

# Zusammenfassung

## 1. JAVA: Aufbau eines Programms und Syntax

### 1.1 Grundlegender Aufbau eines JAVA-Programms

Ein JAVA-Programm besteht aus (mindestens) einer Klasse. Innerhalb dieser Klasse brauchen wir die `main()`-Methode. Diese wird beim Starten des Programms aufgerufen.

```
class Name_Der_Klasse {  
    public static void main(String [] args) {  
        [...]  
    }  
}
```

Listing 1: Aufbau einer JAVA-Klasse

### 1.2 Befehle

Jeder auszuführende Befehl muss in JAVA mit einem \_\_\_\_\_ abgeschlossen werden. Dagegen werden bei Verzweigungen, Schleifen und Methoden mehrere Befehle in Blöcken, welche mit \_\_\_\_\_ eingeschlossen werden, zusammengefasst.

### 1.3 import

JAVA ist grundsätzlich modular in Paketen bzw. *packages* aufgebaut. Wollen wir Befehle und Objekte nutzen, die nicht im „Standardpaket“ von JAVA sind, so müssen wir die entsprechenden packages mit `import` einbinden. (s. beispielsweise *Eingabe über die Konsole*)

*Wichtig:* diese `import`-Befehle müssen **vor** der Klasse (`class Name { [...] }`) geschrieben werden!

### 1.4 Kommentare

Um Code verständlicher zu machen können in JAVA *Kommentare* benutzt werden. Diese werden von JAVA komplett ignoriert und können vom Programmierer dazu genutzt werden, Methoden und Befehle zu beschreiben.

Es gibt zwei Arten von Kommentaren. Beschreibe kurz den Unterschied:

```
// [...] _____  
_____  
  
/* [...] */ _____  
_____
```

## 2. Variablen

Eine Variable ist ein Speicherplatz für Daten, die im Programm verwendet werden können. Bei der *Deklaration* gibt man der Variable einen Typ und einen Namen. Der *Datentyp* gibt an, welche Art von Werten in der Variable gespeichert werden über den der Variablenwert später wieder abgerufen und verändert werden kann.

### 2.1 Variablentypen

Beschreibe zunächst die verschiedenen Variablentypen und gib – sofern bekannt – bei den Datentypen für Zahlen den Zahlbereich an den diese Typen aufnehmen können:

`int` \_\_\_\_\_

`float` \_\_\_\_\_

`double` \_\_\_\_\_

`boolean` \_\_\_\_\_

`char` \_\_\_\_\_

`String` \_\_\_\_\_

*Hinweis:* Neben diesen Standarddatentypen gibt es noch viele weitere und wir können uns auch selbst neue Datentypen erstellen.

### 2.2 Deklaration und Initialisierung

Erzeuge eine Variable `jahr`, die einen ganzzahligen Wert speichern kann und weise ihr den Wert `2016` zu:

```
// Variable erstellen

// Wert zuweisen

/*-----*/
```

Listing 2: Deklaration und Initialisierung einer Variablen

### 3. Ausgabe auf der Konsole

Damit Ergebnisse auch angezeigt werden, müssen wir diese natürlich ausgeben lassen. Die einfachste Ausgabe ist auf der Konsole, hierzu gibt es den Befehl:

```
// Befehl zur Ausgabe
```

$$\text{---}^{\ast}\text{---}^{\ast}\text{---}$$

Listing 3: Ausgabe auf der Konsole

**Tipp:** in Eclipse gibt es hierzu den Kurzbefehl \_\_\_\_\_ gefolgt von der Tastenkombination [Strg]+[Leertaste].

Mit dieser Methode können wir auch mehrere Texte und Variablen hintereinander ausgeben. Hierzu müssen wir diese mit einem \_\_\_\_\_ verbinden.

#### 4. Eingabe über die Konsole

Die einfachste Möglichkeit, wie wir Benutzereingaben von der Konsole einlesen können, ist mit einem **Scanner**-Objekt.

Hierfür müssen wir zunächst das Paket \_\_\_\_\_ – wie im Punkt 1.3 beschrieben – importieren.

Anschließend erzeugen wir uns ein Objekt vom Typ `Scanner` und können dann die Benutzereingabe in eine Variable einlesen:

```
// Scanner-Objekt erstellen
```

```
// Einlesen einer ganzen Zahl
```

$$\text{---}^{\ast}\text{---}^{\ast}\text{---}$$

### Listing 4: Eingabe über die Konsole

*Hinweis:* neben ganzen Zahlen können auch andere Datentypen eingelesen werden. Hierfür gibt es jeweils eine entsprechende Methode des `Scanner`-Objektes.

## 5. Verzweigungen

Verzweigungen dienen dazu, bestimmte Befehle bzw. Programmteile nur unter bestimmten Voraussetzungen auszuführen.

Wir haben bisher die **if-else**-Verzweigungen kennengelernt. Diese stellt eine einfache wenn-dann-sonst-Beziehung her: **Wenn** eine Bedingung erfüllt ist **dann** wird der erste Block ausgeführt **sonst** wird der zweite Block ausgeführt.

### 5.1 Bedingungen

Bedingungen können hierbei z. B. einfache numerische Vergleiche sein:

`a == b` \_\_\_\_\_

`a < b` bzw. `a > b` \_\_\_\_\_

`a <= b` bzw. `a >= b` \_\_\_\_\_

`a != b` \_\_\_\_\_

### 5.2 Logische Operatoren

Mehrere Bedingungen können mit logischen Operatoren miteinander verknüpft werden:

`a && b` \_\_\_\_\_

`a || b` \_\_\_\_\_

*Wichtig hierbei:* da oft die Reihenfolge der Überprüfungen eine Rolle spielt können wir diese mithilfe von Klammern (wie in der Mathematik) beeinflussen.

### 5.3 Beispiel

Ergänze das Beispiel:

```
// Wenn Variable "jahr" größer oder gleich 2015

// dann Ausgabe "Zukunft"

// sonst Ausgabe "Vergangenheit"

/*-----*/
```

Listing 5: **if-else**-Verzweigung

## 6. Methoden

Methoden sind vergleichbar mit mathematischen Funktionen: beim Aufruf übergibt man ihnen bestimmte *Parameter* und die Methoden lösen dann definierte Teilaufgaben. Das Ergebnis dieser Teilaufgaben kann beispielsweise eine Bildschirmausgabe oder ein *Rückgabewert* sein.

Eine Methode wird dazu benutzt, um wiederkehrenden Code zu *kapseln* und den Programmaufbau damit logisch zu strukturieren. Die Algorithmen müssen so nur ein einziges Mal programmiert werden und können immer wieder verwendet werden.

### 6.1 Aufbau einer Methode

Eine Methode sieht im Allgemeinen wie folgt aus:

```
public static boolean istSchaltjahr(int jahr) {  
    if (((jahr%4)==0)&&(((jahr%100)!=0)||((jahr%400)==0))) {  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

Listing 6: Aufbau einer Methode

Erkläre kurz die folgenden Teile davon:

**public:** die Methode ist von überall aufrufbar (genaueres später bei der Objektorientierung)

**static** die Methode ist ohne Objekt (genaueres später bei der Objektorientierung)

**boolean** \_\_\_\_\_

**istSchaltjahr** \_\_\_\_\_

**(int jahr)** \_\_\_\_\_

**{ [...] }** *Methodenrumpf* \_\_\_\_\_

**return [...];** \_\_\_\_\_