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 `2015` zu:

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:

Listing 3: Ausgabe auf der Konsole

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
```

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 diesen 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 Beispiel

Ergänze das Beispiel:

```
// Wenn Variable "jahr" größer oder gleich 2018

// 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

Erkäre 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 [...];` _____

7. Schleifen

Beschreibe, wozu man Schleifen verwendet und welche 2 Schleifenarten wir benutzt haben: _____

1. Schleifenart: _____

2. Schleifenart: _____

8. Arrays

Unter einem Array in Java versteht man einen „Container“, vergleichbar mit einem Schubladenschrank, der in der Lage ist, mehrere Objekte _____ aufzunehmen und zu verwalten.
Wir haben zwei unterschiedliche Möglichkeiten kennengelernt um Arrays zu erzeugen:

8.1 Deklaration und direkte Initialisierung

Vervollständige die Zeile um ein Array mit 5 Elementen zu erzeugen, das die Zahlen 1 bis 5 enthält:

```
int meinArray =
```

Listing 7: Arrays: direkte Initialisierung

Der Nachteil an dieser Möglichkeit ist, _____

_____

_____

8.2 Deklaration ohne Initialisierung

Um den oben genannten Nachteil zu umgehen können wir ein Array auch ohne Initialisierung deklarieren, beispielsweise mit Platz für 100 Werte:

```
int meinArray =
```

Listing 8: Arrays: Deklaration ohne Initialisierung

8.3 Arbeiten mit Arrays

Mit dem Ausdruck _____ können wir dann auf das dritte Element des Arrays zugreifen.
Achtung: der *Index* beginnt bei _____!
Die Länge eines Arrays `meinArray` können wir mit dem Ausdruck _____ bestimmen.