Universität Osnabrück

Informatik B – Objektorientierte Programmierung in Java

Vorlesung 03:

Objektorientierte Programmierung (Teil 3)

Inhalt

- Vererbung
- Beispiel
  - Auto_3
  - Person

Vererbung

- Eines der wichtigsten Designmerkmale objektorientierter Sprachen ist die Zusammenfassung von Variablen und Operationen zu Klassen.
- Eine mindestens ebenso wichtige Eigenschaft ist die der *Vererbung*, also die Möglichkeit, Eigenschaften vorhandener Klassen auf neue Klassen zu übertragen.

Vererbung

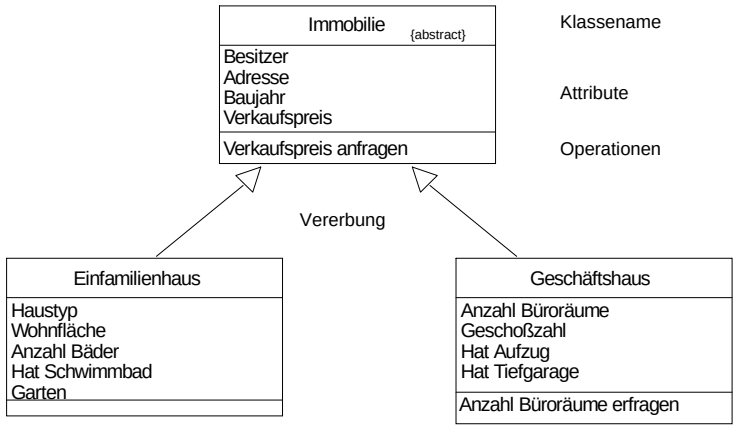
- Einfache Vererbung und Mehrfachvererbung
  - Dabei unterscheidet man zwischen einfacher Vererbung, bei der eine Klasse von maximal einer anderen Klasse abgeleitet werden kann, und Mehrfachvererbung, bei der eine Klasse von mehr als einer anderen Klasse abgeleitet werden kann.
  - In Java (und z.B. Smalltalk) gibt es lediglich Einfachvererbung, um den Problemen aus dem Weg zu gehen, die durch Mehrfachvererbung entstehen können.
  - In C++ gibt es die Mehrfachvererbung.

Vererbung

- Um die Einschränkungen in den Designmöglichkeiten, die bei Einfachvererbung entstehen, zu vermeiden, wurde mit Hilfe der Interfaces eine neue restriktive Art der Mehrfachvererbung eingeführt.
- Interfaces werden später behandelt.

Vererbung

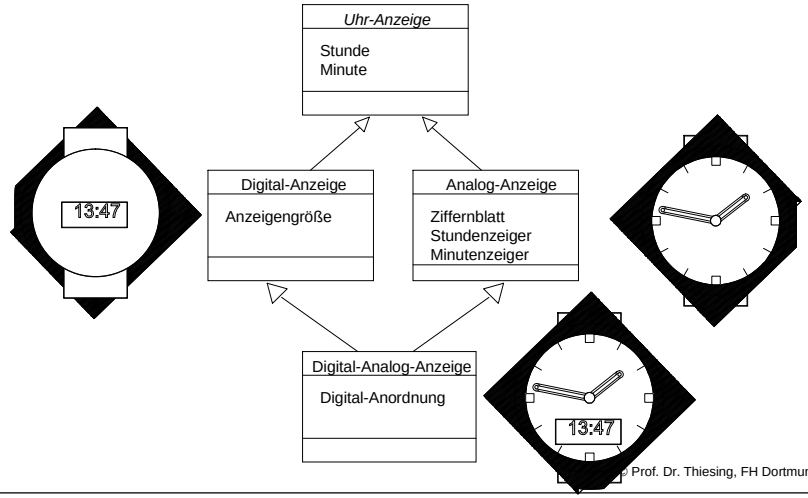
- Beispiel: Einfachvererbung



Notation: UML

Vererbung

- Beispiel: Mehrfachvererbung



Vererbung

- Generalisierung und Spezialisierung
  - Die Zusammenfassung gemeinsamer Eigenschaften von Klassen in einer gemeinsamen Oberklasse wird auch als Generalisierung bezeichnet.
  - Die Ableitung einer Klasse aus einer Oberklasse wird auch Spezialisierung genannt.

