Anwendungs-Entwicklung mit Java

Workshop

Cafer Ucarli & Patrik Balazs, SAP
April 07, 2021

PUBLIC

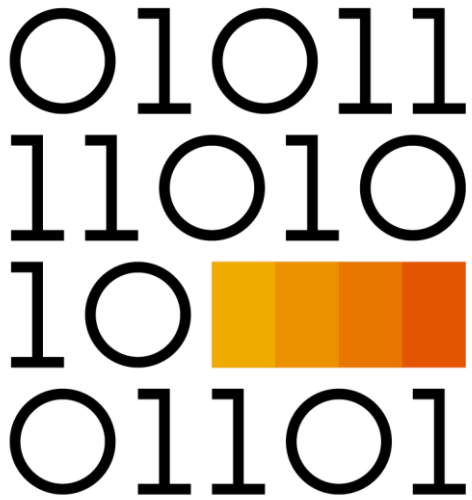
THE BEST RUN SAP®

Programmiersprachen

Die **menschliche Denkweise** unterscheidet sich von der **Arbeitsweise eines Computers**. Diese Diskrepanz wird durch Programmiersprachen minimiert.

Das **Abstraktionsniveau** der Programmiersprache Java ist sehr hoch, was bedeutet, dass der Mensch den Quellcode ohne großen Aufwand verstehen kann.

Java ist eine **objektorientierte Programmiersprache**. So lassen sich reale Szenarien einfacher abbilden.



Hello World

Allgemeine Syntaxregeln

Klassen stellen den grundlegenden Rahmen für Programme dar

Jede Klasse besteht aus einer Folge von **Anweisungen**

Anweisungen sind wohldefinierte Befehle, die der Interpreter zur Laufzeit ausführt

Anweisungen werden in Java mit dem **Semikolon** abgeschlossen

Die main-Methode

Die main-Methode ist eine spezielle Methode in Java und stellt **Startpunkt**, sowie **Endpunkt** einer Anwendung dar

Nur Klassen mit einer main-Methode können von der Laufzeitumgebung **ausgeführt** werden

```
public class Listing0302 {

   public static void main(String[] args) {
      …
   }

}
```

Ausgabe über die Konsole

Der Standard-Ausgabestrom **System.out** bietet verschiedene Methoden, um Strings auszugeben

- **System.out.print(“Text“):** der String wird unverändert und linksbündig ausgegeben
- **System.out.println(“Text“):** der String wird unverändert und linksbündig ausgegeben, bewirkt aber zusätzlich einen Zeilenumbruch

Kommentare

Kommentare sollen die **Lesbarkeit** des Programms verbessern

Java kennt drei unterschiedliche Arten von Kommentaren

- Einzeilige Kommentare
- Kommentarblöcke
- Dokumentationskommentare

Kommentare bewirken bei der Ausführung keine Aktion und werden ignoriert

```
public class Listing0303 {

   public static void main(String[] args) {

      // einzeiliger Kommentar

      /*
       * mehrzeiliger Kommentar
       * mehrzeiliger Kommentar
       */

      /**
       * Dokumentationskommentar
       */

   }

}
```

Aufgabe 1 – Hello World

Schreiben Sie ein Programm, welches den abgebildeten Text auf der Konsole ausgibt.

```
Hello World!
```

Aufgabe 1 – Hello World

```java
public class HelloWorld {

    public static void main(String[] args) {

        //Augabe auf der Konsole
        System.out.println("Hello World!");

    }

}
```

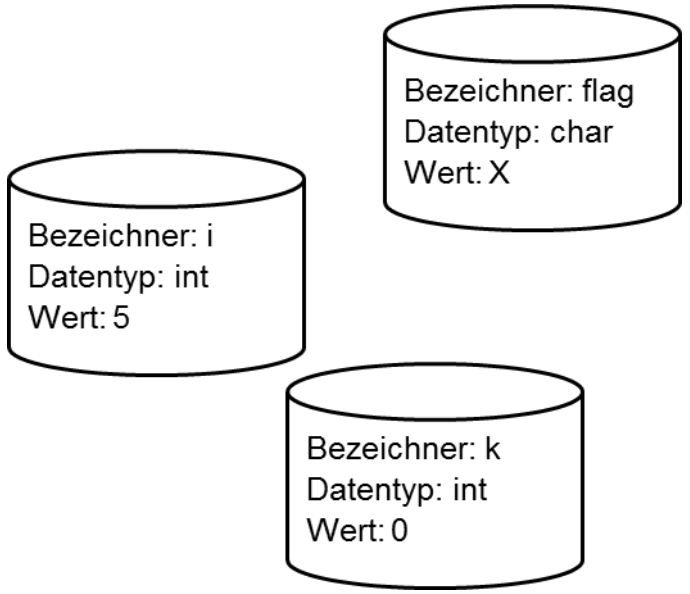
Variablen und Datentypen

Variablen

Eine Variable ist ein **Platzhalter**, der mit verschiedenen Werten belegt werden kann

