Individual Assignment 3
INFS1609 Fundamentals of Business Programming
INFS2609 Programming For Business
Session 2, 2014

Due Dates: 5pm, Friday in Week 13, 31st October 2014

Assignment Description

Individual assignment 3 is graded and contributes to 8% (and an additional 3% of bonus) to your overall results. Please take note of the following issues before attempting the assignment:

- Upon completing an assignment, please submit your program(s) to Moodle so that your submission is time-stamped.
- Please submit all your program(s) in one zip/rar file, which follows the naming convention "zID_Name", such as z1234567_Ben_Choi.
- You should design your own testing data and test your program thoroughly before submitting.
- You should use println ( ) instead of print ( ) for last line of output your program generates.
- You may assume that all input data are correct. Hence you do not need to do input data validation. This is always true throughout all individual assignments unless otherwise specified.
- You will be awarded 8 marks for this individual assignment, provided that (1) you have submitted the assignment before the deadline; (2) you have shown genuine effort in your programs; (3) you have followed the coding style; and (4) your submission is correct.
- You will be awarded 3 bonus marks for this individual assignment, provided that (in addition to above), you have completed the bonus exercise, which is optional.
- Please use the Moodle discussion forum as the media to discuss individual assignments. Raise you questions and discuss your problem solving approaches there if applicable. Do not post your solutions (partial or complete) in the forum.
- Please take care of the coding style of your programs as this is one of the evaluation criteria.

This individual assignment requires you to attempt one exercise (and one bonus (optional) exercise). You are advised to review the materials covered in lecture 1 through 9 before attempting this assignment.

Exercise 1 (8 marks)

1.1 Learning Objectives

- Create a class
- Write data fields, constructors, and public methods
- Simple use of String
- Logic thinking

1.2 Task statements

You are to write two classes in **ToyCar.java** and **TestToyCar.java**

The ToyCar class should contain the following data fields:

- **Model**: Model of the toy car, a String object.
- **Colour**: Colour of the toy car, a String object
- **Trips**: The number of trips made by the top car. This is a non-negative integer.
- **Distance per trip**: The average distance travelled per trip (use double type).
- **Odometer value**: The initial odometer value (use double type). The odometer value on the car displays up to maximum value of 999.9, after which it starts from 000.0 again. (An odometer is an instrument in a car for measuring distance travelled).

There is no need to add any other data fields.

Your ToyCar class should contain at least a default constructor and appropriate accessors and mutators to set and get the values of data fields.

The default constructor should create a new ToyCar object with the following default values:

- **Model**: "No model"
- **Colour**: "block"
- **Trips**: 0
- **Distance per trip**: 0.0
- **Odometer value**: 999.9

In the driver class TestToyCar, you are to read in data about a toy car to create a ToyCar object. The input data includes the model, colour, initial odometer value, the number of trips, and for each trip, the distance travelled (a positive real number).

You may assume that the odometer value and distance travelled has at most one decimal place, and the number of trips is non-negative.

(You will find that there is actually no absolute need to create the ToyCar object, as the task is very simple. However, please create the ToyCar object as instructed anyway, and use accessors to retrieve information from the object, even though the information are in the variables that hold the input data. We want you to have practice on OO).

After all the toy car's information are read, the program is to compute the final odometer value, the average distance travelled per trip, and the maximum difference in distance between two consecutive trips. The program then displays the outputs as shown in the sample runs below.

For example, in sample run #1 below, the difference in distance between trip 1 and trip 2 is 4.8, while the difference in distance between trip 2 and trip 3 is 10.8. Hence the maximum difference is 10.8.

You are to display the real number in one decimal place.

Check sample runs for input and output format; submit your program through Moodle.

1.3 Sample runs

Four sample runs are shown below using interactive input (user's input shown in blue; essential output shown in **bold purple**).

```
Enter model: Zeus
Enter colour: blue
Enter odometer value: 820.3
Enter number of trips: 1
Distance for trip 1: 1563.2
Model: Zeus
Colour: blue
Odometer: 383.5
Trips: 1
Distance per trip: 1563.2
Maximum difference between two consecutive trips: 0.0
```

```
Enter model: Ultimate 2100
Enter colour: black
Enter odometer value: 100
Enter number of trips: 5
Distance for trip 1: 1084.9
Distance for trip 2: 900
Distance for trip 3: 1230.5
Distance for trip 4: 1508
Distance for trip 5: 1450.2
Model: Ultimate 2100
Colour: black
Odometer: 273.6
Trips: 5
Distance per trip: 1234.7
Maximum difference between two consecutive trips: 330.5
```

```
Enter model: Viaxer III
Enter colour: white
Enter odometer value: 980.5
Enter number of trips: 3
Distance for trip 1: 23.8
Distance for trip 2: 19
Distance for trip 3: 8.2
Model: Viaxer III
Colour: white
Odometer: 31.5
Trips: 3
Distance per trip: 17.0
Maximum difference between two consecutive trips: 10.8
```

```
Enter model: Morton Plus
Enter colour: grey
Enter odometer value: 70
Enter number of trips: 0
Model: Morton Plus
```

```
Colour: grey
Odometer: 70.0
Trips: 0
Distance per trip: 0.0
Maximum difference between two consecutive trips: 0.0
```

1.4 Estimated Development time

The time below is an estimate of how much time we expect you to spend on this exercise. If you need to spend a lot more time than this, it is an indication that some help (or more practice) might be needed.

