TP JUnit

Récupérez à l'URL `http://cedric.cnam.fr/~farinone/SMB212/ADistribuer.zip`, le fichier `ADistribuer.zip`. Il contient le `junit-4XXX.jar` utile pour ce TP.

Première partie
Le premier test d'une classe

1°) Sous Eclipse, écrire la classe `SommeArgent` dans le package `junit.monprojet` :

```
package junit.monprojet;
public class SommeArgent {
    private int quantite;
    private String unite;

    public SommeArgent(int amount, String currency) {
        quantite = amount;
        unite = currency;
    }

      public int getQuantite() {
            return quantite;
      }

      public String getUnite() {
            return unite;
      }

      public SommeArgent add(SommeArgent m) {
          return new SommeArgent(getQuantite()+m.getQuantite(), getUnite());
      }
}
```

Les objets de cette classe sont des quantités d'argent dans une certaine unité (ou monnaie) comme $US100.65, £45.87, 100,65 CHF, 54,98 €, etc.

2°) Créer un package `junit.monprojet.test` et dans ce package une classe de tests JUnit 4. Vous aurez surement besoin, si vous ne l'avez pas déjà, du fichier `junit-4XXX.jar` donné dans `ADistribuer.zip` pour l'ajouter aux `.jar` de votre environnement Eclipse. Voir le cours pour l'ajouter.

Remarques sur Eclipse :
a) s'il vous manque des déclarations `import` (ou si vous en avez trop !), vous pouvez, dans Eclipse, les mettre par CTRL+MAJ+O. Ce sera le cas, entre autre, lorsque vous allez utiliser les annotations utiles à JUnit 4.
b) pour indenter correctement tout votre programme taper CTRL A suivi de CTRL I.

3°) On veut écrire une méthode de test qui construit deux sommes d'argent, en fait la somme et vérifie qu'elle est correcte.
Pour cela, il faut enrichir la classe `SommeArgent` de la méthode `equals()` qui définit l'égalité (ou l'équivalence) de deux objets et qui redéfinit la méthode `equals()` de la classe `java.lang.Object` (voir à

`http://download.oracle.com/javase/6/docs/api/java/lang/Object.html#equals(java.lang.Object)`). Deux sommes d'argent sont égales si elles sont de même unité et de même quantité. Écrire cette méthode `equals()` de la classe `SommeArgent`. Elle doit avoir pour signature :

```
public boolean equals(Object anObject)
```

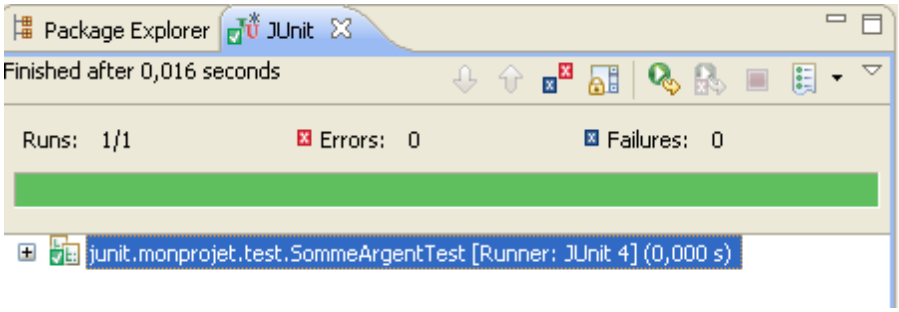
(et pas autrement ;-)).

Remarque : pour bien faire, il faudrait alors aussi redéfinir la méthode `hashCode()`. On pourra s'en passer ici.

4°) Écrire la méthode de test qui construit deux sommes d'argent, en fait la somme et vérifie qu'elle est correcte. Le corps de cette méthode est :

```
SommeArgent m12CHF= new SommeArgent(12, "CHF");  // (1)
SommeArgent m14CHF= new SommeArgent(14, "CHF");
SommeArgent expected = new SommeArgent(26, "CHF");
SommeArgent result = m12CHF.add(m14CHF);    // (2)
Assert.assertTrue(expected.equals(result));     // (3)
```

5°) Lancer ce test. Vous devez obtenir :



"Keep the bar green to keep the code clean."

Seconde partie
D'autres tests pour la classe `SommeArgent`

6°) Euh, en fait, on a oublié de tester la méthode `equals()` de la classe `SommeArgent`. Écrire une méthode de test pour cette méthode `equals()`. Le corps de la méthode de test peut être :

```
SommeArgent m12CHF= new SommeArgent(12, "CHF");
SommeArgent m14CHF= new SommeArgent(14, "CHF");
SommeArgent m14USD= new SommeArgent(14, "USD");

Assert.assertTrue(!m12CHF.equals(null));
Assert.assertEquals(m12CHF, m12CHF);
Assert.assertEquals(m12CHF, new SommeArgent(12, "CHF")); // (1)
Assert.assertTrue(!m12CHF.equals(m14CHF));
Assert.assertTrue(!m14USD.equals(m14CHF));
```

Que teste t-on dans la dernière ligne de ce code ?

