PSE Vorkurs Tag 3

Feedback

Bitte füllt den Feedbackbogen aus, damit wir den Vorkurs für die nächsten Jahre verbessern können. Der Bogen ist anonym und dauert nur 2-3 Minuten.
Ihr findet ihn unter: `https://forms.gle/BxZRFrbm8K7jPtMi6`

Musterlösungen

Tag 2 Lösungen

Präsentations-Cheatsheet

Funktionen und Scopes:

- Funktionen helfen Code auszulagern und mehrfach zu nutzen

```java
public static double avg(double a, double b, double c) {
    return (a + b + c) / 3;
}
```

- `return` gibt einen Wert zurück und beendet die Funktion
- Scopes: Variablen nur im jeweiligen Block sichtbar

```java
int x = 5;
if (x > 0) {
    int y = 10; // y nur hier sichtbar
}
System.out.println(y); // Fehler!
```

Arrays:

- Arrays speichern mehrere Werte gleichen Typs
- Zugriff über Index, Start bei `0`

```java
int[] zahlen = new int[5];
zahlen[0] = 42;
System.out.println(zahlen[0])
```

```
42
```

- Initialisierung auch direkt möglich:

```java
String[] grill = {"Wurst", "Steak", "Mais"};
```

- Mit `.length` über alle Elemente iterieren:

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

Aufgabe 1: Einfache Funktionen Erstellen

1. Erstelle eine Funktion ohne Parameter und ohne Rückgabewert, die den Anfang der Grillung ankündigt:

   ```
   Es ist Grillereizeit meine Freunde!
   ```

   Hinweis: Wenn eine Funktion nur erstellt wird, dann wird sie noch nicht ausgeführt. Sie muss zusätzlich in der `main`-Funktion aufgerufen werden.

2. Erstelle noch eine Funktion, die selbst nichts ausgibt, aber dafür den obigen Text als `String` zurückgibt. Nutze diese neue Funktion um die gleiche Ausgabe wie oben in Aufgabe 1 zu erstellen. Zusätzlich kannst du versuchen die Aufgabe ohne Variable zu lösen.

3. Erstelle eine dritte Funktion, die einen Namen als Parameter entgegennimmt und die Person herzlich zur Grillerei einläd:

   ```
   Komm ran Melanie, es gibt Grillung!
   ```

4. Erstelle eine Funktion, die zwei Zahlen addiert und das Ergebnis der Addition zurückgibt, hierfür aber nur den ++ Operator verwendet (also kein + - * / etc.). Als Erinnerung an den ++ Operator:

   ```java
   int a = 3;
   a++;
   System.out.println(a);
   ```

   ```
   4
   ```

5. Nutze die Additionsfunktion um eine neue Funktion zu erstellen, die zwei Zahlen multipliziert, im Code dieser Funktion dürfen keine Mathematischen Operatoren vorkommen.

6. Nutze die Multiplikationsfunktion um Fakultät zu implementieren.
   Hinweis: Beispielsweise wird die Fakultät von 6 so geschrieben: 6! = 6*5*4*3*2*1

Aufgabe 2: Arrays

1. Erstelle ein Array mit 10 `int`-Werten und versuche das Array auszugeben.

2. Da dies nicht zu klappen scheint, programmiere eine Funktion, die ein Array entgegennimmt und es anschaulich in der Konsole ausgibt.
   Hinweis: Erinnere dich an `Array.length` aus der Präsi.

3. Programmiere eine neue Ausgabe-Funktion, die statt den normalen listenartigen Arrays ein 2-Dimensionales Array entgegennimmt und das gesamte Array anschaulich ausgibt.
   Hinweis: 2-Dimensionale Arrays funktionieren so:

```java
int[][] einArray = new int[3][3]; //ein zweidimensionales Array der Größe 3x3
einArray[0][2] = 3;
System.out.println(einArray[0][2]);
```

```
3
```

4. Programmiere eine Funktion, die eine Seitenlänge als Parameter entgegennimmt und daraus ein quadratisches 2-D Array erstellt. Das Array soll bis auf einen diagonalen Streifen aus Nullen komplett aus Einsen bestehen (siehe Ausgabe). Teste dein Ergebnis mit der Printfunktion die du in Aufgabe 3 programmiert hast.
   Seitenlänge 6:

```
0 0 0 1 1 1
0 0 0 0 1 1
0 0 0 0 0 1
1 0 0 0 0 0
1 1 0 0 0 0
1 1 1 0 0 0
```

   Seitenlänge 10:

```
0 0 0 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1 1
0 0 0 0 0 1 1 1 1 1
1 0 0 0 0 0 1 1 1 1
1 1 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 1 1
1 1 1 1 0 0 0 0 0 1
1 1 1 1 1 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0
1 1 1 1 1 1 1 0 0 0
```

   Wenn du an dieser Aufgabe Spaß hattest gibt es im Codeanhang noch mehr Muster die du versuchen kannst zu generieren.

