

4 Objektorientierte Programmierung mit Java

Bisher wurde Java nur insofern eingesetzt, als es zur Umsetzung der aus funktionalen und zuweisungsorientierten Sprachen bekannten Elemente erforderlich ist. Java ist jedoch im Kern eine objektorientierte Sprache; im Mittelpunkt objektorientierter Sprachen stehen die Konzepte der Klasse und des Objekts.

4.1 Klassen und Objekte

Dieses Kapitel stellt die grundlegenden Mechanismen zum Umgang mit Klassen und Objekten vor. Auf spezielle objektorientierte Prinzipien wie Abstraktion und Vererbung wird erst später eingegangen.

Objekte als Tupel, Klassen als Tupeltypen

Objekte können als Tupel (vgl. funktionale Definition von Tupeln) betrachtet werden, deren Typ durch die zugehörige Klasse gegeben ist. Der folgende Java-Quellcode deklariert eine Klasse (engl.: class) mit dem Namen „Mitarbeiter“ als Typ eines vierstelligen Tupels. Der Klassenkörper in den geschweiften Klammern dieser Klassendeklaration enthält die Beschreibung der vier Tupelkomponenten, in Form von Variablen mit Typ und Namen.

```
class Mitarbeiter {  
    String nachname;  
    String vorname;  
    int personalnummer;  
    boolean istWeiblich;  
}
```

Jedes Tupel dieses Typs ist eine Instanz der Klasse Mitarbeiter und soll einen Mitarbeiter einer Firma beschreiben. Instanzen werden auch als Objekte bezeichnet. Jedes Objekt der Klasse Mitarbeiter besitzt jeweils eigene Exemplare der vier Variablen. Die Variablen in Objekten werden Instanzvariablen oder Attribute genannt. Ihre Namen beginnen üblicherweise mit einem Kleinbuchstaben.

Instanziierung einer Klasse, Zugriff auf Attribute

Wird zur Laufzeit eines Programms ein Objekt von einer Klasse erzeugt, so sagt man auch: „Die Klasse wird instanziiert.“ Die Schaffung neuer Instanzen einer Klasse wird syntaktisch durch das Schlüsselwort **new** ausgedrückt, gefolgt vom Klassennamen. Der Zugriff auf ein Attribut eines Objektes erfolgt, indem man das Objekt angibt, gefolgt von einem Punkt und dem Namen der Variable. Im folgenden Beispiel werden Instanzen der Klasse „Mitarbeiter“ erzeugt und die Attribute belegt:

```
class NeueInstanz {  
  
    public static void main(String args[]) {  
        Mitarbeiter m; // Deklaration eines Attributs für einen Mitarbeiter  
        m = new Mitarbeiter(); // Instanziierung: Kreierung eines neuen Objektes  
  
        // schreibender Zugriff auf Attribute:  
        m.nachname = "Mayer";  
        m.vorname = "Hans";  
        m.personalnummer = 11073;  
        m.istWeiblich = false;  
  
        // lesender Zugriff auf Attribute:  
        System.out.println("Nachname: " + m.nachname);  
    }  
}
```

```

        System.out.println("Vorname:␣" + m.vorname);
        System.out.println("Personalnummer:␣" + m.personalnummer);
        System.out.println("Anrede:␣" + (m.istWeiblich ? "Frau" : "Herr"));

        // Anlegen eines zweiten Objektes:
        Mitarbeiter n = new Mitarbeiter();
        n.nachname = "Bader";
        n.vorname = "Petra";
        n.personalnummer = 865;
        n.istWeiblich = true;
    }
}

