Einstieg in die Informatik mit Java

Vererbung

Gerd Bohlender

Institut für Angewandte und Numerische Mathematik

Gliederung

1. Grundlagen
2. Verdeckte Variablen
3. Verdeckte Methoden
4. Konstruktoren und Vererbung
5. Klassen und `final`
6. Zugriffsrechte und Vererbung
7. Abstrakte Methoden und Klassen

Gliederung

1. Grundlagen
2. Verdeckte Variablen
3. Verdeckte Methoden
4. Konstruktoren und Vererbung
5. Klassen und `final`
6. Zugriffsrechte und Vererbung
7. Abstrakte Methoden und Klassen

Grundlagen

Durch *Vererbung* werden von einer allgemeinen Klasse (*Superklasse*, *Basisklasse*) spezielle Klassen (*Subklassen*, *abgeleitete Klassen*) abgeleitet. Diese erben die Komponenten der Superklasse und besitzen ggfs. weitere Methoden und Variablen.

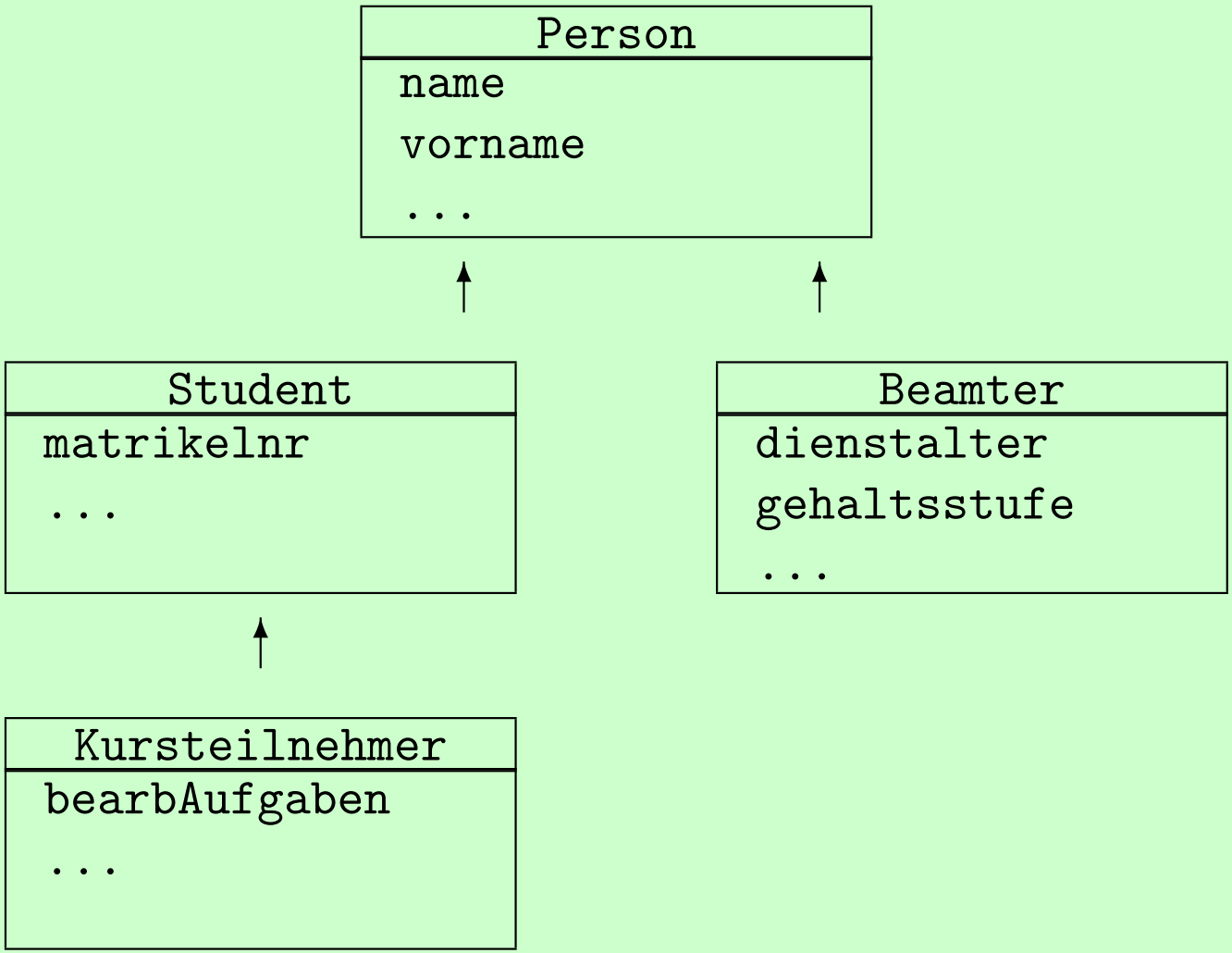
Syntax

```
class Subklasse extends Superklasse {
    // eigene Methoden und Variablen
}
```