5. Programmiere eine Funktion, die eine Zahl n entgegennimmt und ein Array mit den ersten n Zahlen der Fibonacci-Reihe zurückgibt. Teste dein Programm mit der Array-Printfunktion die du oben Programmiert hast.
   Hinweis: Die Fibonacci Reihe fängt mit 1, 1 an und jedes darauffolgende Element ist die Summe der Beiden vorigen Elemente: 1, 1, 2, 3, 5, 8, 13, ...

HIGHPERFORMER-Aufgabe: Sortieren und Performance-Analyse

Java bietet eine Funktion um Zeit akkurat zu Messen: `System.nanoTime()` gibt die Java-interne Zeitmessung in Nanosekunden (10^{-9} Sekunden) aus.

1. Miss die durchschnittliche Laufzeit eines prints.
   Hinweis: Es ist sinnvoll bei einer Laufzeitanalyse die zu messende Funktion mehrmals durchzuführen und mit dem Durchschnitt der Ergebnisse zu arbeiten.

2. Erstelle ein Array der Länge 100 und initialisiere es mit zufälligen Werten zwischen 0 und 99.
   Hinweis: `https://docs.oracle.com/javase/8/docs/api/java/util/Random.html`

3. Überlege dir ein Verfahren zum Sortieren von Arrays und programmiere eine Funktion, die das zufällige Array aus Higherformer-Aufgabe 2. als Parameter entgegennimmt und es sortiert zurückgibt. Miss die durchschnittliche Laufzeit deiner Funktion.

4. Versuche deine Funktion zu optimieren und somit schneller zu machen. Wir (die Vorkurs-Orgas) haben unkreativ wie wir sind einen bekannten Sortieralgorithmus (Bubblesort) implementiert, vielleicht bekommst du es hin unsere Zeit zu schlagen. Unser Algorithmus ist im Codeanhang damit du dessen Laufzeit lokal bei dir testen kannst.
   Hinweis: Bei der Optimierung kann es sehr hilfreich sein nicht nur die gesamte Laufzeit zu analysieren, sondern auch einzelne Teile des Codes, um rauszufinden wo am meisten Zeit eingespart werden kann.

5. Informier dich im Internet über unterschiedliche Sortieralgorithmen und versuche einen der dir gefällt zu implementieren. Natürlich gehört da auch wieder eine Laufzeitanalyse dazu.

Codeanhang

Aufgabe 2.4:
Alle gezeigten Arrays haben Seitenlänge 13, aber deine Funktion sollte die Arrays natürlich je nach Eingabegröße generieren.

```
1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0 1
```

```
1 1 1 1 1 1 0 1 1 1 1 1 1
1 1 1 1 1 0 0 0 1 1 1 1 1
1 1 1 1 0 0 0 0 0 1 1 1 1
1 1 1 0 0 0 0 0 0 0 1 1 1
1 1 0 0 0 0 0 0 0 0 0 1 1
1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 1
1 1 0 0 0 0 0 0 0 0 0 1 1
1 1 1 0 0 0 0 0 0 0 1 1 1
1 1 1 1 0 0 0 0 0 1 1 1 1
1 1 1 1 1 0 0 0 1 1 1 1 1
1 1 1 1 1 1 0 1 1 1 1 1 1
```

```
1 1 0 1 1 1 1 0 1 1 1 1 0
1 0 0 0 1 1 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 1 1 0 0
1 1 0 1 1 1 1 0 1 1 1 1 0
1 1 0 1 1 1 1 0 1 1 1 1 0
1 0 0 0 1 1 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0 0 1 1 0 0
1 1 0 1 1 1 1 0 1 1 1 1 0
1 1 0 1 1 1 1 0 1 1 1 1 0
1 0 0 0 1 1 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
```

Higherformer-Aufgabe 3.:

```java
public static int[] bubbleSort(int[] arr) {
    int n = arr.length;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    return arr;
}
```