© Manfred Kerber
Alexandros Evangelidis

School of Computer Science
University of Birmingham

Worksheet 3

MSc/ICY Software Workshop

Assessed Exercise: 12.5% of the continuous assessment mark.

Submission: Thursday 5 November 2015 2pm

5% late submission penalty within the first 24 hours. No submission after 24 hours. JUnit tests and JavaDoc comments are mandatory. All submissions must pass the tests provided on 28 October. Follow the submission guidelines on `http://www.cs.bham.ac.uk/internal/courses/java/msc/submission.php`.

Exercise 1: (Basic, 25%) Translate the cartoon below into a program, implemented as a class `public class StarRating` with a method `public static String interpret (double rating)`.

(a) For any numeric rating given in the figure (`1.0`, `1.5`, ..., `5.0`) `interpret` should return a textual interpretation as in the cartoon, e.g. `"OK"`, and otherwise `"not rated"`.

(b) Add a method `public static String interpretRange(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`.

xkcd © Randall Munroe; `http://xkcd.com/1098/`

Exercise 2: (Basic, 25%) The digit total of an integer is the integer you get by adding up its digits. For instance the digit totals of `0`, `7`, `12`, `15`, `213`, `999`, and `1111` are `0`, `7`, `3`, `6`, `6`, `27`, and `4`, respectively. Write a class `DigitTotal` with the following methods:

(a) `public static int digitTotal(int n)` that returns the digit total of `n`.

(b) `public static double digitTotalAverage(int max)` that returns the average digit total of the numbers `0`, `1`, `2`, ..., `max` for any positive integer `max`. For instance, the `digitTotalAverage(9)` is computed as $(0+1+\cdots+9)/10.0$, that is, `4.5`. Likewise `digitTotalAverage(19)` is $(0+\cdots+9+1+\cdots+10)/20.0$, which gives `5.0`.

(c) `public static double digitTotalFrequency(int max, int digitTotal)` returns the proportion of numbers which have the given `digitTotal` of all numbers in the range `0`, `1`, ..., `max`. E.g., the `digitTotalFrequency(9, 3)` is `0.1` since only one number out of the first 10 numbers has a digit total of `3`. Likewise `digitTotalFrequency (99, 3)` is `0.04` (since out of the first 100 numbers only the numbers `3`, `12`, `21`, and `30` have a `digitTotal` of 3).

Exercise 3: (Medium, 30%)

(a) An `Employee` is represented by their `name` (of type `String`), `address` (of type `String`), and the 12 monthly `salaries` they receive during the tax year (of type `double[]`). Write a class `public class Employee` with the usual ingredients (field variables as specified above, constructor, getters, setters, and a `toString` method).

(b) Write a class `public class AllEmployees`, which is used to represent the employees on the payroll of a company. Use a variable `ArrayList<Employee> allEmployees` for the representation of the employees of the company. In addition to a constructor and a `toString` method write the following methods:

- `public void add(Employee employee)`
  to add an `employee` to the list of `allEmployees`. Note that you are not expected to check whether the employee is already in the list of all employees.
- `public void delete(Employee employee)`
  to delete an `employee` in the list of `allEmployees`. Note that you are not expected to check whether the employee is in the list of all employees, just delete one occurrence.
- `public double averageMonthlySalary()` to compute the average monthly salary that the company is paying to its employees over the tax year. Note that in the computation of this average the salaries with value `0` should be disregarded (since in the corresponding months the employee was not on the payroll of the company).

Exercise 4: (Advanced, 10%) Write a program that performs certain operations on images in `pgm` format of the P2 type, that is, grey scale type of portable anymap images.

(a) Implement a class `PGMImage` with constructor `PGMImage(String filename)` with fields `int width`, `int height`, `int maxShade` for the maximum grey value of a pixel, and `int[][] pixels` (where the grey value of each pixel is a shade of grey between `0` (for black) and `maxShade` (for white). For simplicity, the constructor should create an empty image whenever something goes wrong, such as an `IOException` occurs.

