II.4.1a Unterklassen und Vererbung (Teil 1)

Mittwoch, 14. November 2018 09:30

3 Themen:

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

1. Generelles Konzept von Ober- u. Unterklassen

Bsp: Student und Angestellter sind ähnliche Klassen, aber nicht identisch wg. der Attribute matrikelnr und stellung.
Methode toString ist in beiden Klassen gleich implementiert (nicht elegant)

implementiert (nicht elegant).

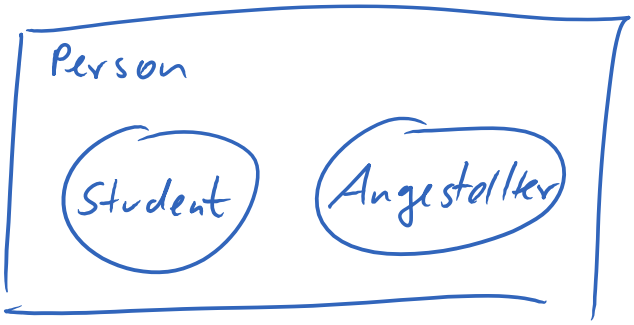
Verbesserung: Setze Klassen zueinander in Beziehung
→ identifiziere identische Teile von Klassen.

Bsp: Neue Klasse Person, die die identischen Teile von Student u. Angestellter zusammenfasst.
Definiere Student und Angestellter als Spezialfälle von Person. (Erweiterungen)
⇒ "extends"

Student u. Angestellter sind jetzt Unterklassen von Person.

D.h: Sie erben alle Eigenschaften der Oberklasse, d.h. alle Attribute und

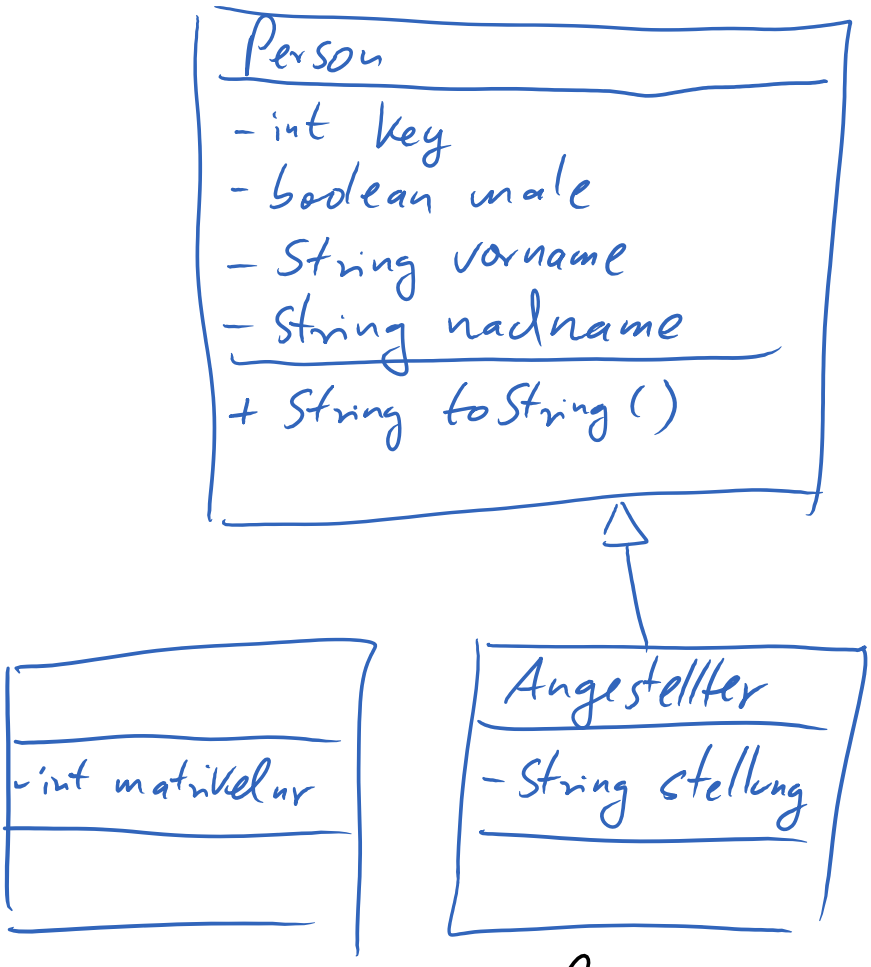
d.h. alle Attribute und Methoden.



