภาษี = " + tax);
12     } catch (FileNotFoundException e) {
13       System.out.println("ไม่พบแฟ้ม: " + e.getMessage());
14     } catch (IOException e) {
15       System.out.println(e.getMessage());
16     }
17   }
```

รหัสที่ 10-10 โปรแกรมคำนวณภาษี (ตารางภาษีถูกอ่านจากแฟ้ม Excel)

จะอ่านข้อมูลจากแฟ้ม Excel อย่างไร ? เมื่อนักเขียนโปรแกรมเผชิญปัญหาในลักษณะนี้ สิ่งแรกที่ควรทำคือ ค้นหาว่า มีใครในโลกแก้ปัญหานี้แล้วบ้าง ถ้ามีคนทำแล้ว และยินดีเผยแพร่ผลงานให้ผู้อื่นใช้ ย่อมเป็นเรื่องดี มีนักเขียนโปรแกรมภาษาจาวาจำนวนมากในโลกที่เผยแพร่ผลงานในลักษณะที่เป็นซอฟต์แวร์ไม่คิดค่าใช้จ่ายในการใช้งาน และก็มีจำนวนมากที่เปิดเผยรหัสต้นฉบับให้ผู้อื่นศึกษาและแก้ไขได้ด้วย หากเราลองใช้กูเกิลค้นด้วยประโยคว่า How to read excel file using java จะพบเอกสารจำนวนมากที่นำเสนอวิธีการอ่านข้อมูลในแฟ้ม Excel ผู้เขียนขอเลือกใช้คลังคลาสของโครงการ Apache POI (http://poi.apache.org) ที่มีผู้กล่าวขานกันมากพอที่จะน่าเชื่อถือว่าใช้งานได้ดี ภายในมีอยู่แพ็กเกจหนึ่งชื่อ HSSF ที่เกี่ยวข้องกับการจัดการแฟ้ม Excel

ขอทบทวนการจัดเก็บข้อมูลใน Excel เล็กน้อย เราเรียกแฟ้ม Excel ว่า workbook ที่ประกอบด้วยหลาย worksheets แต่ละ worksheet เป็นตารางมีหลาย rows และแต่ละ row มีหลาย cells เมท็อด `getTaxTable` ใน รหัสที่ 10-10 ข้างล่างนี้ สร้างอ็อบเจกต์ `HSSFWorkbook` ที่อ่านจากแฟ้ม Excel ที่ได้รับ (บรรทัดที่ 21) ตามด้วยการขออ็อบเจกต์ที่เก็บ worksheet แผ่นที่ 0

(บรรทัดที่ 23), บรรทัดที่ 24 ขอจำนวนแถวที่มีข้อมูลในแผ่นตารางนี้ เพื่อนำมาสร้างอาเรย์สองมิติ (โดยลบจำนวนแถวออกหนึ่ง เพราะไม่สนใจแถวบนที่เป็นข้อความหัวเรื่อง) จากนั้นเข้าวงวน `for` หยิบข้อมูลออกมาทีละแถว (แทนด้วยอ็อบเจกต์แบบ `HSSFRow`) เราข้ามแถวที่ 0 เพราะเป็นหัวเรื่อง แต่ละแถวที่อ่าน เราหยิบ cell ที่ 0 และ 1 มาเก็บในอาเรย์ที่เตรียมไว้ จนได้อาเรย์ที่มีข้อมูลดังตัวอย่างที่แสดงในรูปที่ 10-4 เนื่องจากตัวสร้าง `FileInputStream` และ `HSSFWorkbook` อาจมีการโยนสิ่งผิดปกติ `FileNotFoundException` และ `IOException` ตามลำดับ โดย `getTaxTable` ไม่รู้จะจัดการอย่างไร จึงใช้วิธีโยนต่อ โดยเขียนประกาศไว้ที่หัวเมท็อด (บรรทัดที่ 19) ผู้อ่านอาจสงสัยว่า รู้ได้อย่างไรว่า คำสั่งใดจะโยนสิ่งผิดปกติประเภทใด ถ้าเป็นสิ่งผิดปกติแบบที่ถูกตรวจ ตัวแปลโปรแกรมจะฟ้องเองว่า ที่ตำแหน่งใดของบรรทัดใด อาจโยนสิ่งผิดปกติที่เราไม่ได้เขียนไว้ใน `try-catch` และก็ไม่ได้เขียนที่หัวเมท็อด ทำให้นักเขียนโปรแกรมต้องสนใจทันทีว่าจะทำอย่างไรดีกับสิ่งผิดปกติเหล่านี้ เพราะถ้าไม่ทำอะไรสักอย่าง ตัวแปลโปรแกรมก็จะฟ้องว่าโปรแกรมยังไม่สมบูรณ์ (อย่าลืมลากแฟ้ม poi-scratchpad-3.0.2-FINAL-20080204.jar และ poi-3.0.2-FINAL-20080204.jar ใน c:/java101 ซึ่งคือคลังคลาสของ Apache POI มาวางที่หัวของ JLab เพื่อรวมคลังคลาสเหล่านี้ในโปรแกรมที่พัฒนา ก่อนสั่งโปรแกรมทำงาน)

```
18   private static double[][] getTaxTable(String excelFile)
19                       throws FileNotFoundException, IOException {
20     double[][] table = new double[0][0];
21     HSSFWorkbook wb = new HSSFWorkbook(
22                             new FileInputStream(excelFile));
23     HSSFSheet sheet = wb.getSheetAt(0);
24     int rows = sheet.getPhysicalNumberOfRows() - 1;
25     table = new double[rows][2];
26     for (int i = 1; i <= rows; i++) {
27       HSSFRow r = sheet.getRow(i);
28       HSSFCell cell = r.getCell((short) 0);
29       table[i-1][0] = cell.getNumericCellValue();
30       cell = r.getCell((short) 1);
31       table[i-1][1] = cell.getNumericCellValue();
32     }
33     return table;
34   }
35   public static double calcTax(double[][] taxTable, double income) {
36     double tax = 0.0;
37     for (int i = 0; i < taxTable.length && income > 0; i++) {
38       tax += Math.min(income,taxTable[i][0]) * taxTable[i][1];
39       income -= taxTable[i][0];
40     }
41     return tax;
42   }
..   // เพิ่ม readDouble ของรหัสที่ 10-1 ที่นี่
```

ไม่เอาแถวที่เป็นหัวเรื่อง

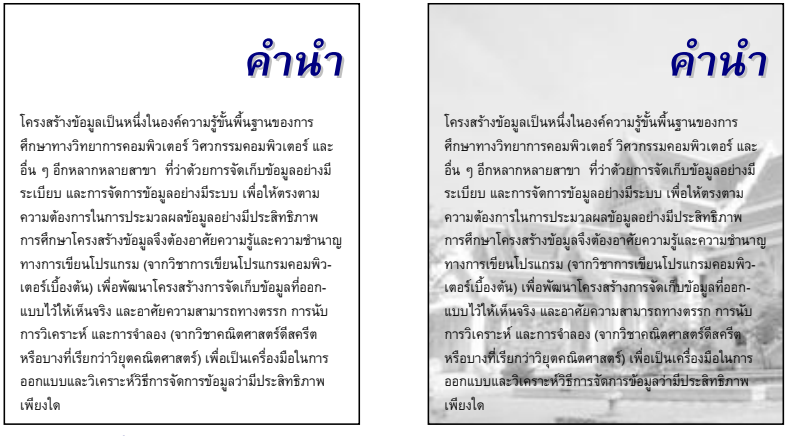
อ่าน เงินได้จำนวนสูงสุดของขั้น

อ่าน อัตราภาษี

รหัสที่ 10-10 โปรแกรมคำนวณภาษี (ตารางภาษีถูกอ่านจากแฟ้ม Excel) (ต่อ)

โปรแกรมเติมภาพฉากหลังในแฟ้มรูปแบบ PDF

ในปัจจุบันแฟ้มรูปแบบ PDF เป็นแฟ้มที่นิยมใช้กันมากในการเก็บและเผยแพร่เอกสาร เนื่องจากสามารถนำแฟ้มแบบนี้มาแสดงบนคอมพิวเตอร์ได้สารพัดยี่ห้อ โดยคงรูปแบบการนำเสนอตัวอักษร รูปภาพได้เหมือนที่ผู้สร้างเอกสารตั้งใจไว้ โปรแกรมที่เราจะเขียนในหัวข้อนี้คือ โปรแกรมที่เติมภาพให้เป็นภาพฉากหลังของทุก ๆ หน้าในเอกสาร PDF รูปที่ 10-5 แสดงตัวอย่างหน้าเอกสารต้นฉบับทางซ้าย เมื่อเติมภาพที่กำหนดให้ไว้เป็นฉากหลังจะได้หน้าเอกสารใหม่ดังรูปขวา



รูปที่ 10-5 เอกสาร PDF ก่อนและหลังเติมภาพฉากหลัง

แล้วเราจะอ่าน แก้ไข และสร้างเอกสารรูปแบบ PDF ได้อย่างไร ปัญหานี้เหมือนกับปัญหาในหัวข้อที่แล้ว ต้องค้นดูว่ามีใครทำอะไรอย่างที่ต้องการหรือไม่ ขอนำเสนอคลังคลาสชื่อ iText (http://www.lowagie.com/iText) ที่มีไว้จัดการเอกสาร PDF รหัสที่ 10-11 มี เมท็อด `main` เปิดแฟ้มขาเข้าด้วยการสร้าง `PDFReader` สร้างแฟ้มขาออกด้วย `PDFStamper` ซึ่งมีความสามารถในการนำแฟ้มขาเข้ามาปรับปรุงให้ได้แฟ้มใหม่ และอ่านแฟ้มภาพด้วยเมท็อด `loadImage` (ซึ่งจะอธิบายภายหลัง) จากนั้นวนเพิ่มภาพนี้ให้ครบทุกหน้าของเอกสาร (บรรทัดที่ 17 ถึง 18) ปิดท้ายด้วยการปิดแฟ้มให้เรียบร้อย ขั้นตอนที่บรรยายมาอาจเกิดสิ่งผิดปกติได้สามประการ คือ หาแฟ้มที่ต้องการใช้ไม่พบ มีปัญหากับตัวเอกสาร PDF และปัญหาอื่น ๆ ที่เกี่ยวกับแฟ้มข้อมูล จึงต้อง `catch` ทั้งสามแบบ (บรรทัดที่ 20 ถึง 26) ซึ่งก็จัดการได้เพียงแค่แสดงคำอธิบายข้อผิดพลาดให้ผู้ใช้ทราบ และหยุดการทำงานของโปรแกรม

`loadImage` เป็นเมท็อดในการอ่านแฟ้มภาพ จะว่าไปแล้ว เมท็อดที่อ่านแฟ้มภาพตัวจริงคือ `Image.getInstance` (บรรทัดที่ 37) แต่เราเพิ่มความสามารถให้รองรับกรณีที่พบปัญหาในการอ่านแฟ้มภาพ โดยลองเปลี่ยนประเภทแฟ้มภาพ (ชื่อภาพเหมือนเดิม) แล้วอ่านใหม่ ในที่นี้เขียนไว้ให้ลองอ่าน 5 ประเภท โดยเขียนให้ `catch` รับสิ่งผิดปกติแบบ `Exception` (บรรทัดที่ 38) แทนการรับสิ่งผิดปกติเกือบทุกประเภท (ยกเว้นแบบ `Error`) ไม่ว่าสิ่งผิดปกติที่เกิดขึ้นจะเป็น

FileNotFoundException หรือ IOException ถือว่าเป็น Exception ทั้งสิ้น (อย่าลืมลากแฟ้ม iText-2.1.3.jar ใน c:/java101 ซึ่งคือคลังคลาสของ iText มาวางที่หัวของ JLab ก่อนสั่งโปรแกรมทำงาน)

```
01 import java.io.*;
02 import com.lowagie.text.*;
03 import com.lowagie.text.pdf.*;
04
05 public class PDFBackground {
06   public static void main(String[] args) {
07     String inFile = "c:/java101/input.pdf";
08     String outFile = "c:/java101/output.pdf";
09     String watermarkFile = "c:/java101/background.gif";
10     try {
11       PdfReader in = new PdfReader(inFile);
12       PdfStamper out = new PdfStamper(in,
13                                new FileOutputStream(outFile));
14       Image img = loadImage(watermarkFile);
15       img.setAbsolutePosition(0, 0);
16       int n = in.getNumberOfPages();
17       for (int i = 1; i <= n; i++)
18         out.getUnderContent(i).addImage(img);
19       out.close();
20     } catch (FileNotFoundException e) {
21       System.out.println("ไม่พบแฟ้ม " + e.getMessage());
22     } catch (DocumentException e) {
23       System.out.println("มีปัญหากับเอกสาร" + e.getMessage());
24     } catch (IOException e) {
25       System.out.println("มีปัญหาเกี่ยวกับแฟ้ม " + e.getMessage());
26     }
27   }
28   private static Image loadImage(String imageFile)
29                                     throws FileNotFoundException {
30     String[] types = {"jpg", "jpeg", "gif", "png", "bmp" };
31     int k = imageFile.lastIndexOf(".");
32     String fn = imageFile;
33     if (k >= 0) fn = imageFile.substring(0, k);
34     String errMsg = "";
35     for (int i = 0; i < types.length; i++) {
36       try {
37         return Image.getInstance(fn + "." + types[i]);
38       } catch (Exception e) {
39         errMsg += e.getMessage() + "\n";
40       }
41     }
42     throw new FileNotFoundException(errMsg);
43   }
44 }
```

เพิ่มภาพให้กับทุกหน้าในเอกสาร

ลบประเภทแฟ้มออก เช่น b.gif -> b

ลองใช้ประเภทแฟ้มภาพแบบที่ i

ถ้าผิด เก็บสาเหตุไว้

ถ้าผิดหมดทั้ง 5 แบบ ให้โยนสิ่งผิดปกติ

รหัสที่ 10-11 โปรแกรมเพิ่มภาพฉากหลังลงในแฟ้ม PDF

โปรแกรมสรุปหัวข้อข่าว

บทที่ 5 ได้นำเสนอ URLStream ที่ใช้ร่วมกับ Scanner เพื่ออ่านข้อมูลจากเว็บเพจที่ต้องการ ซึ่งอาจโยนสิ่งผิดปกติแบบไม่ถูกตรวจระหว่างการอ่าน หัวข้อนี้นำเสนอคลาส WebPage (รหัสที่ 10-12) ที่สร้างจากคลาสมาตรฐานของจาวา มีวิธีใช้งานต่างกันเล็กน้อย คือ ต้องสร้างอ็อบเจกต์ของ WebPage ก่อน แล้วค่อยเรียกเมท็อด getScanner() จึงจะได้ตัวอ่านที่ให้ข้อมูลจากเว็บเพจ หากเกิดปัญหาระหว่างการอ่านเว็บ จะโยนสิ่งผิดปกติแบบที่ถูกตรวจ