Grundlagen

- Im Gegensatz zu C++ ist keine *Mehrfachvererbung* möglich, d.h. die Subklasse kann nur von einer Superklasse abgeleitet werden. Allerdings ist bei *Schnittstellen* Mehrfachvererbung wieder erlaubt.
- Es darf über mehrere Stufen hinweg vererbt werden. Man spricht dann von *indirekten* Sub– und Superklassen, andernfalls von *direkten*.
- Alle Klassen ohne Angabe von `extends` sind abgeleitet von der Superklasse `Object`.
- Konstruktoren, `static`–Initialisierer und `private`–Komponenten werden nicht vererbt.

Beispiel


Person
name
vorname
...
Student
matrikelnr
...
Beamter
dienstalter
gehaltsstufe
...
Kursteilnehmer
bearbAufgaben
...

Gliederung

1. Grundlagen
2. Verdeckte Variablen
3. Verdeckte Methoden
4. Konstruktoren und Vererbung
5. Klassen und `final`
6. Zugriffsrechte und Vererbung
7. Abstrakte Methoden und Klassen

Verdeckte Variablen

Neben den in Kapitel `"OOP"` schon erwähnten Modifizierern existiert zusätzlich der Modifizierer `protected`.
Der Zugriff auf die entsprechende Komponente ist dabei im gleichen Paket (d.h. allgemein im gleichen Verzeichnis) und in allen Subklassen erlaubt.

Achtung

Zugriffsrechte beziehen sich auf die Klasse, nicht auf die Instanzen und werden zur Übersetzungszeit geprüft, nicht erst zur Laufzeit!

Verdeckte Variablen

Gleichnamige Variablen in Sub– und Superklasse mit gleichem oder verschiedenem Typ sind erlaubt. Die Variablen aus der Superklasse bleiben erhalten, sie werden lediglich verdeckt, d.h. der direkte Zugriff aus der Subklasse heraus ist nicht möglich. Abhilfe schafft der Zugriff mit...

Syntax

Klassenname`.`*Variablenname*	bei Klassenvariablen
`super.`*Variablenname*	bei Instanzvariablen und direkter Superklasse
`((`*Superklasse*`)` *Instanz*`).`*Variablenname*	in beiden Fällen und auch bei indirekter Superklasse

Achtung

Zwischen Methodennamen und Variablennamen besteht kein Problem. Diese dürfen sowohl innerhalb einer Klasse als auch in Sub– und Superklasse den gleichen Namen tragen!

Verdeckte Variablen, Beispiel

Beispiel

```
class Super {
      int x = 1;
      int x() {
            return 2;
      }
}

class Sub extends Super {
      int x = 3;
      int x() {
          return 4;
      }
      void info() {
          System.out.println(""+ x + x() + super.x + super.x());
      }

      public static void main (String s){
            Sub instanz = new Sub();
            instanz.info(); // ergibt 3412
      }
}
```

Verdeckte Variablen, Beispiel

Achtung

Anstatt `super.x` kann man auch `((Super) this).x` verwenden!

