Programmieren I

Dr. Klaus Höppner

Hochschule Darmstadt – Wintersemester 2009/2010

Klassen

Bisher haben wir Klassen nur aus einem Grund kennen gelernt:
In Java befindet sich das Hauptprogramm als Methode `public static void main` innerhalb einer Klasse.

Tatsächlich interessant werden Klassen allerdings als Grundelement der Objektorientierten Programmierung:

- Gleichartige Objekte werden durch ein Klassenmodell beschrieben.
- Es können – auch zur Laufzeit – neue Objekte erzeugt werden, die zu dieser Klasse gehören. Dies nennt man Instanzierung.
- So angelegte Objekte können auch wieder zerstört werden.

Beispiel für eine Klasse

Beginnen wir mit einem einfachen Beispiel für eine Klasse:

```
public class Mensch {
    private String vorname;

    public String getVorname() {
        return vorname;
    }

    public void setVorname(String vorname) {
        this.vorname = vorname;
    }

    public void sprich(String inhalt) {
        System.out.println(getVorname()+" sagt: "+inhalt);
    }
}
```


Analyse

Im Beispiel wird eine Klasse `Mensch` definiert.

Diese enthält

- ein Attribut: `vorname`,
- mehrere Methoden:
  - `getVorname`,
  - `setVorname`,
  - `sprich`

Weiterhin wird die Objektreferenz **`this`** eingeführt, die sich jeweils auf das Objekt bezieht, für das eine Methode aufgerufen wird.

Attribute

Attribute beschreiben die Eigenschaften der Objekte oder der Klasse. Hierbei wird unterschieden zwischen:

- *normale* Attribute (manchmal auch Objekt- oder Instanzvariablen genannt):
  Diese beschreiben Eigenschaften einzelner Instanzen, und haben daher auch für jede Instanz einen eigenen Wert.
- statische Attribute (oder Klassenvariablen):
  Diese beschreiben eine Eigenschaft der gesamten Klasse. Daher existiert es nur *einmal*. Das Attribut ist für alle Instanzen sichtbar und zugänglich, hat aber für alle Instanzen den selben Wert!

Methoden

Methoden beschreiben die Funktionalität der Objekte oder der Klasse. Analog zu den Attributen gibt es:

- *normale* Methoden (Instanzmethoden, Objektmethoden), die sich auf eine einzelne Instanz beziehen,
- statische Methoden (Klassenmethoden), die für die Klasse als ganzes aufgerufen werden (statische Methoden können natürlich nur auf Attribute zugreifen, die ebenfalls statisch sind!).

Beispiel

Zur Illustration eine Abwandlung der Klasse `Mensch`:

```
public class Mensch {
    private String vorname;
    private static String nachname;

    public void setVorname(String vorname) {
        this.vorname = vorname;
    }

    public static void setNachname(String nachname) {
        Mensch.nachname = nachname;
    }
}
```


Analyse

Diese Klasse beschreibt Menschen, die alle denselben Nachnamen haben, sich aber in ihrem Vornamen unterscheiden (können).

Aus diesem Grund sind das Attribut `nachname` und die Methode `setNachname` statisch, das Attribut `vorname` und die Methode `setVorname` hingegen nicht.

Die Methode `setVorname` muss daher für eine bestimmte Instanz aufgerufen werden, da diese sich auf jeweils auf ein einzelnes Objekt bezieht.
Die Methode `setNachname` wird hingegen für die Klasse als ganzes aufgerufen.

Beispiel

```
Mensch.setNachname("Meier");
Mensch adam = new Mensch();
adam.setVorname("Adam");

Mensch eva = new Mensch();
eva.setVorname("Eva");
```

Im Beispiel wird deutlich, dass die statische Methode `setNachname` bereits aufgerufen werden kann, bevor überhaupt eine Instanz der Klasse `Mensch` existiert.

Neben der Anweisung `Mensch.setNachname("Meier")` könnte auch die Anweisung `adam.setNachname("Meier")` verwendet werden (nachdem `adam` erzeugt wurde). Dies empfiehlt sich aber nicht, da es missverständlich ist, denn dieser Aufruf ändert den Nachnamen für alle Menschen!

Erzeugen von Instanzen

Betrachten wir folgenden Quelltext, in dem wir die Klasse Mensch verwenden wollen:

```
public class Gott {
    public static void main(String[] args) {
        Mensch adam;
        adam.setVorname("Adam");
        adam.sprich("Ich lebe");
    }
}
```

Dieser Quelltext funktioniert nicht! Wir benutzen die Variable `adam`. Diese bezeichnet in Java aber nicht direkt eine Instanz (und legt also auch keine neue an), sondern eine Referenz bzw. einen Verweis auf eine Instanz von `Mensch`!

Erzeugen von Instanzen (Forts.)

Bevor man die Variable `adam` verwenden kann, muss man dafür sorgen, dass diese auf eine Instanz von `Mensch` verweist. Dies nennt man initialisieren.

Richtig heißt es also:

```
public class Gott {
    public static void main(String[] args) {
        Mensch adam = new Mensch();
        adam.setVorname("Adam");
        adam.sprich("Ich lebe");
    }
}
```

Mit dem Operator **new** wird also eine neue Instanz von `Mensch` angelegt.

