Einstieg in die Informatik mit Java

Objektorientierte Programmierung und Klassen

Gerd Bohlender

Institut für Angewandte und Numerische Mathematik

Gliederung

Gliederung

Die Philosophie

Paradigmen

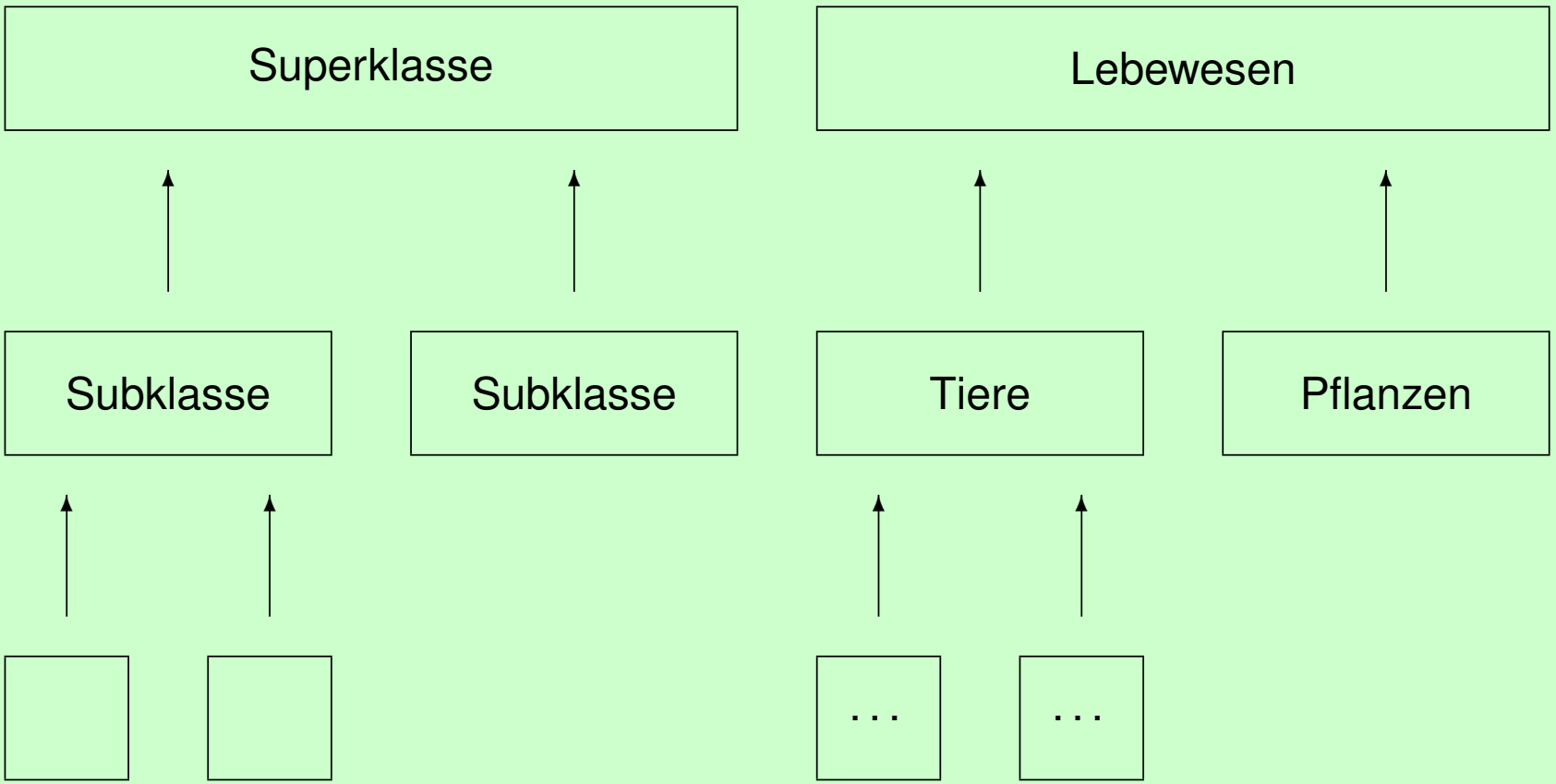
(1) *Generalisierung*: Gemeinsame Strukturen von Objekten werden in Superklassen zusammengefasst.
(2) *Vererbung*: Eigenschaften einer Superklasse werden automatisch an eine Subklasse vererbt.
(3) *Daten-Kapselung*: Variablen und Methoden werden in einem Objekt zusammengefasst. Die interne Struktur wird vor dem Benutzer versteckt, der Zugriff erfolgt nur über genau definierte Schnittstellen.
(4) *Polymorphie*: Eine Methode kann in verschiedenen Subklassen individuell angepasst werden.