Vererbung

- Ableiten einer Klasse
  - Um eine neue Klasse aus einer bestehenden abzuleiten, ist im Kopf der Klasse mit Hilfe des Schlüsselworts **`extends`** ein Verweis auf die Basisklasse anzugeben.
  - Hierdurch erbt die abgeleitete Klasse alle Eigenschaft der Basisklasse, d.h. alle Variablen und alle Operationen.
  - Durch Hinzufügen neuer Elemente oder Überlagerung der vorhandenen Elemente kann die Funktionalität der abgeleiteten Klasse erweitert werden.

Vererbung

- Beispiel:
  - Es wird die neue Klasse **`Cabrio`** definiert, die sich von Auto nur dadurch unterscheidet, dass sie zusätzlich die Zeit, die zum Öffnen des Verdecks benötigt wird, speichern soll:

```
class Cabrio
extends Auto
{  int vdauer;
}
```

  - Es kann nun nicht nur auf die neue Variable **`vdauer`**, sondern auch auf alle Elemente der Basisklasse **`Auto`** zugegriffen werden:

```
Cabrio kfz1 = new Cabrio();
kfz1.name = "MX5";
kfz1.erstzulassung = 1994;
kfz1.leistung = 115;
kfz1.vdauer = 120;
System.out.println("Alter = "+kfz1.alter());
```


Vererbung

- Die Vererbung von Klassen kann beliebig tief geschachtelt werden.
- Eine abgeleitete Klasse erbt dabei jeweils die Eigenschaften der unmittelbaren Oberklasse, die ihrerseits die Eigenschaften ihrer unmittelbaren Oberklasse erbt usw.

Vererbung

- Beispielsweise kann die Klasse **`Cabrio`** verwendet werden, um daraus eine neue Klasse **`ZweisitzerCabrio`** abzuleiten:

```
class ZweisitzerCabrio
extends Cabrio
{
  boolean notsitze;
}
```

Vererbung

- Diese könnte nun verwendet werden, um ein Objekt zu instantiieren, das die Eigenschaften der Klassen **`Auto`**, **`Cabrio`** und **`ZweisitzerCabrio`** hat:

```
ZweisitzerCabrio kfz2 = new ZweisitzerCabrio();
kfz2.name = "911-T";
kfz2.erstzulassung = 1982;
kfz2.leistung = 94;
kfz2.vdauer = 50;
kfz2.notsitze = true;
System.out.println("Alter = "+kfz2.alter());
```


Vererbung

- Nicht jede Klasse darf zur Ableitung neuer Klassen verwendet werden.
- Besitzt eine Klasse den Modifikator **`final`**, ist es nicht erlaubt, eine neue Klasse aus ihr abzuleiten.

Vererbung

- Operationenüberlagerung
  - Neben den Attributen erbt eine abgeleitete Klasse auch die Operationen ihrer Oberklasse (wenn dies nicht durch spezielle Modifikatoren verhindert wird).
  - Daneben dürfen auch neue Operationen definiert werden.
  - Die Klasse besitzt dann alle Operationen, die aus der Oberklasse geerbt wurden und zusätzlich die, die in der Klasse neu definiert wurden.

Vererbung

  - Daneben dürfen auch bereits von der Oberklasse geerbte Operationen neu definiert werden.
  - In diesem Fall spricht man von einer Überlagerung der Operation.
  - Wurde eine Operation überlagert, wird beim Aufruf der Operation auf Objekten dieses Typs immer die überlagernde Version verwendet.

