Vererbung und Polymorphie

Florian Adamsky, B. Sc.

florian.adamsky@iem.thm.de
http://florian.adamsky.it/

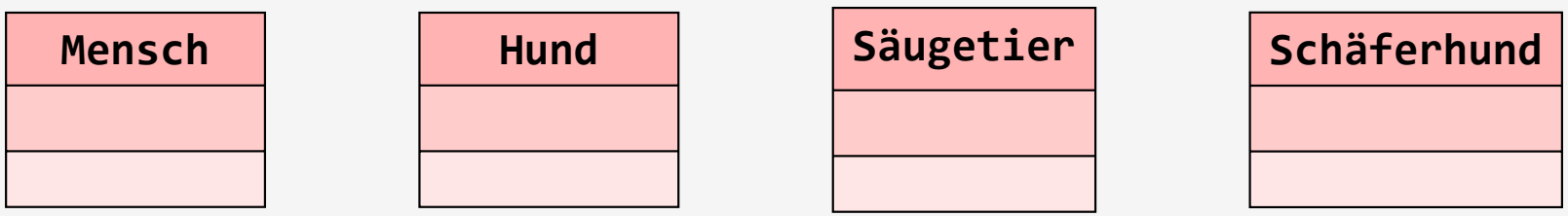
Softwareentwicklung im SS 2014

Outline

Contents

Einführung

- Vererbung und Polymorphie sind fundamentale Konzepte der Objektorientierten Programmierung
- menschliche Intelligenz findet, erkennt und erzeugt Beziehungen zwischen Begriffen



- diesen Abstraktionsvorgang bildet C++ über Vererbung ab

Contents

Was ist Vererbung?

- Vererbung bildet in der OOP eine *ist-ein*-Beziehung ab
  - Der Mensch *ist ein* Säugetier
- Systematische Spezialisierung von oben nach unten
- Die *Unterklasse* erbt damit alle Merkmale der *Oberklasse*

Terminologie

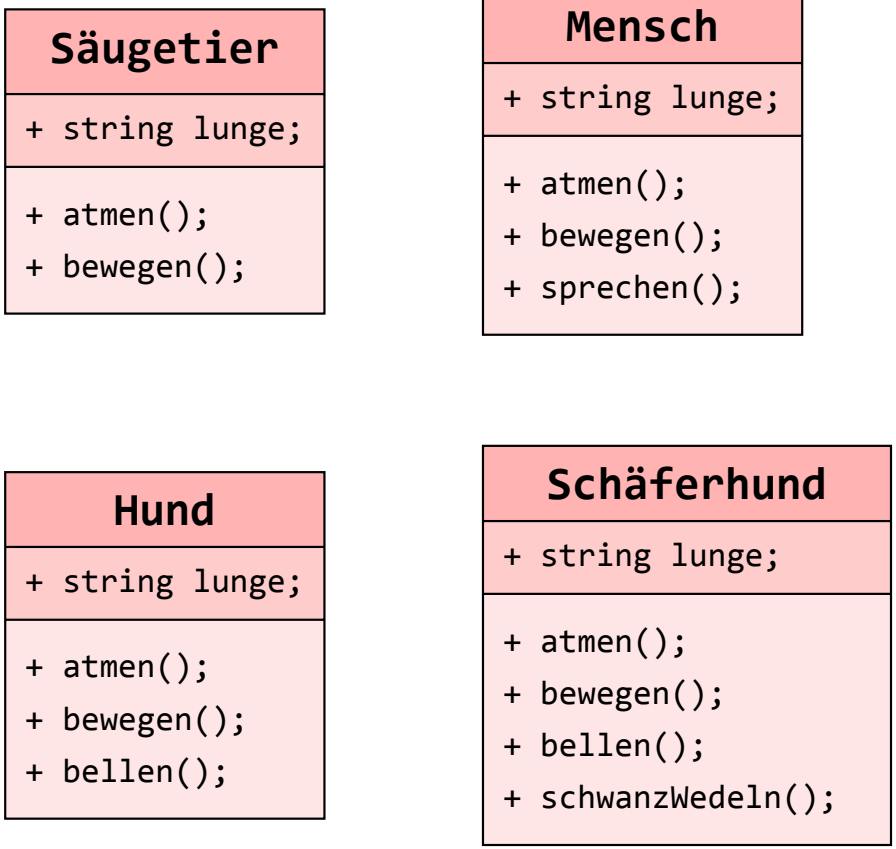
Oberklasse	Basisklasse, Elternklasse, Superklasse
Unterklasse	Kindklasse, Subklasse
Vererbung	Ableitung, Spezialisierung

