II.4.1b Unterklassen und Vererbung (Teil 2)

Mittwoch, 7. Dezember 2016 08:30

3. Verdecken von Attributen, Überschreiben von Methoden

Was passiert, wenn in der Unterklasse Attribute oder Methoden mit demgleichen Namen wie in der Oberklasse definiert sind?
Hierbei werden Attributen. Methoden unterschiedlich behandelt.

Verdecken von Attributen

Hier hängtes vom Typ der Variable / des Ausdrucks ab, welches Attribut man sieht.

p.hochschule sieht das boolesche Attribut der

Oberklasse Person

s. Hochschule sieht das String-Attribut der Unterklasse Student. Das gleichnamige Attribut aus der Klasse Person ist verdeckt.

- Fehlerquelle: Gleichnamiges Attribut nachname in U-Klasse Student würde auch nachname aus Person verdecken. Verdecken v. Attributen ist oft nicht sinnvoll.

Überschreiben von Methoden

Zentrale Programmiertechnik für Programme mit Klassenhierarchien

Bsp.: Methode mahnung in der Oberklasse Person.

Für Studenten zusätzliche Mitteilung, um Exmatrikulation zu verhindern.
Für Angestellte: Zusätzliche Mitteilung an Besoldungsstelle

Methode der Unterklasse überschreibt die gleichnamige Methode der Oberklasse (override).
D.h.: Egal, ob man auf ein Student-Objekt über s oder p zugreift, beim Aufruf von mahnung wird die Methode aus der Unterklasse Student ausgeführt.

Verwendung überschriebener Methoden:

- Sammlung von Objekten der

- Sammlung von Objekten der Oberklasse (z.B. Person)
- Rufe Methode für jedes dieser Objekte auf.
- Für jedes Objekt wird dann die (speziellste) passende Methode ausgeführt.
- Methode "sendeMahnungen" kann schon implementiert werden, bevor die Unterklassen Student u. Angestellter existieren. Es können auch später neue Unterklassen von "Person" dazukommen, in denen "mahnung" neu überschrieben wird.
  ⇒ Stabilität des Codes

Finale Methoden

- Schlüsselwort "final" bei Methoden: Methode kann

nicht überschrieben werden.

- "final" bei Variablen: Konstanten, Wert kann nicht geändert werden.
- "final" bei Klassen: Klasse kann keine Unterklasse haben.

Regeln beim Überschreiben:

- nicht-statische Methoden nur mit nicht-stat. Meth. überschreiben
- statische Meth. nur mit stat. Meth. überschreiben

Bsp: stat. Methode f()
in Klasse Person und Student

```
Student s;
Person p;
p = s;
Student.f() } führt f
s.f()       } aus Klasse
                Student aus

Person.f() } führt f aus
```

p. 11

- Verdecken v. Attributen + Überschr. v. stat. Methoden wird zur Compilezeit aufgelöst (hängt vom (statischen) Typ des Ausdrucks ab)

Überschreiben von nicht-stat. Methoden kann erst zur Laufzeit aufgelöst werden: ad-hoc Polymorphismus, dynamisches Binden

- Überschreibende Methode muss mindestens so sichtbar sein wie die überschriebene Methode.
  (Bsp: Object hat eine public-Methode toString().
  ⇒ toString() kann nur mit public-Methode überschrieben werden.)

- Verdecken v. Attributen: hier dürfen statische/nicht-stat. Attribute stat/n-s Attr. verdecken.

Zugriff auf überschriebene Methoden f(...) u. verdeckte Attribute aus der U-Klasse:

super.f(...)

↑ "super" ist das aktuelle Objekt aus der Sicht der Oberklasse.
"this" ist das aktuelle Objekt aus der Sicht der aktuellen Klasse.

Ist oft sinnvoll, da die überschreibende Methode ähnlich arbeitet wie Methode der Oberklasse.

Zusammenfassung:

1. Verdecken v. Attributen
2. Überschreiben v. Methoden
3. Überladen v. Methoden

1+3 wird zur Compile-Zeit aufgelöst

2 wird zur Laufzeit aufgelöst

3. Überladen v. Methoden

- Versch. Methoden mit gleichem Namen, aber unterschiedlichen Parametern. Ist auch in Klassenhierarchien möglich.

Bsp: mahnung wird jetzt auch überladen.

2-stellige Methode mahnung ist auch in U-Klasse Student sichtbar (Vererbung).

S. mahnung (10,5).

S. mahnung (ges) } führen beide

p.mahnung(ges) die Methode
mahnung(int ges)
aus der Klasse Student
aus.

Bsp: public class Person {
    void mahnung(int ges){..}
}

public class Student
    extends Person {
    void mahnung(double ges){..}
}

Student s;
s.mahnung(5); führt Methode aus kl. Person aus

Grund: Dies ist kein Überschreiben, da die Signatur der Methode mahnung in den beiden Klassen unterschiedlich.

Dies ist Überladen – es wird daher die speziellste passende

Methode genommen.

Bsp: class C {...

:

void f(Person p) {...}

void f(Student s) {...}

:

}

Student s;

Person p;

p=s;

f(s); ⇐ führt diese Methode aus

f(p); ⇐ führt diese Methode aus

Bsp:

void mahnung(int geb){...}

in Person

int mahnung(int geb) {...}

in U-Klasse Student

ist nicht erlaubt

Generell: Gleicher Name für versch. Methoden oder

Attribute / Variablen nur dann wenn sie gleiches Konzept repräsentieren.

Protected:

```
public class A {....
    protected int value;
        ⋮
}
```

in anderem Paket

```
public class B extends A {
        ⋮

    B b;
    b.value ← zugreifbar in Klasse B
        ⋮
}
```