```
01 import java.io.*;
02 import java.net.*;
03 import java.util.Scanner;
04
05 public class WebPage {
06   private Scanner scanner;
07   public WebPage(String page) throws WebPageException {
08     try {
09       URLConnection urlc = new URL(page).openConnection();
10       scanner = new Scanner(urlc.getInputStream());
11     } catch (IOException e) {
12       throw new WebPageException(e.getMessage());
13     }
14   }
15   public Scanner getScanner() {
16     return scanner;
17   }
18 }
```

รหัสที่ 10-12 คลาส WebPage มีเมท็อดคืน Scanner ให้อ่านเว็บเพจ

ตัวสร้าง WebPage รับสตริงที่เก็บตำแหน่งของเว็บเพจ (พารามิเตอร์ชื่อ page) เริ่มด้วยคำสั่ง new URL(page) เพื่อสร้างอ็อบเจกต์ที่แทนตำแหน่ง แล้วเรียก openConnection เพื่อเชื่อมต่อไปยังเว็บเพจ ซึ่งพร้อมส่งข้อมูลให้ตัวอ่านที่เราสร้างเก็บไว้ในตัวแปร scanner (บรรทัดที่ 10) ตัวสร้าง URL อาจโยนสิ่งผิดปกติแบบ MalformedURLException เมท็อด openConnection กับเมท็อด getInputStream อาจโยนสิ่งผิดปกติแบบ IOException เนื่องจาก IOException เป็นคลาสแม่ของ MalformedURLException จึงขอเขียนกลุ่ม catch ให้รับ IOException แบบเดียว แล้วสามารถรับสิ่งผิดปกติได้ทั้งสองแบบ โดยจะไม่จัดการกับสิ่งผิดปกติที่เกิดขึ้น แต่จะสร้างและโยนสิ่งผิดปกติแบบ WebPageException (รหัสที่ 10-5) ให้ผู้เรียกรับทราบแทน การรับและโยนต่อในรูปแบบใหม่เช่นนี้ เป็นเสมือนการเปลี่ยนความหมายของสิ่งผิดปกติจากประเภท IOException ไปเป็น WebPageException อ่านแล้วมีความหมายใกล้เคียงกับสิ่งที่เกิดขึ้น สื่อความหมายมากกว่า

รหัสที่ 10-13 แสดงการใช้คลาส WebPage เพื่ออ่านข้อมูลสรุปหัวข้อข่าวการเมืองจากสำนักข่าวไทยที่ http://news.mcot.net/news_rss/politic.xml ข้อมูลที่อ่านได้มีโครงสร้างดังนี้

```
...
<item>
...
<title>   หัวข้อข่าว   </title>
...
</item>
...
```

เมท็อดในรหัสที่ 10-13 เริ่มด้วยการสร้าง `WebPage` ที่เชื่อมไปยังข้อมูลสรุปข่าวของสำนักข่าวไทย (บรรทัดที่ 7) การสร้างนี้มีโอกาสเกิดสิ่งผิดปกติ จึงเตรียมกลุ่ม `catch` ไว้รองรับ ในที่นี้เราเพียงแค่แสดงสาเหตุของความผิดปกติให้ผู้ใช้ทราบเท่านั้น ถ้าสร้าง `WebPage` ได้สำเร็จ จะขอตัวอ่าน `Scanner` จาก `WebPage` ที่สร้างได้ (บรรทัดที่ 8) แล้วเข้าวงวนอ่านข้อความจากตัวอ่านที่ได้เพื่อนำหัวข้อน่ามาแสดง โดยค้นคำว่า `<item>` (บรรทัดที่ 14) พบเมื่อไร จะค้นหา `<title>` และ `</title>` ต่อ (บรรทัดที่ 16 และ 17) ถ้าพบทั้งคู่จะได้ข้อความที่อยู่ระหว่างสองคำนี้เป็นหัวข้อข่าว (บรรทัดที่ 18) ขอให้ผู้อ่านทำความเข้าใจกับรายละเอียดการทำงานของรหัสที่ 10-13 เอง

```
01 import java.util.Scanner;
02
03 public class SimpleNewsAggregator {
04   public static void main(String[] args) {
05     String page = " http://news.mcot.net/news_rss/politic.xml";
06     try {
07       WebPage webpage = new WebPage(page);
08       Scanner sc = webpage.getScanner();
09       int start, end;
10       int n = 1;
11       boolean itemFound = false;
12       while (sc.hasNext()) {
13         String line = sc.nextLine();
14         if (line.indexOf("<item>") >= 0) itemFound = true;
15         if (itemFound) {
16           if ((start = line.indexOf("<title>")) >= 0) {
17             if ((end = line.indexOf("</title>")) >= 0) {
18               String title = line.substring(start+7, end);
19               System.out.println((n++) + ": \t" + title);
20             }
21             itemFound = false;
22           }
23         }
24       }
25     } catch (WebPageException e) {
26       System.out.println("มีปัญหาเมื่ออ่านเว็บเพจ : " + e.getMessage());
27     }
28   }
29 }
```

หาให้พบ <item> ก่อน

หาและแสดงข้อความที่อยู่ระหว่าง <title> ... </title>

รหัสที่ 10-13 ตัวอย่างการใช้ `WebPage` เพื่อแสดงสรุปข่าวการเมืองจากสำนักข่าวไทย

สุดท้ายนี้

นอกจากกลุ่ม try และกลุ่ม catch แล้ว ยังมีอีกกลุ่มชื่อกลุ่ม finally มีไว้เขียนปิดท้าย จึงเรียกชื่อเต็มว่า try-catch-finally กลุ่ม finally บรรจุคำสั่งที่ระบบจะไปทำทุกครั้งหลังทำกลุ่ม try เสร็จ หรือหลังทำกลุ่ม catch เสร็จ หรือก่อนเกิดการโยนต่อ (กรณีไม่มีใคร catch) นั่นคือ ไม่ว่าการทำงานในกลุ่ม try จะทำได้ตามปกติไม่เกิดความผิดปกติใดๆ หรือว่าทำแล้วเกิดสิ่งผิดปกติ สุดท้ายแล้ว ก็ต้องมาทำคำสั่งในกลุ่ม finally ด้วย

รหัสที่ 10-14 แสดงตัวอย่างการทำงานของ try-catch-finally โปรแกรมนี้ไม่ได้ทำอะไรเป็นเรื่องเป็นราว เป็นเพียงตัวอย่างให้ทำความเข้าใจกับลำดับการทำงานเท่านั้น หากลองไล่ติดตามการทำงาน จะพบว่า

- doSomething(1) เรียก doNothing(1) ที่คืนการทำงานตามปกติ จะแสดงผลที่บรรทัดที่ 10 แล้วไปแสดงอีกครั้งใน finally ที่บรรทัดที่ 16
- doSomething(2) เรียก doNothing(2) เกิดการโยนสิ่งผิดปกติ จะกระโดดไปแสดงผลที่บรรทัดที่ 14 แล้วไปแสดงอีกครั้งใน finally ที่บรรทัดที่ 16
- doSomething(3) เรียก doNothing(3) เกิดการโยนสิ่งผิดปกติ แต่ไม่มีตัวจัดการ doSomething จึงโยนต่อให้ผู้เรียก (คือ main) แต่จะไปทำใน finally ที่บรรทัดที่ 16 ก่อนโยนต่อ

```
01 import java.io.*;
02 import java.net.*;
03 public class TestFinally {
04   public static void main(String[] args) throws IOException {
05     doSomething(1);  doSomething(2);  doSomething(3);
06   }
07   private static void doSomething(int k) throws IOException {
08     try {
09       doNothing(k);
10       System.out.println("Try is sucessfully executed");
11     } catch (MalformedURLException e) {
12       System.out.println("catch MalformedURLException");
13     } catch (FileNotFoundException e) {
14       System.out.println("catch FileNotFoundException");
15     } finally {
16       System.out.println("Finally, I am here");
17     }
18   }
19   private static void doNothing(int k) throws IOException {
20     if (k == 2) throw new FileNotFoundException();
21     if (k == 3) throw new IOException();
22   }
```

รหัสที่ 10-14 ตัวอย่างการทำงานของ try-catch-finally

ได้ผลการทำงานของโปรแกรมในรหัสที่ 10-14 ที่แสดงทางจอภาพข้างล่างนี้

```
Try is sucessfully executed
Finally, I am here
catch FileNotFoundException
Finally, I am here
Finally, I am here
Exception in thread "main" java.io.IOException
  at TestFinally.doNothing(TestFinally.java:25)
  at TestFinally.doSomething(TestFinally.java:12)
  at TestFinally.main(TestFinally.java:8)
```

doSomething(1)

doSomething(2)

doSomething(3)

ย้อนกลับไปดูรหัสที่ 10-11 ซึ่งเป็นโปรแกรมเพิ่มภาพฉากหลังให้กับทุกหน้าในเอกสาร PDF มีการประมวลผลแฟ้ม PDF ขาเข้าและขาออก ตามมารยาทแล้วเราควรปิดแฟ้มหลังจากใช้งานเสร็จ แต่ในรหัสที่ 10-11 นั้น มีแต่การปิดแฟ้มขาออก `out` ด้วยคำสั่ง `out.close()` เพราะถ้าไม่ปิดการบันทึกผลลัพธ์จะไม่สำเร็จ ในขณะที่เราไม่ได้ปิดแฟ้มขาเข้า ซึ่งก็ไม่ได้ทำให้ผลลัพธ์ผิดเพี้ยนไปอย่างไรก็ตาม ตราบใดที่เรายังไม่ปิดแฟ้ม ระบบก็ยังคงต้องกันเนื้อที่หน่วยความจำไว้สำหรับแฟ้มที่ยังไม่ได้ปิด ทำให้สูญเสียทรัพยากรโดยใช่เหตุ ดังนั้น จึงควรปิดแฟ้มทุกครั้งที่มั่นใจว่าเราไม่ต้องการใช้งานแล้ว

รหัสที่ 10-15 แสดงโครงของโปรแกรมที่ปรับปรุงมาจากส่วนของโปรแกรมในรหัสที่ 10-11 โดยเพิ่มกลุ่ม `finally` ปิดท้าย ภายในมีคำสั่งปิดแฟ้มทั้ง `in` และ `out` ทำให้มั่นใจได้ว่า ไม่ว่าการทำงานจะดำเนินการตามปกติได้สำเร็จ หรือดำเนินการแล้วมีปัญหา พบสิ่งผิดปกติ แฟ้มทั้งสองจะถูกปิดตามที่ควรทำ (ถ้า `in` หรือ `out` เป็น `null` ตามค่าเริ่มต้น แสดงว่า ระบบยังไม่เปิดแฟ้มก็ไม่จำเป็นต้องสั่งปิด)

```
...
PdfReader in = null;
PdfStamper out = null;
try {
  in = new PdfReader(inFile);
  out = new PdfStamper(in, new FileOutputStream(outFile));
  ...
} catch (FileNotFoundException e) {
  ...
} catch (DocumentException e) {
  ...
} catch (IOException e) {
  ...
} finally {
  if (in  != null) in.close();
  if (out != null) out.close();
}
```

ทำ try เสร็จ หรือทำ catch ใดเสร็จ ก็ต้องไปทำที่ finally

รหัสที่ 10-15 การเขียนคำสั่งปิดแฟ้มไว้ใน `finally`

เพิ่มเติม

หัวข้อนี้นำเสนอเรื่องเพิ่มเติม เมื่อต้องเขียนเมท็อดของคลาสลูกเพื่อแทนของคลาสแม่ โดยเมท็อดมีการโยนสิ่งผิดปกติ ตามด้วยเนื้อหาของคำสั่ง `assert` ที่มีการทำงานคล้ายกับการโยนสิ่งผิดปกติเมื่อผิดเงื่อนไข แต่มีไว้เพื่อวัตถุประสงค์ที่ต่างกัน

การแทนเมท็อดของคลาสแม่ที่โยนสิ่งผิดปกติ

มีเรื่องจุกจิกเล็กน้อยเกี่ยวกับการเขียนเมท็อดของคลาสลูกเพื่อแทนเมท็อดที่ได้รับทอดจากคลาสแม่ โดยเมท็อดที่คลาสลูกนี้มีการโยนสิ่งผิดปกติที่แตกต่างจากเมท็อดที่คลาสแม่ ก่อนอื่นขอเน้นว่า สิ่งผิดปกติที่กล่าวถึงในหัวข้อนี้จะหมายถึงเฉพาะสิ่งผิดปกติแบบที่ถูกตรวจเท่านั้น ต้องเข้าใจก่อนว่า ที่หัวเมท็อดของแม่ประกาศว่าทำอะไร ก็เสมือนเป็นสัญญาให้ผู้ใช้รับทราบ ลูกก็ต้องปฏิบัติตามสัญญานั้นด้วย ถ้าเมท็อดที่คลาสแม่ประกาศว่าไม่โยนสิ่งผิดปกติอะไร ของลูกที่จะมาแทนของแม่ก็ต้องห้ามโยนเช่นกัน แต่ถ้าเมท็อดของแม่ประกาศว่า อาจโยนแบบ `A` เมท็อดของลูกที่จะแทนเมท็อดนี้ของแม่ก็ต้องประกาศว่าอาจโยนแบบ `A` หรืออาจโยนแบบอื่นที่มองเป็น `A` ได้ (นั่นคือ upcast เป็น `A` ได้ หรืออีกความหมายหนึ่งคือ เป็นคลาสลูกหลานของ `A`) ก็ย่อมทำได้ หรือจะประกาศว่าไม่โยนอะไรก็ได้เช่นกัน อย่างนี้ถือว่าปฏิบัติตามสัญญาเหมือนกับที่แม่ให้ไว้ แต่ถ้าโยนคนละแบบที่ไม่เป็นตามที่กล่าวไว้ข้างต้น ตัวแปลโปรแกรมจะไม่ยอม

```
public class Super {
  public void p() {
    ...
  }
  public void q() throws FileNotFoundException {
    ...
  }
  public void r() throws IOException {
    ...
  }
```

```
public class Sub extends Super {
  @Override public void p() throws IOException { // ผิด แม่ไม่ได้โยน
    ...
  }
  @Override public void q() throws IOException { // ผิด โยนผิดแบบ
    ...
  }
  @Override public void r() throws FileNotFoundException { //ถูกต้อง
    ...
  }
```

รหัสที่ 10-16 ตัวอย่างการแทนเมท็อดของคลาสแม่ที่มีการโยนสิ่งผิดปกติ

ขอนำเสนอด้วยตัวอย่าง รหัสที่ 10-16 แสดงคลาส `Sub` ที่เป็นคลาสลูกของ `Super` มีการแทนเมท็อด `p`, `q` และ `r` พิจารณาเมท็อด `p` ก่อน ในคลาสแม่ไม่ได้ประกาศจะโยนอะไรเลย แต่ `p` ของคลาสลูกเขียนว่าจะโยน `IOException` แบบนี้ทำไม่ได้ มาดูเมท็อด `q` ในคลาสแม่ประกาศว่าจะโยน `FileNotFoundException` แต่ `q` ของคลาสลูกประกาศว่าจะโยน `IOException` ซึ่งไม่ใช่คลาสลูกหลานของ `FileNotFoundException` แบบนี้ก็ทำไม่ได้เช่นกัน แต่ในทางกลับกัน เมท็อด `r` ของคลาสลูกที่แทนของคลาสแม่นั้นทำได้ไม่มีปัญหา

assert : การยืนยันความจริง

ระหว่างการเขียนโปรแกรม นักเขียนโปรแกรมมักมีสมมติฐานอะไรบางอย่างเกี่ยวกับค่าของพารามิเตอร์ ค่าของตัวแปรประจำอ็อบเจกต์ หรือสภาพแวดล้อมอะไรบางอย่างว่าต้องเป็นแบบนั้นแบบนี้ ทำให้บางครั้งอาจละเลยไม่เขียนคำสั่งตรวจสอบสมมติฐานเหล่านั้น (เพราะมั่นใจ) สิ่งที่นักเขียนโปรแกรมบางคนอาจทำก็คือ เขียนเป็นหมายเหตุกำกับไว้ในโปรแกรมเพื่อให้ผู้อ่านโปรแกรมรับทราบสมมติฐานเหล่านั้น แต่เนื่องจากโปรแกรมส่วนใหญ่ได้รับการปรับปรุงอย่างสม่ำเสมอ เพื่อเพิ่มความสามารถและเพื่อแก้ไขจุดบกพร่อง จึงมักพบในภายหลังว่า สมมติฐานที่ตั้งไว้ ณ ตำแหน่งต่าง ๆ ของโปรแกรมที่นักเขียนโปรแกรมเดิมเคยวางไว้ กลับไม่เป็นจริงอีกต่อไปโดยไม่มีใครรู้ และอาจก่อให้เกิดปัญหาในภายหลัง ดังนั้น แนวปฏิบัติที่สำคัญมากคือ เมื่อใดที่เขียนเป็นหมายเหตุให้ “คน” อ่าน ก็ควรเขียนเป็นคำสั่งให้ “เครื่อง” ตรวจสอบด้วย ถ้าพบในขณะทำงานว่าผิดเงื่อนไขที่ตรวจสอบ ก็ให้โยนสิ่งผิดปกติทันที

นอกจากจาวาจะมีคำสั่ง `throw` และกลไกการจัดการสิ่งปกติเพื่อเพิ่มความทนทานให้กับซอฟต์แวร์ตามที่ได้นำเสนอมาแล้ว จาวายังมีคำสั่ง `assert` ที่มีลักษณะการทำงานคล้ายการกับการโยนสิ่งผิดปกติ แต่มีวัตถุประสงค์ต่างกัน คือ ใช้เพื่อเพิ่มความเชื่อถือได้ของซอฟต์แวร์ ก่อนอื่นมาทำความเข้าใจกับการใช้งาน `assert` กันก่อน ซึ่งมีรูปแบบดังนี้

`assert` นิพจน์บูลีน ;	หรือ	`assert` นิพจน์บูลีน : นิพจน์ ;

เมื่อ `assert` ทำงาน ระบบจะประเมินค่าของนิพจน์บูลีนด้านขวาของ `assert` ถ้าเป็นจริง ระบบจะทำงานต่อไป แต่ถ้าเป็นเท็จ ระบบจะสร้างและโยนสิ่งผิดปกติของคลาส `AssertionError` ถ้าเป็นรูปแบบทางขวามือข้างบน จะนำค่าของนิพจน์หลังเครื่องหมาย : ส่งให้ตัวสร้างสิ่งผิดปกติ รหัสที่ 10-17 แสดงตัวอย่างการตรวจสอบค่าของพารามิเตอร์