Die Größe des reservierten Speichers ist abhängig vom gewählten **Datentyp**

Bezeichner ermöglichen das Ansprechen von Variablen im Programmablauf



Deklaration von Variablen

Durch Angabe von **Datentyp** und **Bezeichner** wird eine Variable deklariert, d.h. dem Compiler bekannt gegeben

Deklarationen werden wie jede Anweisung mit einem **Semikolon** abgeschlossen

Üblicherweise beginnen Variablennamen mit einem **kleinen Buchstaben**

Hinweis: Java ist **„case-sensitiv“**, unterscheidet also zwischen Groß- und Kleinschreibung!

```
public class Listing0401 {

   public static void main(String[] args) {

      int i;
      boolean error;
      char char1;
      String text;

   }

}
```

Initialisierung von Variablen

Der Zuweisungsoperator = besitzt zwei Operanden und weist der Variablen auf der linken Seite den Wert des Ausdrucks auf der rechten Seite zu

Hinweis: in Java müssen Werte **explizit initialisiert** werden, d.h. mit einem Wert belegt werden, sonst kommt es zu einem Compilerfehler!

```
public class Listing0402 {

   public static void main(String[] args) {

      int i = 42;
      boolean error = true;
      char char1;
      String text;

      char1 = 'M';
      text = "Hallo Welt";

   }

}
```

Elementare Datentypen

Java kennt 8 sogenannte *elementare Datentypen*

Elementare Datentypen sind fest in der Programmiersprache verankert und können durch ein entsprechendes Schlüsselwort angesprochen werden

Datentyp	Größe	Wertebereich
boolean	1 Byte	true, false
char	2 Byte	\u0000 bis \uFFFF
byte	1 Byte	-128 bis +127
short	2 Byte	-32.768 bis +32.767
int	4 Byte	-2.147.483.648 bis +2.147.483.647
long	8 Byte	-9.233.372.036.854.775.808 bis +9.233.372.036.854.775.807
float	4 Byte	$+/-1{,}4e^{-45}$ bis $+/-3{,}4028235e^{+38}$
double	8 Byte	$+/-4{,}9e^{-324}$ bis $+/-1{,}7976931348623157e^{+308}$

Zeichenketten

Ketten von beliebigen Zeichen werden als *String* bezeichnet

Strings werden in Java mit Hilfe der **Anführungszeichen** realisiert

Steuer- und Sonderzeichen können in Java mit Hilfe einer Escape-Sequenz realisiert werden

Escape-Sequenz	Beschreibung
\n	Zeilensprung
\t	Tabulatorsprung
\\	Backslash
\"	Anführungszeichen
\'	Hochkomma

Aufgabe 2 – Variablen

Schreiben Sie ein Programm, welches die drei Variablen:

- name (String)
- age (int)
- gender (char)

deklariert, initialisiert und über die Konsole ausgibt.

```
Name: Daniel
Alter: 34
Geschlecht: m
```

Aufgabe 2 – Variablen

```java
public class Aufgabe2 {

    public static void main(String[] args) {

        //Deklaration
        String name;
        int age;
        char gender;

        //Initialisierung
        name    = "Daniel";
        age     = 34;
        gender  = 'm';

        //**********ODER
        String name1 = "Daniel";
        int age1     = 34;
        char gender1 = 'm';

        //Ausgabe
        System.out.println("Name: "  + name);
        System.out.println("Alter: " + age);
        System.out.println("Geschlecht: " + gender);
    }

}
```

Operatoren

Operatoren

Operatoren sind Zeichen, mit denen Daten manipuliert werden können

Mit Hilfe von Operanden und Operatoren können beliebig komplexe Ausdrücke abgebildet werden

In Java wird zwischen **numerischen Operatoren, Vergleichsoperatoren, logischen Operatoren** und bitweisen Operatoren unterschieden