Gliederung

1. Grundlagen
2. Verdeckte Variablen
3. Verdeckte Methoden
4. Konstruktoren und Vererbung
5. Klassen und `final`
6. Zugriffsrechte und Vererbung
7. Abstrakte Methoden und Klassen

Verdeckte Methoden

- Methoden können auch im Zusammenhang mit Vererbung *überladen* werden, d.h. sie tragen den gleichen Namen, aber eine andere Signatur.
- Hat in der Subklasse eine Methode den gleichen Namen und die gleiche Signatur wie in der Superklasse, so wird bei einer Objektreferenz automatisch die Methode der Subklasse verwendet (*Polymorphie*). Der Zugriff auf die Methode der Superklasse erfolgt dann über

  `super.`*Methodenname*

Achtung

Außerhalb der Subklasse ist ein Zugriff nicht möglich, da der Typ von Klassen dynamisch zur Laufzeit bestimmt wird und somit eine Referenz der Superklasse auch auf die Subklasse verweisen darf!

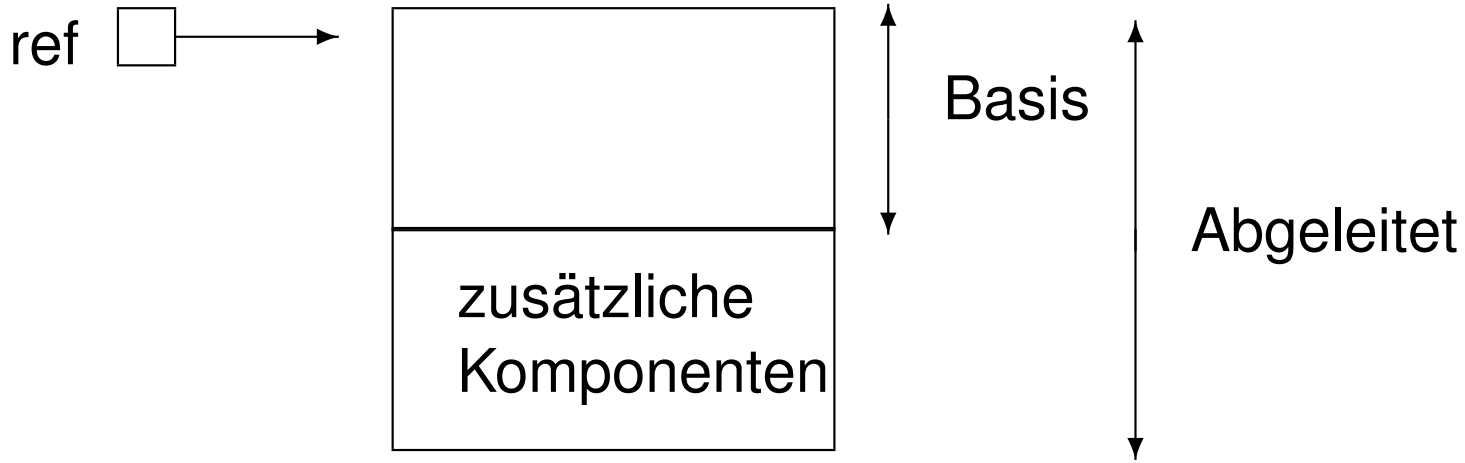
Verdeckte Methoden

Beispiel

```
class Basis {
      void info() {
            System.out.println ("Basis");
      }
}
class Abgeleitet extends Basis {
      void info() {
            System.out.println ("abgeleitet");
      }

      public static void main (String s){
            Abgeleitet a = new Abgeleitet();
            a.info(); // abgeleitet
            ((Basis)a).info(); // auch: abgeleitet
            Abgeleitet b = new Basis(); // Fehler!
            Basis b = new Abgeleitet(); // ok!
            b.info(); // abgeleitet
      }
}
```

Konversionen



Damit gelten offenbar die folgenden Regeln für Konversionen zwischen Basisklasse und abgeleiteter Klasse:

(1) Ein Objekt der abgeleiteten Klasse kann automatisch in ein Objekt der Basisklasse konvertiert werden, da es alle notwendigen Komponenten enthält (*Up–Cast*).

