TU Berlin
Software and Business Engineering
Prof. Dr. Ingo Weber
Dr. Tobias Heindel
Richard Hobeck

WiSe 2021/22
Aufgabe 10
Block 3
10. Januar 2022

Aufgabe 10 *Programmieren I*

Hinweise

- Die Abgabe dieser Übungsaufgaben muss bis spätestens Sonntag, den 16. Januar 2022 um 23:59 Uhr im ISIS-Kurs erfolgt sein. Es gelten die Ihnen bekannten Übungsbedingungen.
- Lösungen zu diesen Aufgaben sind als gezippter Projektordner abzugeben. Eine Anleitung zum Zippen von Projekten finden Sie auf der Seite des ISIS-Kurses. *Bitte benutzen Sie einen Dateinamen der Form* `VornameNachname.zip`.
- Bitte beachten Sie, dass Abgaben im Rahmen der Übungsleistung für die Zulassung zur Klausur relevant sind. Durch Plagiieren verwirken Sie sich die Möglichkeit zur Zulassung zur Klausur in diesem Semester.

Aufgabe 10.1 *Behandlung von Ausnahmen* (3 Punkte)

Erstellen Sie die Klasse `Flatten` wie in Abbildung 1 und ergänzen Sie die Methoden `listFlatten` und `max`, sodass alle auftretenden Ausnahmen geeignet behandelt werden und außerdem die Ausgabe wie folgt aussieht.

```
1
2
3
null
[1, 2, 3, null]
1
2
3
null
[1, 2, 3, null, 1, 2, 3, null]
3.0
3.0
2.0

Process finished with exit code 0
```

Das bedeutet, dass der `catch`-Block für

```
res.addAll(listFlatten((List) x));
```

dafür sorgen muss, dass `x` auch dann geeignet behandelt wird, wenn es keine Liste ist.

Hinweise Lassen Sie die `main`-Methode unverändert und benutzen Sie `try-catch`-Blöcke.

Aufgabe 10.2 Bubblesort *in Listen—nebenläufig* **(1 Punkt)**

Vervollständigen Sie die Implementierung der Klasse `Bubbler` in Abbildung 2 durch einen Konstruktor und eine `run`-Methode, sodass die `main`-Methode dann zehn `Bubbler`-Threads startet, die alle die `bubbleSort`-Methode mit *derselben* Eingabe ausführen.
Die Ausgabe soll dann ungefähr wie folgt sein (bis auf die Reihenfolge der Zeilen).

```
Thread[Thread-3,5,main] has run.
Thread[Thread-4,5,main] has run.
Thread[Thread-0,5,main] has run.
Thread[Thread-2,5,main] has run.
Thread[Thread-1,5,main] has run.
Thread[Thread-5,5,main] has run.
\o/
Thread[Thread-7,5,main] has run.
Thread[Thread-8,5,main] has run.
Thread[Thread-6,5,main] has run.                    1
Thread[Thread-14,5,main] has run.
 .
 .
 .
Thread[Thread-17,5,main] has run.
Thread[Thread-13,5,main] has run.
Thread[Thread-10,5,main] has run.
Thread[Thread-18,5,main] has run.

Process finished with exit code 0
```

Achtung! Die Ausgabe des Programms kann mitunter etwas länger sein. In Abhängigkeit von der Infrastruktur, hält das Programm u.U. nicht in kürzerer Zeit oder *gar nicht*. Das Programm kann aber terminieren, und zwar mit einer Ausgabe wie oben.

Tipp Deshalb, verwenden Sie diesen Link

`https://trinket.io/java/a75897d846?showInstructions=true`,

um Ihr Programm zu testen; dort scheint die „richtige“ Infrastruktur vorhanden zu sein.

Aufgabe 10.3 *Zufallspartikel als Threads* **(2 Punkte)**

Erstellen Sie eine Variation der Zufallspartikel, sodass mehrere Partikel sich “gleichzeitig” bewegen (und unterschiedliche Buchstaben hinterlassen). Am einfachsten ist es, wenn Sie die Klasse `ZufallsPartikel` als Erweiterung von `Thread` umschreiben (und die Klassenvariable vom Typ `ZufallsPartikel` geeignet ersetzen).