Numerische Operatoren

Numerische Operatoren ermöglichen die Verarbeitung von arithmetischen Ausdrücken

Ausdruck mit Operator	Bedeutung
x + y	Addiere y zu x
x - y	Subtrahiere y von x
x * y	Multipliziere x mit y
x / y	Dividiere x durch y
x % y	Berechne den Divisionsrest von x / y
x++	Inkrementiere x und gib den alten Wert an den Ausdruck zurück
++x	Inkrementiere x und gib den neuen Wert an den Ausdruck zurück
x--	Dekrementiere x und gib den alten Wert an den Ausdruck zurück
--x	Dekrementiere x und gib den neuen Wert an den Ausdruck zurück

Konsoleneingabe

Konsoleneingabe

Die Klasse *Scanner* im Paket *java.util* stellt Methoden zur Verfügung, um Werte bestimmter Datentypen von der Konsole einzulesen

Hinweis: dem Konstruktor muss der Wert *System.in* mitgegeben werden!

```
import java.util.Scanner;

public class Listing0601 {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);
      int i = scanner.nextInt();
      System.out.println(i);
      scanner.close();

   }

}
```

Aufgabe 3 – Operatoren und Konsoleneingabe

Schreiben Sie ein Programm, welches zwei Ganzzahlen von der Konsole

- Einliest
- Addiert
- Ergebnis auf der Konsole ausgibt.

```
Bitte geben Sie eine Zahl ein:
10
Bitte geben Sie eine weitere Zahl ein:
5
Ergebnis: 15
```

Aufgabe 3 – Operatoren und Konsoleneingabe

```java
public class Aufgabe3 {

    public static void main(String[] args) {

        //Scanner erzeugen
        Scanner scanner = new Scanner(System.in);

        //Zahlen abfragen
        System.out.println("Bitte geben Sie eine Zahl ein:");
        int value1 = scanner.nextInt();
        System.out.println("Bitte geben Sie eine weitere Zahl ein:");
        int value2 = scanner.nextInt();
        scanner.close();

        //Berechnung
        int result = value1 + value2;

        //Ausgabe
        System.out.println("Ergebnis: " + result);
    }

}
```

Kontrollstrukturen

Logische Operatoren

Logische Operatoren können dazu verwendet werden, **logische Aussagen** miteinander zu verknüpfen

Ausdruck mit Operator	Bedeutung
a && b	Logische Und-Verknüpfung (mit Kurzschlussauswertung)
a || b	Logische Oder-Verknüpfung (mit Kurzschlussauswertung)
!a	Logische Verneinung

Vergleichsoperatoren

Logische Bedingungen für zwei Werte eines elementaren Datentyps können durch Vergleichsoperatoren realisiert werden

Hinweis: der Vergleichsoperator wird oft mit dem Zuweisungsoperator verwechselt!

Ausdruck mit algebraischem Operator	Ausdruck mit Operator in Java	Bedeutung
x = y	x == y	x ist gleich y
x ≠ y	x != y	x ist ungleich y
x > y	x > y	x ist größer als y
x < y	x < y	x ist kleiner als y
x ≥ y	x >= y	x ist größer gleich y
x ≤ y	x <= y	x ist kleiner gleich y

Kontrollstrukturen

Kontrollstrukturen erlauben es, den Programmablauf durch **Verzweigungen und Wiederholungen** zu steuern

Java kennt verschiedene Verzweigungen (if-Verzweigung, Switch-Case-Verzweigung), verschiedene Schleifen (while-Schleife, for-Schleife, do/while-Schleife), sowie rekursive Methoden

Die if-Verzweigung

Die if-Verzweigung ist eine Anweisung, die abhängig von einer Bedingung zwischen **unterschiedlichen Anweisungsfolgen** auswählt

Ist die Bedingung wahr, wird der Anweisungsblock direkt nach der Bedingung ausgeführt, ansonsten wird der Anweisungsblock nach **else** ausgeführt

Der else-Zweig ist optional, kann also weggelassen werden

```
public class Listing0701 {

   public static void main(String[] args) {

      int a = 3, b = 4, c;

      if (a > b) {
         c = a - b;
      } else {
         c = b - a;
      }

   }

}
```

Die if-else-if-Leiter

Mehrere Alternativen von Fällen können mit Hilfe der if-else-if-Leiter abgebildet werden

Die if-else-if-Leiter verschachtelt mehrere if-Anweisungen zu einer sogenannten kaskadierten Verzweigung

```
public class Listing0702 {

   public static void main(String[] args) {

      int age = 34;

      if (age > 59) {
         System.out.println("Senior");
      } else if (age > 17) {
         System.out.println("Erwachsener");
      } else if (age > 9) {
         System.out.println("Jugendlicher");
      } else {
         System.out.println("Kind");
      }

   }

}
```

Aufgabe 4 – Verzweigungen

Schreiben Sie ein Programm, welches die 4 Grundrechenoperationen beherrscht.

