II.4.1b Unterklassen und Vererbung (Teil 2)

Freitag, 24. November 2017 09:30

3. Verdecken v. Attributen u. Überschreiben v. Methoden

Was passiert bei gleichnamigen Attr. bzw. Methoden in Ober- u. U-Klasse?

Bei Attributen (+ statischen Methoden): Verdecken

Bei nicht-statischen Methoden: Überschreiben

Verdecken v. Attributen

- Bsp: Attribut hochschule sowohl in Kl. Person als auch in Kl. Student

  Bei Person: gibt an, ob Person schon Hochschulabschluss hat

  Bei Student: Name der aktuellen Hochschule

- Attribut der U-Klasse verdeckt das gleichnamige Attribut d. O-Klasse.
  Es hängt v. Typ der Var. ab, über die auf ein Objekt der U-Klasse zugegriffen wird. Sichtbar ist das Attribut, das dem Typ der Var. entspricht.
- Verdecken v. Attributen wird statisch (zur Compile-Zeit) aufgelöst, denn schon zur Compile-Zeit ist klar, welchen Typ Variable habe.
- Fehlerquelle: Ungewolltes Verdecken v. Attributen d. O-Klasse, da Attrib. in U-Klasse unnötigerweise erneut deklariert wurde.

Überschreiben v. Methoden

Methode mahnung.
Bei mahnung in U-Klasse Student/Angestellter müssen noch mehrere zusätzliche Anweisungen ausgeführt werden.

Bei Aufruf der Methode "mahnung" für ein Student-Objekt wird immer die Implementierung aus der U-Klasse Student ausgeführt.

s.mahnung, p.mahnung } führt die gleiche Implem. aus, falls p = s.

Verwendung v. überschr. Methoden:
- Sammlung v. Objekten d. Oberklasse
- Für alle Objekte darin wird eine Methode f aufgerufen.
- Für jedes Objekt wird dann die Methode aus der richtigen U-Klasse ausgeführt.
- Wenn Objekt zu keiner U-Klasse gehört, dann wird die Implem. aus d. Oberklasse ausgeführt.

- Erst zur Laufzeit kann entschieden werden, welche Implementierung einer überschriebenen Methode ausgeführt wird:
  dynamisches Binden bzw. ad hoc Polymorphismus

Polymorphismus: Eine Fkt. kann auf Argumente versch. Typen angewendet werden.

- ad hoc Polymorphismus:
  je nach Argumenttyp wird unterschiedl. Implementierung ausgeführt (typisch für OO: Java, Haskell, ...)
- parametrischer Polymorphismus:
  eine Implem. für Argumente versch. Typen.
  (typisch für fkt. Sprachen: Haskell, Java, ... seit 1.5)

Vorteil v. überschr. Methoden:
- elegante Prog.
- "sende Nachrichten" kann geschrieben werden, bevor

es U-Klassen v. Person gibt.

- Man kann später neue U-Klassen v. Person implementieren und ggf. "mahnung" überschreiben, ohne "sendeMahnungen" zu ändern.

Finale Methoden

- Methoden mit Schlüsselwort final dürfen in U-Klassen nicht überschrieben werden.
- Stellt sicher, dass Nutzer einer Prog-Bibliothek nur diese Implementierung der Methode nutzen können.
- finale Klasse: Klasse ohne Unterklassen
- finale Attribute/Variablen: Wert kann nicht verändert werden.

Anmerkungen zum Überschreiben v. Methoden:

- Eine Methode der Oberklasse darf nur mit Methode in der U-Klasse überschrieben werden wenn Methode der U-Klasse

mindestens so sichtbar ist wie Methode d. O-Klasse.

```
public class Person {
        public void f(){...}
        :
}
public class Student extends Person{
        private void f(){...}
        :
}
```

Verboten!