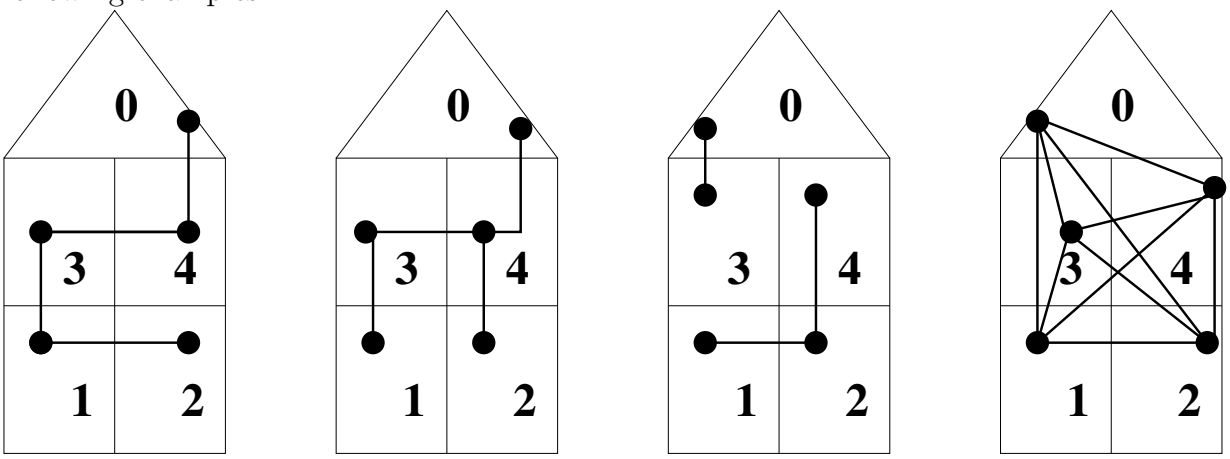
(b) Implement a method `public void invert()` that inverts the image so that dark grey values go to bright ones and vice versa.

(c) Implement a method `public void rotateClockwise()` that rotates the current image clockwise by 90 degrees.

(d) Implement a method `public void exportToP1(String filename)`, which exports the image to a `pbm` file of P1 type, i.e. a black-and-white type of portable anymap images. Compute the black-and-white version of your grey scale image as follows: Let `double threshold` be the average grey scale value of all pixels in the image. Then, each pixel whose grey value has the value `threshold` or any larger value should be white in the exported file. Each pixel whose grey value has a value less than `threshold` should be black. [Note that for P1, `maxShade` should be set to 1.]

(e) Testing should be done by comparing manipulated pictures to given sample output pictures. You do not have to write JUnit tests of your own for this exercise.

Exercise 5: (Advanced, 10%) Graphs are an important data structure in Computer Science. They can be used to represent reachability between different nodes. For instance, assume a house with 5 rooms and electric connectivity between them. The mains are connected to room 1. One possible question now is whether all rooms are connected in the following examples.



The connectivity can be represented by two-dimensional arrays of type `int`, where the integer indicates the distance. For instance, in these cases by the matrices:

```
{{0, 0, 0, 0, 1},     {{0, 0, 0, 0, 1},     {{0, 0, 0, 1, 0},       {{0, 1, 1, 1, 1},
 {0, 0, 1, 1, 0},      {0, 0, 0, 1, 0},      {0, 0, 1, 0, 0},        {1, 0, 1, 1, 1},
 {0, 1, 0, 0, 0},      {0, 0, 0, 0, 1},      {0, 1, 0, 0, 1},        {1, 1, 0, 1, 1},
 {0, 1, 0, 0, 1},      {0, 1, 0, 0, 1},      {1, 0, 0, 0, 0},        {1, 1, 1, 0, 1},
 {1, 0, 0, 1, 0}}      {1, 0, 1, 1, 0}}      {0, 0, 1, 0, 0}}        {1, 1, 1, 1, 0}}
```

Graphs (a), (b), and (d) share the property that all the nodes (0, 1, 2, 3, and 4) can be reached from all other nodes (that is, the graph is *connected*). This is not the case for the graph (c). Graph (d) has the property that it is *fully connected*, that is, each node is connected to each other node.

Note that in general, the values in the matrix can stand for distances (e.g., distances between cities) and that such a matrix is not necessarily symmetric (e.g., if there are one way roads, there may be a connection from 1 to 2, but no connection from 2 to 1).

Write a class `Graph` with a constructor `public Graph(String filename)` that reads in a file that contains data of the following form (the example represents graph (a)):

```
5
0 4 1
1 2 1
1 3 1
2 1 1
3 1 1
3 4 1
4 0 1
4 3 1
```

That is, the first number states the number of nodes in the graph, all other lines consist of exactly three integer numbers, first two nodes that are connected and lastly the distance between these two nodes.

Write a method `public boolean isFullyConnected()` that checks for a graph whether it is fully connected or not.