(2) Dagegen kann ein Objekt der Basisklasse nicht automatisch in ein Objekt der abgeleiteten Klasse konvertiert werden, da Komponenten fehlen.
Ist es aber “in Wirklichkeit” ein Objekt der abgeleiteten Klasse, dann geht eine explizite Konversion mittels eines *Down–Cast*.

Konversionen, Beispiel

Beispiel

```
// Konversion von Basis nach Abgeleitet:
Basis b = new Basis();
Abgeleitet a = b; // Fehler!
Abgeleitet a = (Abgeleitet) b; // Fehler!

// Konversion von Abgeleitet nach Basis:
Abgeleitet a = new Abgeleitet();
Basis b = a; // okay,
// implizite Konversion der Referenz

Abgeleitet c = (Abgeleitet) b; // okay,
// b referenziert Objekt vom Typ Abgeleitet
```

Polymorphe Methoden, Beispiel

Beispiel

```
class Basis {
     void info() {
          System.out.println("Basis");
     }
}

class Abgeleitet extends Basis {
     void info() {
          System.out.println("abgeleitet");
     }
}

class Haupt {
     public void static main (String [] args) {
          Basis [] a = {new Abgeleitet(),
                        new Basis(),
                        new Abgeleitet() };

          for (int i=0; i<a.length; i++){
               a[i].info(); // ergibt: abgeleitet Basis abgeleitet
          } //
     }
}
```

Polymorphie

Polymorphie

Der Typ der Klasse, zu der eine Instanzmethode gehört (im obigen Beispiel die Methode `info()`), wird erst zur Laufzeit bestimmt (*späte Bindung, Polymorphie*).

Keine Polymorphie

Bei Methoden mit den Wortsymbolen `final`, `static` oder `private` erfolgt die Bestimmung bereits zur Kompilerlaufzeit (*frühe Bindung*) $\longrightarrow$ keine Polymorphie.

Gliederung

1. Grundlagen
2. Verdeckte Variablen
3. Verdeckte Methoden
4. Konstruktoren und Vererbung
5. Klassen und `final`
6. Zugriffsrechte und Vererbung
7. Abstrakte Methoden und Klassen

Konstruktoren und Vererbung

Konstruktoren werden nicht vererbt, allerdings kann der Konstruktor der direkten Superklasse aufgerufen werden.

Syntax

```
super(Parameterliste);
```

- Der Aufruf muss als erste Anweisung im Konstruktor der Subklasse stehen.
- Ohne expliziten Aufruf wird am Anfang der Standardkonstruktor der Superklasse aufgerufen, d.h. der Konstruktor der Subklasse wird ergänzt um `super();` Das bedeutet, dass jeder Konstruktor implizit mit dem Aufruf von `super()` beginnt. Bei der Superklasse wird der Konstruktor der Klasse `Object` aufgerufen.

Konstruktoren und Vererbung, Beispiel

Beispiel

```
class Basis {
      int x;
      Basis() { // Standardkonstruktor
            x = 5;
      }
}
class Abgeleitet extends Basis {
      int y;
      Abgeleitet() { // Standardkonstruktor
            super(); // Standardkonstruktor der
            y = 6; // Superklasse
      }

      public static void main(String s){
            Abgeleitet a = new Abgeleitet();
            System.out.println (a.x); // ergibt 5
            System.out.println (a.y); // ergibt 6
      }
}
```

Konstruktoren und Vererbung, Achtung

Achtung

Nicht erlaubt ist der Aufruf der Konstruktors der Superklasse innerhalb von bedingten Anweisungen, da dann der Aufruf nicht an erster Stelle erfolgt!