- Der Benutzer soll die Größe des „Spielfelds“ eingeben nun auch die Anzahl der Zufalls-Partikel.
- Jedes Partikel soll sich so lange bewegen, bis es das Feld verlassen würde, und schließlich die `run()`-Methode verlassen (wodurch dann der Thread auch terminiert).

Die Ausgabe soll dann so wie in Abbildung 3 sein.

[1] Wie viele Threads global dann wirklich gestartet werden ist in hohem Maße vom Zufall abhängig.

```java
import java.util.LinkedList;
import java.util.List;

public class Flatten {
    public static List listFlatten(List t) {
        List res = new LinkedList();

        for (Object x : t) {
            System.out.println(x);

            res.addAll(listFlatten((List) x));
        }

        return res;
    }

    public static double max(Integer[] a, int i, int j) {
        int max = 0;
        for (int k = i; k <= j; k++) {
            max = Math.max(max, a[k]);
        }
        return max;
    }

    public static void main(String[] args) {
        Integer[] ints = {1, 2, 3};
        List l = new LinkedList();

        for (Integer i : ints) {
            l.add(i);
        }
        l.add(null);
        l.add(((LinkedList<?>) l).clone());

        l = listFlatten(l);

        System.out.println(l);

        System.out.println(max(ints, 2, 4));
        System.out.println(max(ints, 2, 2));
        ints[2] = null;
        System.out.println(max(ints, 0, 2));
    }
}
```

Abbildung 1: Programm mit Ausnahmen

```java
import java.util.LinkedList;
import java.util.List;

public class BubbleSort {
    static void bubbleSort(List<Integer> l) {
        int n = l.size();
        for (int i = 0; i < n - 1; i++)
            for (int j = 0; j < n - i - 1; j++)
                synchronized (l) {
                    if (l.get(j) > l.get(j + 1)) {
                        // swap elements
                        l.add(j, l.remove(j + 1));
                    }
                }
    }

    static Boolean sorted(List<Integer> l) {
        Boolean res = true;
        for (int i = 0; i < l.size() - 1; res = res & (l.get(i) <= l.get(++i))) ;
        return res;
    }

    public static void main(String[] args) {
        List<Integer> x = new LinkedList<>();
        final int SIZE = 100;
        Thread[] threads = new Thread[SIZE / 10];
        do {
            for (int i = 0; i < SIZE; i++)
            x.add((int) Math.round(SIZE  * java.lang.Math.random()));

            for (int i = 0; i < threads.length; i++) {
                threads[i] = new Bubbler(x);
            }
            for (int i = 0; i < threads.length; i++) {
                threads[i].start();
            }
        } while (sorted(x));

        System.out.println("\\o/");


    }
}

class Bubbler extends Thread {

  // The solution goes here.

}
```

Abbildung 2: *Bubblesort*-Methode

```
Bitte geben Sie die Breite des Feldes ein: 30
Bitte geben Sie die Höhe des Feldes ein: 30
Bitte geben Sie Anzahl der Zufallspartikel ein: 4
--------------------------------
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|               ccc            |
|       cc  bbb cc             |
|      ccc  bbb cab            |
|     ccccbbbb  cabb           |
|     ccbbbbbb bbbbbbdd        |
|Cccccc  bb bb bbbbbbbbb       |
|       cbbbbbccbbbbbbbb  bb   |
|        bbbbbb bbbbbbbbbbbb   |
|           bbbbbbb  b ddbbb   |
|    aaa     b  aaaa     b b   |
| aaaaaa           a     b bbdd|
| a    a           a    bb bbdd|
| aaaaaaaa         a   dbbbbbdd|
|Aaa  aaaaaaa   aaaa   bbbbbb  |
|      aa   aaaaaaaa   bbbbbbb |
|                a     bbbbbbbB|
|                a     bbbd  b |
|                       bbd    |
|                      dddd    |
|                  dddd        |
|                  dddD        |
--------------------------------

Process finished with exit code 0
```

Abbildung 3: Mehrere Zufallspartikel