```

Konstrukturen

Konstrukturen sind spezielle Methoden, die automatisch mit der Erzeugung einer neuen Instanz aufgerufen werden. Ihre Aufgabe ist im allgemeinen, diese neue Instanz in einen geeigneten Startzustand zu versetzen, d.h. ihre Attribute sinnvoll zu initialisieren. Jede Klasse besitzt, wenn nichts weiter angegeben ist, einen parameterlosen Standardkonstruktor (engl.: default constructor), der alle Attribute mit einem Standardwert initialisiert (z. B. `int`-Variablen mit 0). Ein Konstruktor hat stets denselben Namen wie die dazugehörige Klasse. Weil ein Konstruktor kein Resultat liefert, kann bei seiner Deklaration kein Resultatstyp angegeben werden. Daher sind im Rumpf von Konstrukturen `return`-Anweisungen ausschliesslich in der Form `return;` erlaubt. Ein Konstruktor wird im Zusammenhang mit `new` aufgerufen, wenn ein neues Objekt kreiert wird. Die aktuellen Parameter für einen Konstruktor-Aufruf werden hierbei in einem Instanziierungsausdruck nach dem Klassennamen in runden Klammern durch Kommata getrennt angegeben. Der Wert dieses Ausdrucks ist das neu erzeugte und initialisierte Objekt.

Syntax eines Instanziierungsausdrucks:

<Klasseninstanziierung> ::= 'new' <Klassenbezeichner> '(' <Parameter> (',' <Parameter>)* ')'

Beispiel:

```

class Mitarbeiter {
    String nachname;
    String vorname;
    int personalnummer;
    boolean istWeiblich;

    // Deklaration eines Konstruktors:
    Mitarbeiter(String nn, String vn, int pn, boolean w) {
        nachname = nn;
        vorname = vn;
        personalnummer = pn;
        istWeiblich = w;
    }

    public static void main(String args[]){
        Mitarbeiter m,n;
        // Konstruktor-Aufrufe:
        m = new Mitarbeiter("Mayer", "Hans", 11073, false);
        n = new Mitarbeiter("Bader", "Petra", 865, true);
    }
}

```

4.2 Methoden (Instanzmethode)

Innerhalb einer Klasse beschreibt jede Methodendeklaration ohne das Schlüsselwort **static** eine Instanzmethode (häufig kurz „Methode“). Eine Inkarnation einer Instanzmethode nimmt automatisch Bezug auf die Instanz der Klasse, der die Methode zugeordnet ist. Eine Instanzmethode einer Klasse wird aufgerufen, indem man den Namen des jeweiligen Objekts angibt, gefolgt vom Methodennamen und den aktuellen Parametern in runden Klammern. Dieses Objekt ist dann das implizite Argument des Aufrufs.

```
class Mitarbeiter {
    String nachname;
    String vorname;
    int personalnummer;
    boolean istWeiblich;

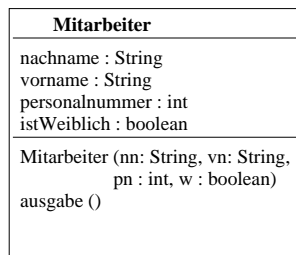
    // Konstruktor:
    Mitarbeiter(String nn, String vn, int pn, boolean w){
        nachname = nn;
        vorname = vn;
        personalnummer = pn;
        istWeiblich = w;
    }

    // eine Instanzmethode:
    void ausgabe(){
        System.out.print(istWeiblich ? "Frau" : "Herr");
        System.out.print(vorname + " ");
        System.out.print(nachname);
        System.out.println("(" + personalnummer + ")");
    }
}

class InstanzmethodenBeispiel {
    public static void main(String args[]){
        Mitarbeiter m1 = new Mitarbeiter("Schmidt", "Martin", 341, false);
        Mitarbeiter m2 = new Mitarbeiter("Bader", "Petra", 865, true);
        // zwei Aufrufe der obigen Instanzmethode:
        m1.ausgabe();
        m2.ausgabe();
    }
}
```

