II.4.1a Unterklassen und Vererbung (Teil 1)

Dienstag, 8. Dezember 2020 16:01

1. Generelles Konzept der Ober- und Unterklassen
2. Erzeugung von Objekten in Klassenhierarchien
3. Überschreiben von Methoden und Verdecken von Attributen

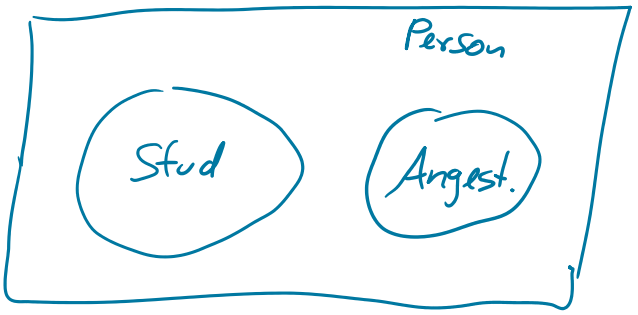
## 1. Generelles Konzept der Ober- und Unterklassen

Klassen Stud und Angestellte sind ähnlich, aber nicht gleich.
Methode toString ist in beiden Klassen gleich implementiert.
⇒ nicht elegant

Besser: Versuche, Gemeinsamkeiten verschiedener Klassen in Oberklasse zusammenzufassen.

⇒ Stud und Angestellte sollten Unterklassen der Klasse Person sein.

⇒ Alle Eigenschaften der Oberklasse werden an Unterklassen vererbt.
Unterklassen können zusätzliche Attribute u. Methoden haben.

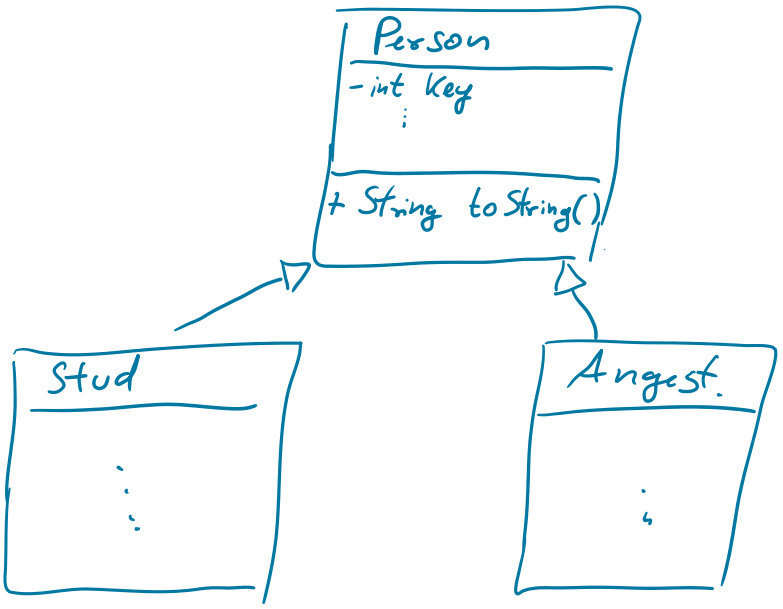



Java hat nur Einfachvererbung: Jede Klasse hat nur eine direkte Oberklasse.

Die "oberste" Klasse ist "Object".
Wenn eine Klasse kein "extends" im Klassenkopf hat

Die oberste Klasse ist Object.
Wenn eine Klasse kein "extends" im Klassenkopf hat, dann wird automatisch "extends Object".

## Klassendiagramm:




Grund für Einfachvererbung: Sonst wäre unklar, aus welcher Oberklasse geerbt werden soll, wenn Eigenschaften gleich heißen.

