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

WiSe 2020/21
Aufgabe 10
Block 4
25. Januar 2021

Abgabetermin: 31. Januar 2021

Aufgabe 10 *Einführung in die Programmierung mit Java*

Hinweise

- Die Abgabe dieser Übungsaufgaben muss bis spätestens Sonntag, den 31. Januar 2021 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.

```
import java.util.LinkedList;

public class ExceptionHandling{

    public static LinkedList listFlatten(LinkedList l){
        LinkedList res = new LinkedList();
        for (Object x :l) {
            System.out.println(x);
            res.addAll((LinkedList)x);
        }
        return res;
    }

    public static double mean(Integer[] a, int i, int j){
        int sum = 0;
        for(int k= i; k<=j;k++){
            sum = sum + a[k];
        }
        return sum / (1+j-i);
    }

    public static void main(String[] args) {
        Integer[] ints = {1,2,3};
        LinkedList l = new LinkedList();
        l.add(null);
        for(Integer i : Arrays.asList(ints)){
    l.add(i);
}
        l = listFlatten(l);

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

Abbildung 1: Programm mit Ausnahmen

Aufgabe 10.1 *Behandlung von Ausnahmen* (2 Punkte)

Erstellen Sie die Klasse `ExceptionHandling` wie in Abbildung 1 und ergänzen Sie die Methoden `listFlatten` und `mean`, sodass alle auftretenden Ausnahmen geeignet behandelt werden. Geben Sie auch jeweils geeignete Fehlermeldungen aus. Wenn Sie alle Fehlermeldungen auskommentieren, soll die Ausgabe des Programms folgende sein.

```
null
1
2
3
3.0
3.0
1.0

Process finished with exit code 0
```

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

```
public class BubbleSort {

    static void 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]) {
                    // swap temp and arr[i]
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
    }

    static Boolean sorted(int[] arr){
        Boolean res = true;
        for(int i=0;i<arr.length-1;res = res & (arr[i]<= arr[++i]));
        return res;
    }

    public static void main(String[] args) {
        int[] x = new int[100];
        Thread[] threads = new Thread[x.length/10];

        do {
            for(int i=0;i<x.length;i++)
                x[i] = (int) (x.length*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("oops");
    }
}

class Bubbler extends Thread{

    // the solution goes here
}
```

Abbildung 2: *Bubblesort*-Methode

Aufgabe 10.2 Bubblesort *mit Threads* (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.

```
⋮

Thread[Thread-2378,5,main] has run.
oops
Thread[Thread-2380,5,main] has run.
Thread[Thread-2379,5,main] has run.
Thread[Thread-2381,5,main] has run.
Thread[Thread-2383,5,main] has run.
Thread[Thread-2385,5,main] has run.
Thread[Thread-2386,5,main] has run.
Thread[Thread-2382,5,main] has run.
Thread[Thread-2388,5,main] has run.
Thread[Thread-2387,5,main] has run.
Thread[Thread-2384,5,main] has run.
Thread[Thread-2389,5,main] has run.