Sonst:

```
Student s = ...;
Person  p = s;
p.f();
```

- nicht-statische Methode darf nur mit nicht-statischer Methode überschrieben werden (und umgekehrt)

Bei statischen Methoden:
Verdecken statt Überschreiben

Falls Person u. Student beide eine statische Methode f() mit gleicher Signatur haben:

```
Student s = ...;
Person  p = s;
p.f()   ← steht für
```

p.f(); ← steht für Person.f()
Student.f()

Verdecken v. Attributen u. v. statischen Methoden wird zur Compile-Zeit aufgelöst (hängt v. Typ d. Var. ab, über die zugegriffen wird).

Überschreiben v. nicht-statischen Methoden wird zur Laufzeit aufgelöst.

Da Verdecken v. Attributen beim Compilieren aufgelöst wird, darf gleichnamiges Attrib. in O-Klasse statisch sein u. in U-Klasse nicht (oder umgekehrt).

Zugriff auf überschriebene Methode v. U-Klasse aus

- sinnvoll, da die Methode der U-Klasse oft auf den Code der Methode der O-Klasse aufbaut
- super.f(...)

d.h.:
this . das aktuelle Objekt als Objekt der eige-

neu Klasse

super : das aktuelle Objekt als Objekt der direkten Oberklasse

this.f() : führe f für das akt. Objekt aus (als Obj. der eigenen Klasse)

super.f() : ... als Obj. der Oberklasse

this(...) : } analog in Kon-
super(...) : } struktoren

this.x } analog für
super.x } Attribute

Zusammenfassung

1. Überladen von Methoden
2. Verdecken v. Attributen (u. statischen Methoden)
3. Überschreiben v. (nicht-statischen) Methoden

1. Überladen v. Methoden

- Versch. Methoden mit gleichem Namen, aber unterschiedl. Parametern.
- wird zur Compile-Zeit aufgelöst
- Überladen auch in Klassen-

Hierarchien mögl, d.h.
Versch. Methoden können in
Versch. Klassen sein.

p.mahnung (10, 5) } führt die
s.mahnung (10,5) } 2-stellige „mahnung"-Methode aus Person aus

s.mahnung (ges) } führt die
p.mahnung (ges) } 1-stellige „mahnung" aus Student aus.

Bsp: in Klasse Person
void f(int x) {...}

in Klasse Student
void f(double x) {...}

Student s= ....
s.f(5); ← führt f(int..) aus Person aus

Grund: f wird überladen (nicht überschrieben). Daher stehen in Klasse Student sowohl
f(int..) als auch
f(double..) zur Verfügung.

Beim Überladen wird immer die speziellste passende Methode ausgeführt.

Bsp: in anderer Klasse ex.

die Methoden

void f(Person p) {...}

void f(Student s) {...}.

```
Student s = ...;
Person p = s;
f(s);   // führt diese Methode aus
f(p);   // führt diese Methode aus
```

Überladen wird zur Compile-Zeit aufgelöst.

2. Überschreiben v. Methoden

- versch. Methoden mit gleichem Namen u. gl. Parametern in Ober- und Unterklasse.
- welche Methode verwendet wird, hängt v. Typ d. akt. Objekts (zur Laufzeit) ab.

Bsp: toString()

Diese Methode gibt es in Klasse Object. Sie kann in U-Klassen überschrieben werden, muss aber nicht.

3. Verdecken v. Attributen

- v. Typ der Variable, über die

zugegriffen wird, hängt ab, welches Attribut gemeint ist.

Zugriffsspezifikation "protected":

- Zugriff mögl aus allen U-Klassen, auch dann, wenn sie in anderen Paketen liegen.
- geeignet f. Komponenten, die auch in U-Klassen, aber nicht überall sichtbar sein sollen.

Man kann public Attribut mit private Attribut verdecken:

```
public class Person {
    public int x;
    :
}
```

```
public class Student
        extends Person {
    private int x;
    :
}
```

Bei Zugriff auf s.x mit Student s; aus anderer Klasse als Student: Fehler, da Zugriff auf private Variable.