## Zugriff auf Objekte in Klassenhierarchien

- Zuweisung von Objekt der U-Klasse an Variable der O-Klasse: erlaubt, denn jedes Objekt der U-Klasse ist auch eines der O-Klasse.
  Allerdings kann es in der U-Klasse zusätzliche Attribute geben.
  Implizite Typanpassung vom speziellen Typ der U-Klasse zum allgemeinen Typ der O-Klasse.
  Bei prim. Datentypen:

```
double d = 2;
  ↑        ↑ int
 double
```

↑ double  " int

- Zuweisung zwischen 2 Unterklassen: verboten
  (Stud und Angestellte sind in keiner U-Klassen-Relation.)
- Zugriff auf Attribute der U-Klasse über Variable der O-Klasse: verboten
  Grund: Man kann i.A. nicht (zur Compile-Zeit) sicherstellen, dass die Variable der O-Klasse wirklich auf ein Objekt der U-Klasse zeigt.
- Zuweisung von Objekt der O-Klasse an Variable der U-Klasse: verboten
  Grund: Man kann i.A. nicht (zur Compile-Zeit) sicherstellen, dass das Objekt der O-Klasse wirklich auch zur richtigen U-Klasse gehört.

  Abhilfe: explizite Datentypanpassung

  (Stud) p

  Wenn p wirklich auf ein Stud-Objekt gezeigt hat, dann wird hierbei aus p wieder das ursprüngliche Stud-Objekt (mit der ursprünglichen Matrikelnr.).

  Anders als bei prim. Datentypen. Dort kann Information bei der Datentypanpassung verloren gehen.

  (int) 2.5

  wandelt den double-Wert in den int-Wert 2. Der Nachkomma-Teil wird hierbei gelöscht.

  Falls p nicht auf ein Stud-Objekt zeigt, dann führt

  (Stud) p

(Stud) p

zu einer Exception (=> Prog. Abbruch).

Um zur Laufzeit zu überprüfen, welchen Typ ein Objekt hat:

instanceof

Verwendung von instanceof lässt sich oft vermeiden durch geeignetes Überschreiben von Methoden.

## Konzeptionelles Modell

- Jedes Objekt der U-Klasse enthält ein Obj. der O-Klasse.
- p=s: p zeigt auf den Teil des Stud-Objekts, der einem Person-Objekt entspricht.
  Daher ist Attribut matrikelnr über p nicht sichtbar.

## 2. Erzeugung von Objekten in Klassenhierarchien

Erzeugung von Objekten:

new und Aufruf eines Konstruktors

Jetzt: Wie funktioniert dies bei Ober- und Unterklassen?

- Aufruf von Konstruktoren der O-Klasse aus dem Konstruktor der U-Klasse: super(...).
  O-Klasse ist eindeutig wegen Einfachvererbung.
- super(...) ist auch möglich bei Konstruktoren mit Argumenten.
- Man kann in einem Konstruktor auch Konstruktoren der eigenen Klasse aufrufen:

  this(...)

D.h.:

- this, this.... : Zugriff auf das aktuelle Objekt (in nicht-statischer Methode oder Konstruktor)
- this(...) : Aufruf eines anderen Konstruktors der gleichen Klasse (im Konstruktor)
- super.... : später
- super(...) : Aufruf eines Konstruktors der O-Klasse (im Konstruktor)

```
public Stud(...) {
Person p = new Person(....);
    ⋮
}
```

↑ ist nicht das neue Objekt, das vom Stud-Konstruktor erzeugt wird

this(...) oder super(...) muss die erste Anweisung im Konstruktor sein. Wenn sie fehlt, dann ergänzt Java automatisch die 1. Anweisung:

```
super();
```

Wenn man in einer Klasse keinen Konstruktor schreibt, dann wird der parameterlose Konstruktor automatisch ergänzt.

Mögliche Fehlerquelle:
In Person wird ein Konstruktor mit Parametern implementiert, in Stud wird kein Konstruktor implementiert.
⇒ Fehler.

=> Fehler.

Grund: Java erzeugt automatisch den Konstruktor:

```
public Stud () {
    super();
}
```

Vorgehen beim Erzeugen neuer Objekte:

1. Erst werden Attribute auf den Initialwert ihres Typs gesetzt (z.B. 0 bei int, null bei Klassen-Datentypen, ...)
2. Dann werden die initialen Zuweisungen ausgeführt, falls vorhanden.
3. Schließlich wird der Konstruktor ausgeführt.