Vorteile

- Es entsteht weniger Quelltext.
- Es bestehen weniger Fehlermöglichkeiten.
- Das Programm erhält eine bessere Strukturierung.
- Der Quelltext ist besser wartbar.

Die Philosophie

Beispiel



Gliederung

Definition von Klassen

Syntax

Modifizierer `class` *Bezeichner* `{`*Elementliste*`}`

Die Elementliste besteht aus *Datenelementen* (d. h. Variablen) und *Methoden* (d.h. Funktionen), die diese Daten berarbeiten (vgl. in Pascal: `record` und in C/C++: `struct`, `class`).

Zur Datenkapselung kann die Sichtbarkeit der Elemente mit Hilfe von vorangestellten *Modifizierern* geregelt werden (siehe nächster Abschnitt).

Gliederung

Datenkapselung

Mögliche Modifizierer

nichts	Standard-Rechte, auch *friendly* genannt. Element kann u. a. in der Klasse selber sowie in allen Klassen des gleichen Pakets verwendet werden.
`public`	Element kann von jeder Klasse verwendet werden.
`private`	Element kann nur von der eigenen Klasse verwendet werden.
`protected`	später ...

- Datenelemente werden aus Gründen der Datenkapselung meistens mit dem Modifizierer `private` versehen.
- Die Klasse selber darf nur Standardrechte oder `public`–Rechte besitzen.
- In einer Datei dürfen mehrere Klassen definiert werden, jedoch darf nur eine davon `public`–Rechte besitzen. Diese Klasse muss den gleichen Namen wie die Datei tragen.

Beispiel zur Klassendefinition und Datenkapselung

Nachfolgend wird der neue Datentyp Point definiert:

```
class Point {
    private double x,y;     // geschützt vor unberecht. Zugriff
    public double getx () {
        return x;
    }
    public double gety () {
        return y;
    }
    void move (double dx, double dy) {
        x += dx; y += dy;
    }

    public static void main (String[] args) {
        Point p = new Point ();  // bildet Instanz der Klasse
    }                            // Point durch Aufruf des
}                                // Standardkonstruktors
                                 // --> siehe Kapitel 11.4
```

Achtung

Unterscheidung zwischen Klasse und Instanz!

Gliederung

Instanzen

Klasse Datentyp,
es wird noch kein Speicher für Komponenten reserviert.

Instanz Variable des Datentyps, auch Objekt genannt
es wird für Komponenten Speicher reserviert
(→ *Instanzvariablen*, *Instanz–Methoden*).

Von einer Klasse können viele Instanzen (bzw. Objekte) gebildet werden. Die Instanzen sind dann zwar vom selben Datentyp, belegen aber nicht die selben Speicherbereiche. Dadurch sind sie voneinander unabhängig handhabbar.
Eine Ausnahme bilden Komponenten mit dem Modifizierer `static`. Diese werden für die gesamte Klasse nur einmal angelegt, unabhängig davon, ob keine, eine oder mehrere Instanz(en) der Klasse gebildet wurden (→ *Klassenvariablen*, *Klassenmethoden*).

Achtung

Im Gegensatz zu C++ ist keine separate Deklaration und Definition der Methoden möglich!

Gliederung

Zugriff auf Elemente

Erinnerung: Syntax

Modifizierer `class` *Bezeichner* `{`*Elementliste*`}`

Elementvariablen und Elementmethoden werden auch unter dem Begriff *Komponenten* zusammengefasst.
Innerhalb der Klasse ist der Zugriff auf alle Komponenten erlaubt, und zwar einfach durch Angabe des entsprechenden Bezeichners. Wird der Bezeichner durch eine gleichnamige lokale Variable verdeckt, kann mit

`this.`*Bezeichner*

auf die Komponenten der gerade aktiven Instanz zugegriffen werden. `this` ist dabei eine Referenz auf die momentan betrachtete Instanz.

Achtung

`this` ist nur für Instanz-, nicht für statische Variablen und Methoden zulässig, da es von ihnen nicht mehrere zu unterscheidende Werte geben kann.

Zugriff auf Elemente

Beispiel zu `this`

```
public class Point2{
    private double x,y;              // Instanzvariablen
    public double getx () {
        return x;
    }
    public double gety () {
        return y;
    }
    void move(double x, double y){
        this.x = x;
// Ueberschreiben der Inst.var.
        this.y = y;
// mit den gleichnamigen Var. x,y
     }

    public static void main (String s){
          Point2 p2 = new Point2();  // Instanz erzeugen
    }
}
```

Zugriff auf Elemente

Außerhalb der Klasse, d.h. in einer anderen Klasse, ist der Zugriff nur auf sichtbare Komponenten erlaubt. Diese sind folglich mit öffentlichen Zugriffsrechten bzw. für den Fall, dass sich die Klasse im gleichen Verzeichnis befindet, mit Standardrechten ausgestattet.