Was wird vererbt und was nicht?

- Unterklassen erben alle zugreifbaren Member der Basisklasse:
  - Konstruktor und Destruktor
  - Attribute / Klassenvariablen
  - Methoden und Operatoren
- Was wird nicht vererbt?
  - `friend` Funktionen
  - Member die als `private` deklariert sind

Beispiel

Ohne Vererbung



Beispiel

Ohne Vererbung

Säugetier

```
+ string lunge;
+ atmen();
+ bewegen();
```

Mensch

```
+ string lunge;
+ atmen();
+ bewegen();
+ sprechen();
```

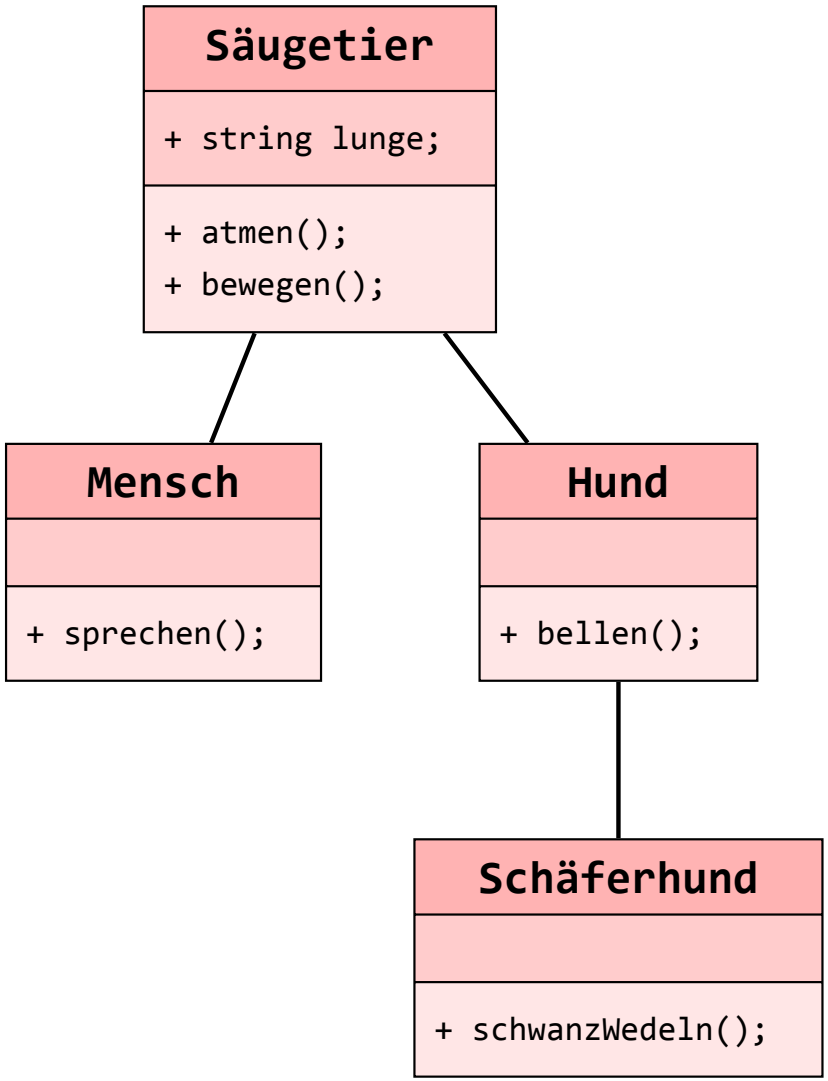
Hund

```
+ string lunge;
+ atmen();
+ bewegen();
+ bellen();
```