Graphische Beschreibung von Klassen, UML

Wir haben gesehen, dass sich Klassen im wesentlichen aus zwei Elementen, den Attributen und den Methoden zusammensetzen. Zur übersichtlichen Darstellung der wesentlichen Bestandteile einer Klasse, aber auch zur Modellierung der Beziehungen zwischen mehreren Klassen hat sich die UML (Universal Modelling Language) etabliert. Für obiges Beispiel erhält man etwa folgendes Klassendiagramm:



Offensichtlich besteht ein Klassendiagramm aus drei Teilen: Dem Klassenbezeichner in der ersten Zeile; den Attributen in der zweiten Zeile, wobei Attributbezeichner und Attributtyp durch einen Doppelpunkt getrennt sind; und schließlich in der dritten Zeile aus den verschiedenen Methoden. Die UML macht hierbei bezüglich der Signatur der Methoden keine detaillierte Vorgabe. Je nach gewünschter Exaktheit der Spezifikation kann im Klassenmodell nur der Methodenbezeichner oder auch die gesamte Signatur angegeben werden. Der Konstruktor wird zwar im UML-Standard nicht angegeben, im Rahmen dieses einführenden Kurses dürfte es jedoch hilfreich sein, den Konstruktor im UML-Modell anzugeben.

Im weiteren werden die graphischen Modellierungsmethoden der UML schrittweise erweitert werden.

Klassenvariablen

Im Gegensatz zu Instanzvariablen (Attribute) existiert von einer Klassenvariable für alle Objekte einer Klasse nur ein Exemplar. Dieses Exemplar ist auch dann vorhanden, wenn zu einem Zeitpunkt keine Instanz der Klasse existiert. Klassenvariablen werden auch statische Variablen genannt, da ihre Deklarationen mit dem Schlüsselwort **static** geschrieben werden. (Bei dem mittlerweile wohlbekannten **static** innerhalb der Deklaration der Methode **main** handelt es sich analog um eine statische Methode!) Bei der Deklaration von Instanzvariablen dagegen fehlt dieser Modifikator. Der Zugriff auf eine Klassenvariable erfolgt durch die Angabe des zugehörigen Klassennamens, gefolgt von einem Punkt und dem Namen der Variablen. Ein typisches Beispiel für eine Klassenvariable ist ein Zähler, dessen Wert die Anzahl der bereits erzeugten Instanzen einer Klasse angibt.

```
class Mitarbeiter {
    String nachname;
    String vorname;
    int personalnummer;
    boolean istWeiblich;
    static int anzahl = 0; // eine Klassenvariable , initialisiert mit 0

    Mitarbeiter(String nn, String vn, int pn, boolean w) {
        nachname = nn;
        vorname = vn;
        personalnummer = pn;
        istWeiblich = w; anzahl++;
    }
}

class AnzahlBeispiel {
    public static void main(String[] args){
        System.out.println("Anzahl:␣" + Mitarbeiter.anzahl);
        Mitarbeiter m1 = new Mitarbeiter("Mayer", "Hans", 11073, false);
        System.out.println("Anzahl:␣" + Mitarbeiter.anzahl);
        Mitarbeiter m2 = new Mitarbeiter("Bader", "Petra", 865, true);
        System.out.println("Anzahl:␣" + Mitarbeiter.anzahl);
    }
}
```

4.3 Klassenmethoden oder statische Methoden

Instanzmethoden sind Methoden, bei deren Ausführung auf ein Objekt Bezug genommen wird. Im Gegensatz dazu ist eine Klassenmethode eine globale Methode für die ganze Klasse, nicht für eine bestimmte Instanz. Sie sind wie Klassenvariablen am Schlüsselwort `static` zu erkennen und aus früheren Kapiteln (2) wohlbekannt: Selbstdefiniert Methoden, wie etwa die Methode zur Berechnung der Binomialkoeffizienten (4); aber auch die `main`-Methode. Sie alle sind mit dem Schlüsselwort `static` deklariert und konnten verwendet werden, ohne eine Instanz der entsprechenden Klasse zu erzeugen.