Syntax

Klassenname.*Komponenten*	bei Klassenkomponenten
Instanzname.*Komponenten*	bei Instanzkomponenten

Beispiel (erweiterte Klasse `Point2`)

```
...                                          // im Hauptprogramm:
System.out.println ("x-Koodinate von p" + p.getx());
System.out.println (p.x);
// syntaktisch richtig,
...
// aber Zugriff verboten!
```

Gliederung

Konstruktoren

Beim letzten Beispiel gab es die Möglichkeit, Werte für die Variablen `x` und `y` über die Methode `move` vorzugeben.

Beispiel

```
...                                  // im Hauptprogramm:
Point2 p2 = new Point2();            // erzeuge Instanz
p2.move(1000, 1000);
// belegt x,y mit 1000
...
```

Achtung

Nachteil: Die Methode `move` muss für jedes neue Objekt aufgerufen werden! Besser ist die Verwendung eines *Konstruktors*.

Konstruktoren

- Konstruktoren erlauben es sehr komfortabel Instanzvariablen schon bei der Erzeugung einer Instanz/eines Objektes zu *initialisieren*.
- Ein Konstruktor ist eine spezielle Methode ohne Ergebnistyp (auch nicht `void`), die den gleichen Namen wie die Klasse trägt.
- Konstruktoren können *überladen* werden, d.h. es dürfen mehrere Konstuktoren existieren.
- Konstruktoren ohne Argumente werden *Standardkonstruktoren* genannt. Werden keine Konstruktoren definiert, so erzeugt der Kompiler automatisch den Standardkonstruktor, andernfalls nicht.
- Konstruktoren können gegenseitig mittels `this` aufgerufen werden. Dieser Aufruf ist allerdings in jedem Konstruktor nur als erste Anweisung erlaubt.

Konstruktoren

Beispiel

```
public Point (double x, double y) { // 1.Konstruktor
    this.x = x;
    this.y = y;
}
public Point () { // 2.Konstruktor = Standardkonstruktor
    x = 0; // alternativ ueber Aufruf
    y = 0; // des 1.Konstruktors: this (0,0);
}
// im Hauptprogramm:
public static void main (String[] args) {
Point q = new Point (1,2);    // Aufruf des 1.Konstr.
Point r = new Point ();       // Aufruf des 2.Konstr.
}
```

Mittels `new` wird der benötigte Speicher beschafft, und die angegebenen Anweisungen im Konstruktor ausgeführt (evtl. noch weitere Operationen).

Gliederung

Garbage Collection

Im Gegensatz zu Pascal (`new`, `delete`) und C++ (`new`, `delete`, Konstruktor, Destruktor) gibt es in Java im Wesentlichen kein Gegenstück zum Konstruktor. Der Speicher von nicht mehr referenzierten Objekten wird früher oder später automatisch freigegeben, im schlimmsten Fall erst am Programmende → *Garbage collection*.

Syntax

```
System.gc();
```

Beispiel

```
String s = "blub";
...
s = null; // Ref. auf "blub" geht verloren,
// sofern keine weitere Referenz ex.
```

Garbage Collection

Beispiel

```
class Demo { } ...
Demo d = new Demo(); //Instanz der Klasse Demo
...
d = new Demo(); //neue Instanz, alte geht verloren
```

Achtung

Selbst beim expliziten Aufruf des *Garbage collectors* muss der Speicher nicht komplett freigegeben werden!

`finalize`

Sind weitere Aufräumarbeiten wie das Zählen lebender Objekte, Abbau der Internetverbindung usw. nötig, dann kann man eine Methode `finalize()` definieren. Diese wird aufgerufen, bevor ein Objekt vom *Garbage collector* freigegeben wurde.

Syntax

```
protected void finalize () throws Throwable {
...
}
```

Lebenszyklen von Klassen und Instanzen

(1) Ein Klasse wird „geladen“ (z. B. von der Festplatte in den Hauptspeicher), sobald sie benötigt wird und wieder entfernt, sobald sie nicht mehr benötigt wird.

(2) Instanzen werden i. allg. mit `new` erzeugt. Sobald keine Referenz mehr auf die Instanz existiert, kann sie vom Java–System über die automatische *Garbage collection* wieder entfernt werden. Dies kann allerdings später oder auch gar nicht geschehen (siehe oben).