Vererbung

- Das folgende Beispiel erweitert die Klasse **`ZweisitzerCabrio`** um die Operation **`alter()`**, welches nun in Monaten ausgegeben werden soll:

```
class ZweisitzerCabrio
extends Cabrio
{
  boolean notsitze;

  int alter()
  {
    return 12 * (2004 – erstzulassung);
  }
}
```


Vererbung

- Da die Operation **`alter()`** bereits aus der Klasse **`Cabrio`** geerbt wurde, die sie ihrerseits von **`Auto`** geerbt hat, handelt es sich um eine Überlagerung.
- Zukünftig würde dadurch in allen Objekten vom Typ **`ZweisitzerCabrio`** bei Aufruf von **`alter()`** die überlagerte Version, bei allen Objekten des Typs **`Auto`** oder **`Cabrio`** aber die alte Version verwendet werden.
- Es wird immer die Variante aufgerufen, die dem aktuellen Objekt beim Zurückverfolgen der Vererbungslinie am nächsten liegt.

Vererbung

- Dynamisches Binden
  - Nicht immer kann bereits der Compiler entscheiden, welche Variante einer überlagerten Operation er aufrufen soll.
  - Ein Objekt einer abgeleiteten Klasse ist zuweisungskompatibel zu einer Variablen einer übergeordneten Klasse.
  - Es darf also beispielsweise ein Cabrio-Objekt ohne weiteres einer Variablen vom Typ **`Auto`** zugewiesen werden.

Vererbung

- Die Variable vom Typ **`Auto`** kann während ihrer Lebensdauer Objekte verschiedenen Typs enthalten, (insbesondere vom Typ **`Auto`**, **`Cabrio`** und **`ZweisitzerCabrio`**).
- Damit kann natürlich nicht schon zum Zeitpunkt der Übersetzung durch den Compiler entschieden werden, welche Version einer überlagerten Operation aufgerufen werden soll.
- Der Compiler muss also Code generieren, um dies zur Laufzeit zu entscheiden.
- Man bezeichnet dies auch als *dynamisches Binden*.

Vererbung

- Dieses Verhalten wird in C++ durch virtuelle Methoden realisiert und muss mit Hilfe des Schlüsselwortes **`virtual`** explizit angeordnet werden.
- In Java ist eine explizite Deklaration nicht nötig, denn Operationenaufrufe werden immer dynamisch interpretiert.
- Der dadurch verursachte Aufwand ist nicht zu vernachlässigen und liegt deutlich über den Kosten eines Operationenaufrufs in C++.

Vererbung

- Es gibt drei Möglichkeiten, dafür zu sorgen, dass eine Operation nicht dynamisch interpretiert wird.
- Dabei wird mit Hilfe zusätzlicher Modifikatoren dafür gesorgt, dass die betreffende Operation nicht überlagert werden kann (statisches Binden):
  - Operationen, die mit dem Modifikator **`private`** versehen werden, sind in abgeleiteten Klassen nicht sichtbar und können daher nicht überlagert werden.
  - Bei Operationen, die mit dem Modifikator **`final`** versehen sind, deklariert der Programmierer explizit, dass sie nicht überlagert werden sollen.
  - Auch bei Klassenoperationen werden statisch gebunden.

Vererbung

- Aufrufen einer Oberklassenoperation über **`super`**
  - Wird eine Operation **`x`** in einer abgeleiteten Klasse überlagert, wird die ursprüngliche Operation **`x`** verdeckt.
  - Aufrufe von **`x`** beziehen sich immer auf die überlagernde Variante.
  - Oftmals ist es allerdings nützlich, die verdeckte Oberklassenoperation aufrufen zu können, beispielsweise, wenn deren Funktionalität nur leicht verändert werden soll.

Vererbung

- In diesem Fall kann mit Hilfe des Ausdrucks **`super.x()`** die Operation der Oberklasse aufgerufen werden.
- Der kaskadierte Aufruf von Oberklassenoperationen (wie in **`super.super.x()`**) ist nicht erlaubt.