```
private void heatReactor(int temp) {
  assert (200 <= temp && temp <= 1000) : temp;
  reactor.setHeat(temp);
  ...
}
```

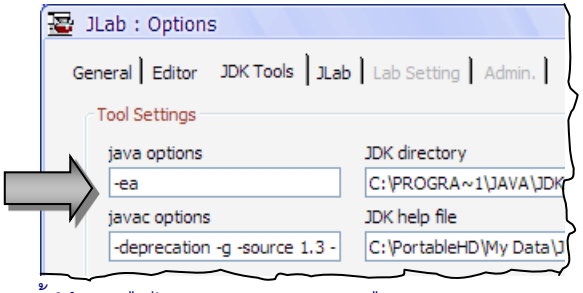
อุณหภูมิที่ถูกต้องต้องอยู่ระหว่าง 200 ถึง 1000

รหัสที่ 10-17 การใช้ `assert` เพื่อตรวจสอบค่าของพารามิเตอร์

ผู้อ่านอาจสงสัยว่า ทำไมต้องมี `assert` ด้วย ในเมื่อที่ผ่านมาเรา เราใช้คำสั่ง `throw new IllegalArgumentException(...)` ก็ได้ผลคล้ายกัน แล้วจะต่างกันตรงไหน ข้อแตกต่างที่ตอบแบบกำปั้นทุบดินคือ สองคำสั่งนี้โยนสิ่งผิดปกติต่างชนิดกัน แต่สาเหตุที่เราใช้ `assert` มีสองประเด็นคือ

- เราใช้ `assert` กับเงื่อนไขที่เราคิดว่าไม่มีทางเป็นเท็จ แต่เราใช้การโยนสิ่งผิดปกติกับเงื่อนไขที่มีโอกาสเป็นเท็จได้ หากเงื่อนไขหลัง `assert` ที่เราคิดว่าต้องเป็นจริงเสมอนี้กลับเป็นเท็จย่อมแสดงว่า ตัวโปรแกรมมีจุดบกพร่องแน่ ๆ จากตัวอย่างในรหัสที่ 10-17 ให้สังเกตว่า เมท็อดนี้เป็น `private` ผู้ที่จะเรียกเมท็อดนี้ได้คือ ผู้เขียนคลาสที่เมท็อดนี้อยู่เท่านั้น ผู้อื่นเรียกใช้ไม่ได้ ดังนั้น หากขณะทำงานกลับพบว่าไม่เป็นจริง ตัวนักเขียนโปรแกรมนั่นเองต้องเป็นผู้เขียนผิด และเนื่องจาก `AssertionError` เป็นคลาสลูกของคลาส `Error` ซึ่งเป็นแบบที่ระบุความรุนแรงระดับเดียวกับสิ่งผิดปกติจากระบบ ซึ่งปกติจะไม่ `catch` กัน ตัวโปรแกรมต้องหยุดทำงานทันที
- หากมีคำสั่ง `assert` เพื่อตรวจสอบเงื่อนไขระหว่างการทำงานกระจายตามที่ต่าง ๆ ในโปรแกรม [5] โดยระหว่างการทดสอบโปรแกรม (ในระยะเวลาที่นานพอควร) พบว่า ไม่ผิดเงื่อนไขหลัง `assert` ใด ๆ เลย ย่อมเพิ่มความเชื่อมั่นในโปรแกรมที่ทดสอบ พร้อมนำไปใช้งานจริง หากจะมีการผิดเงื่อนไข ก็ขอให้เกิดขณะยังอยู่ในระหว่างการทดสอบจะได้แก้ไขจุดบกพร่องได้ นักเขียนโปรแกรมอาจกังวลว่า การตรวจสอบเงื่อนไขหลังคำสั่ง `assert` อาจเป็นภาระเพิ่มเติมที่ทำให้ระบบทำงานช้าลง ประเด็นนี้ไม่ต้องห่วง เพราะเราสามารถสั่งให้ระบบปิดทางหรือเปิดทางให้คำสั่ง `assert` ทำงานหรือไม่ก็ได้ โดยทั่วไปจะเปิดให้ `assert` ทำงานในช่วงการทดสอบโปรแกรม และปิดเมื่อใช้งานจริง การเปิดหรือปิดทางทำงานของ `assert` ทำได้โดยการเพิ่มตัวเลือก –ea หรือ –da ขณะสั่งให้ระบบทำงานด้วยคำสั่ง java.exe (ผ่านทาง command prompt ที่เคยนำเสนอในบทที่ 7) เช่น ต้องการสั่งคลาส A ทำงานโดยเปิดให้คำสั่ง `assert` ทำงาน ก็สั่งด้วย `java -ea A` ในกรณีที่ใช้ JLab เขียนโปรแกรม ระบบจะตั้ง –ea ให้อัตโนมัติ หรือถ้าต้องการเปลี่ยนแปลง สามารถตั้งค่า –ea หรือ –da ได้โดยเลือกเมนู Tools → Options จะปรากฏกล่องโต้ตอบ ให้เลือกที่ JDK Tools แล้วเติมค่าที่ต้องการในช่อง java options ดังแสดงเป็นตัวอย่างในรูปที่ 10-6

[5] เมื่อปี ค.ศ. 2005 มีรายงาน (http://users.encs.concordia.ca/~chalin/papers/TR-2005-001-r2.pdf) การสำรวจซอฟต์แวร์ต่าง ๆ พบว่า ซอฟต์แวร์แบบ open source มีจำนวนบรรทัดเฉลี่ยที่ทำหน้าที่คล้ายกับคำสั่ง `assert` กระจายอยู่ในตัวซอฟต์แวร์ประมาณ 5.1% ของจำนวนบรรทัดทั้งหมดของตัวซอฟต์แวร์ ในกรณีของซอฟต์แวร์ระบบปิดมีประมาณ 3.27% สำหรับ Microsoft Office มีประมาณ 250,000 บรรทัด (คิดเป็น 1% ของจำนวนบรรทัดทั้งหมด)



รูปที่ 10-6 การตั้งให้ทำหรือไม่ทำ `assert` โดยเลือกเมนู Tools →Options ใน JLab

แล้วเราจะเขียน `assert` ที่ใดกันบ้าง มีข้อแนะนำในการเขียนคำสั่ง `assert` ดังนี้

- ตรวจสอบเงื่อนไขที่ต้องเป็นจริงก่อนเริ่มทำงานในเมท็อด โดยจะใช้กับเมท็อดที่เป็น `private` (เขียนเองใช้เอง) หรือเป็นแบบใช้เฉพาะในแพ็กเกจเดียวกัน (เขียนใช้กันภายในกลุ่มที่พัฒนาซอฟต์แวร์เดียวกัน) ก่อนจะใช้เมท็อดมักมีสมมติฐานที่ควรเป็นจริงก่อนการเรียกใช้ ถ้าเป็นเมท็อดที่ใช้ส่วนตัวและให้เพื่อนในกลุ่มใช้ ควรต้องปฏิบัติตามอย่างเคร่งครัด เพราะตัวเองกับกลุ่มเดียวกันเองเป็นผู้ตั้งเงื่อนไข หากเงื่อนไขไม่เป็นจริง แสดงว่าเป็นจุดบกพร่อง (ดังตัวอย่างในรหัสที่ 10-17)
- ตรวจสอบเงื่อนไขหลังจากที่เมท็อดทำงานเสร็จ เพื่อตรวจสอบย้ำอีกครั้งว่า ที่เราตั้งใจให้เมท็อดทำงานนั้น ทำได้ตามวัตถุประสงค์หรือไม่ ตัวอย่างเช่น เขียนเมท็อดให้เรียงลำดับข้อมูลในอาเรย์ ก็สามารถตรวจสอบว่าข้อมูลในอาเรย์เรียงลำดับจริงหรือไม่ ดังแสดงในรหัสที่ 10-18 (การตรวจสอบในลักษณะนี้ กระทำได้กับเมท็อดทุกประเภท)

```
public static void sort(int[] d) {
   ... // รหัสที่ทำการเรียงลำดับ

   for (int i=0; i<d.length-1; i++) {
     assert d[i] <= d[i+1];   // ตัวซ้ายต้องไม่มากกว่าตัวขวา
   }
}
```

รหัสที่ 10-18 การใช้ `assert` เพื่อตรวจสอบผลหลังการทำงาน

- ตรวจสอบสมมติฐานที่มั่นใจแทนการเขียนหมายเหตุกำกับคำสั่ง ตัวอย่างเช่น นักเขียนโปรแกรมมั่นใจว่าผลจาก `getNumberOfStudents()` ในรหัสที่ 10-19 มีค่าเกินศูนย์แน่ จึงเขียน `assert` เพื่อให้ตรวจความมั่นใจนั้น

```
...
int numberOfStudents = myClass.getNumberOfStudents();
assert numberOfStudents > 0 : numberOfStudents;
...
```

รหัสที่ 10-19 การใช้ `assert` เพื่อตรวจสอบค่าตามที่คาดไว้

หรือในกรณีของคำสั่ง switch-case ที่นักเขียนโปรแกรมมักมั่นใจว่า เขียน case ครอบคลุมทุกกรณีแล้ว ก็ควรเขียน assert false ไว้ที่กรณี default ดังตัวอย่างในรหัสที่ 10-20 เรามั่นใจว่า มีสามกรณีเท่านั้น จึงเขียนไว้สาม case ถ้าเกิด flavor มีค่าอื่นที่ไม่ใช่สามกรณีที่คาดไว้ จะไปทำ assert เนื่องจากค่าที่ตามหลังเป็น false จึงเสมือนการบังคับให้เกิดข้อผิดพลาดขึ้น

```
...
switch(flavor) {
  case VANILLA :  ...   break;
  case COCONUT :  ...   break;
  case LEMON   :  ...   break;
  default      :  assert false : flavor;
}
```

รหัสที่ 10-20 การใช้ assert false ที่ default เมื่อมั่นใจว่าไม่มีทางเกิดขึ้น

สิ่งหนึ่งที่ต้องระวังเป็นพิเศษในการใช้ assert ก็คือ นิพจน์บูลีนที่เขียนเพื่อตรวจสอบเงื่อนไขนั้นต้องไม่ก่อให้เกิดการเปลี่ยนแปลงกับสถานะหรือค่าของตัวแปรใด ๆ เช่น คำสั่ง

```
assert ++c < 20 : c;
```

มีการเปลี่ยนค่าของ c ในการตรวจสอบ เนื่องจากผู้สั่งโปรแกรมทำงานสามารถปิดหรือเปิดให้ assert ทำหรือไม่ทำงาน ซึ่งให้ผลการทำงานต่างกัน (ถ้า assert ทำงาน c จะมีค่ามากกว่า assert ไม่ทำงาน) ซึ่งไม่ถูกต้อง น่าจะเขียน ++c; ก่อน แล้วตามด้วย assert c<20 :c;

อย่าลืมว่า เราควรเขียน assert กำกับคำสั่งภายในเมท็อดหนึ่ง ขณะที่กำลังเขียนเมท็อดนั้น เพราะถ้ารอไปเขียนในภายหลัง อาจลืมสมมติฐานต่าง ๆ ที่คิดไว้ตอนเขียนเมท็อด ทำให้เขียน assert ไม่ครบ การเขียน assert กระจายไว้ทั่ว ๆ ไม่ได้มีข้อเสีย ไม่ต้องห่วงเรื่องเวลาการทำงาน เพราะเราปิดไม่ให้ assert ทำงานก็ได้ (โดยไม่ต้องเปลี่ยนรหัสต้นฉบับ หรือรหัสเครื่องแต่อย่างใด) ขอเน้นอีกครั้งว่า assert ช่วยเพิ่มความเชื่อถือได้ให้กับโปรแกรม ในขณะที่การโยนและจัดการสิ่งผิดปกติช่วยเพิ่มความทนทานให้กับโปรแกรม

แบบฝึกหัด

1. รหัสที่ 10-11 แสดงการใช้ try-catch ซึ่งมีการดักจับสิ่งผิดปกติสามแบบ อยากทราบว่า
   - ถ้าจะยุบการ catch ทั้งสามรวมเหลือแค่ catch(IOException e) จะได้หรือไม่
   - ถ้ายุบเป็น catch(RuntimeException e) จะได้หรือไม่
   - ถ้ายุบเป็น catch(Exception e) จะได้หรือไม่

- ถ้ายุบเป็น `catch(Error e)` จะได้หรือไม่
- ถ้ายุบเป็น `catch(Throwable e)` จะได้หรือไม่
- ถ้ายุบเป็น `catch(Object e)` จะได้หรือไม่

ในกรณีที่ทำไม่ได้ ให้หาเหตุผลประกอบด้วย สำหรับกรณีที่ยุบรวมได้ จะต้องเขียนคำสั่งภายใน `catch` อย่างไร จึงจะสามารถแสดงข้อความให้ผู้ใช้รับทราบถึงเหตุการณ์ที่เกิดขึ้นได้สามกรณีเหมือนที่ทำใน รหัสที่ 10-11 โดยใช้แค่ `catch` เดียว (อนึ่งการยุบรวมที่เสนอให้ทำในแบบฝึกหัดนี้เป็นสิ่งที่ไม่ควรเขียนในทางปฏิบัติ เพียงต้องการทดสอบความเข้าใจเท่านั้น)

2. หากลองเปลี่ยนรหัสที่ 10-11 ให้ปิดแฟ้ม `in` และ `out` ในกลุ่ม `finally` ตามข้อแนะนำในรหัสที่ 10-15 จะพบว่ายังมีปัญหาอยู่ จงแก้ไขให้ถูกต้อง

3. จงปรับปรุงคลาส `ArrayList` ในแบบฝึกหัดที่ 15 ของบทที่ 9 ให้ตรวจสอบและโยนสิ่งผิดปกติในเมท็อดต่าง ๆ ที่ให้บริการดังนี้
   - `add(x)` : `x` ต้องไม่เป็น `null`
   - `add(i,x)` : `x` ต้องไม่เป็น `null` และ $0 \le i \le$ `size()`
   - `remove(i)` : $0 \le i \le$ `size()-1`
   - `size()` : ไม่มีอะไรต้องตรวจสอบ
   - `set(i, x)` : `x` ต้องไม่เป็น `null` และ $0 \le i \le$ `size()-1`
   - `get(i)` : $0 \le i \le$ `size()-1`

4. เมื่อสั่งโปรแกรมต่อไปนี้ทำงานจะเกิดสิ่งผิดปกติอะไร ที่บรรทัดใด จงหาสาเหตุและวิธีแก้ไข