Schäferhund

```
+ string lunge;
+ atmen();
+ bewegen();
+ bellen();
+ schwanzWedeln();
```

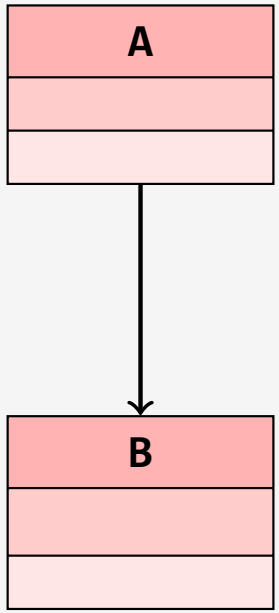
Vererbung



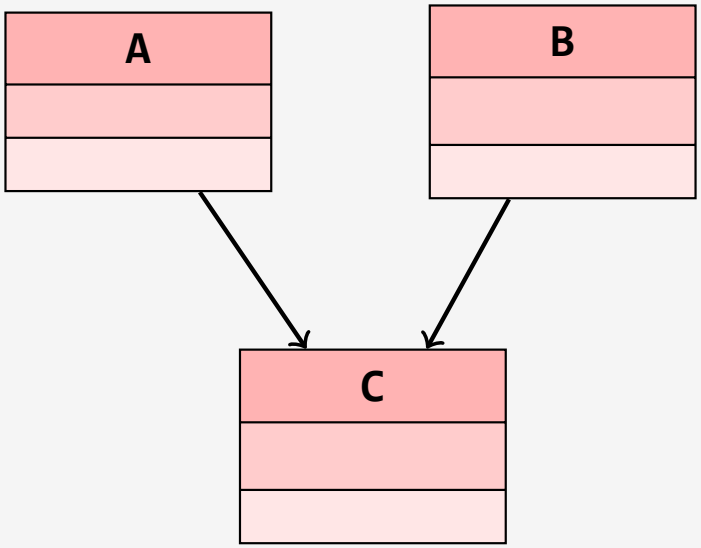
Vorteile von Vererbung

- Klassen sind besser wiederverwendbar
- Weniger Code
- Dadurch weniger Bugs

Einfache Vererbung

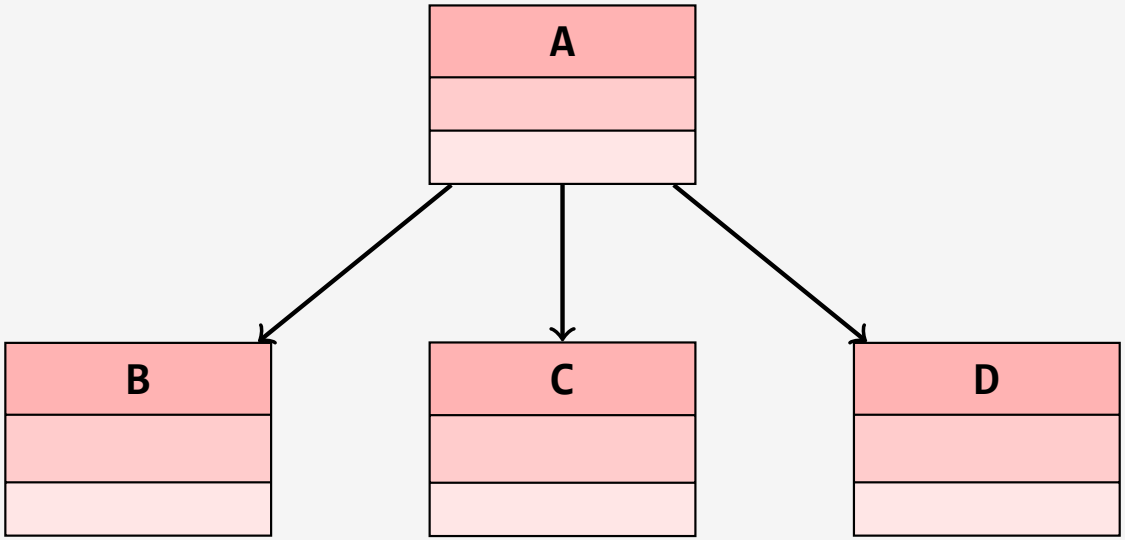


Mehrfach Vererbung

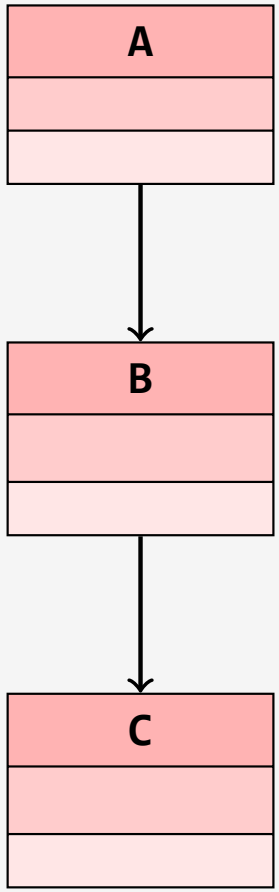


- nur in wenigen Sprachen möglich: C++, Perl, Python