Statische Methoden werden aufgerufen, indem vor dem Methodennamen den Namen der Klasse angegeben wird, wobei Klassen- und Methodennamen durch einen Punkt getrennt sind. Befinden sich Aufruf und Deklaration innerhalb derselben Klasse, so genügt beim Aufruf der Name der Methode ohne vorangestellten Klassennamen. Auch die mehrfach verwendete Funktion `System.out.println()` ist demzufolge statische Methode, da sie über einen Klassennamen, nämlich `System` aufgerufen wird.

```
class Mitarbeiter {
    String nachname;
    String vorname;
    int personalnummer;
    boolean istWeiblich;
    static int anzahl = 0;

    // Konstruktor :
    Mitarbeiter(String nn, String vn, int pn, boolean w){
        nachname = nn;
        vorname = vn;
        personalnummer = pn;
        istWeiblich = w; anzahl++;
    }

    // zwei Klassenmethoden :
    static int liesAnzahl() {
        return anzahl;
    }

    static void druckeAnzahl() {
        System.out.print("Anzahl der Mitarbeiter: ");
        // ein Aufruf einer Klassenmethode (ohne Klassennamen):
        System.out.println(liesAnzahl());
    }
}

class KlassenmethodenBeispiel {
    public static void main(String args[]) {

        Mitarbeiter chef = new Mitarbeiter("Knoll", "Alois", 341, false);

        // Aufrufe der obigen Klassenmethoden (mit Klassennamen):
        int a = Mitarbeiter.liesAnzahl();
        Mitarbeiter.druckeAnzahl();
    }
}
```

Klassen als Typen

Klassen sind zulässige Typen für Variablen und Parameter. Variablen, deren Typ eine Klasse ist, können Instanzen dieser Klasse zugewiesen werden. Um auszudrücken, dass in einer Variable gerade kein Objekt gespeichert ist, gibt es den speziellen Wert **null**. Dieser Wert wird jeder Instanz- und jeder Klassenvariable eines Referenztyps bei ihrer Deklaration implizit zugewiesen.

Beispiel: Die Klasse `Mitarbeiter` wird variiert, so dass sie nun auch eine Instanzvariable mit Namen `vorgesetzter` enthält. Diese Variable ihrerseits ist wiederum vom Typ `Mitarbeiter`.

```
class Mitarbeiter {
    String nachname;
    String vorname;
    int personalnummer;
    boolean istWeiblich;
    Mitarbeiter vorgesetzter; // eine Klasse als Typ einer Instanzvariable

    // Konstruktor:
    Mitarbeiter(String nn, String vn, int pn, boolean w, Mitarbeiter v) {
        nachname = nn;
        vorname = vn;
        personalnummer = pn;
        istWeiblich = w;
        vorgesetzter = v;
    }
}

class NullBeispiel {
    public static void main(String args[]){
        Mitarbeiter m, chef; // eine Klasse als Typ von lokalen Variablen
        chef = new Mitarbeiter("Schmidt", "Martin", 341, false, null);
        m = new Mitarbeiter("Mayer", "Hans", 11073, false, chef);
    }
}
```

Listing 7: 'Mitarbeiter und ihre Vorgesetzten'

In diesem Beispiel wird **null** verwendet, um zu verdeutlichen, dass das im Objekt `chef` gespeicherte Objekt keinen Vorgesetzten besitzt.

