Präsenzaufgaben 12 18.12./19.12.2019

Die Lösung der Aufgaben wird am Ende der Übung von Ihnen vorgestellt.

Aufgabe 1: Schreiben Sie eine Funktion, die einen `IntStream` mit den Potenzen von 2 zurückgibt, solange sie in eine int-Variable passen ($2^0 \ldots 2^{30}$).

```
public static IntStream powersOfTwo()
```

Folgende `Stream`-Methoden könnten nützlich sein: *`iterate, limit, range, map.`*

Aufgabe 2:

Testen Sie Ihre Funktionen mit einfachen JUnit-Tests (JUnit 4). Erstellen Sie einen neuen JUnit-Test in Eclipse mit *File → New → JUnit Test Case.*
Alle Methoden, vor denen die Annotation `@Test` steht, werden automatisch als Testfälle erkannt. Ersetzen Sie zunächst die automatisch erzeugte Methode durch:

```
@Test
public void multTest1() {
      IntStream test = MyStreams.powersOfTwo();  // MyStreams: Name Ihrer
                                                 // Stream-Klasse
      assertEquals(31, test.count());
}
```

Starten Sie Ihren Test mit *Run As../JUnit Test.* Beobachten Sie die Ausgabe auf der linken Seite. Der erste Parameter der `assertEquals`-Anweisung ist der erwartete Wert. Ändern Sie ihn ab und provozieren Sie damit eine Fehlermeldung.

Aufgabe 3: Schreiben Sie eine Funktion

```
public static void testStream(IntStream stream, int num)
```

die die ersten `num` Werte des übergebenen Streams auf dem Bildschirm ausgibt.

Folgende `Stream`-Methoden könnten nützlich sein: *`forEach.`*

Aufgabe 4: Schreiben Sie eine Funktion

```
public static int[] toArray(ArrayList<Integer> list)
```

die einen Stream verwendet, um eine `ArrayList<Integer>` in ein **`int`**-Array umzuwandeln.

Folgende Stream-Methoden könnten nützlich sein: *`list.stream, toArray, mapToInt.`*

Aufgabe 5: Schreiben Sie eine Funktion

```
public static double[] getRandNumbers(int cnt)
```

die einen Stream verwendet, um ein Feld mit *`cnt`* Zufallszahlen zwischen 0 (inkl.) and 1 (exkl.) zu erzeugen und zurückzugeben.

Folgende Stream-Methoden könnten nützlich sein: *`generate, limit.`*

Aufgabe 6: Sehen Sie sich Kapitel 3 des folgenden Dokuments an:

https://esb-dev.github.io/mat/junit.pdf

Es handelt sich um eine Kurzreferenz zu JUnit 4. Fügen Sie anschließend Ihrer Testklasse die folgenden Tests hinzu:

- Überprüfen Sie mit einem JUnit-Test, ob
  `public static double[] getRandNumbers(int cnt)`
  eine `ArithmeticException` wirft, wenn `cnt` negativ ist. Ändern Sie anschließend die Funktion so ab, dass der Test korrekt durchlaufen wird.
- Überprüfen Sie mit einem JUnit-Test, ob `getRandNumbers(int cnt)`
  tatsächlich ein Feld zurückgibt, in dem alle Elemente zwischen 0 (einschließlich) und 1 (ausschließlich) liegen.

Aufgabe 7: Schreiben Sie eine Funktion
`public static Stream<Character> getCharStream(String s)`

die einen Stream mit den Zeichen von `s` zurückgibt.
Folgende Stream-Methoden könnten nützlich sein: *range, mapToObj.*

Aufgabe 8: Schreiben Sie eine Funktion
`public static int getMax(int[] arr)`

die den größten Wert des Felds *arr* zurückgibt. Benutzen Sie dazu Java-Streams. Parallelisieren Sie Ihren Algorithmus.
Folgende Stream-Methoden könnten nützlich sein: *Arrays.stream, getAsInt, max, parallel*.

Aufgabe 9: Schreiben Sie eine Funktion
`public static int[] getPos(String x, char c)`

die ein Feld mit allen Positionen zurückgibt, an denen das Zeichen `c` im String `x` vorhanden ist. Parallelisieren Sie die Funktion und nutzen Sie dazu Java-Streams.

Beispiel: Ein Funktionsaufruf mit `x="banana"` und `c='a'` als Parametern sollte das Feld `[1,3,5]` zurückgeben.
Folgende Stream-Methoden könnten nützlich sein: *range, filter.*

Aufgabe 10: Schreiben Sie eine Funktion
`public static boolean isPrime(long z)`

die Streams benutzt, um zu ermitteln, ob `z` eine Primzahl ist. Parallelisieren Sie Ihren Algorithmus. Überprüfen Sie Ihr Programm mit der Zahl 100000000000000181 (eine Primzahl). Vergleichen Sie die Laufzeit der seriellen und der parallelen Version. Sie können *System.nanoTime()* für Ihre Zeitmessungen benutzen.

Folgende Stream-Methoden könnten nützlich sein: *noneMatch, range.*