Vererbung

- Vererbung und Konstruktoren
  - Konstruktorenverkettung
    - Wenn eine Klasse instantiiert wird, garantiert Java, dass ein zur Parametrisierung des **new**-Operators passender Konstruktor aufgerufen wird.
    - Daneben garantiert der Compiler, dass auch der Konstruktor der Oberklasse aufgerufen wird.
    - Dieser Aufruf kann entweder explizit oder implizit geschehen.

Vererbung

- Falls als erste Anweisung innerhalb eines Konstruktors ein Aufruf der Operation **`super`** steht, wird dies als Aufruf des Konstruktors der Oberklasse interpretiert.
- **`super`** wird wie eine normale Operation verwendet und kann mit oder ohne Parameter aufgerufen werden.
- Der Aufruf muss natürlich zu einem in der Oberklasse definierten Konstruktor passen.
- Falls als erste Anweisung im Konstruktor kein Aufruf von **`super`** steht, setzt der Compiler an dieser Stelle einen impliziten Aufruf **`super();`** ein und ruft damit den parameterlosen Konstruktor der Oberklasse auf.

Vererbung

- Falls ein solcher Konstruktor in der Oberklasse nicht definiert wurde, gibt es einen Compiler-Fehler.
- Alternativ zu diesen beiden Varianten, einen Konstruktor der Oberklasse aufzurufen, ist es auch erlaubt, mit Hilfe der **`this(…)`**-Operation einen anderen Konstruktor der eigenen Klasse aufzurufen, dies allerdings nur als erste Anweisung im Konstruktor.
- Um die oben erwähnten Zusagen einzuhalten, muss dieser allerdings selbst direkt oder indirekt einen Konstruktor der Oberklasse aufrufen.

Vererbung

- Der Default-Konstruktor
  - Das Anlegen von Konstruktoren in einer Klasse ist optional.
  - Falls in einer Klasse überhaupt kein Konstruktor definiert wurde, erzeugt der Compiler beim Übersetzen der Klasse automatisch einen parameterlosen Default-Konstruktor.
  - Dieser enthält lediglich einen Aufruf des parameterlosen Konstruktors der Oberklasse.

Vererbung

- Aufruf von Konstruktoren
  - Konstruktoren werden nicht vererbt.
  - Alle Konstruktoren, die in einer abgeleiteten Klasse benötigt werden, müssen neu definiert werden, selbst wenn sie nur aus einem Aufruf des Konstruktors der Oberklasse bestehen.
  - Durch diese Regel wird bei jedem Neuanlegen eines Objekts eine ganze Kette von Konstruktoren aufgerufen.
  - Da nach den obigen Regeln jeder Konstruktor zuerst den Konstruktor der Oberklasse aufruft, wird die Initialisierung von oben nach unten in der Vererbungshierarchie durchgeführt:
    - Zuerst wird der Konstruktor der Klasse **`Object`** ausgeführt, dann der der ersten Unterklasse usw., bis zuletzt der Konstruktor der zu instantiierenden Klasse ausgeführt wird.

Vererbung

- Destruktorenverkettung
  - Im Gegensatz zu den Konstruktoren werden die Destruktoren eines Ableitungszweiges nicht automatisch verkettet.
  - Falls eine Destruktorenverkettung erforderlich ist, kann sie durch explizite Aufrufe des Destruktors der Oberklasse mit Hilfe der Anweisung **`super.finalize()`** durchgeführt werden.

Beispiel

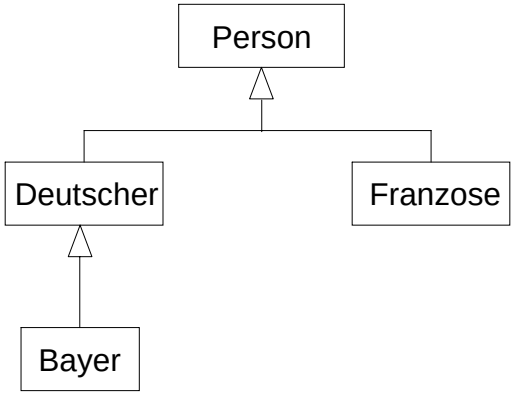
- Auto_3
- Person

Beispiel: Person

- Zu schreiben ist ein einfaches Programm, welches die Daten von Personen unterschiedlicher geographischer Herkunft verwaltet und jede Person einen landestypischen Gruß ausgeben lässt.