```
import java.util.ArrayList;
import java.util.Arrays;
public class StringBin {
  private ArrayList list = new ArrayList();
  public void add(String s) {
    list.add(s);
  }
  public String[] toArray() {
    String[] out = new String[list.size()];
    for (int i = 0; i <= list.size(); i++)
      out[i] = (String)list.get(i);
    return out;
  }
  public static void main(String[] args) {
    StringBin b = new StringBin();
    b.add("A"); b.add("B"); b.add("C");
    String[] s = b.toArray();
    System.out.println(Arrays.toString(s));
  }
}
```

5. เมื่อสั่งโปรแกรมต่อไปนี้ทำงานจะเกิดสิ่งผิดปกติอะไร ที่บรรทัดใด จงหาสาเหตุและวิธีแก้ไข

```
import java.util.Date;
public class Person {
  private String id;
  private String name;
  private Date birthDate;
  public Person(String i, String n) {
    this.id = i;
    this.name = n;
    // ให้วันที่อ็อบเจกต์ถูกสร้างเป็นวันเกิด
    Date birthDate = new Date();
  }
  public String toString() {
    return "[id=" + id + ",name=" + name+
           ",date=" + birthDate.toString() + "]";
  }
  public static void main(String[] args) {
    Person p = new Person("A101", "สมชาย");
    System.out.println(p.toString());
  }
}
```

6. โปรแกรมสรุปหัวข้อข่าวที่เขียนในรหัสที่ 10-13 ยังไม่ค่อยดี หากข้อมูลที่อ่านได้ไม่ได้ถูกจัดรูปแบบตามที่กำหนด เช่น ถ้า <item> อยู่บรรทัดเดียวกับ <title> และบรรทัดหนึ่งมีหลาย <item> (เช่น ข่าวจาก http://www.bangkokpost.com/rss/src/topstories.xml ของบางกอกโพสต์) จะทำให้การทำงานผิดพลาด จงปรับปรุงรหัสที่ 10-13 ให้รองรับลักษณะของข้อมูลในรูปแบบที่วาง <item> ... <title> ... </title> ... </item> ที่อยู่บรรทัดเดียวกันก็ได้ ข้ามหลายบรรทัดก็ได้ (สามารถนำผลจากแบบฝึกหัดในบทที่ 5 มาใช้ได้)

7. จงเขียนคลาส `WorldTime` ให้เป็นลูกของคลาส `Time` (ในบทที่ 9) คลาสนี้มีตัวสร้างรับชื่อเมือง เพื่อตั้งเวลาตามเวลาปัจจุบันที่เมืองนั้น เช่น `new WorldTime("Tokyo")` และ `new WorldTime("Bangkok")` จะได้เวลาของเมืองโตเกียวและกรุงเทพ ตามลำดับ เวลาของเมืองอ่านได้จาก http://www.koalanet.com.au/world-time-zones.rss ข้อมูลที่อ่านได้มีรูปแบบของข้อมูลเดียวกับแบบฝึกหัดข้อที่แล้ว บรรจุเวลาของเมืองสำคัญต่าง ๆ ทั่วโลกเกือบ 400 เมือง ภาระที่ต้องทำคือหาหัวข้อที่มีชื่อเมืองที่ต้องการ เช่น เวลาที่กรุงเทพฯ จะมีหัวข้อเป็น Bangkok - 21:56 (9:56 pm) ที่โตเกียวมีหัวข้อเป็น Tokyo - 23:56 (11:56 pm) เป็นต้น แล้วนำหัวข้อที่อ่านได้มาสกัดเอาเฉพาะชั่วโมงและนาทีมาตั้งให้กับอ็อบเจกต์ของ `WorldTime` นอกจากนี้ให้สร้างสิ่งผิดปกติใหม่ชื่อว่า `TimeNotFoundException` ที่จะถูกโยนเมื่อเกิดปัญหาระหว่างการขออ่านเวลาปัจจุบันของเมืองที่ได้รับ

8. แทนที่จะเขียนรหัสที่ 10-13 ให้มีเมท็อด `main` ที่สั่งทำงานแล้วแสดงสรุปหัวข้อ เราควรเปลี่ยนให้เป็นเมท็อด `getTitles` ที่มีหัวเป็น `ArrayList getTitles(String page)`

ซึ่งรับเพจที่ต้องการให้ไปอ่านแล้วดึงเฉพาะหัวข้อต่าง ๆ เก็บใส่อ็อบเจกต์ของ ArrayList ที่ถูกคืนเป็นผลลัพธ์ จะได้บริการที่คล่องตัวและนำไปใช้ประโยชน์ได้หลากหลายกว่า จงเขียนเมท็อด getTitles นี้ พร้อมกับออกแบบให้โยนสิ่งผิดปกติที่เหมาะสมกับเหตุการณ์ต่าง ๆ ที่มีโอกาสเกิดขึ้น ให้ผู้เรียกเมท็อดได้รับทราบและจัดการ (อ่านรายละเอียดของ ArrayList ในแบบฝึกหัดข้อที่ 3)

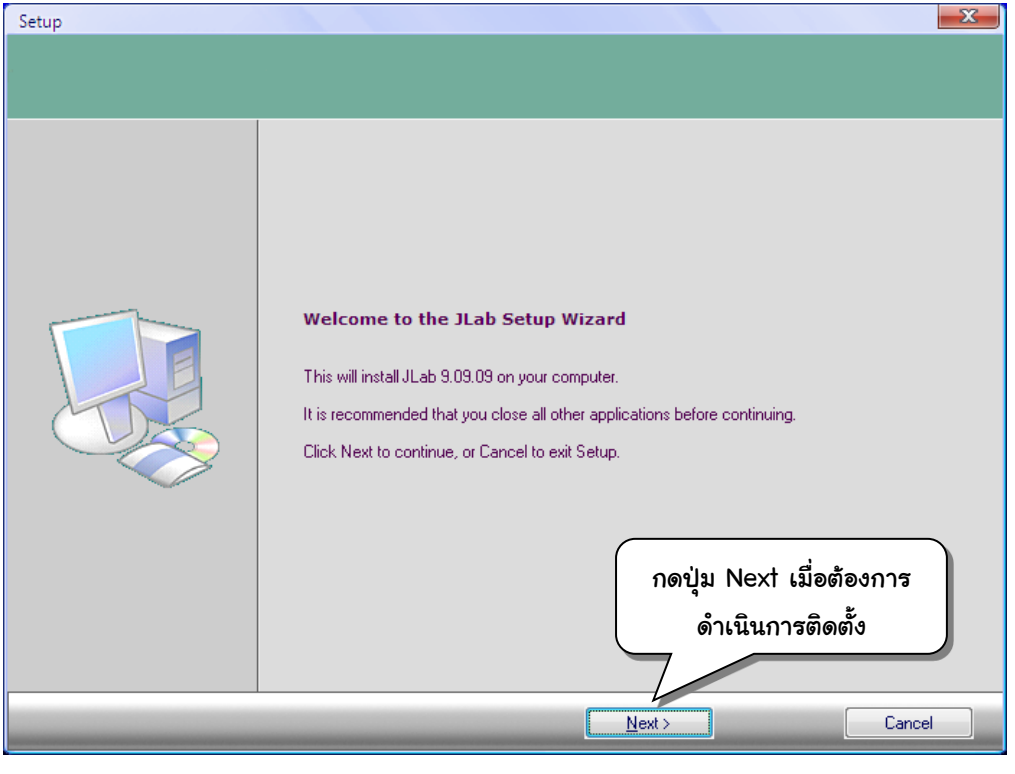
9. จงทำตัวเป็นตัวแปลโปรแกรม เพื่อหาตำแหน่งในคลาสข้างล่างนี้ที่ผิดกฎ และเสนอวิธีแก้ไข

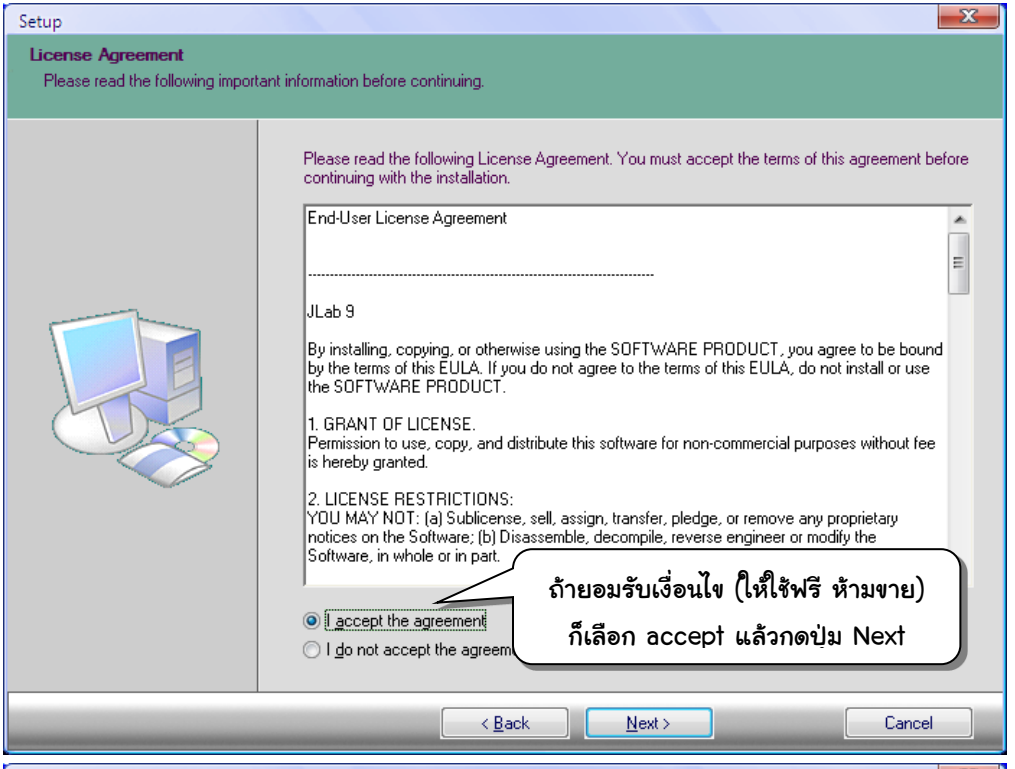
```
public class A {
  public void a() {
    throw new IOException();
  }
  public void b() throws IOException {
  }
  public void c() throws IllegalArgumentException {
  }
}
```

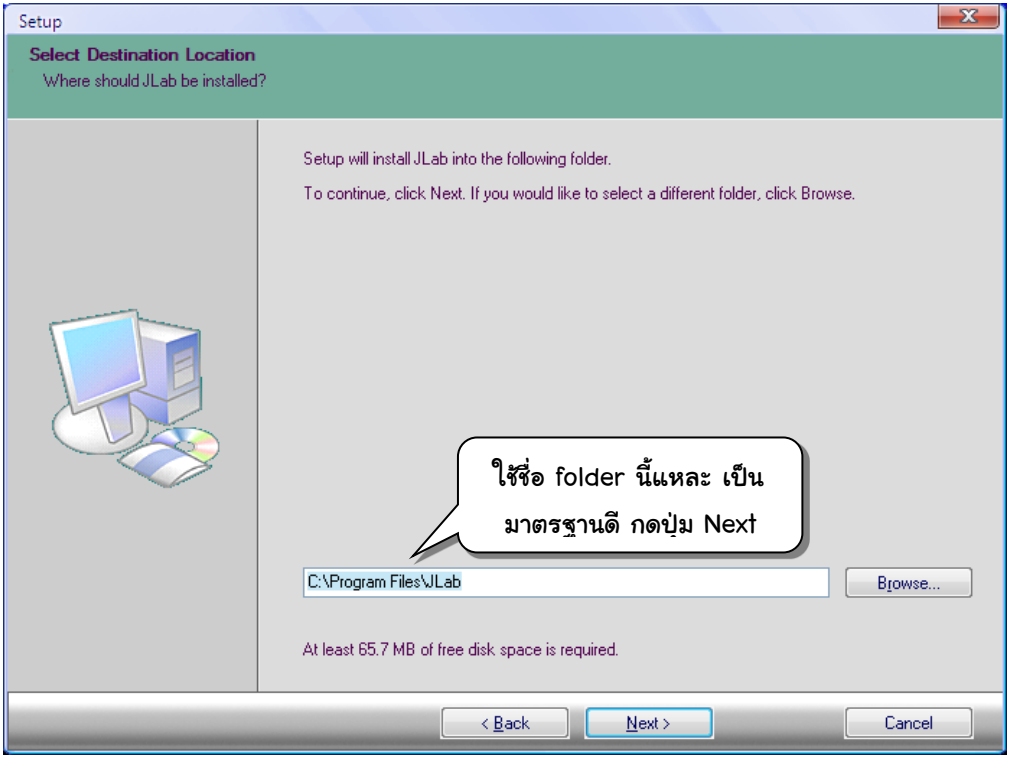
```
public class B extends A {
  public void b()  {
    throw new IOException();
  }
  public void c() throws Exception {
    throw new IllegalArgumentException();
  }
  public void d()  {
    throw new IllegalArgumentException();
  }
}
```

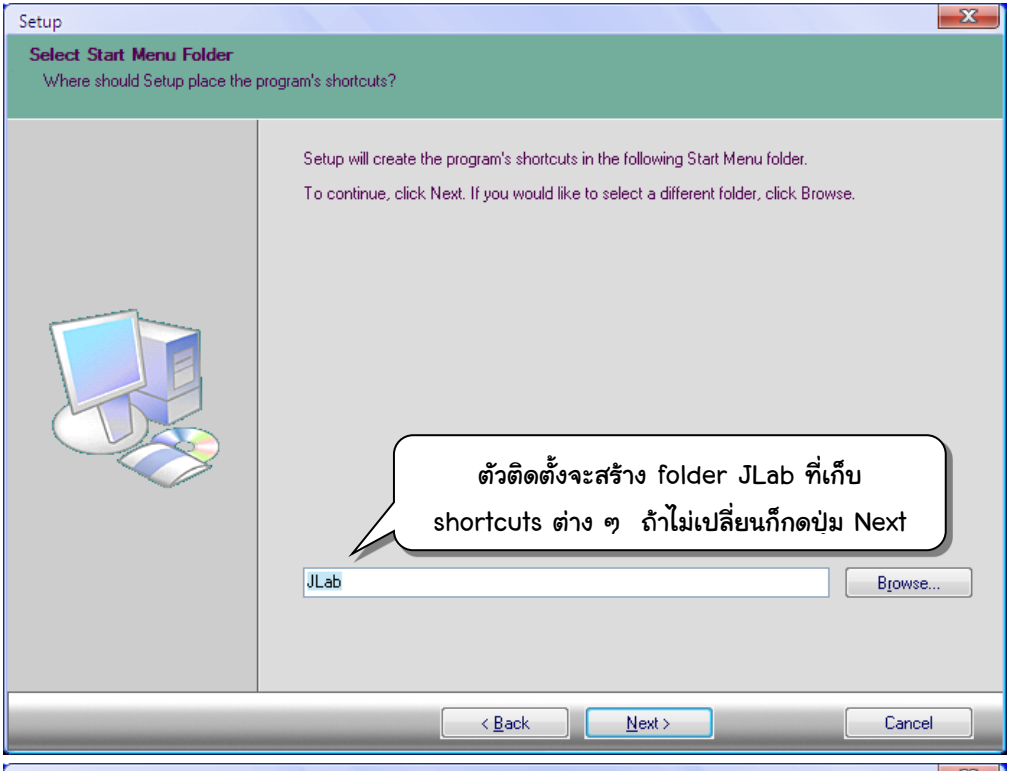
การติดตั้ง JLab

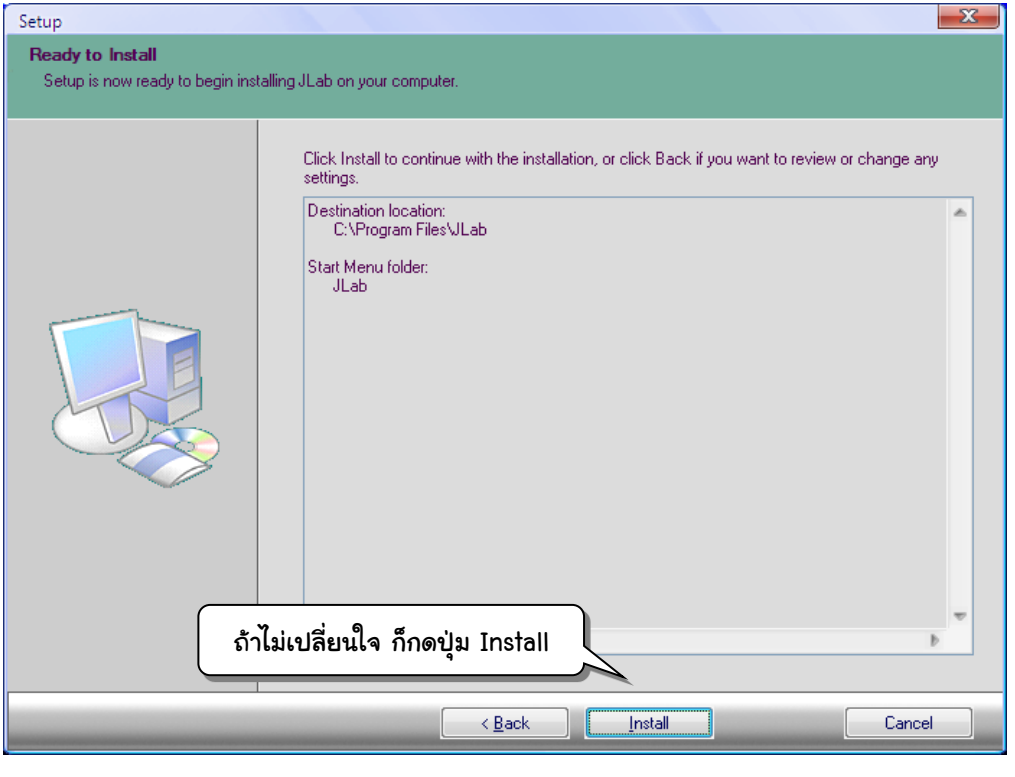
JLab คือซอฟต์แวร์ช่วยพัฒนาโปรแกรมที่เราใช้เป็นเครื่องมือสำหรับการเขียนโปรแกรมในหนังสือเล่มนี้ ผู้อ่านสามารถติดตั้ง JLab โดยใส่แผ่นซีดีรอมหลังปกหนังสือนี้เข้าเครื่องคอมพิวเตอร์ แล้วสั่งให้แฟ้ม setupJLab90909-jdk.exe ทำงาน รอสักครู่ระบบจะเริ่มติดตั้งเป็นขั้นตอนดังต่อไปนี้

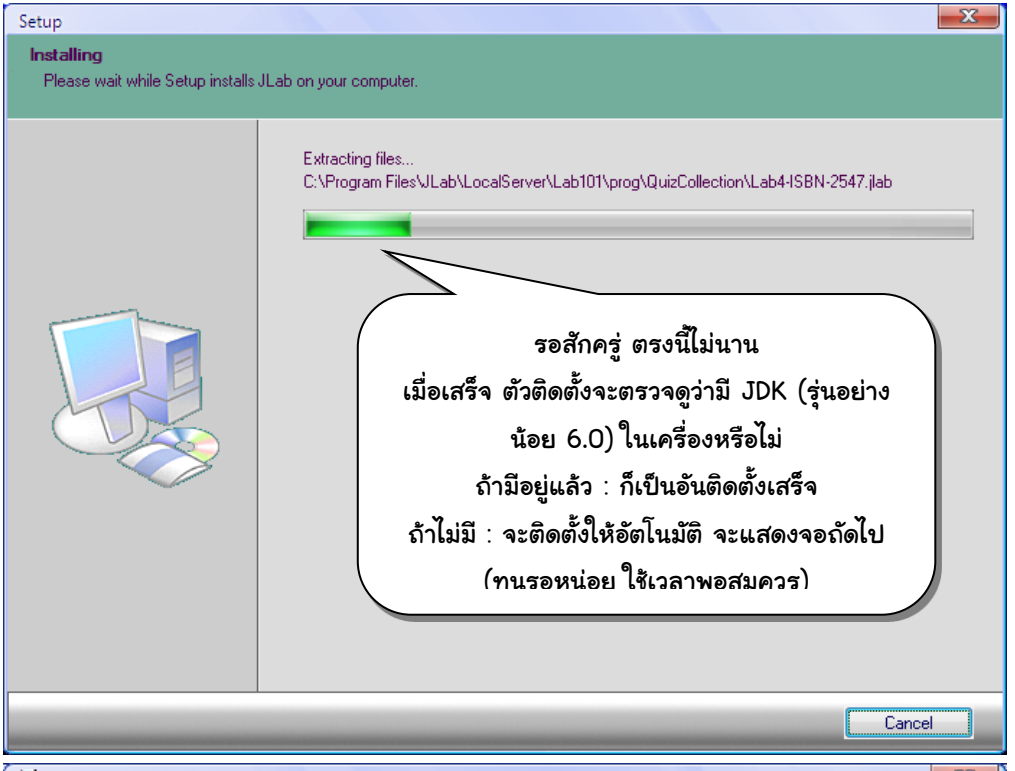


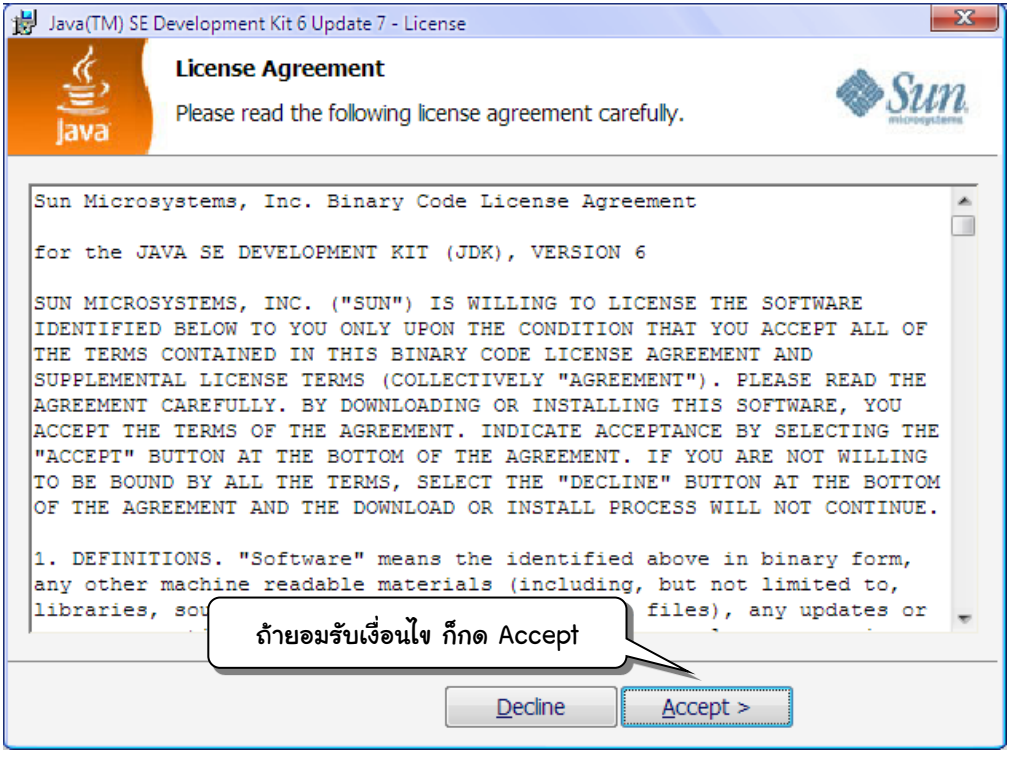

Setup
License Agreement
Please read the following important information before continuing.
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
End-User License Agreement
JLab 9
By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE PRODUCT.
1. GRANT OF LICENSE.
Permission to use, copy, and distribute this software for non-commercial purposes without fee is hereby granted.
2. LICENSE RESTRICTIONS:
YOU MAY NOT: (a) Sublicense, sell, assign, transfer, pledge, or remove any proprietary notices on the Software; (b) Disassemble, decompile, reverse engineer or modify the Software, in whole or in part.
I accept the agreement
I do not accept the agreem
ถ้ายอมรับเงื่อนไข (ให้ใช้ฟรี ห้ามขาย)
ก็เลือก accept แล้วกดปุ่ม Next
< Back
Next >
Cancel

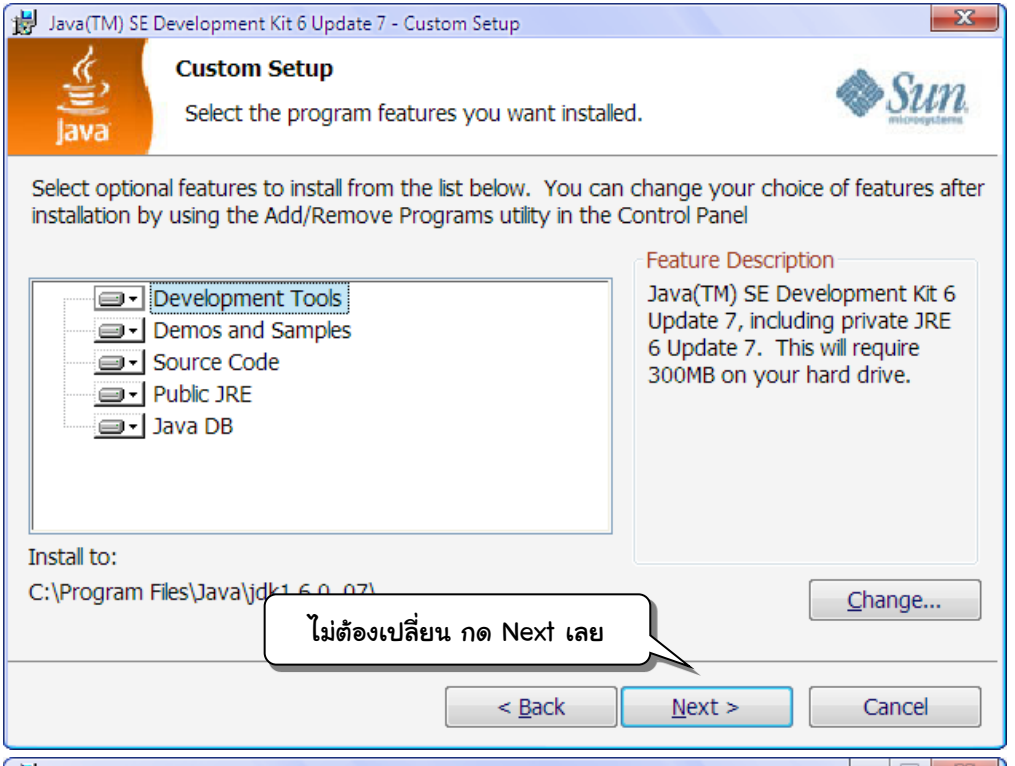

Setup
Select Destination Location
Where should JLab be installed?
Setup will install JLab into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
ใช้ชื่อ folder นี้แหละ เป็น
มาตรฐานดี กดปุ่ม Next
C:\Program Files\JLab
Browse...
At least 65.7 MB of free disk space is required.
< Back
Next >
Cancel

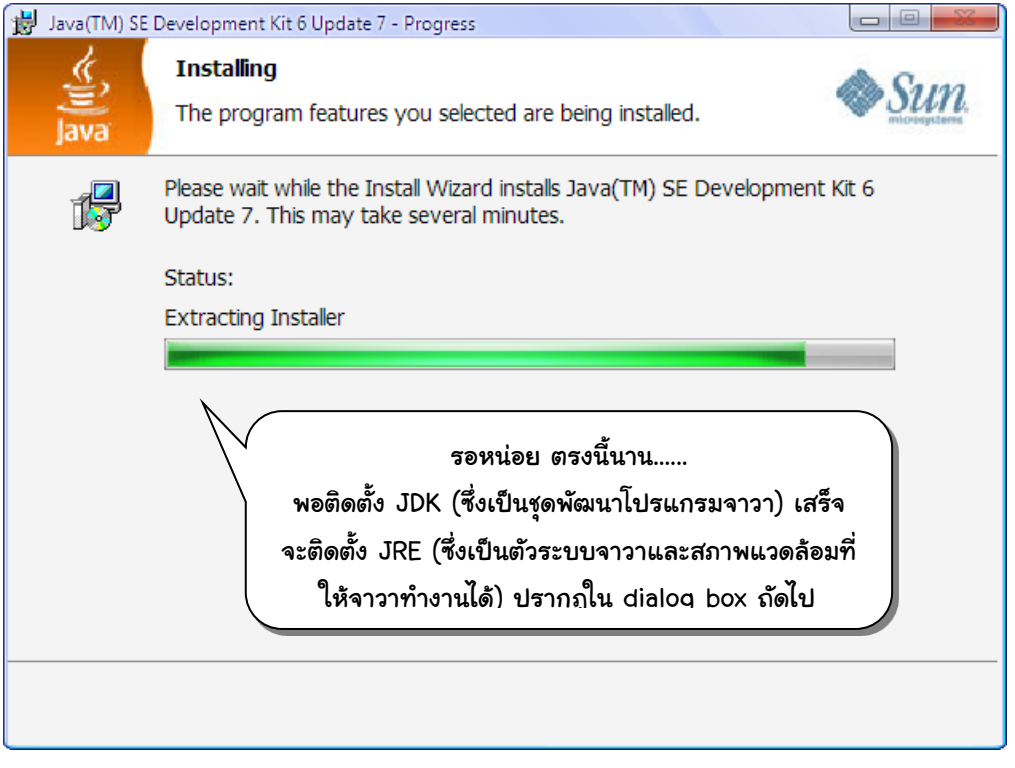

Setup
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
ตัวติดตั้งจะสร้าง folder JLab ที่เก็บ shortcuts ต่าง ๆ ถ้าไม่เปลี่ยนก็กดปุ่ม Next
JLab
Browse...
< Back
Next >
Cancel

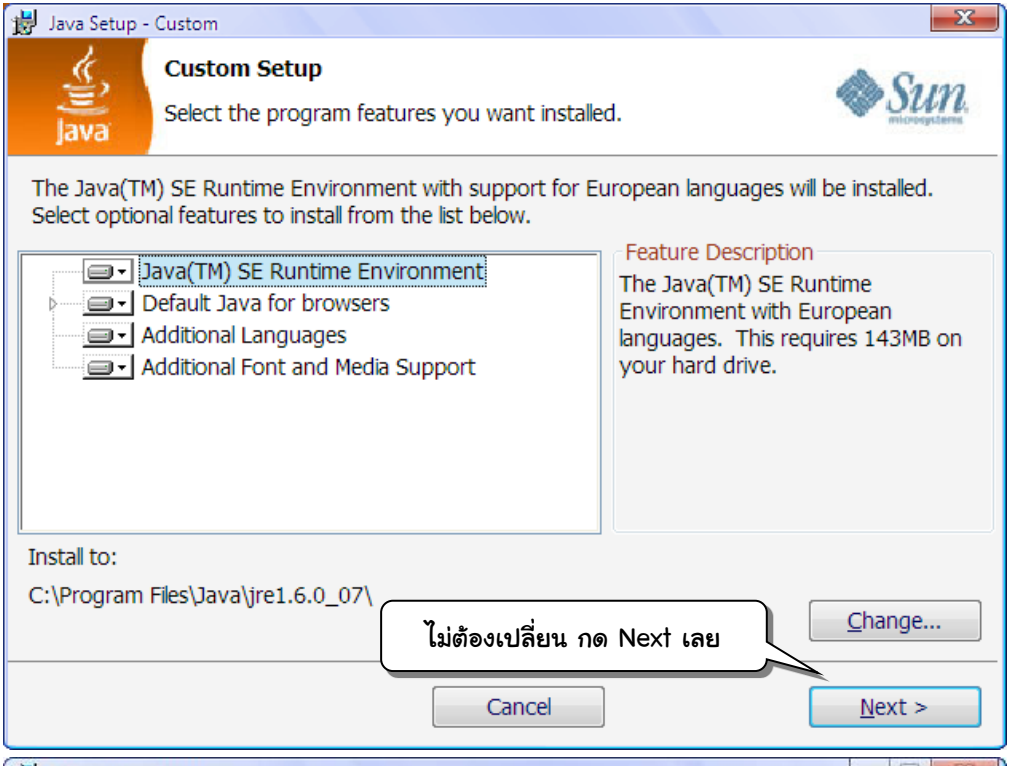

Setup
Ready to Install
Setup is now ready to begin installing JLab on your computer.
