© Manfred Kerber
Alexandros Evangelidis
David McDonald

School of Computer Science
University of Birmingham
Wednesday 24 October 2018

# Worksheet 4

## MSc/ICY Software Workshop

**Assessed Worksheet: 2% of the module mark.**
**Submission Deadline is Thursday, 8 November, at 2pm via Canvas.**
**5% late submission penalty within the first 24 hours. No submission after 24 hours. Follow the submissions guidelines on Canvas. JavaDoc comments are mandatory. All submissions must pass the tests provided on 1 November. For Exercises 1 – 3 submit own tests.**

**Exercise 1: (Basic, 30%)** Insertion sort (for more details see, e.g., `https://en.wikipedia.org/wiki/Insertion_sort`) is a simple sorting algorithm which sorts arrays from the start by going once through the array so that the initial part of the array is sorted and the rest of the array contains the elements that need still to be dealt with. The current element is put into its correct place in the sorted part and the elements following it are moved one up to make room for it. For instance, with the bar indicating the separation of the sorted part and the elements to be considered, and * denoting the most recent element inserted, we get:

```
     [ 4,  3,  6,  1,  9,  2]
->   [ 4*|,3,  6,  1,  9,  2]  4 put in its place
->   [ 3*, 4,| 6,  1,  9,  2]  3 put in its place
->   [ 3,  4,  6*|,1,  9,  2]  6 put in its place
->   [ 1*, 3,  4,  6|, 9,  2]  1 put in its place
->   [ 1,  3,  4,  6,  9*|,2]  9 put in its place
->   [ 1,  2*, 3,  4,  6,  9|] 2 put in its place
```

The algorithm terminates when the second part of the array is empty. Give a Java implementation of insertion sort using loops, `public static int[] insertionSort(int[] numbers)`, in a class `InsertionSort`. NOTE: Since any sorting algorithm would pass the tests which we will provide, there will be an inspection by your tutor of whether you actually implemented `insertionSort`. You will get marks only for an implementation of `insertionSort`.

**Exercise 2: (Medium, 30%)** Implement a class `Student` with the field variables `private String registrationNumber` and `private int[] marks`. The mark array has a size of 14. Its elements represent (in this order) 8 assessed worksheets (worth 2, 2, 2, 2, 1, 1, 1, and 1 percent of the total module mark, respectively), 4 in-class tests (worth 2, 2, 1, and 3 percent of the total module mark), a team project (worth 10 percent of the total module mark), and an examination result (worth 70 percent of the total module mark). Write a constructor, setters, getters, and a `toString()` method. Furthermore write a method that allows one to set a single assignment mark, `public void setAssignmentMark(int assignmentNumber, int mark)`, where the `assignmentNumber` is to be a number between 1 and 14, representing the 14 component marks in the order presented above, each given out of 100.
Also, write a method `public double totalMark()` that computes the total mark of a student, rounded to one decimal place. If a mark of `-1` has been entered for one piece of assessment, then this is to mean that the student has been granted for this piece of assessment extenuating circumstances and that the mark should be discarded from the computation of the total mark. If marks with a total weight of more than 50 percent of the total mark have been waived, then the returned total mark should be `-1`.
Finally, write a method `public boolean passed()` that returns `true` if the total mark of the student is greater than or equal to 50 and false otherwise.
In the case of a total mark of `-1`, the method `passed()` should throw an `IllegalArgumentException`.
For instance, with the entries corresponding to
`int[] samsMarks   = {50, 60, 65, 60, 65, 70, 55, 66, 60, 73, 65, 45, 68, 54};`,
`int[] billysMarks = {50, 60, -1, 60, 65, 70, 55, 66, 60, 73, 65, 45, 68, 54};`,
`Student sam = new Student("1111111", samsMarks);`, and
`Student billy = new Student("1111112", billysMarks);`
`sam.totalMark()` should result in `56.5`; `billy.totalMark()` in `56.3`; `sam.passed()` and `billy.passed()` should both result in `true`.

**Exercise 3: (Advanced, 30%)** In the lecture we have seen how grey value pictures given in form of a P2 type PGM (Portable GrayMap) image of a given `width` and `height` can be represented by a two-dimensional array of type `short[][] image = new short[height][width]`, in which each entry in the array is a grey value between 0 and 255, inclusively. Similarly, coloured pictures are represented in PPM (Portable PixMap) format of type P3. The PPM file contains for each pixel 3 values, the RGB values (each between 0 and 255, inclusively) for the colours red, green, and blue separately. The corresponding image can then be represented by a three-dimensional array of type `short[][][] image = new short[height][width][colour]`, in which the three values for the three colours are stored in indices 0, 1, and 2 in the third dimension corresponding to the three colours red, green, and blue, respectively.
Write a class `PPMImage` with the five field variables

```
private int width;
private int height;
private int maxShade;
private String typeOfFile; // here:  "P3"
private short[][][] pixels;
```

a constructor `public PPMImage(String filename)`, getters and setters, and a method `public short[][] makeGrey(String filename)` which returns an array of type `short[][]` of suitable size containing the corresponding grey values so that each grey value is the rounded average of the corresponding three colour values. If the `filename` is not the empty string the method should also save the image in the corresponding file as a PGM image.

**Exercise 4: (Debugging, 10%)** In a class `Measurement.java`, assume a method, `public static int measurement()`, that returns a measurement of an instrument. As values only `1` or `2` are returned. If the instrument malfunctions the method throws an `IllegalArgumentException`.
This can be simulated, for instance, by the following method:

```
public static int measurement() {
    int res = (int) (Math.random() * 3);
    if (res == 0) {
        throw new IllegalArgumentException();
    } else {
        return res;
    }
}
```

You want to call the method `n` times and store the results in an array, `int[] result = new int[n]`. If the instrument has malfunctioned you want store a value of `-1`. This is supposed to be done by the following method, but it does not work properly.

```
public static int[] measurementSeries(int n) {
    int[] result = new int[n];
    for (int i = 0; i < n; i++) {
        if (measurement() == -1) {
            result[i] = -1;
            throw new IllegalArgumentException();
        }
        else {
            result[i] = measurement();
        }
    }
    return result;
}
```

- What are the problems? Write your answer at the top of the file as part of the JavaDoc comment.
- How can the problems be resolved? Change the code accordingly.