Assignment 1

Abgabe bis zum 20. April 2021 (Dienstag)um 23:59 Uhr

Geben Sie Ihr Assignment bis zum 20. April 2021 (Dienstag) um 23:59 Uhr auf folgender Webseite ab:
https://assignments.hci.uni-hannover.de
Ihre Abgabe muss aus einer einzelnen zip-Datei bestehen, die den Quellcode und ggf. ein PDF-Dokument bei Freitextaufgaben enthält. Beachten Sie, dass Dateien mit Umlauten im Dateinamen nicht hochgeladen werden können. Entfernen Sie die daher vor der Abgabe die Umlaute aus dem Dateinamen. Überprüfen Sie nach Ihrer Abgabe, ob der Upload erfolgreich war. Bei technischen Problemen, wenden Sie sich an Patric Plattner oder Dennis Stanke. Es wird eine Plagiatsüberprüfung durchgeführt. Gefundene Plagiate führen zum Ausschluss von der Veranstaltung. In dieser Veranstaltung wird ausschließlich die Java/JavaFX Version 11 verwendet. Code und Kompilate anderer Versionen sind nicht zulässig. Ihre Abgaben sollen die vorgegebenen Dateinamen verwenden. Wiederholter Verstoß gegen diese Regel kann zu Punktabzügen führen.

Aufgabe 1: Bubblesort

In dieser Aufgabe sollen Sie den Sortieralgorithmus Bubblesort in Java implementieren. Arbeiten Sie in einer selbst erstellten `Bubblesort.java` Datei, die eine `Bubblesort` Klasse beinhaltet.

a) Erstellen Sie zunächst eine `public static void main(String[] args)` Methode.

Erstellen Sie anschließend eine Methode `static String arrayToString(int[] arr)`, welche den Inhalt eines beliebigen Integer Arrays als String zurückgibt:

Beispiel:

```
int[] bsp = {1,2,3,4,5};
System.out.println(arrayToString(bsp));
```

Ausgabe:

```
[1, 2, 3, 4, 5]
```

b) Implementieren Sie eine Methode `static int[] randomArray(int n)`, die ein Integer-Array der Länge `n` zurückgibt. Dieses Array soll mit Zufallszahlen im Interval $[0, 99]$ gefüllt sein und noch nicht explizit sortiert sein. Falls Sie Hilfe brauchen um Zufallszahlen zu generieren, schauen sie in die Lösungshinweise zu dieser Aufgabe.

Erstellen Sie anschließend drei Arrays mit dieser Methode in der `main` Methode mit den Längen 10, 20 und 30 und geben Sie diese auf die Kommandozeile aus.

Beispiel:

```
int[] randArr = randomArray(10);
System.out.println(arrayToString(randArr));
```

Vorlesung: apl. Prof. Matthias Becker becker@hci.uni-hannover.de
Übung: Patric Plattner patric.plattner@hci.uni-hannover.de
Organisation: M.Sc. Dennis Stanke dennis.stanke@hci.uni-hannover.de

Ausgabe:

```
[43, 23, 73, 59, 9, 3, 21, 93, 22, 63]
```

c) Implementieren Sie die Methode `static void bubbleSort(int[] arr)`, die das gegebene Array mit dem Bubblesort Algorithmus **absteigend** sortiert. Der Bubblesort Algorithmus funktioniert wie folgt:

1. Durchlaufen Sie das Array aufsteigend.
2. Vergleichen Sie jedes Element (`i`) des Arrays mit dem Element, welches sich im Array direkt daneben befindet (`i+1`).
3. Ist das Element `i+1` größer als das Element `i`, dann tauschen Sie die beiden Elemente miteinander.
4. Wiederholen Sie die Schritte 1 bis 3 solange bis Sie einmal über das Array gegangen sind, ohne Elemente zu vertauschen.

Sortieren Sie nun die drei Arrays, die Sie in Teilaufgabe `b)` erstellt haben und geben Sie diese einmal vor dem Sortieren und einmal nach dem Sortieren auf die Konsole aus.