7°) Dans ces deux méthodes de tests il y a des initialisations communes. Écrire un code, dans la classe de test, qui regroupe ces initialisations dans une seule méthode externe. Vérifier que ces initialisations sont bien effectuées avant chaque lancement de méthode de tests.

7.1°) Comment se nomme ces ensembles d'objets qui sont créés à chaque exécution d'une méthode de test ?

7.2°) De manière similaire, si après chaque exécution de méthodes de tests, on veut exécuter un code commun (de désallocation par exemple), comment doit on procéder ? Vérifier votre réponse par le code. On voudrait obtenir une exécution des tests qui affichent :

```
1ime passage avant exécution d'une méthode de test
1ime passage APRES exécution d'une méthode de test
2ime passage avant exécution d'une méthode de test
2ime passage APRES exécution d'une méthode de test
```

8°) Le code donné ci dessus pour la méthode `add()` qui ajoute deux sommes d'argent, n'est pas vraiment correct : que se passe t il si ces deux sommes ne sont pas de la même unité ? On se propose dans ce cas de faire lever une exception `UniteDistincteException` et de modifier alors la méthode en la signant :

```
public SommeArgent add(SommeArgent m) throws UniteDistincteException
```

Il faut pour cela, ajouter dans votre projet cette classe `Exception` :

```
public class UniteDistincteException extends Exception {
	private SommeArgent somme1, somme2;

	public UniteDistincteException(SommeArgent sa1, SommeArgent sa2) {
		somme1 = sa1;
		somme2 = sa2;
	}

	public String toString() {
		return "unité distincte : " + somme1.getUnite() + " != " +
somme2.getUnite();
	}
}
```

et modifier la méthode `add()` par :

```
public SommeArgent add(SommeArgent m) throws UniteDistincteException {
	if (!m.getUnite().equals(this.getUnite())) {
		throw new UniteDistincteException(this, m);
	}
	else return new SommeArgent(getQuantite()+m.getQuantite(), getUnite());
}
```

8.1°) Comme la méthode `add()` a été modifiée, corriger les éventuelles erreurs qui sont apparues dans votre projet.

8.2°) Ecrire une méthode de test, qui construit deux objets de la classe `SommeArgent` avec des unités distinctes et vérifier que, dans ce cas, une exception `UniteDistincteException` est levée.

Troisième partie : un porte monnaie

On veut regrouper ces diverses sommes d'argent dans un porte monnaie et, pour cela construire une nouvelle classe `PorteMonnaie`. Une première version de cette classe peut être :

```
import java.util.HashMap;
```

```
public class PorteMonnaie {
	HashMap<String, Integer> contenu;

	public HashMap<String, Integer> getContenu() {
		return contenu;
	}

	public PorteMonnaie() {
		contenu = new HashMap<String, Integer>();
	}

	public void ajouteSomme(SommeArgent sa) {
		// à définir cf. question suivante
	}
}
```

9°) On veut ajouter dans cette classe la possibilité d'ajouter des sommes d'argent. Les spécifications du porte monnaie sont :
"Si un ajoute 12 € et qu'il n'y a pas déjà d'euro dans le porte monnaie, cette somme est simplement mise. Si il y avait déjà 10 €, il y aura désormais 22 €."
Coder cette spécification dans la méthode `ajouteSomme()`.
Remarque : vous aurez peut être besoin des méthodes utilitaires de la classe `Vector` accessible à l'URL `http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html`

10°) Ecrire une méthode de `public String toString()` qui affiche le contenu du porte monnaie.

11°) Construire, dans le package de test, une classe de test pour la classe `PorteMonnaie`, contenant deux méthodes de tests (au moins) qui vérifient la bonne cohérence de la méthode `ajouteSomme()` (cf. les spécifications du porte monnaie à la question 9°). On pourra par exemple écrire la méthode
`public boolean equals(Object obj)`
dans la classe `PorteMonnaie` qui retourne `true` si l'instance et le porte-monnaie passé en paramêtres ont les mêmes devises en même quantité.

Conclusion

On a contruit des classes et à chaque étape on a contruit un test. On a employé la méthodologie : "code a little, test a little, code a little, test a little, ...". Il est vrai qu'on aura pu faire mieux : commencer à écrire les tests puis écrire le code puis écrire les tests suivants, puis le code suivant, etc. Relisez le TP : c'est en effet possible.

Bibliographie

Ce TP est grandement inspiré de celui se trouvant à `http://junit.sourceforge.net/doc/testinfected/testing.htm`. Bon, OK, c'est parfois une simple localisation ;-). Ah si, ce problème utilise JUnit 4, alors que cette URL donne une solution avec JUnit 3. De plus je l'ai adapté à Eclipse. Enfin j'ai utilisé une `HashMap` générique plutôt qu'un `Vector` non générique dans les questions sur le porte monnaie.