- Hier können Sie die Aufgabe 3 zur Hilfe nehmen
- Datentyp für Operator: char
- Einlesen eines char: scanner.next().charAt(0);

```
Bitte geben Sie eine Zahl ein:
10
Bitte den Operator eingeben:
*
Bitte eine weitere Zahl eingeben:
5
Berechnung: 10 * 5 = 50
```

Aufgabe 4 – Verzweigungen

```java
public static void main(String[] args) {

    //Scanner erzeugen
    Scanner scanner = new Scanner(System.in);

    //Zahlen abfragen
    System.out.println("Bitte geben Sie eine Zahl ein:");
    int value1 = scanner.nextInt();
    System.out.println("Bitte den Operator eingeben:");
    char operator = scanner.next().charAt(0);
    System.out.println("Bitte geben Sie eine weitere Zahl ein:");
    int value2 = scanner.nextInt();
    scanner.close();
```

```java
//Berechnung
int result;
if(operator == '+') {
    result = value1 + value2;
} else if (operator == '-') {
    result = value1 - value2;
} else if (operator == '*') {
    result = value1 * value2;
} else {
    result = value1 / value2;
}

//Ausgabe
System.out.println("Ergebnis: " + result);
```

Wiederholungen

Sich wiederholende Tätigkeiten immer wieder auszuführen ist ein wesentlicher Bestandteil der Programmierung

Wiederholungen können mit Hilfe von **Schleifen** (Iterationen) oder Selbstaufrufen (Rekursionen) realisiert werden

Die while-Schleife

Bei der while-Schleife wird eine bestimmte Anweisungsfolge (Schleifenrumpf) wiederholt **solange eine bestimmte Bedingung** (Schleifenbedingung) wahr ist

Eine while-Schleife benötigt stets eine Anweisung, die dazu führt, dass die Schleifenbedingung falsch wird (Gefahr der Endlosschleife)

Da zu Beginn der Schleife die Bedingung geprüft wird, spricht man auch von einer **kopfgesteuerten Schleife**

```
public class Listing0705 {

   public static void main(String[] args) {

      int i = 0;

      while (i < 10) {
         System.out.println(i);
         i++;
      }

   }

}
```

Die do/while-Schleife

Im Gegensatz zur while-Schleife wird bei der do/while-Schleife der Schleifenrumpf **immer mindestens einmal durchlaufen**

Da die Bedingung hier am Ende der Schleife geprüft wird, spricht man hier von einer fußgesteuerten Schleife

```
public class Listing0707 {

   public static void main(String[] args) {

      int i = 0;

      do {
         System.out.println(i);
         i++;
      } while (i < 10);

   }

}
```

Die for-Schleife

Bei der for-Schleife handelt es sich um eine indexgesteuerte Schleife, auch **Zählschleife** genannt

Durch den Index wir bereits zu Schleifenbeginn festgelegt, wie oft die Schleife durchlaufen wird

```
public class Listing0706 {

   public static void main(String[] args) {

      for (int i = 0; i < 10; i++) {
         System.out.println(i);
      }

   }

}
```

Abbruchbefehle für Schleifen

Die von Mehrfachverzweigungen bekannte **break-Anweisung** sorgt dafür, dass eine Schleife ungeachtet der Bedingung komplett verlassen wird

Mit der Anweisung **continue** wird der aktuelle Schleifendurchgang abgebrochen und die Schleife wird mit dem nächsten Durchlauf fortgeführt

```
public class Listing0708 {

   public static void main(String[] args) {

      for (int i = 0; i < 10; i++) {

         if (i == 6) {
            break;
         }

         if (i % 2 == 0) {
            continue;
         }

         System.out.println(i);

      }

   }

}
```

Aufgabe 5 – Schleifen

Schreiben Sie ein Programm, welches von 1 – 10 zählt und diese in der Konsole in einer Zeile ausgibt

```
1 2 3 4 5 6 7 8 9 10
```