Beispiel:

```
int[] randArr = randomArray(10);
System.out.println(arrayToString(randArr));
bubbleSort(randArr);
System.out.println(arrayToString(randArr));
```

Output:

```
[89, 63, 19, 33, 62, 66, 83, 6, 51, 20]
[89, 83, 66, 63, 62, 51, 33, 20, 19, 6]
```

Bestehenskriterien

- Ihr Code kompiliert.
- Ihre `bubbleSort()` Methode sortiert Arrays korrekt **absteigend**.
- Erstellen und sortieren die Arrays so, wie sie in den jeweiligen Teilaufgaben vorgegeben sind.

Lösungshinweise

- Um Zufallszahlen zu generieren, können Sie ein Objekt der Klasse `java.util.Random` erstellen (`java.util.Random rand = new java.util.Random();`). Mit der Methode `nextInt(int bound)` erhalten Sie einen zufälligen Integer in dem Intervall $[0, \text{bound}[$ (also excl. `bound`). Sie müssen nicht für jede neue Zufallszahl ein neues `Random` Objekt erstellen. Für weitere Informationen schauen Sie sich die Methode in der Java 11 API an: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Random.html#nextInt(int)

Aufgabe 2: Primzahlensieb

In dieser Aufgabe sollen Sie das Sieb des Eratosthenes (https://de.wikipedia.org/wiki/Sieb_des_Eratosthenes) implementieren. Arbeiten sie in einer selbst erstellten `Primes.java` Datei, die eine `Primes` Klasse beinhaltet.

a) Erstellen Sie zunächst eine `public static void main(String[] args)` Methode.

Erstellen Sie anschließend eine Methode `static String arrayToString(int[] arr)`, welche den Inhalt eines beliebigen Integer Arrays als String zurückgibt:

Beispiel:

```
int[] bsp = {1,2,3,4,5};
System.out.println(arrayToString(bsp));
```

Ausgabe:

```
[1, 2, 3, 4, 5]
```

b) Implementieren Sie die Methode `static int[] fillArray(int n)`. Das von dieser Methode zurückgegebene Array soll von `2` an aufsteigend alle Zahlen bis `n` enthalten. Anders ausgedrückt enthält dieses Array somit alle Zahlen im Intervall $[2, n] \subset \mathbb{N}$.

Erstellen Sie in der `main` Methode ein Array, das alle Zahlen im Interval $[2, 25] \subset \mathbb{N}$ enthält, mithilfe der Methode und geben Sie dieses auf der Konsole aus.

Beispiel:

```
...
int[] array = fillArray(10);
System.out.println(arrayToString(array));
...
```

Ausgabe:

```
[2, 3, 4, 5, 6, 7, 8, 9, 10]
```

c) Implementieren Sie die Methode `static void filterArray(int[] arr, int index)`. Diese soll alle Vielfachen des Elements (außer das Element selbst) am angegebenen Index auf den Wert -1 ändern. Diese Methode muss nur auf Ausgaben der `fillArray(int n)` Methode funktionieren.

Streichen Sie alle Vielfachen der ersten drei Primzahlen aus dem Array der Teilaufgabe `b)` und geben Sie das Array danach nochmal auf die Konsole aus.

Beispiel:

```
...
int[] array = fillArray(20);
filterArray(array, 0);        /* array[0] == 2 */
System.out.println(arrayToString(array));
filterArray(array, 1);        /* array[1] == 3 */
```

```
System.out.println(arrayToString(array));
...
```

Ausgabe:

```
[2, 3, -1, 5, -1, 7, -1, 9, -1, 11, -1, 13, -1, 15, -1, 17, -1, 19, -1]
[2, 3, -1, 5, -1, 7, -1, -1, -1, 11, -1, 13, -1, -1, -1, 17, -1, 19, -1]
```

d) Implementieren Sie die Methode `static void primes(int n)`. Diese soll als Eingabe eine Zahl `n` bekommen und alle Primzahlen $\leq n$ in der Konsole ausgeben. Nutzen Sie für die Implementierung die Methoden, die Sie in den Teilaufgaben `b)` und `c)` bereits geschrieben haben.

