II.4.1b Unterklassen und Vererbung (Teil 2)

Montag, 19. November 2018 11:30

3. Verdecken von Attributen und Überschreiben von Methoden

Attribute + statische Methoden:
verdecken

nicht-statische Methoden:
überschreiben

Was passiert, wenn Attribute bzw. Methoden in O-Klasse und U-Klasse gleich heißen?

Attribute:

Ein namensgleiches Attribut der U-Klasse verdeckt das Attribut der O-Klasse.

Dies würde auch passieren, wenn die Attribute in O-Klasse und U-Klasse den gleichen Typ haben (⇒ Fehlerquelle). Auf welches Attribut zugegriffen wird, hängt am Typ der Variable, über die zugegriffen wird.

s. hodschule

p. hodschule

greifen auf unterschiedl. Attribute zu, obwohl s und p beide auf das gleiche Student-Objekt zeigen. Aber s hat Typ Student, p hat den Typ Person.

⇒ wird zur Compilezeit (statisch) aufgelöst.

Überschreiben von nicht-statischen Methoden

Bsp: Mahnung wird in Klassen Person, Student, Angestellter leicht unterschiedlich implementiert.

S.mahnung(10) } führt Methode aus Klasse Student aus
P.mahnung(10) }

S und P sind Var. v. Typ Student bzw. Person, zeigen beide auf ein Student-Objekt

Überschreiben v. Methoden wird erst zur Laufzeit aufgelöst, denn es hängt vom

Typ des Objekts zur Laufzeit ab, welche Methode ausgeführt wird.
(overriding = Überschreiben)

Typische Verwendung:

- Sammlung von Objekten der Oberklasse, manche davon sind auch Objekte von U-Klassen
- Rufe Methode f für alle Objekte in der Sammlung.
- Für jedes Objekt wird jetzt die Methode aus der passenden Klasse ausgeführt.
- Ist viel eleganter als eine Fallunterscheidung nach den einzelnen Typen (kein "instanceof" nötig).

- Methode der Oberklasse kann implementiert werden, bevor es entspr. U-Klassen gibt bzw. man kann später neue U-Klassen hinzufügen.

Bsp: ergänze weitere U-Klasse "Gast". Wenn man darin Mahnung nicht überschreibt, dann wird bei p.mahnung(gebühr) die Methode aus "Person" ausgeführt, wenn p ein Gast ist. Aber man kann "mahnung" in "Gast" auch beliebig überschreiben.

dynamisches Binden / ad-hoc Polymorphismus

Polymorphismus = Vielgestaltigkeit

Eine Methode kann für Objekte/Argumente verschiedener Typen verwendet werden.

ad hoc Polymorphismus:

Je nach Typ wird eine andere Implementierung ausgeführt.

(typisch für OO-Sprachen)

parametrischer Polymorphismus:

Eine Implementierung für Argumente verschiedener Typen.

(typisch für funktionale Sprachen)

In Java gibt es (seit

Java 5) Beide Formen des Polymorphismus.

- Überschreibende Methode muss mindestens so sichtbar sein wie die überschriebene Methode der Oberklasse.

Bsp: f ist public in Person
f ist private in Student

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

← würde auf die private Student-Methode zugreifen

⇒ verboten

- statische/nicht-stat. Methoden dürfen nur mit stat/nicht-stat. Methoden

"Überschrieben" werden

- Bei statischen Methoden findet Verdecken statt Überschreiben statt.

Bsp: Person u. Student haben beide eine statische Methode

```
static void f() {...}

Student s=...;
Person p=...;
p=s;
Student.f(..);
Person.f(..);
s.f(..);   <- ruft Student.f(..) auf
p.f(..);   <- ruft Person.f(..) auf
```

- Beim Verdecken von Attributen ist die Sichtbarkeit und

die Auswahl static/nicht-static beliebig.

Manchmal möchte man sicherstellen, dass Methode nicht überschrieben werden kann: final

- final bei Methoden: dürfen nicht überschrieben werden
- final bei Klassen: darf keine U-Klassen haben
- final bei Attributen: Wert darf nicht verändert werden (Konstante)

Zugriff auf überschriebene Methode aus der U-Klasse

- sinnvoll, da die überschreibende Methode ähnlich zur

überschriebenen Methode

- super.methode(...)

Bedeutungen von this/super:

- this.... : Zugriff auf Eigenschaften des aktuellen Objekts

  super.... : Zugr. auf Eigenschaften des akt. Objekts aus Sicht der Oberklasse

- this(...) ← Aufruf eines Konstruktors der eigenen Klasse
  super(...) bzw. der Oberklasse (aus Konstr. heraus)

Durch super... kann man auf analoge Weise auf verdeckte Attribute der Oberklasse zugreifen.

Zusammenfassung:

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

1. Überladen

- Verschiedene Methoden mit gleichem Namen, aber unterschiedlicher Parameteranzahl oder -typ.
- Wird vom Compiler aufgelöst, indem die speziellste passende Methode genommen wird.
- Überladen auch in Klassenhierarchien möglich, versch. Methoden können in Ober- und U-Klasse stehen.

Bsp: mahnung wird überladen

p.mahnung(10,5) oder
s.mahnung(10,5)

führt beides die 2-stellige "mahnung"-Methode aus "Person" aus (wird an Student vererbt)

p.mahnung(10)
s.mahnung(10)

} führt beides die Methode aus "Student" aus, da sie die entspr. "mahnung"-Methode aus Person überschreibt.

Bsp:

in Person:

```
void mahnung(int geb){...}
```

in Student

```
void mahnung(double geb){...}
```

```
p=s;
```

```
p = s;
p.mahnung(10);
s.mahnung(10);
```

führt beides die Methode aus "Person" aus.

Überschreiben findet nur statt, wenn die Typen der Parameter gleich sind.
Sonst ist es Überladen und die spezifischste passende Methode wird ausgeführt.

Bsp: in anderer Klasse

① `void f(Person p) {...}`
② `void f(Student s) {...}`

```
Student s = new Student();
Person p = s;
```

f(s); ← führt ② aus

f(p); ← führt ① aus

2. Überschreiben

- bei gleichen Parametertypen
- hängt vom Typ des Objekts zur Laufzeit ab
- Bsp: public String toString() {...} ist schon in Object vorhanden.

Kann in jeder U-Klasse überschrieben werden.

3. Verdecken

- wird zur Compile-Zeit aufgelöst
- hängt vom Typ der Var. ab

protected: Zugreifbar

auch in U-Klassen, die
in anderen Paketen liegen

⇒ zum Kapseln benutzen,
wenn Komponente in U-Klassen
gebraucht wird