5 Grundlegende Programmierelemente

5.1 Datentypen

Bei der Programmierung in Java erinnern wir uns noch an diverse Datentypen. Also die Charakterisierung der möglichen Wertzuweisung einer Variable.

Typ	Beschreibung	Wertebereich / Beispiel
`boolean`	Boolescher Wert	`true`, `false`
`char`	einzelnes Zeichen	alle Unicode-Zeichen (Tastatur)
`byte`	eine ganze Zahl (max. 8 Bit)	$-2^7 \ldots 2^7 - 1$
`short`	eine ganze Zahl (max. 16 Bit)	$-2^{15} \ldots 2^{15} - 1$
`int`	eine ganze Zahl (max. 32 Bit)	$-2^{31} \ldots 2^{31} - 1$
`long`	eine ganze Zahl (max. 64 Bit)	$-2^{63} \ldots 2^{63} - 1$
`float`	Fließkommazahl (max. 32 Bit)	Beispiel: $3,14159f$
`double`	Fließkommazahl (max. 64 Bit)	Beispiel: $-1,79 * 10^{38}$
`String`	Zeichenkette (Wörter / Sätze etc.)	Beispiel: „Das ist ein String!“
`int[ ]`	ganzzahliges Feld (Array)	Beispiel: $3, 1, 4, 1, 5, 9$

Java unterscheidet zwischen **zwei Datentypen**. Zum einen gibt es die *primitiven Typen*. Dazu zählen z.B. `boolean`, `char`, ... , `double`. Zum anderen gibt es die *Referenztypen*. Als solche werden Typen bezeichnet, die entweder primitive Typen enthalten oder aus solchen zusammengesetzt werden. So zum Beispiel Objekte, Strings und Arrays.
Die wichtigen Datentypen werden in einem gesonderten Kapitel behandelt. Dort finden sich auch verschiedene Operationen und Beispiele für die Anwendung der entsprechenden Datentypen.

5.2 Operatoren

In Java gibt es diverse Operatoren, die bei der Programmierung hilfreich sein können.

Java-Operator	Beschreibung	Anmerkung
+	Addition	
-	Subtraktion	
*	Multiplikation	
/	Division	Liefert den Quotienten von **x und y**. Sind beide Zahlen ganzzahlig, so auch der Quotient (z.B. 11/5 liefert 2).
%	Modulo	Divisionsrest (z.B 9%4 = 1)
++	Inkrement	`i++` entspricht dann `i+1`

Zusätzlich existieren noch weitere Operatoren die **Verknüpfung** zweier Variablen ermöglichen.

Java-Operator	Beschreibung	Anmerkung
=	Zuweisung	Der Variablen auf der linken Seite wird der auf der rechten Seite des =-Zeichen stehende Wert zugewiesen.
==	Vergleich	Ermöglicht den Vergleich von **primitiven Datentypen**. Liefert als Rückgabe `true` oder `false`.
<	Kleiner	Liefert `true` oder `false`.
<=	Kleiner gleich	
>	Größer	
>=	Größer gleich	
!=	Ungleich	Ermöglicht den Vergleich von **primitiven Datentypen**. Liefert als Rückgabe `true` oder `false`.
!	logisches NICHT	Kehrt die Wertzuweisung der nachfolgenden Variable für die nächste Operation um. Aus `true` wird `false`.
||	logisches ODER	Entweder die eine, die andere **oder** beide Bedingungen sind erfüllt.
&&	logisches UND	Es müssen beide Bedingungen erfüllt sein.

5.3 Klassendefinition

Wir erinnern und, dass es sich bei Java um eine sogenannte **objektorientierte Programmiersprache** handelt. Um ein Objekt überhaupt erzeugen zu können, benötigen wir einen *Bauplan*, der alle nötigen Informationen enthält. Diesen *Bauplan* bezeichnet man auch als **Klasse**.
Dabei gilt folgende Vorgabe:

```
(<Zugriffsart>) class <Bezeichner> (extends <Oberklasse>) {...}
```

Hierbei ist die Angabe der `<Zugriffsart>` notwendig, `extends <Oberklasse>` hingegen ist optional.

Beispiel:

```
public class Square{
  /**
  * Deklaration der Attribute
  * Ganzzahlige Attribute für Seitenlänge und Text-Variable für die Farbe werden
  * deklariert.
  **/
  private int length;
  private String color;

  /**
  * Methodendefinition
```

```
    * Konstruktor zur Erzeugung des Objekts hat den gleichen Namen wie die Klasse.
    **/
    Square(int side1){
      length = side1;
      color = "Red";
    }

    public double area(){
      return length*length;
    }
  } //Ende der Klassendefintion
```

5.4 Methodendeklaration

Die eben angesprochene Klasse beinhaltet im Allgemeinen Methoden, also „Fähigkeiten“, die die erzeugten Objekte der Klasse besitzen.
Möchte man eine solche Methode deklarieren, so muss diese Deklaration die folgende Form haben:

```
(<Zugriffsart>) <Rückgabewert> <Bezeichner> (<Parameter>) {...}
```

Wie bei der Klasse ist die Definition der `<Zugriffsart>` verpflichtend. Die Angabe `<Parameter>` hat die Form `<Datentyp> <Bezeichner>`.
Die Definition des `<Rückgabewert>` bestimmt, welchen Datentyp die Methode bei Aufruf zurückliefert. Die möglichen Belegungen sind die unter 5.1 genannten, sowie weitere Datentypen. Die Angabe von `void` als Rückgabewert sagt aus, dass die Methode keine Rückgabe liefert.