Process finished with exit code 0
```

Abbildung 3: Erwartete Form der Ausgabe[1]

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 in Abbildung 3.

Tipp Deshalb, verwenden Sie den Link `https://trinket.io/java/43df35c03f`, um Ihr Programm zu testen; dort scheint die „richtige“ Infrastruktur vorhanden zu sein.

Aufgabe 10.3 *Langtons Ameisen als Threads* (2 Punkte)

Als letzte Variation von Langtons Ameise (aus Aufgabe 4.1) sollen die Ameisen jetzt Threads sein und es sollen mehrere gleichzeitig laufen. Das bedeutet, dass die Klasse `LangtonsAnt` insbesondere die Klasse `Thread` erweitern soll.

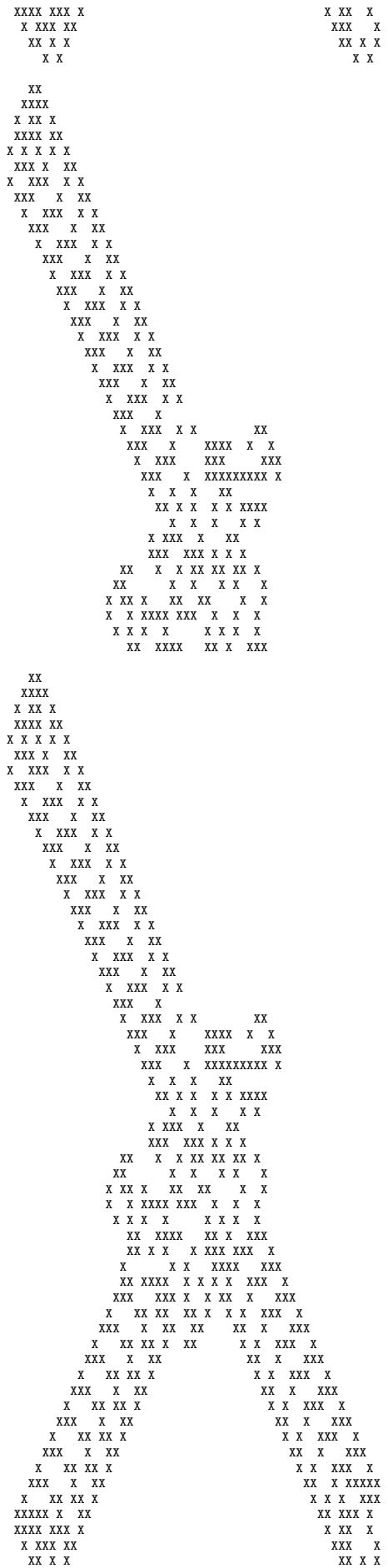
- Der Benutzer soll die Größe des „Spielfelds“ eingeben, die Größe der zufälligen Mitte und nun auch die Anzahl der Ameisen.
- Jede Ameise soll so lange laufen, bis Sie das Feld verlassen würde, dann das derzeitige Spielfeld ausgeben, und schließlich die `run()`-Methode verlassen.

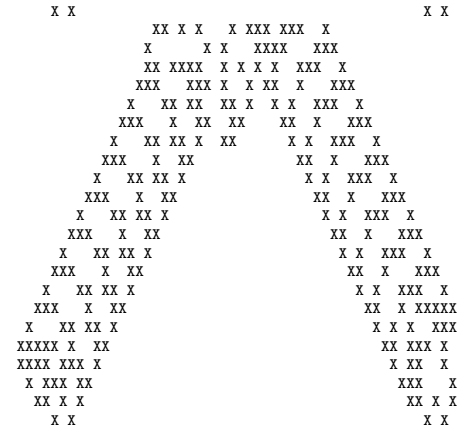
Folgende Ausgabe – die zeigt, dass sich die Ameisen bei der Ausgabe am Ende gegenseitig „unterbrechen“ können – resultierte aus einem Beispielablauf.

```
Bitte geben Sie die Größe des Feldes ein:60
Bitte geben Sie die Größe der zufälligen Mitte ein:8
Bitte geben Sie die Anzahl der Ameisen ein:3


  XXXX
 X XX X
 XXXX XX
X X X X X
 XXX X  XX
X  XXX  X X
 XXX   X  XX
  X  XXX  X X
   XXX   X  XX
    X  XXX  X X
     XXX   X  XX
      X  XXX  X X
       XXX   X  XX
        X  XXX  X X
         XXX   X  XX
          X  XXX  X X
           XXX   X  XX
            X  XXX  X X
             XXX   X
              X  XXX  X X       XX
               XXX   X   XXXX  X  X
                X  XXX    XXX    XXX
                 XXX   X  XXXXXXXXX X
                  X  X  X   XX
                   XX X X  X X XXXX
                     X  X  X   X X
                   X XXX  X   XX
                   XXX  XXX X X X
                 XX   X  X XX XX XX X
                XX      X  X   X X   X
               X XX X   XX  XX    X  X
               X  X XXXX XXX  X  X  X
                X X X  X     X X X  X
                  XX  XXXX   XX X  XXX
                  XX X X   X XXX XXX  X
                 X      X X   XXXX   XXX
                 XX XXXX  X X X X  XXX  X
                XXX   XXX X  X XX  X   XXX
               X   XX XX  XX X  X X  XXX  X
              XXX   X  XX  XX   XX  X   XXX
             X   XX XX X  XX     X X  XXX  X
            XXX   X  XX           XX  X   XXX
           X   XX XX X             X X  XXX  X
          XXX   X  XX               XX  X   XXX
         X   XX XX X                 X X  XXX  X
        XXX   X  XX                   XX  X   XXX
       X   XX XX X                     X X  XXX  X
      XXX   X  XX                       XX  X   XXX
     X   XX XX X                         X X  XXX  X
    XXX   X  XX                           XX  X XXXXX
   X   XX XX X                             X X X  XXX
  XXXXX X  XX                               XX XXX X
```

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





```
Process finished with exit code 0
```