Beispiel: Person

- Klassendiagramm



Beispiel: Person

- Die oberste Klasse in der Hierarchie, **`Person`**, muss Operationen und Attribute bereitstellen, welche für alle Personen gleichermaßen gelten:

```
class Person
{
  String name;

  Person (String name)
  { this.name = name;
  }

  String getName()
  { return name;
  }
}
```


Beispiel: Person

- Eine Operation zum Grüßen wurde vorerst nicht aufgenommen, weil wir noch gar nicht wissen können, wie eine Person grüßt, deren Nationalität wir nicht kennen.

Beispiel: Person

- Unterklassen

```
class Franzose extends Person
{
  Franzose (String name)
  { super(name);
  }

  String getGruss()
  { return "Bonjour";
  }
}
```

Beispiel: Person

- Unterklassen

```
class Deutscher extends Person
{
  Deutscher (String name)
  { super(name);
  }

  String getGruss()
  { return "Guten Tag";
  }
}
```


Beispiel: Person

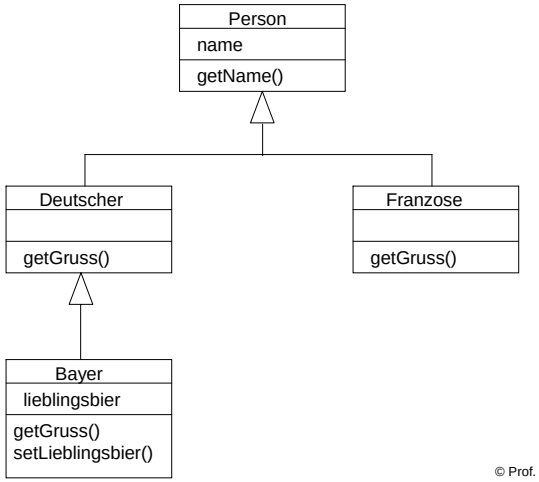
- Überschreiben von Operationen

```
class Bayer extends Deutscher
{
  String lieblingsbier;

  Bayer (String name)
  { super(name);
  }
  void setLieblingsbier(String bier)
  { lieblingsbier = bier;
  }
  String getGruss()
  {return "Gruess Gott. Hoast a " + lieblingsbier + "?";
  }
}
```


Beispiel: Person

- Erweitertes Klassendiagramm



Beispiel: Person

- Benutzung der Klassen: das Programm

```
class GrussAusgabe
{
    public static void main(String[] args)
    {
       Franzose jean = new Franzose("Jean");
       Deutscher hans = new Deutscher("Hans");
       Bayer sepp = new Bayer("Sepp");
       sepp.setLieblingsbier("Franziskaner Weisse");

       System.out.println(jean.getName() + " : " + jean.getGruss());
       System.out.println(hans.getName() + " : " + hans.getGruss());
       System.out.println(sepp.getName() + " : " + sepp.getGruss());
    }
}
```

Beispiel: Person

- Auswahl der auszuführenden Operation
  - Wie findet der Java-Interpreter die richtige, auszuführende Operation, wenn ein Objekt aus einer Vererbungshierarchie aufgerufen wird?
    - Sie folgt einem einfachen Prinzip:
      - Beim Aufruf einer Operation sucht der Java-Interpreter zunächst in der Klasse des beauftragten Objekts, ob dort diese Operation vorhanden ist.
      - Ist dies der Fall, so wird die Operation ausgeführt.
      - Wird die Operation in der Klasse nicht gefunden, so wird die Suche in der nächsthöheren Klasse fortgesetzt und so weiter.
      - Spätestens in der obersten Klasse der Hierarchie wird die Operation gefunden und ausgeführt.

Beispiel: Person