Click Install to continue with the installation, or click Back if you want to review or change any settings.
Destination location:
C:\Program Files\JLab
Start Menu folder:
JLab
ถ้าไม่เปลี่ยนใจ ก็กดปุ่ม Install
< Back
Install
Cancel

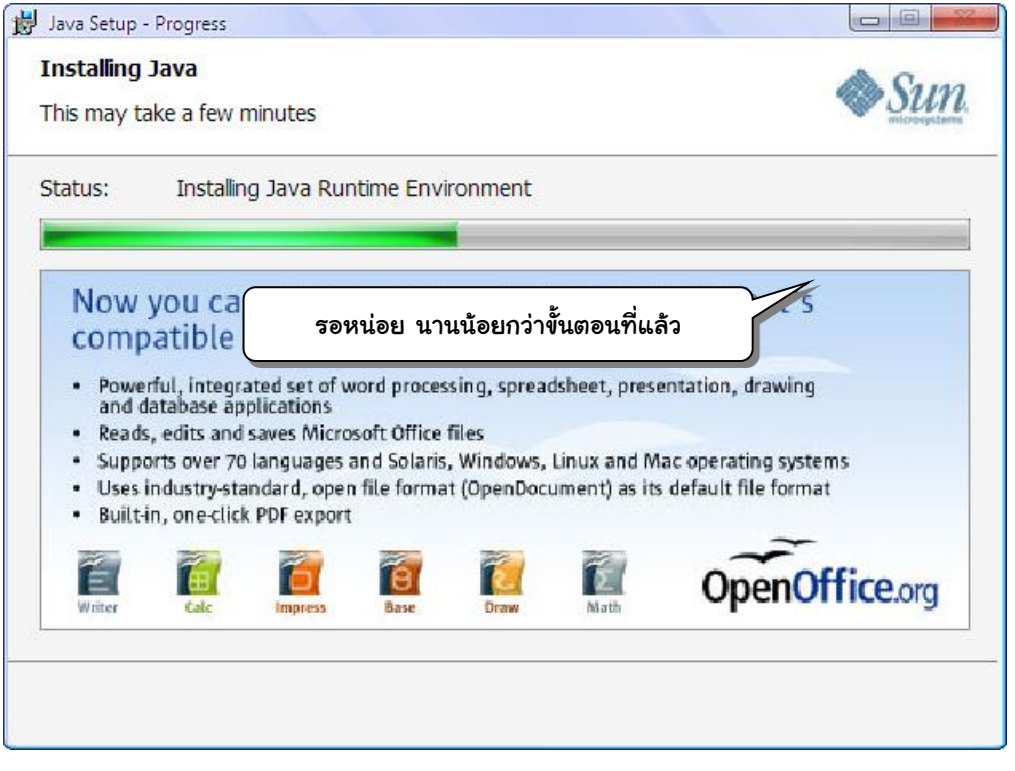

Setup
Installing
Please wait while Setup installs JLab on your computer.
Extracting files...
C:\Program Files\JLab\LocalServer\Lab101\prog\QuizCollection\Lab4-ISBN-2547.jlab
รอสักครู่ ตรงนี้ไม่นาน
เมื่อเสร็จ ตัวติดตั้งจะตรวจดูว่ามี JDK (รุ่นอย่างน้อย 6.0) ในเครื่องหรือไม่
ถ้ามีอยู่แล้ว : ก็เป็นอันติดตั้งเสร็จ
ถ้าไม่มี : จะติดตั้งให้อัตโนมัติ จะแสดงจอถัดไป
(ทนรอหน่อย ใช้เวลาพอสมควร)
Cancel


Java(TM) SE Development Kit 6 Update 7 - License
License Agreement
Please read the following license agreement carefully.
Sun Microsystems, Inc. Binary Code License Agreement
for the JAVA SE DEVELOPMENT KIT (JDK), VERSION 6
SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, YOU ACCEPT THE TERMS OF THE AGREEMENT. INDICATE ACCEPTANCE BY SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ALL THE TERMS, SELECT THE "DECLINE" BUTTON AT THE BOTTOM OF THE AGREEMENT AND THE DOWNLOAD OR INSTALL PROCESS WILL NOT CONTINUE.
1. DEFINITIONS. "Software" means the identified above in binary form, any other machine readable materials (including, but not limited to, libraries, so files), any updates or
ถ้ายอมรับเงื่อนไข ก็กด Accept
Decline
Accept >

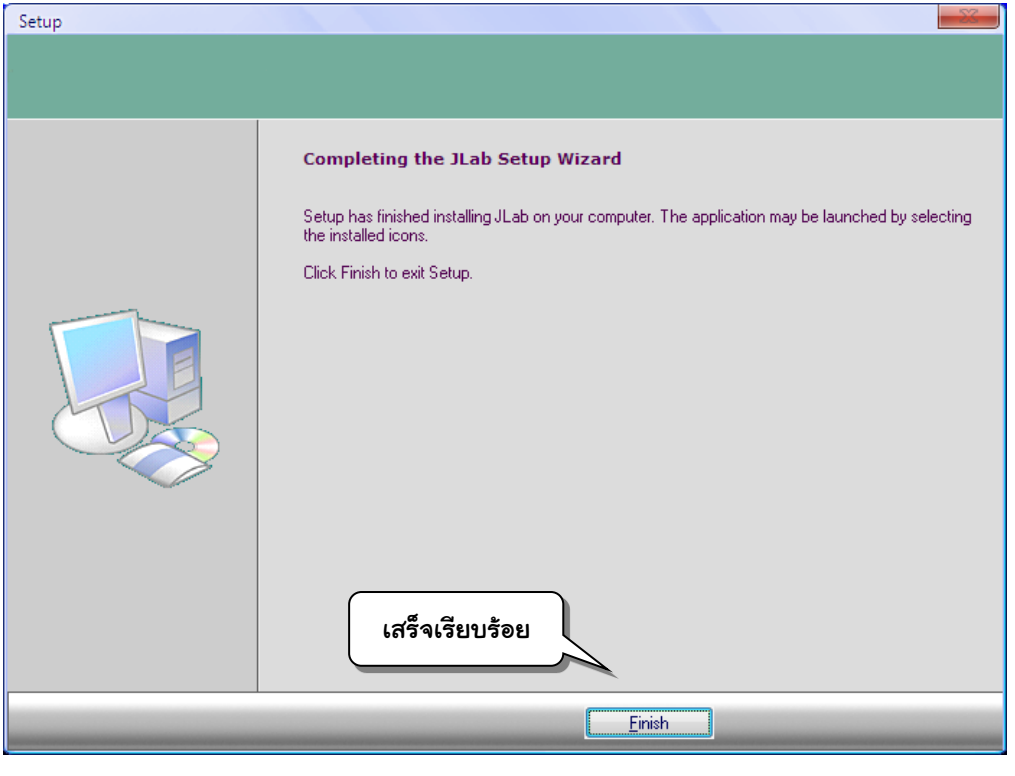

Java(TM) SE Development Kit 6 Update 7 - Custom Setup
Custom Setup
Select the program features you want installed.
Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel
Development Tools
Demos and Samples
Source Code
Public JRE
Java DB
Feature Description
Java(TM) SE Development Kit 6 Update 7, including private JRE 6 Update 7. This will require 300MB on your hard drive.
Install to:
C:\Program Files\Java\jdk1.6.0_07\
Change...
ไม่ต้องเปลี่ยน กด Next เลย
< Back
Next >
Cancel


Java(TM) SE Development Kit 6 Update 7 - Progress
Installing
The program features you selected are being installed.
Please wait while the Install Wizard installs Java(TM) SE Development Kit 6 Update 7. This may take several minutes.
Status:
Extracting Installer
รอหน่อย ตรงนี้นาน.....
พอติดตั้ง JDK (ซึ่งเป็นชุดพัฒนาโปรแกรมจาวา) เสร็จ
จะติดตั้ง JRE (ซึ่งเป็นตัวระบบจาวาและสภาพแวดล้อมที่
ให้จาวาทำงานได้) ปรากฏใน dialog box ถัดไป


Java Setup - Custom
Custom Setup
Select the program features you want installed.
The Java(TM) SE Runtime Environment with support for European languages will be installed.
Select optional features to install from the list below.
Java(TM) SE Runtime Environment
Default Java for browsers
Additional Languages
Additional Font and Media Support
Feature Description
The Java(TM) SE Runtime Environment with European languages. This requires 143MB on your hard drive.
Install to:
C:\Program Files\Java\jre1.6.0_07\
ไม่ต้องเปลี่ยน กด Next เลย
Change...
Cancel
Next >

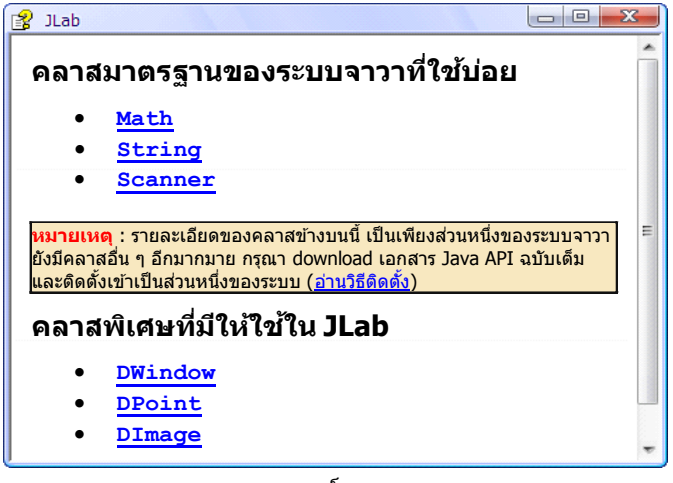

Java Setup - Progress
Installing Java
This may take a few minutes
Status: Installing Java Runtime Environment
รอหน่อย นานน้อยกว่าขั้นตอนที่แล้ว
Powerful, integrated set of word processing, spreadsheet, presentation, drawing and database applications
Reads, edits and saves Microsoft Office files
Supports over 70 languages and Solaris, Windows, Linux and Mac operating systems
Uses industry-standard, open file format (OpenDocument) as its default file format
Built-in, one-click PDF export
Writer
Calc
Impress
Base
Draw
Math
OpenOffice.org


Java(TM) SE Development Kit 6 Update 7 - Complete
Java(TM) SE Development Kit 6 Update 7 Successfully Installed
Product Registration is FREE and includes many benefits:
* Notification of new versions, patches, and updates
* Special offers on Sun products, services and training
* Access to early releases and documentation
When you click Finish, product and system data will be collected and the JDK product registration form will be presented. If you do not register, none of this information will be saved.
For more information on what data Registration collects and how it is managed and used, see the ProductRegi...
Product Registration Information
Java
Sun
ติดตั้ง JDK, JRE เสร็จ
Finish


Setup
Completing the JLab Setup Wizard
Setup has finished installing JLab on your computer. The application may be launched by selecting the installed icons.
Click Finish to exit Setup.
เสร็จเรียบร้อย
Finish

การติดตั้ง Java API Help File

หลังจากติดตั้ง JLab แล้ว หากกดปุ่ม F1 เพื่อขอดูเอกสารการใช้งานคลาสมาตรฐานในระบบวาจา (Java API help file) จะพบวินโดว์ดังรูปข้างล่างนี้ ซึ่งแสดงคลาสมาตรฐานเพียงสามคลาสที่เราใช้บ่อยในหนังสือเล่มนี้ และคลาสพิเศษของ JLab อีกสามคลาส



เนื่องจากบริษัทซันฯ ผู้เป็นเจ้าของลิขสิทธิ์ตัวเอกสารของคลังคลาสมาตรฐานในระบบจาวาฉบับเต็ม สงวนสิทธิ์ไม่อนุญาตให้บันทึกและเผยแพร่แฟ้มเอกสารในซีดีรอม แต่อนุญาตให้ดาวน์โหลดมาใช้ส่วนตัวจากเว็บไซต์ http://www.allimant.org/javadoc ผู้เขียนจึงแนะนำอย่างยิ่งว่าให้ผู้อ่านดาวน์โหลดเอกสารนี้มาติดตั้งในเครื่องตนเอง เมื่อไปที่เว็บไซต์นี้ ให้คลิกที่บรรทัด J2SE 6 Documentation (ในปัจจุบันจาวารุ่นล่าสุดคือรุ่นที่ 6) จะได้แฟ้มชื่อ j2se6.zip เมื่อ unzip จะได้แฟ้มชื่อ j2se6.chm ซึ่งต้องทำสำเนาไปเก็บไว้ในโฟลเดอร์ c:\Program Files\JLab (ซึ่งคือโฟลเดอร์ที่ติดตั้ง JLab) เพียงเท่านี้ ก็สามารถกดปุ่ม F1 เพื่อเรียกใช้เอกสารฉบับเต็มได้ทันที

ภาคผนวก ข DWindow