Zugriffsarten, Zugriffsmethoden

In der Beispielklasse ist das Attribut `vorname` privat und daher nur innerhalb der Klasse sichtbar und zugänglich.
Um auf den Wert dieses Attributs zugreifen oder diesen gar verändern zu können, wurden zwei öffentliche *Zugriffsmethoden* definiert, nämlich eine zum Setzen (`set...`) und eine zum Lesen (`get...`).

Eine solche Konstruktion kommt sehr häufig vor. Solche Zugriffsmethoden nennt man Akzessoren, *get/set-Methoden* oder einfach *getter* bzw. *setter*.

Motivation

Im Quelltext des Beispiels wurde für die neu angelegte Instanz von `Mensch` zunächst der Vorname gesetzt und dann die Methode zum Sprechen aufgerufen.

Was ergibt nun folgender Quelltext?

```
Mensch adam = new Mensch();
adam.sprich("Ich lebe");
```

Es wird ausgegeben:

```
null sagt: Ich lebe
```


Problem

Mit dem **`new`**-Operator wird ein Mensch angelegt. Was passiert dabei?

Die Attribute (Objektvariablen) werden initialisiert:

- elementare Datentypen mit dem Wert 0,
- Objektreferenzen mit **`null`**, d. h. sie verweisen auf *nichts*.

Im Beispiel äußert sich dies dadurch, dass als Vorname tatsächlich „`null`“ ausgegeben wird. Würde man hingegen versuchen, für ein Null-Objekt eine Methode aufzurufen, erhielte man eine *Nullpointer*-Exception!

Wie kann man nun dafür sorgen, dass ein Attribut mit einem anderen Wert initialisiert wird? Dieses Problem stellt sich insbesondere, wenn für ein Attribut keine set-Methode existiert.

Konstruktoren

Beim Anlegen einer Instanz der Klasse wird ein *Konstruktor* aufgerufen.
Im Konstruktor wird also bestimmt, *was* passiert, wenn eine neue Instanz der Klasse erzeugt wird.

Ein Konstruktor wird dabei definiert als Methode der Klasse

- mit dem Namen der Klasse als Methodenname,
- ohne Angabe eines Rückgabetyps (auch nicht `void`!)

Beispiel: Neue Klasse `Mensch`

```
public class Mensch {
    private String vorname;

    public Mensch(String vorname) {
        this.vorname = vorname;
    }

    public String getVorname() {
        return vorname;
    }
    public void setVorname(String vorname) {
        this.vorname = vorname;
    }
    public void sprich(String inhalt) {
        System.out.println(getVorname()+" sagt: "+inhalt);
    }
}
```


Verwenden des Konstruktors

Nun kann man den neuen Konstruktor mit direkter Angabe des Vornamens verwenden:

```
Mensch adam = new Mensch("Adam");
adam.sprich("Ich lebe");
```

Es ist übrigens zulässig, mehrere Konstruktoren zu definieren, solange anhand der Signatur der Parameter eindeutig entschieden werden kann, welcher gemeint ist.

Rückschau

In der ersten Version des Quelltextes wurde die Instanz der Klasse `Mensch` mit folgender Anweisung erzeugt:

```
Mensch adam = new Mensch();
```

Wurde hier auch schon ein Konstruktor aufgerufen? Ja, und zwar der Konstruktor, der keine Parameter braucht (deswegen `Mensch()`).

Aber der wurde doch gar nicht definiert? Existiert der automatisch?

Der Standardkonstruktor

Der parameter-lose Konstruktor hat einen speziellen Namen:

Standardkonstruktor

Der Standardkonstruktor existiert automatisch und ist dabei wie folgt definiert:

```
public Mensch() {
}
```

(Er tut also *nichts*)

. . . allerdings existiert er nur automatisch, wenn man *keinen* eigenen Konstruktor mit Parametern definiert!

Der Standardkonstruktor (Forts.)

Dies bedeutet für das Beispiel der Klasse Mensch:

1. Solange der Konstruktor von `Mensch` mit Angabe eines Vornamens als Parameter noch nicht definiert war, konnte man den Standardkonstruktor verwenden.
2. Nach Definition des Konstruktors mit Parameter existiert der Standardkonstruktor jedoch nicht mehr automatisch.
3. Daher ist es nun nur noch möglich, Instanzen von `Mensch` mit Verwendung des Konstruktors mit direkter Übergabe des Vornamens als Parameter anzulegen.
4. Soll man weiterhin den Standardkonstruktor verwenden können, muss man ihn jetzt also explizit definieren.

Beispiel 1

Hier wird der Standardkonstruktor in seiner ursprünglichen Bedeutung definiert, d. h. er tut nichts.

```
public class Mensch {
    private String vorname;

    public Mensch(String vorname) {
        this.vorname = vorname;
    }
    public Mensch() {
    }

    // Methodendefinitionen ...
}
```

Legt man eine neue Instanz damit an, ist der Vorname somit **null**.

Beispiel 2

Hier wird der Standardkonstruktor nun so definiert, dass er den Konstruktor, der die Angabe des Vornamens als Parameter erwartet, mit dem String „N.N.“ aufruft:

```
public class Mensch {
    private String vorname;

    Mensch(String vorname) {
        this.vorname = vorname;
    }
    Mensch() {
        this("N.N.");
    }

    // Methodendefinitionen ...
}
```

Somit ist für neue Instanzen, die mit dem Standardkonstruktor angelegt wurden, der Vorname *N.N.*.