Aufgabe 5 – Schleifen

```java
//********do/while-Schleife******************
int counter = 1;
do {
    System.out.print(counter+" ");
    counter = counter + 1;
}while(counter <= 10);
```

```java
//*********while-Schleife*********************
int counter1 = 1;
while(counter1 <= 10) {
    System.out.print(counter1+" ");
    counter1++;
}
```

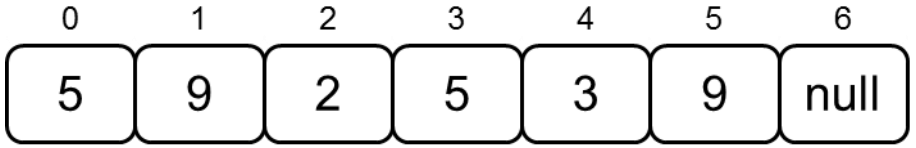
```java
//**********for-Schleife**********************
for(int counter2 = 1; counter2 < 11 ; counter2++) {
    System.out.print(counter2 + " ");
}
```

Felder

Felder

Wenn eine **große Menge an Daten** verarbeitet werden soll, greift man auf spezielle Datenstruktur-Variablen, die sogenannten Felder (*Arrays*), zurück

Die einzelnen „Speicherplätze“ in einem Feld werden als Elemente bezeichnet, die über einen **Index** direkt angesprochen werden können



Erzeugung von Feldern

Bei der Erzeugung muss immer die Länge des Feldes (d.h. die Anzahl der Elemente) angegeben werden

Jedes Feld verfügt über das Attribut length, welches die Länge des Feldes enthält

```
public class Listing0801 {

   public static void main(String[] args) {

      int[] numbers1 = new int[5];
      int[] numbers2 = { 9, 66, 0, 2, 35 };

   }

}
```

Zugriff auf die Elemente eines Feldes

Der Zugriff auf die Elemente eines Feldes erfolgt über die Angabe des entsprechenden Index

Hinweis: der **Index beginnt bei Java bei 0**!

```
public class Listing0802 {

   public static void main(String[] args) {

      int[] numbers = { 9, 66, 0, 2, 35 };

      for (int i = 0; i < numbers.length; i++) {
         System.out.println(numbers[i]);
      }

   }

}
```

Die Klasse ArrayList

Die Klasse ArrayList stellt eine **Liste** auf Basis eines Feldes dar

Die Länge einer Liste ist im Gegensatz zu einem Feld dynamisch

```
import java.util.ArrayList;

public class Listing0805 {

   public static void main(String[] args) {

      ArrayList<Integer> numbers = new ArrayList<>();

      for (int i = 0; i < 5; i++) {
         numbers.add(i);
         System.out.println(numbers.get(i));
      }

   }

}
```

Die for-each-Schleife

Die for-each-Schleife ermöglicht das **elementweise durchlaufen** von Feldern

Hinweis: mit der for-each-Schleife können Werte **nur gelesen**, nicht geändert werden!

```
public class Listing0803 {

   public static void main(String[] args) {

      int[] numbers = { 9, 66, 0, 2, 35 };

      for (int i : numbers) {
         System.out.println(i);
      }

   }

}
```

Aufgabe 6 – Felder

Schreiben Sie ein Programm, welches

- Zweierpotenzen von 2^0 bis 2^7 berechnet
- in einem Feld speichert
- auf dem Bildschirm ausgibt.

→ Berechnung einer Potenz:
(int) Math.pow(basis, hochzahl);

```
Zweierpotenzen:
1
2
4
8
16
32
64
128
```