Konstruktoren und Vererbung, Beispiel

Beispiel

Ablauf der Instanziierung:

```
class Vater {
      int x = 1;
}
class Sohn extends Vater {
      int y = 2;
}
...
Sohn s = new Sohn();
```

Konstruktoren und Vererbung

(1) Die Referenz wird angelegt und mit `null` initialisiert.

(2) Es wird Speicher für zwei Werte vom Typ `int` auf dem *Heap* (Freispeicher) beschafft.

(3) Die Komponenten `x` und `y` werden mit 0 initialisiert.

(4) Den Komponenten `x` und `y` werden die Werte 1 bzw. 2 zugewiesen, und es werden ggfs. weitere Initialisiererblöcke ausgeführt.

(5) Der Konstruktor (hier: Standardkonstruktor) der Klasse `Sohn` wird aufgerufen. Dieser ruft als erstes den Standardkonstruktor der Basisklasse `Vater` und der wiederum den Standardkonstruktor der Klasse `Object` auf.

(6) Die Variable `s` erhält eine Referenz auf das neu erzeugte Objekt.

Gliederung

1. Grundlagen
2. Verdeckte Variablen
3. Verdeckte Methoden
4. Konstruktoren und Vererbung
5. Klassen und `final`
6. Zugriffsrechte und Vererbung
7. Abstrakte Methoden und Klassen

Klassen und `final`

Die Angabe von `final` ist bei Klassen, Methoden und Instanzen möglich.
Dies ist bei Klassen und Methoden sinnvoll, die nicht weiter spezialisiert werden können. Es erhöht die Effizienz, da eine frühe Bindung erfolgen kann. Das bedeutet, dass Methoden eventuell auch *inline* erzeugt werden (kein Aufruf, keine Parameterübergabe usw.).
Desweiteren bietet es einen Schutz vor *trojanischen Pferden*, die die Funktionalität der Klasse ersetzen könnten.

(1) Klassen mit dem Wortsymbol `final` dürfen nicht vererbt werden.

(2) Methoden mit dem Wortsymbol `final` dürfen nicht vererbt werden, d.h. bei einer Methode mit gleicher Signatur in der abgeleiteten Klasse tritt ein Fehler auf, dagegen ist eine Methode mit einer anderen Signatur erlaubt.

(3) Instanzen mit dem Wortsymbol `final` sind konstant und dürfen folglich nicht verändert werden.

Klassen und `final`

Achtung

Die Komponenten der Instanz sind damit allerdings nicht automatisch konstant!

Klassen und `final`

Beispiel

```
class Punkt {
    int x, y;
    final void verschiebe (int dx, int dy) {
        x += dx; y += dy;
    }
}

class FarbPunkt extends Punkt {
    Color c;
    void verschiebe (Punkt p) { // ok!
        x += p.x; y += p.y; // andere Signatur
    }
    void verschiebe (int dx, int dy) { // Fehler!
        x += dx; y += dy; // gleiche Signatur
    }

    public static void main(String s){
        final Punkt p = new Punkt ();
        p = new Punkt (); // Fehler!
        p = null; // Fehler!
        p.x = 7; // erlaubt!
    }
}
```

Gliederung

1. Grundlagen
2. Verdeckte Variablen
3. Verdeckte Methoden
4. Konstruktoren und Vererbung
5. Klassen und `final`
6. Zugriffsrechte und Vererbung
7. Abstrakte Methoden und Klassen

Zugriffsrechte und Vererbung

Die Zugriffsrechte von Methoden dürfen beim Überschreiben in der abgeleiteten Klasse nicht reduziert werden. Die Reihenfolge lautet dabei:

`private` $\longrightarrow$ `default` $\longrightarrow$ `protected` $\longrightarrow$ `public`

Achtung

Die Umkehrung der Reihenfolge ist verboten, da aufgrund der Polymorphie ein Objekt der Basisklasse in Wirklichkeit zur abgeleiteten Klasse gehören kann! In diesem Fall müssen polymorphe Methoden zugreifbar bleiben!
Bei Variablen und überladenen Methoden existiert diese Einschränkung nicht!