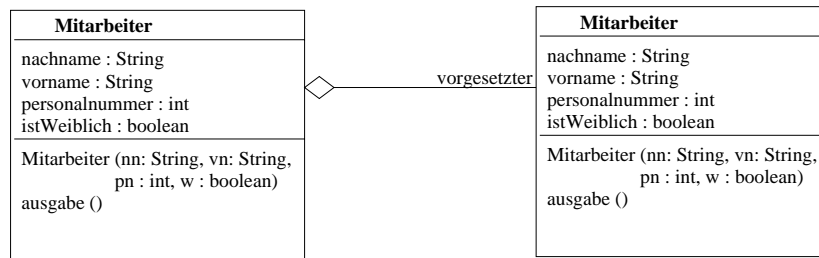
Instanziierungsausdrücke: Die Instanziierung einer Klasse mit **new** ist ein Ausdruck, dessen Typ die Klasse ist. Ein solcher Ausdruck kann beispielsweise nach `-stehen`, um ein Objekt anzugeben, mit dem eine Variable initialisiert werden soll:

```
Mitarbeiter chef = new Mitarbeiter("Schmidt", "Martin", 341, false, null);
Mitarbeiter m = new Mitarbeiter("Mayer", "Hans", 11073, false, chef);
```

Verschachtelte Instanziierungsausdrücke: Wird als Argument eines Konstruktors ein Objekt erwartet, so kann bei seinem Aufruf ebenfalls ein **new**-Ausdruck stehen. Hierbei entstehen Verschachtelungen von **new**-Ausdrücken, auch Konstruktor-Terme genannt:

```
Mitarbeiter m = new Mitarbeiter("Mayer", "Hans", 11073, false, new Mitarbeiter("Schmidt", "Martin", 341, false, null));
```

Im Programm (7) werden erstmals Klassen wie Typen von Variablen verwendet. Die Klasse `Mitarbeiter` verwendet hier ein Objekt namens `vorgesetzter`, das ebenfalls Instanz der Klasse `Mitarbeiter` ist. Eine solche Beziehung zwischen zwei Klassen, bei der Objekte einer Klasse B Attribute einer Klasse A sind nennt man Aggregation. Graphisch wird dies durch folgendes UML-Diagramm beschrieben:



4.4 Werte und Referenzen, Grundtypen und Referenztypen

Zwischen den Grundtypen (**int**, **boolean**, **double**, ...) auf der einen Seite und den Referenztypen, d.h. den Klassen und Feldtypen auf der anderen Seite gibt es einen wesentlichen Unterschied: Eine Variable eines Grundtyps enthält einen Wert dieses Grundtyps. Eine Variable eines Referenztyps enthält eine Referenz (oder auch Verweis oder Zeiger ; engl.: reference oder pointer) auf ein Objekt (und nicht das Objekt selbst), oder aber die spezielle Referenz null. Der in Java vordefinierte Typ **Object** stellt den allgemeinsten Referenztyp dar. Eine Variable vom Typ **Object** kann daher Werte eines beliebigen Referenztyps annehmen.

Beispiel: Es werden drei Variablen initialisiert und verglichen; der Typ der ersten ist ein Grundtyp, die Typen der anderen beiden sind Referenztypen, nämlich ein Feldtyp bzw. eine Klasse:

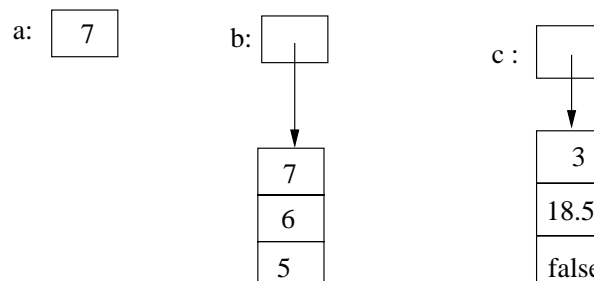
```
int a = 7;
int b[] = {- 7, 6, 5};
Triple c = new Triple(-3, 18.5, false);
```

Die Klasse **Triple** habe dabei folgende Gestalt:

```
class Triple {
    short s;
    double d;
    boolean b;

    Triple(short ss, double dd, boolean bb) {
        s = ss;
        d = dd;
        b = bb;
    }
}
```

Nach Initialisierung dieser drei Variablen kann man sich den Inhalt des Arbeitsspeichers wie folgt vorstellen:



Die Variablen **b** und **c** enthalten also jeweils eine Referenz auf ein Feld bzw. ein Objekt.