- 70 minutes

1.5 Marking scheme

This exercise constitutes 8 marks. 6 marks are allocated to the correctness of the program that you are supposed to develop. 2 marks are allocated to design and style. Mistakes that result in penalty include (but not limited to) the following:

- No name (under style/design)
- Poor indentation (under style/design)
- Poor naming of variable or method (under style/design)
- Messy logic, even if the code produces correct result (under style/design)

IMPORTANT: Programs which fail to compile will be awarded 0 marks.

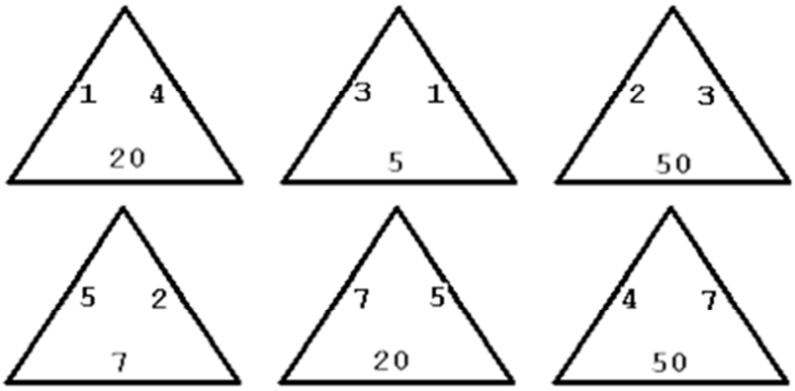
Exercise 2 (Bonus – 3 mark)

2.1 Learning Objectives

- Writing recursive program

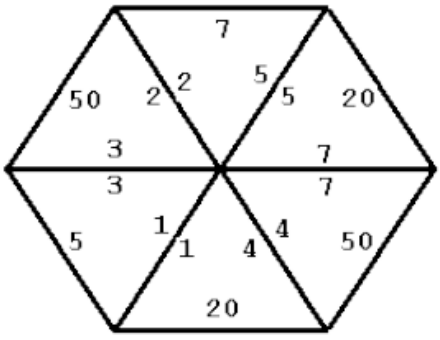
2.2 Task statements

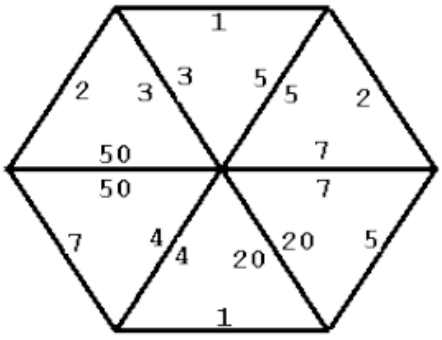
You are given six equilateral triangles of the same size whose edges are numbered, as shown in the example below.



You are given six equilateral triangles (same sizes) whose edges are numbered, as in the example above. Your task is to form a **hexagon** (a 6-sided polygon) by matching the edges of the triangles. The edges common to two triangles match if they have the same number on them. You may move or rotate the triangles but you can't flip them. The score of the hexagon is the sum of the numbers on the exterior edges.

For the above example, the hexagon may be assembled in the following ways:





Write a complete Java program that computes the maximum hexagon score from a set of input triangles. Your program should read its input from a .dat file and output to the interactive screen.

The input consists of 18 non-negative integers, each on a line, denoting the edge values of the six input triangles. The edge numbers in the input are given in a clockwise manner, one triangle after another (see sample run). If no such hexagon can be formed, your program should output "IMPOSSIBLE" (without the quotes).

Your output is a single integer denoting the maximum score.

A demonstration program DemoAssignment3.java to show how you may read data from a text file demo.dat is given.

2.3 Sample runs

Essential output is shown in **bold purple**.

Example 1: Assuming that the following is the content of the text file maze.dat:

A sample run with the input file is shown below.

```
1
4
20
3
1
5
2
3
50
5
2
7
7
5
20
4
7
50
```

The output is as follows.

```
152
```

2.4 Estimated Development time

The time below is an estimate of how much time we expect you to spend on this exercise. If you need to spend a lot more time than this, it is an indication that some help (or more practice) might be needed.

- 45 minutes

2.5 Important Notes

You should test your program with different maze.dat files. For convenience in testing, your program may request for the filename of the input file and read that specified file accordingly.

2.6 Marking scheme

This exercise constitutes 3 bonus marks. 2 marks are allocated to the correctness of the program that you are supposed to complete or write. 1 mark is allocated to design and style. Mistakes that result in penalty include (but not limited to) the following:

- No name (under style/design)
- Poor indentation (under style/design)
- Poor naming of variable or method (under style/design)
- Messy logic, even if the code produces correct result (under style/design)

IMPORTANT: Programs which fail to compile will be awarded 0 marks.