- sollte vermieden werden

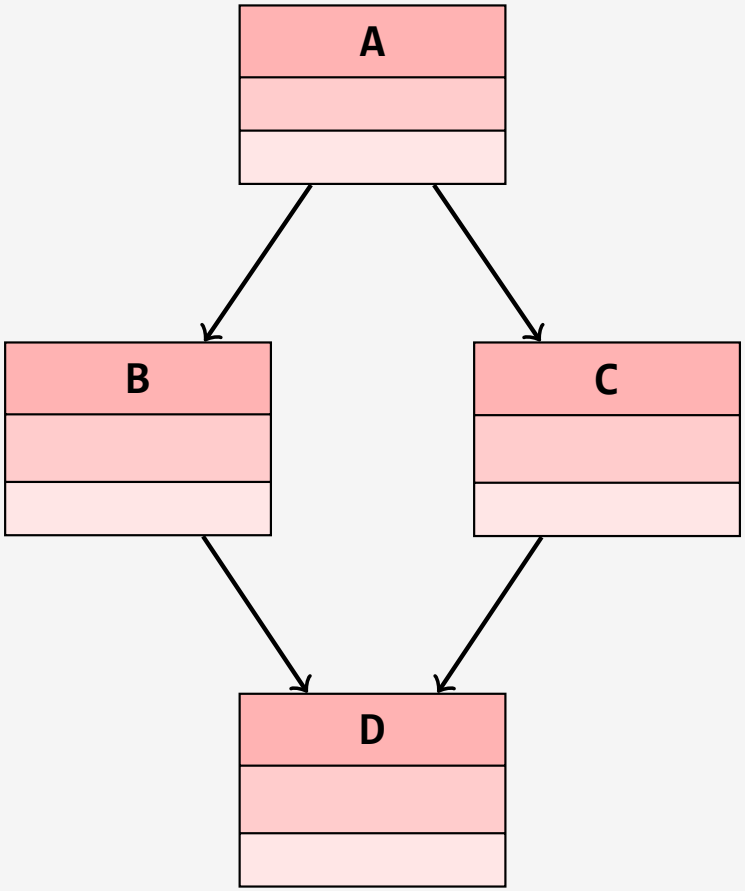
Hierarchische Vererbung



MultiLevel Vererbung



Hybride Vererbung



Zugriffsschlüsselwörter (access specifier)

private Nur für Attribute/Methoden die innerhalb (intern) der Klassen verwendet werden.

public Stellen Schnittstellen nach außen (Main) dar.

protected Dort zu benutzen, wo eine Subklasse (abgeleitete Klasse) die Methoden/Attributen nochmals benötigt.

