

## **Hausaufgaben 3**

**23./24.10.2024**

Abgabe der Lösung am 29.10.2024

### **StringBuilder -Test**

Schreiben Sie eine Klasse `LaufzeitVergleichStrings`. Die Klasse enthält einen `String`, einen `StringBuilder` und zwei `long` mit den Namen `startTime` und `stopTime`. Wir möchten mit der Klasse vergleichen, wie viel Zeit einzelne Operationen mit einem `String` und einem `StringBuilder` benötigen. Um die Zeit zu messen, verwenden Sie `startTime = System.nanoTime();` vor der Operation und `stopTime = System.nanoTime();` nach der Operation. Mit `stopTime - startTime` kann dann die verstrichene Zeit in Nanosekunden gemessen werden.

- a) Konstruktor: `public LaufzeitVergleichStrings(String s)`  
`//initialisiert den String und den StringBuilder mit dem uebergebenen`  
`//String. Dabei soll fuer beide Zuweisungen die benoetigte Zeit`  
`//gemessen und auf der Konsole ausgegeben werden.`
- b) Schreiben Sie eine `main`, die folgende Zeilen beinhaltet:  
`LaufzeitVergleichStrings test;`  
`test = new LaufzeitVergleichStrings("Hallo");`  
Die Ausgabe soll folgendermaßen aussehen:  
`Constructor time for String: 1700ns`  
`Constructor time for StringBuilder: 28200ns`  
Hinweis: Die Zahlen werden sehr wahrscheinlich anders sein als die hier dargestellten.
- c) Schreiben Sie die Funktion: `public void setCharAt(int index, char value)`  
Fügen Sie der `main` die folgenden Zeilen hinzu:  
`test.setCharAt(0, 'h');`  
`test.setCharAt(3, 'o');`  
Dies soll zu folgender Ausgabe führen:  
`setCharAt() time for String: 11285100ns`  
`setCharAt() time for StringBuilder: 8300ns`  
`setCharAt() time for String: 8900ns`  
`setCharAt() time for StringBuilder: 600ns`
- d) Schreiben Sie die Funktion: `public String toString()`, die folgende Ausgabe erzeugen soll:  
`String: haloo`  
`StringBuilder: haloo`
- e) Schreiben und testen Sie weiterhin die Funktionen:  
`public void delete(int start, int end)`  
`public void deleteCharAt(int index)`  
`public insert(int offset, char value)`  
`public insert(int offset, char[] value)`  
`public append(char[] value)`

## Zahlenschungel

Im Zahlenschungel springen die Zahlen von Baum zu Baum. Jeder Sprung einer Zahl kostet Kraft und dekrementiert die Zahlen.

- Eine gerade Zahl springt einen Baum nach rechts und fällt eine Ebene nach unten. Dies dekrementiert die Zahl um 1.
- Eine ungerade Zahl springt drei Bäume nach rechts und fällt dabei eine Ebene nach unten. Dies dekrementiert die Zahl um 3.
- Eine gesprungene Zahl überschreibt nicht einfach die am Zielort vorhandene Zahl, sondern addiert sich zu dieser.
- Zahlen, die auf der untersten Ebene angekommen sind, springen nicht mehr weiter und bleiben stehen.
- Pro Tag springt jede Zahl im Dschungel, die größer als 1 ist, genau ein einziges Mal.
- Der Zahlenschungel ist zyklisch. Der Baum am weitesten links ist rechts vom Baum, der am weitesten rechts ist.
- Jede Zahl im Zahlenschungel ist positiv oder 0 und kleiner als 100.

Schreiben Sie die Klasse `Zahlenschungel` mit dem folgenden Konstruktor und den genannten Methoden:

a) Konstruktor: `public Zahlenschungel(int[][] arr)`

Der Konstruktor überprüft, ob das gegebene Array nicht ausgefranst ist.

Ein Dschungel mit nur einem Baum sieht zum Beispiel so aus:

```
int[][] singleTree = {{0,0,4,6,2,11}};
```

Ein Dschungel mit zwei Bäumen sieht zum Beispiel so aus:

```
int[][] twoTrees = {{0,0,4,6,2,11},{0,1,5,2,5,7}};
```

Somit gibt das `x` für `arr[x][y]` an, um welchen Baum es sich handelt, angefangen bei `x=0` mit dem Baum am weitesten links. Das `y` gibt die Ebene des Baums an, wobei angefangen bei `y=0` die unterste Ebene startet.

b) `public String toString()`

Die Ausgabe des obengenannten Beispiels `singleTree` soll folgendermaßen aussehen:

```
11
02
06
04
00
00
```

Die Ausgabe des obengenannten Beispiels `twoTrees` soll folgendermaßen aussehen:

```
11 07
02 05
06 02
04 05
00 01
00 00
```

c) **public** void simulateDay()

Ein Tag im Zahlenschungel soll simuliert werden. Achten Sie darauf, dass jede Zahl, die größer als 1 ist, nur ein einziges Mal springt. Das obengenannte Beispiel `twoTrees` soll nach einem Tag folgendermaßen aussehen:

```
00 00
04 08
02 01
01 05
02 04
00 00
```

d) **public** boolean simulationFinished()

Gibt `true` zurück, wenn ein weiterer Tag im Zahlenschungel zu keiner Veränderung führt, sonst `false`.

e) **public** static void main(String[] args) {

```
    int[][] twoTrees= {{0,0,4,6,2,11},{0,1,5,2,5,7}};
    Zahlenschungel test2 = new Zahlenschungel(twoTrees);
    System.out.println(test2);
    while(!test2.simulationFinished()) {
        test2.simulateDay();
        System.out.println(test2);
    }
}
```

soll folgende Ausgabe erzeugen:

```
11 07
02 05
06 02
04 05
00 01
00 00

00 00
04 08
02 01
01 05
02 04
00 00

00 00
00 00
07 04
01 01
02 00
03 01

00 00
```

00 00  
00 00  
04 05  
00 00  
03 02

00 00  
00 00  
00 00  
00 00  
02 03  
03 02

00 00  
00 00  
00 00  
00 00  
00 00  
03 03