ภาคผนวกนี้สรุปการใช้งาน รายละเอียดของหัวเมท็อดและตัวสร้างของคลาส `DWindow` พร้อมกับคลาสเล็ก ๆ ที่ใช้ประกอบอีกสองคลาสคือ `DImage` และ `DPoint` ที่เราได้อ้างอิงถึงในตัวอย่างต่าง ๆ ที่นำเสนอในหนังสือเล่มนี้ สามคลาสนี้เป็นคลาสพิเศษในแพ็กเกจ `jlab.graphics` ที่ผู้เขียนพัฒนาขึ้นเพื่ออำนวยความสะดวกในการเขียนโปรแกรมที่แสดงผลด้วยรูปภาพและภาพกราฟิกส์บนวินโดว์ เพื่อให้ตัวอย่างแลดูน่าสนใจกว่าการเขียนโปรแกรมที่แสดงผลแต่ข้อความเพียงอย่างเดียว คลาสทั้งสามนี้จึงไม่ใช่ส่วนหนึ่งของคลาสมาตรฐานในระบบจาวา

DWindow

`DWindow` เป็นคลาสที่อำนวยความสะดวกในการแสดงภาพ หรือรูปทรงเรขาคณิตพื้นฐานในวินโดว์ ผู้ใช้เพียงสั่ง `new` อ็อบเจกต์ `DWindow` จะได้วินโดว์ปรากฏบนจอภาพ พร้อมให้ลากเส้นตรง วาดสี่เหลี่ยม วาดวงรี อ่านภาพจากแฟ้มภาพมาแสดง ขอแผนที่จุดภาพของภาพที่ปรากฏบนวินโดว์ไปประมวลผล เปลี่ยนภาพโดยการตั้งแผนที่จุดภาพใหม่ รวมทั้งให้บริการบันทึกภาพที่ปรากฏบนวินโดว์ไว้ในแฟ้มภาพได้อีกด้วย ผู้ใช้สามารถใช้บริการตั้งสี ผสมสี แยกสี การทำภาพจาง การหยุดการทำงานชั่วขณะ เพื่อแสดงภาพเคลื่อนไหวที่น่าสนใจได้ นอกจากนี้ยังมีบริการการอ่านพิกัดของตัวชี้เมาส์ การให้โปรแกรมหยุดรอจนกว่าผู้ใช้จะคลิกเมาส์ด้วย ประเด็นที่ผู้ใช้ควรรู้เล็กน้อยก่อนใช้งานมีดังนี้

- **พิกัด** : มุมซ้ายบนของบริเวณที่ใช้วาดรูปมีพิกัดเป็น (0,0) และค่าพิกัด x เพิ่มขึ้นเมื่อไปทางขวา ในขณะที่ค่าพิกัด y เพิ่มขึ้นเมื่อลงด้านล่าง

- **สี** : ควรใช้สีที่ตั้งเป็นค่าคงตัวไว้แล้วในคลาส เช่น `DWindow.RED`, `DWindow.BLACK` เป็นต้น หรือใช้เมท็อดประจำคลาส `DWindow.mixRGB(r, g, b)` ในการผสมสีเอง
- **ตัวชี้เมาส์** : พิกัดของตัวชี้เมาส์จะเป็นอ็อบเจกต์ประเภท `DPoint` ซึ่งมีเมท็อด `getX()` และ `getY()` ที่สามารถเรียกใช้เพื่อขอพิกัด *x* และ *y* ตามลำดับของตัวชี้เมาส์บนวินโดว์

ค่าคงตัวประจำคลาส

ค่าคงตัวประจำคลาส `DWindow` ประกอบด้วยสีมาตรฐานมีชื่อกำกับ เพื่อความสะดวกในการใช้งานดังนี้

- `BLACK, BLUE, BROWN, CYAN, GRAY, GREEN, MAGENTA, ORANGE, PINK, PURPLE, RED, VIOLET, WHITE, YELLOW`

ตัวสร้าง

`DWindow()`

หน้าที่	ตัวสร้างวินโดว์ที่มีขนาด 200 × 200 โดยแสดงทันทีหลังการ `new` และปรากฏทับวินโดว์อื่น ๆ ให้เห็นเด่นชัดตลอดเวลา

`DWindow(double width, double height)`

หน้าที่	ตัวสร้างวินโดว์ที่มีขนาด `width` × `height` โดยแสดงทันทีหลังการ `new` และปรากฏทับวินโดว์อื่น ๆ ให้เห็นเด่นชัดตลอดเวลา
พารามิเตอร์	`width` – ความกว้างของบริเวณที่แสดงผลได้ของวินโดว์ (หน่วยเป็นจุดภาพ) `height` – ความสูงของบริเวณที่แสดงผลได้ของวินโดว์ (หน่วยเป็นจุดภาพ)

เมท็อด

`void setForeground(int c)`

หน้าที่	ตั้งสีที่ใช้ในการวาดบนวินโดว์ หลังการสร้างวินโดว์ใหม่ จะให้สีในการวาดเป็นสีดำ
พารามิเตอร์	`c` – ค่าของสี

`void setBackground(int c)`

หน้าที่	ตั้งสีพื้นหลังของวินโดว์ หลังการสร้างวินโดว์ใหม่ จะให้สีขาวเป็นสีพื้น
พารามิเตอร์	`c` – ค่าของสี

`void clearBackground()`

หน้าที่	ล้างภาพที่ปรากฏบนวินโดว์ออกให้หมด แล้วแสดงสีพื้นหลังที่ได้ตั้งไว้

`void setSize(int width,  int height)`

หน้าที่	ตั้งขนาดของวินโดว์
พารามิเตอร์	`width` – ความกว้างของบริเวณที่แสดงผลได้ของวินโดว์ (หน่วยเป็นจุดภาพ) `height` – ความสูงของบริเวณที่แสดงผลได้ของวินโดว์ (หน่วยเป็นจุดภาพ)

void drawString(String s, double size, double x, double y)

หน้าที่ แสดงสตริงบนวินโดว์ สีของตัวอักษร คือ สีที่ตั้งไว้ด้วยเมท็อด `setForeground` ครั้งแรกที่มีการเรียกใช้เมท็อดนี้ อาจต้องใช้เวลาสักครู่ในการโหลดแบบอักษรเข้าสู่หน่วยความจำ

พารามิเตอร์ `s` - สตริงที่ต้องการแสดง
`size` - ขนาดตัวอักษร
`x` - ตำแหน่งขอบซ้ายของสตริงที่จะแสดงบนวินโดว์ (หน่วยเป็นจุดภาพ)
`y` - ตำแหน่งขอบบนของสตริงที่จะแสดงบนวินโดว์ (หน่วยเป็นจุดภาพ)

void drawLine(double x1, double y1, double x2, double y2)

หน้าที่ ลากเส้นตรงบนวินโดว์ จากพิกัด (`x1,y1`) ถึง (`x2,y2`) โดยใช้สีตามที่ตั้งไว้ด้วยเมท็อด `setForeground` เป็นสีเส้น

พารามิเตอร์ `x1` - พิกัด `x` ของจุดปลายจุดที่ 1
`y1` - พิกัด `y` ของจุดปลายจุดที่ 1
`x2` - พิกัด `x` ของจุดปลายจุดที่ 2
`y2` - พิกัด `y` ของจุดปลายจุดที่ 2

void drawRect(double x, double y, double w, double h)

หน้าที่ วาดเส้นขอบของสี่เหลี่ยมผืนผ้า ขนาด `w × h` บนวินโดว์ มีมุมซ้ายบนอยู่ที่ (`x,y`) และใช้สีตามที่ตั้งไว้ด้วยเมท็อด `setForeground` เป็นสีของเส้นขอบ

พารามิเตอร์ `x` - พิกัด `x` ของมุมซ้ายบนของสี่เหลี่ยม
`y` - พิกัด `y` ของมุมซ้ายบนของสี่เหลี่ยม
`w` - ความกว้างของสี่เหลี่ยม
`h` - ความสูงของสี่เหลี่ยม

void fillRect(double x, double y, double w, double h)

หน้าที่ วาดสี่เหลี่ยมผืนผ้าทึบ ขนาด `w × h` บนวินโดว์ โดยมีมุมซ้ายบนอยู่ที่ (`x,y`) บริเวณภายในสี่เหลี่ยมที่วาดมีสีตามที่ได้ `setForeground` ไว้

พารามิเตอร์ `x` - พิกัด `x` ของมุมซ้ายบนของสี่เหลี่ยม
`y` - พิกัด `y` ของมุมซ้ายบนของสี่เหลี่ยม
`w` - ความกว้างของสี่เหลี่ยม
`h` - ความสูงของสี่เหลี่ยม

void drawEllipse(double xc, double yc, double w, double h)

หน้าที่ วาดเส้นขอบของวงรี ที่มีขนาด `w × h` บนวินโดว์ มีจุดศูนย์กลางอยู่ที่ (`xc,yc`) และใช้สีตามที่ตั้งไว้ด้วยเมท็อด `setForeground` เป็นสีของเส้นขอบ

พารามิเตอร์ `xc` - พิกัด `x` ของจุดศูนย์กลางของวงรี
`yc` - พิกัด `y` ของจุดศูนย์กลางของวงรี
`w` - ความกว้างของวงรี
`h` - ความสูงของวงรี

void fillEllipse(double xc, double yc, double w, double h)

หน้าที่ วาดวงรีทึบ ที่มีขนาด `w × h` บนวินโดว์ มีจุดศูนย์กลางอยู่ที่ (`xc,yc`) บริเวณภายในวงรีมีสีตามที่ได้ `setForeground` ไว้

พารามิเตอร์ `xc` - พิกัด `x` ของจุดศูนย์กลางของวงรี
`yc` - พิกัด `y` ของจุดศูนย์กลางของวงรี
`w` - ความกว้างของวงรี, `h` - ความสูงของวงรี

void draw(DImage img, double xc, double yc)

หน้าที่ วาดรูปภาพ จากอ็อบเจกต์ภาพ `img` มีจุดศูนย์กลางของภาพอยู่ที่ `(xc,yc)`

พารามิเตอร์
- `img` - อ็อบเจกต์ภาพ
- `xc` - พิกัด `x` ของจุดศูนย์กลางของวงรี
- `yc` - พิกัด `y` ของจุดศูนย์กลางของวงรี

void drawString(int c, String s, double size, double x, double y)

หน้าที่ แสดงสตริงบนวินโดว์ แบบระบุสี ต้องกำหนดสี ขนาดของตัวอักษร และตำแหน่งมุมซ้ายบนวินโดว์ ครั้งแรกที่มีการเรียกใช้เมท็อดนี้ อาจต้องใช้เวลาสักครู่ในการโหลดแบบอักษรเข้าสู่หน่วยความจำ

พารามิเตอร์
- `c` - สีของสตริง
- `s` - สตริงที่ต้องการแสดง
- `size` - ขนาดตัวอักษร