Venn-Diagramm

Java hat nur Einfachvererbung, d.h., jede Klasse hat höchstens eine direkte Oberklasse.
Grund: Sonst wäre nicht klar, aus welcher Oberklasse man Eigenschaften erbt, wenn sie in beiden Oberklassen gleich heißen.
(Simulation v. Mehrfachvererbung durch "Interfaces": später)

Klassendiagramm:



Vererbung sowohl für statische als auch für nicht-statische Attribute u. Methoden.

Wenn es in Klasse Person eine statische Methode f gäbe, dann könnte man sie sowohl mit

Person.f(...) als auch mit

Student.f(...) aufrufen.

U-Klassen können zusätzliche Methoden enthalten. Diese dürfen auch auf Attribute der Oberklasse zugreifen.

Bsp:
in Klasse Student:

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

Datentypanpassung + Zugriff

- Zuweisung v. Objekten der U-Klasse an Variablen der Oberklasse: erlaubt, implizit wird aus Objekt der U-Klasse ein Objekt der

U-Klasse.

Sei primitiven Datentypen:

double d = 5;
↑
Typ int

Anders als bei prim. Datentypen gelen die zusätzlichen Informationen des speziellen Typs nicht verloren (im Bsp. die matrikelnr von s).

- Zuweisung zwischen 2 U-Klassen: verboten
(Bsp: Student und Angestellter sind keine U-Klassen voneinander).
- Zugriff auf Eigenschaften von der Oberklasse aus:

s.key und p.key liefert dasselbe

p.matrikelnr: verboten

Kein Zugriff auf Eigenschaften der U-Klasse von Variable der O-Klasse. Grund: Es kann i.A. nicht sicher gestellt werden, ob Var. der O-Klasse auf Objekt der richtigen U-Klasse zeigt.

- Zuweisung v. O-Klasse an Var. der U-Klasse: verboten
  Grund: Man kann zur Compile-Zeit i.A. nicht sicherstellen, dass O-Klassen-objekt zur richtigen U-Klasse gehört.
- Explizite Datentypanpassung von O-Klasse zur U-Klasse möglich: (Student) p

Wenn p vorher auf ein Student-Objekt gezeigt hat, dann gelingt die Datentypanpassung und matrikelnr ist wieder sichtbar.
Ansonsten: Fehler (Exception)

Man kann zur Laufzeit überprüfen, welchen Typ ein Objekt hat: instanceof

(Liefert "true" für alle Klassen, zu denen ein Objekt gehört.

z.B. p instanceof Student } beide
p instanceof Person } true)

Meist kommt man ohne "instanceof" aus.

Es gibt in Java eine Oberklasse Object, die Oberklasse aller Klassen.

⇒ Wenn man bei einer Klasse kein "extends" schreibt, wird automatisch "extends Object" ergänzt.

Konzeptionell

- In jedem Objekt der U-Klasse "steckt" ein der O-Klasse.
- Variable der O-Klasse sieht immer nur den Teil des Objekts, der zur O-Klasse gehört.

2. Erzeugung von

Objekten in Klassen-hierarchien

Wie bisher:

- Aufruf von Konstruktoren mit "new".
- Aufruf des Konstruktors der O-Klasse aus dem Konstruktor der U-Klasse mit super

Ist nicht dasselbe wie

```
new Person()
```

↑ erzeugt ein neues Personen-Objekt, aber dies ist dann verschieden von dem neuen Objekt, das im Konstruktor der U-Klasse gerade erzeugt wird

Super funktioniert auch

bei Konstruktoren mit Argumenten. Überladen wird wie bisher aufgelöst.

Aufruf eines anderen Konstruktors der eigenen Klasse:

this (...).

this. ... Eigenschaft des aktuellen Objekts

this (...) Aufruf eines Konstruktors der eigenen Klasse (nur aus anderem Konstruktor heraus).

super. .... später

super (...) Aufruf eines Konstruktors der O-Klasse (nur

aus Konstr. heraus)

Aufruf this(...) oder super(...) darf nur als 1. Anweisung in einem Konstruktor stehen.

Wenn keine solche Anweisung am Anfang des Konstruktors steht, dann ergänzt Java automatisch die 1. Anweisung

```
super();
```

Fehlerquelle, falls in der O-Klasse kein parameterloser Konstruktor existiert.

Wenn in Person nur ein Konstruktor mit 4 Parametern geschrieben wird und kein Konstruktor in Student

geschrieben wird => Fehler.
Grund: Dann erzeugt Java den Konstruktor

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

Vorgehen beim Erzeugen neuer Objekte:

1. Attribute werden auf Standardwert ihres Typs gesetzt (z.B. 0 bei int, null bei Klassen-Datentypen,..)
2. Attribute werden auf Initialwert gesetzt, falls in Deklaration vorhanden.
3. Konstruktor ausführen (beginnt mit Aufruf eines

anderen Konstruktors)