II.4.1a Unterklassen und Vererbung (Teil 1)

Mittwoch, 30. November 2016 09:30

Ziel

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

---

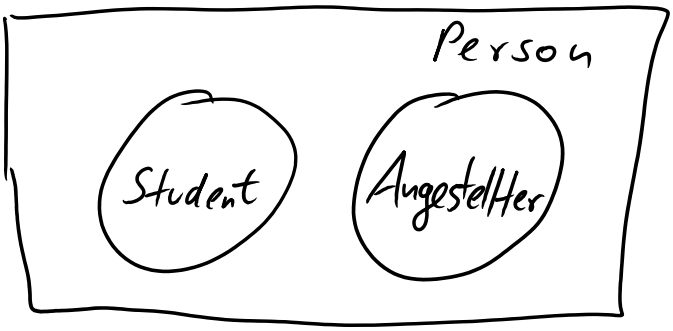
1. Generelles Konzept v. Unter- und Oberklassen

- Student u. Angestellter sind sehr ähnlich.
- Daher sind Methoden wie toString in beiden Klassen

gleich implementiert.
⇒ nicht elegant

- Bislang haben Klassen keine Beziehung zueinander.
- Jetzt: Identifiziere Gemeinsamkeiten ⇒
  Klasse Person
- Studenten und Angestellte sind Spezialfälle von Personen.

⇒
Definiere Student und Angestellter als Unterklassen der Klasse Person ("extends").



- Unterklassen erben alle Eigenschaften der Oberklasse
  ⇒ Jeder Student hat key, ...,
  hashname, toString()
  die in der Oberklasse Person
  deklariert wurden.
- In jeder Unterklasse können weitere Attribute + Methoden definiert sein. Diese können auch auf Eigenschaften aus der Oberklasse zugreifen.

Klasse Student könnte z.B. die folgende Methode enthalten:

```
public void setzeKeyZurueck(){
        key = 0;
}
```

Dann kann man bei jedem Objekt s vom Typ Student

s.setKeyZurueck().

- Datentypanpassung + Zugriff:
  - p=s; ← Zuweisung von Unterklasse zu Oberklasse ✓
    implizite Anpassung vom Speziellen zum allgemeinen Typ (ähnlich wie von int nach double)
    Unterschied: Die speziellen Eigenschaften des Unterklassen-Objekts gehen dabei nicht verloren (im Bsp. matrikelnr).
  - s=a; ↯
    Student u. Angestellter sind "unvergleichbar" (auch wenn sie eine gemeinsame Oberklasse Person haben)
  - p.matrikelnr ↯
    p zeigt i.A. auf ein Objekt

Person, das evtl. keine Matrikelnr hat.
⇒ es hängt vom Typ der Variable/des Ausdrucks ab, auf welche Eigenschaften man zugreifen kann.

- `S=p;` ✓ Zuweisung von Oberklasse zu Unterklasse

  (analog zur Zuweisung eines double-Werts an eine int-Variable)

- `S=(Student) p;`

  gelingt nur, wenn p vorher auf ein Student-Objekt gezeigt hat. Ansonsten: Exception.

  Abhilfe: Verwendung von

"instanceof".

Im Bsp:

p instanceof Student == true

p instanceof Person == true

p instanceof Angestellter == false

(Meist lässt sich instanceof vermeiden.)

- Konzeptionelles Modell für Objekte in Klassenhierarchien:

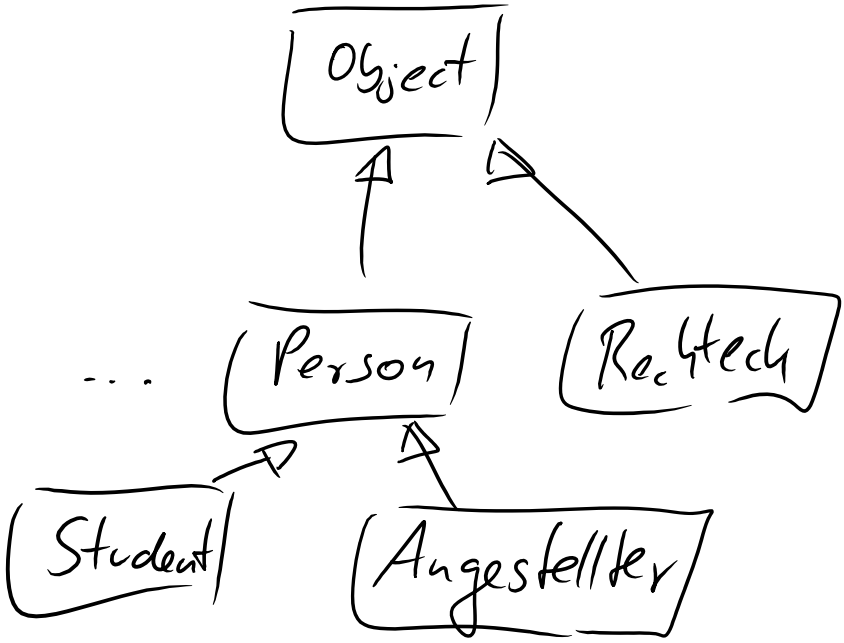
  Jedes Student-Objekt enthält ein Person-Objekt.

  Beim Zugriff über Student-Variable s: Suche erst nach Eigenschaften, die in Unterklasse Student definiert werden, danach in der Oberklasse.

Beim Zugriff über Person-Var.p:
Suche nur nach Eigenschaften,
die in Oberklasse def. wurden.

2. Konstruktoren in Klassenhierarchien

- Aufruf mit new Person().
- In Java gibt es eine Klasse Object, die Oberklasse aller anderen Klassen ist.



- Um ein Objekt der Unterklasse zu erzeugen, will man oftmals

Zunächst ein Objekt der Oberklasse erzeugen u. danach die weiteren Attribute setzen.
=> Schlüsselwort super.
(Zum Aufruf des Konstruktors der direkten Oberklasse).
=> Jede Klasse (außer Object) hat genau eine direkte Oberklasse.
↑ Keine Mehrfachvererbung in Java!

- Es kann mehrere Konstruktoren in der Oberklasse geben. Welcher Konstruktor gewählt wird, hängt von den Typen der Argumente von super ab (Überladung).
- Aufruf von super(...), this(...) nur als erste Anweisung des Konstruktors
- Aufruf eines anderen Konstruktors

der eigenen Klasse mit
this(...).
- Wenn die erste Anweisung
eines Konstruktors weder
this(...) noch super(...)
ist, dann ergänzt Java
automatisch die erste An-
weisung: super();
(Fehlerquelle, wenn die Ober-
klasse keinen parameterlosen
Konstruktor besitzt).