- `x` - ตำแหน่งขอบซ้ายของสตริงที่จะแสดงบนวินโดว์ (หน่วยเป็นจุดภาพ)
- `y` - ตำแหน่งขอบบนของสตริงที่จะแสดงบนวินโดว์ (หน่วยเป็นจุดภาพ)

void drawLine(int c, double x1, double y1, double x2, double y2)

หน้าที่ ลากเส้นตรงบนวินโดว์ แบบระบุสี จากพิกัด `(x1,y1)` ถึง `(x2,y2)`

พารามิเตอร์
- `c` - สีของเส้น
- `x1` - พิกัด `x` ของจุดปลายจุดที่ `1`
- `y1` - พิกัด `y` ของจุดปลายจุดที่ `1`
- `x2` - พิกัด `x` ของจุดปลายจุดที่ `2`
- `y2` - พิกัด `y` ของจุดปลายจุดที่ `2`

void drawRect(int c, double x, double y, double w, double h)

หน้าที่ วาดเส้นขอบของสี่เหลี่ยมผืนผ้า แบบระบุสี ขนาด `w × h` บนวินโดว์ โดยมีมุมซ้ายบนอยู่ที่ `(x,y)`

พารามิเตอร์
- `c` - สีของเส้นขอบ
- `x` - พิกัด `x` ของมุมซ้ายบนของสี่เหลี่ยม
- `y` - พิกัด `y` ของมุมซ้ายบนของสี่เหลี่ยม
- `w` - ความกว้างของสี่เหลี่ยม
- `h` - ความสูงของสี่เหลี่ยม

void fillRect(int c, double x, double y, double w, double h)

หน้าที่ วาดสี่เหลี่ยมผืนผ้าทึบ แบบระบุสี ขนาด `w × h` บนวินโดว์ โดยมีมุมซ้ายบนอยู่ที่ `(x,y)`

พารามิเตอร์
- `c` - สีภายในสี่เหลี่ยม
- `x` - พิกัด `x` ของมุมซ้ายบนของสี่เหลี่ยม
- `y` - พิกัด `y` ของมุมซ้ายบนของสี่เหลี่ยม
- `w` - ความกว้างของสี่เหลี่ยม
- `h` - ความสูงของสี่เหลี่ยม

void drawEllipse(int c, double xc, double yc, double w, double h)

หน้าที่ วาดเส้นขอบของวงรี แบบระบุสี ที่มีขนาด `w × h` บนวินโดว์ โดยมีจุดศูนย์กลางอยู่ที่ `(xc,yc)`

พารามิเตอร์
- `c` - สีของเส้นขอบ
- `xc` - พิกัด `x` ของจุดศูนย์กลางของวงรี
- `yc` - พิกัด `y` ของจุดศูนย์กลางของวงรี
- `w` - ความกว้างของวงรี
- `h` - ความสูงของวงรี

void fillEllipse(int c, double xc, double yc, double w, double h)

หน้าที่ วาดวงรีทึบ แบบระบุสี ที่มีขนาด `w × h` บนวินโดว์ โดยมีจุดศูนย์กลางอยู่ที่ `(xc,yc)`

พารามิเตอร์
- `c` - สีภายในวงรี
- `xc` - พิกัด `x` ของจุดศูนย์กลางของวงรี
- `yc` - พิกัด `y` ของจุดศูนย์กลางของวงรี
- `w` - ความกว้างของวงรี
- `h` - ความสูงของวงรี

int getWidth()

หน้าที่ คืนความกว้างของวินโดว์

ผลที่คืน ความกว้างของวินโดว์

int getHeight()

หน้าที่ คืนความสูงของวินโดว์

ผลที่คืน ความสูงของวินโดว์

void setRepaintDuringSleep(boolean enable)

หน้าที่ ตั้งให้วินโดว์แสดงภาพขณะ `sleep` หรือไม่ ในกรณีที่มีการวาดภาพเป็นจำนวนมาก กระทำบ่อย ๆ โดยมีการสั่งให้ `sleep` คั่นกลาง หากเราตั้งให้ไม่แสดงภาพใหม่ทุกครั้งที่วาด แต่รวบยอดไปแสดงใหม่ตอน `sleep` จะทำให้การเปลี่ยนแปลงภาพที่ราบเรียบไม่กระพริบ ดูสบายตา

พารามิเตอร์ `enable` - ค่า `true` คือให้วินโดว์แสดงภาพใหม่เฉพาะเมื่อมีการ `sleep`, ค่า `false` คือให้แสดงภาพใหม่ทุกครั้งที่เปลี่ยนแปลง

void sleep(long millis)

หน้าที่ สั่งให้หยุดการทำงานชั่วขณะ เป็นเวลาตามที่ระบุไว้ใน `millis` มีหน่วยเป็นมิลลิวินาที

พารามิเตอร์ `millis` - ช่วงเวลาที่ต้องการหยุดทำงานชั่วขณะ (หน่วยเป็นมิลลิวินาที)

DPoint getMouse()

หน้าที่ คืนพิกัดของตัวชี้เมาส์บนวินโดว์

ผลที่คืน พิกัดของตัวชี้เมาส์ (อ่านรายละเอียดเมท็อดของ `DPoint`)

DPoint waitForMouseClicked()

หน้าที่ รอจนกว่าจะมีการคลิกเมาส์

ผลที่คืน พิกัดของตัวชี้เมาส์เมื่อมีการคลิกปุ่มเมาส์ (อ่านรายละเอียดเมท็อดของ `DPoint`)

void onMouseMoved(DPoint p)

หน้าที่ ระบบเรียกเมท็อดนี้ให้อัตโนมัติเมื่อมีการเลื่อนเมาส์บนวินโดว์นี้ เมื่อใดที่ต้องการดักเหตุการณ์การเลื่อนเมาส์บนวินโดว์ ให้เขียนเมท็อดนี้ที่คลาสลูกของ `DWindow`

พารามิเตอร์ `p` - พิกัดของตัวชีเมาส์บนวินโดว์ขณะเกิดการเลื่อนเมาส์

void onMouseDragged(DPoint p)

หน้าที่ ระบบเรียกเมท็อดนี้ให้อัตโนมัติเมื่อมีการลากเมาส์บนวินโดว์นี้ การลากเมาส์คือการเลื่อนเมาส์พร้อมกับกดปุ่มเมาส์ เมื่อใดที่ต้องการดักเหตุการณ์การลากเมาส์บนวินโดว์ ให้เขียนเมท็อดนี้ที่คลาสลูกของ `DWindow`

พารามิเตอร์ `p` - พิกัดของตัวชีเมาส์บนวินโดว์ขณะเกิดการลากเมาส์

void fade(double level)

หน้าที่ ทำทุกจุดภาพบนวินโดว์ในจางลง ตามระดับความจางที่มีค่าตั้งแต่ 0 ถึง 1, 0 หมายถึงไม่จางลงแต่อย่างใด, 1 หมายถึงจางจนหมด (เสมือนการล้างวินโดว์)

พารามิเตอร์ level – ระดับความจาง

int[][] getPixmap()

หน้าที่ คืนแผนที่จุดภาพของวินโดว์ เป็นอาเรย์สองมิติมีจำนวนแถวและจำนวนสดมภ์เท่ากับความกว้างและความสูงของวินโดว์ เช่น วินโดว์มีขนาดกว้าง×สูงเป็น 200×100 จะได้อาเรย์ขนาด 200 แถว 100 สดมภ์ การใช้แต่ละช่องของอาเรย์ที่ใช้ดัชนีของพิกัด x ก่อนแล้วจึงตามด้วยดัชนีของพิกัด y ในรูปแบบ p[x][y] p[x][y] คือ จำนวนเต็มที่แทนค่าสีของจุดภาพที่พิกัด (x,y) บนวินโดว์

ผลที่คืน อาเรย์สองมิติที่เก็บแผนที่จุดภาพของวินโดว์

void setPixmap(int[][] p)

หน้าที่ แสดงภาพใหม่บนวินโดว์ตามแผนที่จุดภาพ p ที่ได้รับ พร้อมกับปรับขนาดของวินโดว์ให้พอดีกับขนาดของแผนที่จุดภาพที่ได้รับ แผนที่จุดภาพเป็นอาเรย์สองมิติมีจำนวนแถวและจำนวนสดมภ์เท่ากับความกว้างและความสูงของวินโดว์ เช่น วินโดว์มีขนาดกว้าง×สูงเป็น 200×100 จะได้อาเรย์ขนาด 200 แถว 100 สดมภ์ โดยที่ p[x][y] คือ จำนวนเต็มที่แทนค่าสีของจุดภาพที่พิกัด (x,y) บนวินโดว์

พารามิเตอร์ p – อาเรย์สองมิติที่เก็บแผนที่จุดภาพที่ต้องการให้แสดง

static int getR(int c)

หน้าที่ คืนค่าของแม่สีแดงของสี c มีค่าระหว่าง 0 ถึง 255

พารามิเตอร์ c – ค่าสี

ผลที่คืน ค่าของแม่สีแดง

static int getG(int c)

หน้าที่ คืนค่าของแม่สีเขียวของสี c มีค่าระหว่าง 0 ถึง 255

พารามิเตอร์ c – ค่าสี

ผลที่คืน ค่าของแม่สีเขียว

static int getB(int c)

หน้าที่ คืนค่าของแม่สีน้ำเงินของสี c มีค่าระหว่าง 0 ถึง 255

พารามิเตอร์ c – ค่าสี

ผลที่คืน ค่าของแม่สีน้ำเงิน

static int mixRGB(double r, double g, double b)

หน้าที่ คืนค่าสีที่ได้จากการผสมแม่สีแดง เขียว น้ำเงิน

พารามิเตอร์ r – ค่าของแม่สีแดง มีค่าระหว่าง 0 ถึง 255

g – ค่าของแม่สีเขียว มีค่าระหว่าง 0 ถึง 255

b – ค่าของแม่สีน้ำเงิน มีค่าระหว่าง 0 ถึง 255

ผลที่คืน ค่าสีที่ได้จากการผสม

static int mixARGB(double a, double r, double g, double b)

หน้าที่ คืนค่าสีที่ได้จากการผสมแม่สีแดง เขียว น้ำเงิน และค่าอัลฟา. ค่าอัลฟา (alpha) มีไว้กำหนดระดับความทึบของสี 0 แทนไม่ทึบเลย (คือโปร่งใส จนมองไม่เห็น) 255 แทนทึบสุด ๆ การทาสีสองสีที่มีความโปร่งใสบ้าง (เช่น 100) ซ้อนทับกัน สีก็จะผสมกันเอง