Aufgabe 6 – Felder

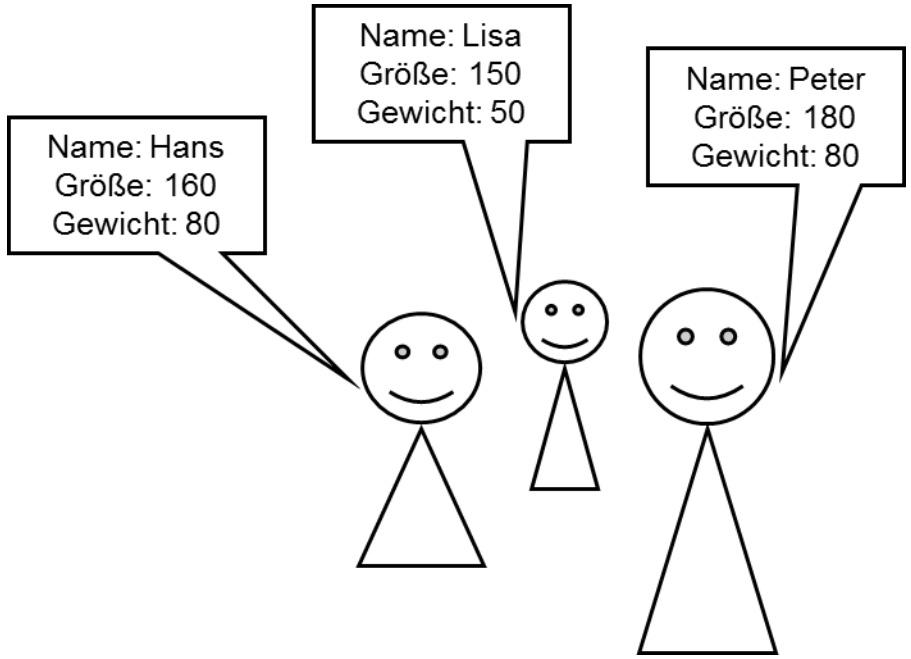
```java
public class Aufgabe6 {

    public static void main(String[] args) {

        //Array anlegen
        int[] results = new int[8];

        //Array füllen
        for(int i = 0; i < results.length; i++) {
            results[i] = (int) Math.pow(2, i);
        }

        //Ausgabe
        System.out.println("Zweierpotenzen:");
        for(int result: results) {
            System.out.println(result);
        }

    }
```

Objektorientierung

Die Grundidee der Objektorientierung

Die reale Welt besteht aus **Objekten** mit individuellen **Eigenschaften** und individuellem **Verhalten**

Für ein einfacheres **Verständnis** werden Objekte kategorisiert, also zu sinnhaften Einheiten verbunden



Begrifflichkeiten in der Objektorientierung

Eine Kategorie von ähnlichen Objekten bezeichnet man als **Klasse**

Konkrete Ausprägungen bzw. Instanzen einer Klassen werden wiederum als **Objekte** bezeichnet

Die Eigenschaften von Objekten werden als **Attribute**, das Verhalten als **Methoden** bezeichnet

Definition von Klassen und Attributen

Klassen werden in Java mit dem Schlüsselwort **class** definiert

Attribute einer Klasse werden dem gleichen Prinzip wie Variablen definiert

Die Angabe des Zugriffsrecht legt die Sichtbarkeit des Attributs fest

```
public class Animal {

   // Attribute
   private String name;
   private float size;
   private float weight;
   private int age;

}
```

Aufgabe 7 – Klassen

Erstellen Sie die Klasse Plant mit den rechts stehenden Attributen.

Attribut	Datentyp	Sichtbarkeit
type	String	private
size	float	private
isToxic	boolean	private

Aufgabe 7 – Klassen

```
public class Plant {

    private String type;
    private float size;
    private boolean isToxic;

}
```

Methoden

Eine Methode stellt das Verhalten eines Objekts dar, Methoden können Änderungen am Objekt durchführen oder andere Anweisungen ausführen.

Eine Methode besteht aus einem **Methodennamen**, einer Liste von **Eingabeparameter** (optional), einem **Rückgabewert**, sowie dem Methodenrumpf

Rückgabewerte bei Methoden

Methoden können entweder genau einen **Rückgabewert oder keinen Rückgabewert** besitzen

Methoden mit genau einem Rückgabewert müssen vor dem Methodennamen den Datentyp des Rückgabewerts angeben und am Ende des Methodenrumpfes immer die Anweisung **return** besitzen

Methoden ohne Rückgabewert müssen mit dem Schlüsselwort **void** deklariert werden

```
public class Animal {
…
   // Methoden
   public void setAttributes(String name, float size,
      float weight, int age) {
      this.name = name;
      this.size = size;
      this.weight = weight;
      this.age = age;
   }

   public void displayAttributes() {
      System.out.print(name + " - " + size + " - "
         + weight + " - " + age);
   }

   public String getName() { return name; }
   public float getSize() { return size; }
   public float getWeight() { return weight; }
   public int getAge() { return age; }

}
```

Aufgabe 8 – Methoden

Erweitern Sie die Klasse Plant aus Aufgabe 7
um die nachfolgenden Methoden.