Tabelle: Zugriffserlaubnis der einzelnen Typen[1]

Zugriffstyp	Eigene Klasse	Abgeleitete Klasse	Fremde Klasse
`private`	ja	nein	nein
`public`	ja	ja	ja
`protected`	ja	ja	nein

[1] Aus C++-Programmierung, Andre Willms, Addision-Wesley, S.180

C++-Syntax

```
class UnterKlasse : public Oberklasse { /* ... */ }

class UnterKlasse : protected Oberklasse { /* ... */ }

class UnterKlasse : private Oberklasse { /* ... */ }

class UnterKlasse : Oberklasse { /* ... */ }

class UnterKlasse : public Oberklasse1, public Oberklasse2 { /* ... */ }
```

Oberklasse	public wird	protected wird	private wird
public	public	protected	private
protected	protected	protected	private
private	private	private	private

Konstruktor 1

- Beim Erstellen eines Objekts der Unterklasse wird der Konstruktor der Oberklasse automatisch aufgerufen wenn der Konstruktor keine Argumente erwartet

```
class BasisKlasse {
public:
    BasisKlasse() { /* ... */ }
};

class UnterKlasse : public BasisKlasse {
public:
    UnterKlasse(int x) { /* ... */ }
};
```

Konstruktor 2

- Wenn der Konstruktor der Basisklasse Argumente erwartet, muss dieser explizit mit der constructor initialization list aufgerufen werden

```
class BasisKlasse {
public:
    BasisKlasse(int x) { /* ... */ }
};

class UnterKlasse : public BasisKlasse {
public:
    UnterKlasse(int x) : BasisKlasse(x) { /* ... */ }
};
```

Contents

Was ist Polymorphie?

- Polymorphie/Polymorphismus (griechisch)

  polys "*viele, mehrere*"
  morphē "*Form, Gestalt*"

- Zwei wesentliche Formen:
  - statisches Binden (zur Compile-Zeit)
  - dynamisches Binden (zur Laufzeit)

Statisches Binden

```
class BasisKlasse {
public:
    void methode () { cout << "Basisklasse vorgegeben" << endl; }
};

class UnterKlasse : public BasisKlasse {
public:
    void methode () { cout << "Unterklasse ueberschrieben" << endl; }
};

int main (void) {
    Unterklasse objekt;
    objekt.methode();
}
```

Statisches Binden

```
class BasisKlasse {
public:
    void methode () { cout << "Basisklasse vorgegeben" << endl; }
};

clas
publ

};

int main (void) {
    Unterklasse objekt;
    objekt.methode();
}
```

Ausgabe

```
$ ./programm
Unterklasse ueberschrieben
```

Dynamisches Binden

```
class BasisKlasse {
public:
    virtual void methode () { cout << "Basisklasse vorgegeben"; }
};

class UnterKlasse : public BasisKlasse {
public:
    void methode () { cout << "Unterklasse ueberschrieben"; }
};

int main (void) {
    BasisKlasse* objekt;
    objekt = new Unterklasse();
    objekt.methode();
}
```

Dynamisches Binden

```
class BasisKlasse {
public:
    virtual void methode () { cout << "Basisklasse vorgegeben"; }
};

clas
publ

};

int main (void) {
    BasisKlasse* objekt;
    objekt = new Unterklasse();
    objekt.methode();
}
```

Ausgabe

```
$ ./programm
Unterklasse ueberschrieben
```

Danke

Fragen?

florian.adamsky@iem.thm.de
http://florian.adamsky.it/