Zugriffsrechte und Vererbung, Beispiel

Beispiel

```
class Mutter {
      private void a() {
      ...
      }
      void b() {
      ...
      }
      protected void c() {
      ...
      }
      public void d() {
      ...
      }
}

class Tochter extends Mutter {
       // Ueberschreiben von Methoden der Superklasse Mutter
}
```

Zugriffsrechte und Vererbung

Folgende Varianten sind beim Überschreiben der Methoden aus der Superklasse `Mutter` möglich:

- `a()`: kein Überschreiben möglich, allerdings ist eine Neudefinition mit beliebigen Rechten möglich.
- `b()`: Überschreiben mit `default`, `protected` oder `public` möglich.
- `c()`: Überschreiben mit `protected` oder `public` möglich.
- `d()`: Überschreiben mit `public` möglich.

Gliederung

1. Grundlagen
2. Verdeckte Variablen
3. Verdeckte Methoden
4. Konstruktoren und Vererbung
5. Klassen und `final`
6. Zugriffsrechte und Vererbung
7. Abstrakte Methoden und Klassen

Abstrakte Methoden und Klassen

Syntax

```
abstract class Klassenname {
    abstract Modifizierer Ergebnistyp Methodenname (...);
}
```

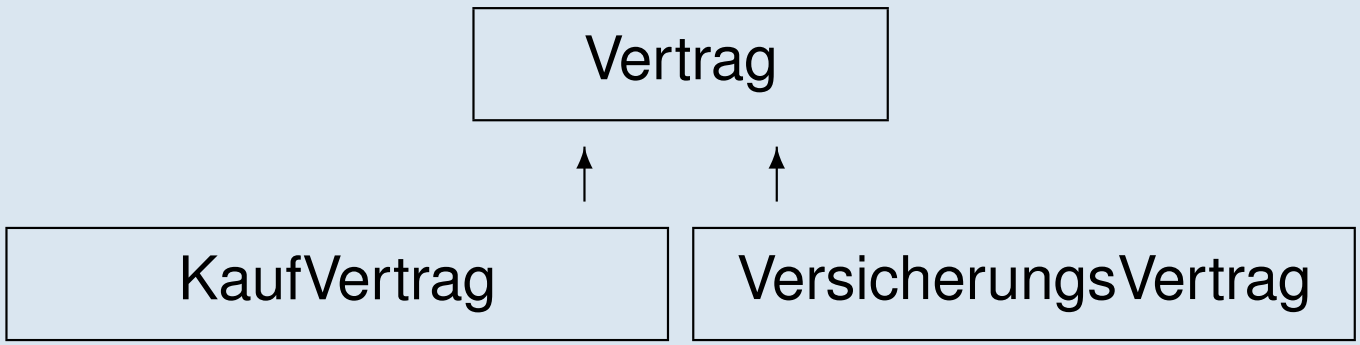
Achtung

Abstrakte Methoden dürfen keinen Rumpf besitzen!

- Ist eine Methode einer Klasse abstrakt, so muss auch die Klasse selber abstrakt sein.
- Von abstrakten Klassen dürfen keine Instanzen gebildet werden. Die Klassen dienen nur zur Vorlage bei der Vererbung.
- Die abgeleitete Klasse kann abstrakt sein, muss aber nicht, falls alle abstrakten Methoden durch konkrete Implementierungen überschrieben werden.

Abstrakte Methoden und Klassen

Beispiel



Eine abstrakte Klasse kann auch als *Protokollklasse* aufgefaßt werden, sie definiert nur ein *Interface*, keine Implementierung.

Ähnlich zu abstrakten Klassen verhalten sich die beiden folgenden Fälle:

- Sind alle Konstruktoren `private`, so können ebenfalls keine Instanzen gebildet werden
- Schnittstellen (*Interfaces*)