Methode	Rückgabewert	Sichtbarkeit	Beschreibung
setAttributes(String, float, boolean)	void	public	Setzen aller Attribute
displayAttributes()	void	public	Ausgabe aller Attribute
toxic()	boolean	public	Rückgabe der Information, ob Pflanze giftig ist oder nicht

Aufgabe 8 – Methoden

```java
public class Plant {

    private String type;
    private float size;
    private boolean isToxic;

    public void setAttributes(String pType,
            float pSize, boolean pToxic) {

        type    = pType;
        size    = pSize;
        isToxic = pToxic;
    }

    public void displayAttributes() {
        System.out.println("Typ: " + type);
        System.out.println("Größe: " + size);
        System.out.println("Giftig: " + isToxic);
    }

    public boolean toxic() {
        return isToxic;
    }

}
```

Deklaration und Erzeugen von Objekten

Analog zu den elementaren Datentypen können auch für Klassen Variablen – sogenannte **Referenzvariablen** – definiert werden

Beim Erzeugen eines Objekts mit Hilfe des new-Operators wird der bei der Deklaration reservierte Speicherplatz durch das Objekt belegt

Hinweis: nach dem new-Operator muss immer ein Konstruktor der Klasse stehen!

```
public class Listing0903 {

   public static void main(String[] args) {

      Animal a1 = new Animal();
      Animal a2 = new Animal();
      Animal a3 = new Animal();

   }

}
```

Zugriff auf Attribute und Aufruf von Methoden

Erlauben die Zugriffsrechte den Zugriff auf ein Attribut, kann über die deklarierte Referenzvariable und einem **nachgestellten Punkt auf das Attribut zugegriffen** werden

Auch sichtbare Methoden werden über diese Syntax aufgerufen

```
public class Listing0905 {

   public static void main(String[] args) {

      Animal a1 = new Animal();

      a1.setAttributes("Hans", 6f, 750f, 50);
      String name = a1.getName();
      float size = a1.getSize();
      float weight = a1.getWeight();
      int age = a1.getAge();
      a1.displayAttributes();

   }

}
```

Aufgabe 9 – Objekte

Schreiben Sie ein ausführbares Programm, welches

- mehrere Pflanzen erzeugt
- diesen Werte zuweist
- und die Werte aller Attribute auf der Konsole ausgibt.

```
Typ: Eiche
Größe: 3.5
Giftig: false

Typ: Sonnenblume
Größe: 1.0
Giftig: false

Typ: Roter Fingerhut
Größe: 0.1
Giftig: true
```

Aufgabe 9 – Objekte

```java
public class MyPlant {

    public static void main(String[] args) {

        //Objekt erzeugen und Methode aufrufen
        Plant plant1 = new Plant();
        plant1.setAttributes("Eiche", 3.5f, false);
        plant1.displayAttributes();

        //zweite Pflanze
        Plant plant2 = new Plant();
        plant2.setAttributes("Sonnenblume", 1.0f, false);
        plant2.displayAttributes();

        //dritte Pflanze
        Plant plant3 = new Plant();
        plant3.setAttributes("Roter Fingerhut", 0.1f, true);
        plant3.displayAttributes();
    }
}
```

Zufallszahl

Die Klasse *Random* im Paket *java.util* stellt Methoden zur Verfügung, um Werte bestimmter Datentypen zufällig zu bestimmen

- Zufällige Ganzzahl:
  - random.nextInt();

```
import java.util.Random;

public class Aufgabe1 {

    public static void main(String[] args) {

        Random random = new Random();
        int zufallszahl = random.nextInt();

    }

}
```

Abschlussaufgabe – Lottospiel

Schreiben Sie ein ausführbares Programm zum Spielen von Lotto.

- Lesen Sie dazu 6 Werte vom Benutzer ein und speichern diese in einem Feld
- Ziehen sie 6 Zufällige Zahlen mit der Klasse Random und speichern diese Ebenfalls in einem Feld
- Vergleichen Sie die zwei Felder miteinander und ermitteln Sie die Treffer

Thank you.

Cafer Ucarli
Vocational Training Student

SAP SE
T +49 62277 59555
M +49 171 1901745
E cafer.ucarli@sap.com

Patrik Balazs
Vocational Training Student

SAP SE
T +49 62277759533
M +4917621243682
E patrik.balazs@sap.com