© Manfred Kerber
Alexandros Evangelidis

School of Computer Science
University of Birmingham

Worksheet 3

MSc/ICY Software Workshop

Assessed Exercise: 9% of the continuous assessment mark.

Submission: Thursday 3 November 2016 2pm

5% late submission penalty within the first 24 hours. No submission after 24 hours.

JavaDoc comments are mandatory. Follow the submission guidelines on

`https://canvas.bham.ac.uk/courses/21955/modules/items/581785`.

The public tests will be provided a week before the submission deadline.

Exercise 1: (Basic, 25%) Bubble sort is a very simple sorting algorithm which swaps adjacent elements in an array which are out of order until the whole array is sorted. E.g.:

```
    [4, 3, 6, 1, 9, 2]
->  [4, 3, 6, 1, 2, 9]
->  [4, 3, 1, 6, 2, 9]
->  [4, 1, 3, 6, 2, 9]
->  [1, 4, 3, 6, 2, 9]  Now position 0 in correct place. Rerun from back.
->  [1, 4, 3, 2, 6, 9]
->  [1, 4, 2, 3, 6, 9]
->  [1, 2, 4, 3, 6, 9]  Now positions 0 and 1 in correct place. Rerun from back.
->  [1, 2, 3, 4, 6, 9]  Now all in correct place.
```

Give a Java implementation of bubble sort using loops, `public static int[] bubbleSort( int[] numbers)` in a class `BubbleSort`.

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 `BubbleSort`. You will get marks only for an implementation of `BubbleSort`.

Exercise 2: (Basic, 25%) Translate the cartoon below into a program, implemented in a class `StarRating` with a method `public static String interpret(double rating)`.

Concretely, write a method `public static String interpret(double rating)` that interprets any rating $0 \leq x \leq 5$, (which could be interpreted as the rating averaged over several individual ratings), and otherwise returns `"not rated"`. A `"CRAP"` rating starts at `0.0` and may extend up to, but excluding, `4.0`; `"OK"` should start from `4.0`, `"EXCELLENT"` from `4.5`, and `"[HAS ONLY ONE REVIEW]"` from `5.0`.

© Randall Munroe `xkcd.com/1098/`

Exercise 3: (Medium, 20%) Write a method that returns an ArrayList of all those integers which do not contain a particular digit, that is, a method `public static ArrayList< Integer> allIntegersWithout(int from, int to, int omittedDigit)` in a class `Omit`. E.g., `allIntegersWithout(20, 45, 3)` contains the following elements: 20, 21, 22, 24, 25, 26, 27, 28, 29, 40, 41, 42, 44, and 45.

Exercise 4: (Medium, 15%) When transferring data mistakes may happen. In order to be able to detect and correct mistakes so-called parity bits are used. For instance, if we want to transfer data such as in the following 3×5-matrix (two-dimensional array) of bits:

```
0 1 1 1 0
1 0 0 0 1
1 0 1 0 1
```

the matrix is extended to a 4×6-matrix such that each line has added a 0 if the sum in the line is even and 1 if it is odd. Likewise, the columns are extended. That is, we get:

```
0 1 1 1 0 1
1 0 0 0 1 0
1 0 1 0 1 1
0 1 0 1 0 0
```

Write a method `public static int[][] addParity(int[][] a)` that takes a two-dimensional array (consisting of 0s and 1s) of size $n \times m$ and returns an array of size $(n+1) \times (m+1)$ with the parity bits added.
Furthermore write a method `public static boolean checkParity(int[][] a)` that returns `true` if in an extended matrix with parity bits the parity bits follow the rules and `false` otherwise. (Do all this in a class called `Parity`.)

Exercise 5: (Advanced, 15%) Following the previous exercise, assume that in the transfer at most one bit has been transferred incorrectly. If one bit has been transferred incorrectly find it and correct it, that is, write in a class `ParityCorrection` a method `public static int[][] correctMatrix(int[][] a)` that takes a two-dimensional array (with parity bits) that may contain up to one incorrect bit and returns a corrected two-dimensional array (with parity bits) that does not contain any incorrect bits. To this end write two methods `public static int findRow(int[][] a)` that finds a row which is not formed according to the rules and `public static int findColumn(int[][] a)` that finds a column which is not formed according to the rules (each returning `-1` if everything is formed according to the rules).
For instance, for `a` as

```
0 1 1 1 0 1
1 0 1 0 1 0
1 0 1 0 1 1
0 1 0 1 0 0
```

`findRow(a)` should return 1 and `findColumn(a)` should return 2, that is, the field to be corrected is at position (1,2). (Note that as usual we start counting with 0.)
The corrected matrix is:

```
0 1 1 1 0 1
1 0 0 0 1 0
1 0 1 0 1 1
0 1 0 1 0 0
```