พารามิเตอร์ a - ค่าอัลฟา มีค่าระหว่าง 0 ถึง 255
r - ค่าของแม่สีแดง มีค่าระหว่าง 0 ถึง 255
g - ค่าของแม่สีเขียว มีค่าระหว่าง 0 ถึง 255
b - ค่าของแม่สีน้ำเงิน มีค่าระหว่าง 0 ถึง 255
ผลที่คืน ค่าสีที่ได้จากการผสม

void loadImage(String fileName)

หน้าที่ อ่านแฟ้มภาพมาแสดงบนวินโดว์ ปรับขนาดวินโดว์ตามขนาดภาพ ชื่อภาพต้องระบุให้ครบเช่น "c:/temp/pic.jpg" แต่ถ้าเป็นแฟ้มภาพที่อยู่ในแฟ้ม jlab ก็ระบุเฉพาะชื่อได้ ประเภทแฟ้มภาพที่อ่านได้คือ jpg, gif, png, และ bmp
พารามิเตอร์ fileName - ชื่อแฟ้มภาพ

void loadImage(String fileName, double w, double h)

หน้าที่ อ่านแฟ้มภาพมาแสดงบนวินโดว์ ปรับขนาดของภาพและวินโดว์ให้เป็น wxh ชื่อภาพต้องระบุให้ครบเช่น "c:/temp/pic.jpg" แต่ถ้าเป็นแฟ้มภาพที่อยู่ในแฟ้ม jlab ก็ระบุเฉพาะชื่อได้ ประเภทแฟ้มภาพที่อ่านได้คือ jpg, gif, png, และ bmp
พารามิเตอร์ fileName - ชื่อแฟ้มภาพ
w - ความกว้างของวินโดว์
h - ความสูงของวินโดว์

void saveImage(String fileName)

หน้าที่ บันทึกภาพบนวินโดว์ลงในแฟ้มภาพ. ชื่อภาพต้องระบุให้ครบเช่น "c:/temp/pic.jpg" ประเภทแฟ้มภาพที่บันทึกได้คือ jpg, gif, png, และ bmp
พารามิเตอร์ fileName - ชื่อแฟ้มภาพ

DImage

DImage ผลิตอ็อบเจกต์เพื่อเก็บรูปภาพ เพื่อใช้คำสั่ง w.draw(img, xc, yc) ให้วินโดว์ w (แบบ DWindow) วาดรูป img โดยตำแหน่งศูนย์กลางของรูปอยู่ที่พิกัด (xc, yc)

ตัวสร้าง

DImage(String imageFile)

หน้าที่ ตัวสร้างรูปที่อ่านมาจากแฟ้มภาพที่กำหนดให้ อ่าน gif, jpg, png, bmp ได้
พารามิเตอร์ imageFile - แฟ้มภาพที่ต้องการอ่าน
สิ่งผิดปกติ IllegalArgumentException - เมื่ออ่านภาพไม่ได้

เมท็อด

int getWidth()

หน้าที่ คืนความกว้างของรูปนี้
ผลที่คืน ความกว้างของรูปนี้ (หน่วยเป็นจุดภาพ)

int getHeight()

หน้าที่ คืนความสูงของรูปนี้
ผลที่คืน ความสูงของรูปนี้ (หน่วยเป็นจุดภาพ)

void scaleTo(double s)

หน้าที่ ปรับขนาดของรูป ตามค่า `s` เช่น `s = 0.5` ทำให้รูปมีขนาดเป็น `50%` ของรูปดั้งเดิม หรือ `s = 2.0` ทำให้รูปมีขนาดเป็น `200%` ของรูปดั้งเดิม
พารามิเตอร์ `s` - ค่าการย่อหรือขยายรูป

void scaleTo(double sx, double sy)

หน้าที่ ปรับขนาดของรูป โดยกำหนดการย่อขยายความกว้างและสูงแยกจากกัน
พารามิเตอร์ `sx` - ค่าการย่อหรือขยายความกว้างของรูป
`sy` - ค่าการย่อหรือขยายความสูงของรูป

DPoint

`DPoint` มีไว้ผลิตอ็อบเจกต์แทนพิกัดบนวินโดว์ อ็อบเจกต์ของ `DPoint` ถูกผลิตจาก

- ผลจากคำสั่ง `w.getMouse()` ได้อ็อบเจกต์แบบ `DPoint` ที่แทนพิกัดของตัวชี้เมาส์บนวินโดว์ `w`
- เมื่อตัวชี้เมาส์เลื่อนในวินโดว์ `w` ระบบจะเรียกเมท็อด `onMouseMoved` ของวินโดว์ `w` ให้อัตโนมัติ หากเลื่อนเมาส์พร้อมกดปุ่ม ระบบจะเรียกเมท็อด `onMouseDragged` ของวินโดว์ `w` ให้อัตโนมัติ ในสองกรณีนี้ สิ่งที่ส่งมาให้เมท็อดดังกล่าวคือ อ็อบเจกต์ของ `DPoint` ที่แทนพิกัดของตัวชี้เมาส์บนวินโดว์ (หมายเหตุ : สองเมท็อดนี้มักเขียนในคลาสลูกของ `DWindow` เมื่อต้องการดักจับเหตุการณ์การเลื่อนเมาส์)

เมท็อด

int getX()

หน้าที่ คืนพิกัด `x` ของจุดนี้
ผลที่คืน พิกัด `x` ของจุดนี้

int getY()

หน้าที่ คืนพิกัด `y` ของจุดนี้
ผลที่คืน พิกัด `y` ของจุดนี้

บรรณานุกรม

หนังสือและแหล่งความรู้ทางการเขียนโปรแกรมมีมากมาย ผู้เขียนขอแสดงรายชื่อหนังสือและเว็บไซต์ที่ผู้เขียนได้เคยอ่าน และนำความรู้จากแหล่งเหล่านี้กลั่นกรองจนได้หนังสือเล่มนี้

[1] Barnes, David and Kölling, Michael, “Objects First with Java A Practical Introduction using BlueJ, 3rd edition” , Pearson Education (2006)

[2] Bloch, Joshua, “Effective Java Programming Language Guide”, Addison-Wesley Professional (2001)

[3] Bronson, Gary, “Java for Engineers and Scientists”, Brooks Cole (2003)

[4] Cohoon, James and Davidson, Jack, “Java Program Design”, McGraw-Hill (2003)

[5] Cooper, Doug, “Oh Pascal: Turbo Pascal 6.0”, W. W. Norton & Company (1992)

[6] Downey, Allen, “How to Think Like a Computer Scientist, Java Version”, Green Tea Press (2003) (http://www.greenteapress.com/thinkapjava)

[7] Gosling, James, Joy, Bill, et.al., “The Java Language Specification, Third Edition”, Prentice Hall (2005)

[8] Guzdial, Mark, Ericson, Barbara, “Introduction to Computing & Programming in Java™: A Multimedia Approach”, Prentice Hall (2006)

[9] Sierra, Kathy and Bates, Bert, “Head First Java, 2nd Edition”, O'Reilly Media, Inc.(2005)

[10] Stein, Lynn Andrea, “Interactive Programming In Java”, (http://www.cs101.org/ipij)

[11] Zakhour, Sharon, Hommel, Scott, et.al., “The Java Tutorial: A Short Course on the Basics, 4th Edition”, Prentice Hall PTR (2006)

[12] สมชาย ประสิทธิ์จูตระกูล, “การออกแบบและวิเคราะห์อัลกอริทึม”, สวทช (2544)

[13] สมชาย ประสิทธิ์จูตระกูล, “โครงสร้างข้อมูล : ฉบับวาจาจาวา”, สำนักพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย (2550)

[14] http://java.sun.com/docs/books/tutorial : “The Java™ Tutorials”

[15] http://javabat.com : “java practice problems”

[16] http://jtf.acm.org : “ACM Java Task Force”

[17] http://nifty.stanford.edu : “Nifty Assignments”

[18] http://www.cs101.org : “Rethinking CS101 project”

ดัชนี

ก, ข

ค

จ

ช, ซ

ด, ต

ถ, ท, น

บ, ป

ผ, พ, ฟ, ภ

ม, ย

ร, ล

ว

ศ, ส

ห

อ

A

B

C

D

E

F

G, H

I

J, L

M, N

O, P, R

S

T

U, V, W

สมชาย ประสิทธิ์จูตระกูล จบการศึกษาปริญญาตรีสาขาวิศวกรรมคอมพิวเตอร์ (เกียรตินิยมอันดับหนึ่งเหรียญทอง) จากจุฬาลงกรณ์มหาวิทยาลัย เมื่อปี พ.ศ. ๒๕๒๖ ได้รับพระราชทานทุนมูลนิธิ “อานันทมหิดล” เพื่อศึกษาต่อในปี พ.ศ. ๒๕๒๘ จบการศึกษาปริญญาโทและปริญญาเอกสาขาวิทยาศาสตร์คอมพิวเตอร์ เมื่อปี พ.ศ. ๒๕๓๐ และ พ.ศ. ๒๕๓๔ ตามลำดับ จากมหาวิทยาลัยอิลลินอยส์ ณ เมืองเออร์บานา-แชมเปน สหรัฐอเมริกา เข้ารับราชการเป็นอาจารย์ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัยตั้งแต่ปี พ.ศ. ๒๕๒๗ จนถึงปัจจุบัน มีผลงานด้านตำราดังนี้

- เริ่มเรียนเขียนโปรแกรม : ฉบับวาจาจาวา, พ.ศ. ๒๕๕๒
- โครงสร้างข้อมูล : ฉบับวาจาจาวา, พ.ศ. ๒๕๕๐
- การออกแบบและวิเคราะห์อัลกอริทึม, พ.ศ. ๒๕๔๔
- ภินทนคณิตศาสตร์, พ.ศ. ๒๕๔๔
- คณิตศาสตร์ดีสครีตเชิงประยุกต์ (เขียนร่วมกับ ดร. วิทยา วัชระวิทยากุล) , พ.ศ. ๒๕๓๖