Geben Sie mit dieser Methode alle Primzahlen bis 20 und alle Primzahlen bis 100 auf der Konsole aus.

Beispiel:

```
...
primes(15)
primes(50)
...
```

Ausgabe:

```
[2, 3, 5, 7, 11, 13]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Bestehenskriterien

- Ihr Code kompiliert.
- Ihre Ausgaben entsprechen den Ausgaben der Beispiele.
- Die Methoden aus Teilaufgaben `b)` und `c)` sind implementiert und funktionieren auch mit anderen Eingaben wie in der Aufgabe beschrieben.
- Ihre Methode aus der Teilaufgabe `d)` nutzt die Methoden aus den Teilaufgaben `b)` und `c)`.
- Sie implementieren die Beispielaufrufe in der `main` Methode, so wie sie in den Teilaufgaben beschrieben sind.

Aufgabe 3: Debugging

Jedes Assignment wird eine Debugging-Aufgabe enthalten. In diesen Aufgaben werden Sie einen fehlerhaften Java Codeabschnitt bekommen. Diese Fehler können **syntaktisch** oder **semantisch** sein. Es kann sich dabei um Compiler- oder Laufzeitfehler handeln. Sie dürfen den Code kompilieren und ausführen um die Fehler zu finden. Arbeiten Sie in der Template-Datei `Debug.java`. Diese finden Sie im Stud.IP. Nutzen Sie Kommentare, um die Fehler zu Kennzeichnen. Gegeben ist der folgende Codeabschnitt:

```
ckass Debug {
    // Method prints all numbers in an array that are even.
    static void 1printEvenNumbers(int[] numbers) {
        for (int i = 0; i < numbers.length; ++i) {
            if (numbers[i] \% 2 == 1) {
                System.out.println(numbers[i]);
            }
        }
    }
    int[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    //output should be as follows:
    //2
    //4
    //6
    //8
    Debug.printEvenNumbers(arr);
}
```

a) Der Codeabschnitt enthält zwischen 3-5 Fehler. Kompilieren und führen Sie den Code aus. Suchen Sie anhand der Fehlermeldungen des Compilers oder der Laufzeitfehlermeldungen Fehler im Code und korrigieren Sie diese. Kommentieren Sie am Ende der jeweiligen Zeile, was Sie korrigiert haben.

Erstellen Sie während der Fehlerkorrektur einen Blockkommentar unter dem Code, in dem Sie die Fehler dokumentieren. Schreiben Sie die Zeile(n) des Fehlers auf und beschreiben Sie den Fehler. Kopieren sie die Fehlermeldung, sofern es eine gab, anhand welcher Sie den Fehler erkannt haben. Die Dokumentation soll wie in folgendem Beispiel aussehen:

```
public class Debug { //K: class falsch geschrieben (ckass)

...

/*
...
Zeile 1: class Keyword falsch geschrieben (ckass):
Fehlermeldung:
**************
Debug.java:1: error: class, interface, or enum expected
public ckass Debug {
       ^
**************
Der Compiler erwartet eines der drei oben angegebenen Keywords, hat aber nur das falsch
    ↪ geschriebene ckass bekommen.
...
*/
```

Bestehenskriterien

- Ihr korrigierter Code funktioniert so wie in den Kommentaren im Code beschrieben.
- Alle korrigierten Fehler haben einen kurzen Kommentar, der beschreibt, warum die Stelle im alten Code nicht funktioniert hat.
- Alle Fehlermeldungen (Compiler und Laufzeit) sind in einem Blockkommentar `/* ... */` unter dem Code in der `Debug.java`.

Lösungshinweise

- Im Code vorgegebene Kommentare beschreiben immer das **korrekte** Verhalten des Programms.