Beispiel:

```
/**
* Öffentliche Methode hello gibt auf dem Bildschirm "Hallo XYZ" aus, wenn "XYZ" beim
* Aufruf übergeben wurde.
**/
public void hello(String name){
  System.out.println("Hallo " + name);
}
public double umfang(double radius){
  return 2*radius*3,14159;
}

/**
* Die Methode goToSleep hat keinen Rückgabewert und keine Parameter.
* Sie ruft nacheinander die Methoden undress, wash, brushTeeth und lieDown auf.
**/
public void goToSleep(){
  undress();
  wash();
  brushTeeth();
  lieDown();
}
```

5.5 Variablendefinition

Innerhalb von Klassen, aber auch in Methoden benötigen wir Variablen, mit denen wir arbeiten können. Diese müssen zunächst deklariert werden. Auch hier gibt es eine Deklarationsvorschrift:

```
(<Zugriffsart>) <Typ> <Bezeichner> (= <Wert>)
```

Die direkte Wertzuweisung mittels `= <Wert>` kann, muss aber nicht, direkt bei der Variablendeklaration gemacht werden.
Bei dieser 'Wertzuweisung' ist wichtig zu beachten, dass **Referenztypen** im Allgemeinen mit dem `new`-Operator erzeugt werden müssen. Dies gilt nicht für den Referenztyp `String`.

Beispiel:

```
private int anzahl;
int tage = 15;
boolean healty;

int[] counter = new int[Größe];
```

5.6 Zugriffsart

Bei der Definition bzw. Deklaration von Klassen, Methoden und Variablen wird immer nach der ominösen `<Zugriffsart>` verlangt. Diese gibt an, wer auf das Objekt und seine Methoden und Variablen zugreifen kann. Dabei gibt es die folgenden Unterscheidungen:

- **public**
  Innerhalb einer Klasse sind die Konstruktoren, Methoden und Variablen sichtbar. Sollen diese auch von Objekten außerhalb der Klasse verwendet werden, definiert man sie als `public`.
  Deklariert man eine Klasse als `public`, so können andere Klassen Instanzen dieser Klasse erzeugen.

- **private**
  Dem Gegenüber steht `private`. Diese Zugriffsart erlaubt den Zugriff nur innerhalb der Klasse selbst. Das bedeutet auch, dass z.B. Methoden oder Variablen, die als `private` deklariert wurden, für andere nicht sichtbar sind.

- **protected**
  Zusätzlich gibt es die Zugriffsart `protected`. Diese dritte Zugriffsart betrifft die Klasse, sowie alle derzeit existierenden und zukünftigen Subklassen.
  Auf als `protected` deklarierte Konstruktoren, Methoden und Instanzvariablen kann nur von Subklassen zugegriffen werden.
  Befinden sich zwei Klassen im gleichen `package`, können diese jeweils auf die `protected` Bereiche der anderen zugreifen.

- **package (auch friendly oder default)**
  Der `default`-Modus tritt immer dann in Kraft,wenn keine ausdrückliche Zugriffsart angegeben wird.

<Zugriffsart>	Beschreibung
public	Der Zugriff ist immer möglich.
private	Der Zugriff ist nur innerhalb der Klasse möglich.
protected	Der Zugriff ist von Klassen innerhalb des gleichen Package möglich. Ebenso kann von Subklassen auf die protected Elemente zugegriffen werden.
package	Ein Zugriff ist innerhalb der Klasse und von anderen Klassen des gleichen Package möglich. Der Zugriff ist von einer Subklasse aus nicht möglich.

5.7 Einlesen von der Tastatur

Um einen Text von der Taststur einzulesen gibt es das Objekt `System.in`. Ohne nähere Erklärung der Hintergründe folgt hier zunächst eine Vorstellung, wie eine Ganzzahl von der Tastatur eingelesen werden kann.

```
Scanner eingabe = new Scanne(System.in);
int zahl = eingabe.nextInt();
```

Mit `Zeile 1` haben wir ein Objekt erzeugt, das eine Methode zum Einlesen einer Ganzzahl hat. Diese Methode kann über `eingabe.nextInt();` zeilenweise von der Tastatur lesen. Das Ergebnis dieser Methode ist vom Typ `int` (Integer).

Beispiel:

```
import java.util.*;

class Einlesen{
  public static void main(String[] args){
    int a,b;
    Scanner eingabe = new Scanner(System.in);
    System.out.println("Bitte geben Sie Zahl a ein:");
    a = eingabe.nextInt();
    System.out.println("Bitte geben Sie Zahl b ein:");
    b = eingabe.nextInt();

    System.out.println("Sie haben die Zahlen a = " + a + " und b = " + b + " eingegeben.");
  }
}
```

Eine **kleine Besonderheit** besteht beim Einlesen eines String. Die Methode `eingabe.next()` erkennt über das Betätigen der Enter-Taste nicht, dass die Eingabe beendet wurde. Hier wäre ein doppeltes Enter-Drücken notwendig.
Um dies zu umgehen bietet das Objekt `eingabe` die Methode `eingabe.nextLine();`. Diese beendet das Einlesen der Tastatureingabe automatisch mit dem Betätigen der Enter-Taste.