COMP101 Lab 10: Seventh Assessed Coursework
Arrays

worth 18% of the final mark

The completion of the implementation and report as described below will obtain 90% of the marks. To obtain the final 10% students should also complete the Extended Requirements. This may involve doing some additional reading beyond what has been presented in lectures or more complex programming. Only attempt the extended requirements if you are confident with programming.

Learning Outcomes. This addresses the following learning outcomes, to

- Be able to implement, compile, test and run Java programmes, comprising more than one class, to address a particular software problem;
- Understand how to include arithmetic operators and constants in a Java program;
- Demonstrate the ability to employ various types of selection constructs in a Java program;
- Demonstrate the ability to employ repetition constructs in a Java program;
- Be able to employ a hierarchy of Java classes to provide a solution to a given set of requirements.
- Demonstrate the ability to use simple data structures like arrays in a Java program.

Introduction. Dowload the file `ThreeDice.java` from the COMP101 module website (the link is right below that of this assignment). This assignment builds upon that class. **DO NOT ALTER THE SOURCE CODE OF `ThreeDice.java` IN ANY FASHION!**

`ThreeDice.java` implements a class that records the rolls of three dice. We will consider the dice to be regular six-sided dice (i.e. dice with the integers 1-6, each number occurring with equal chance). The ThreeDice class constructor takes three integer parameters as input, each parameter representing the number shown on one of the dice. (The constructor will store these three numbers in order from smallest to largest, useful for other methods in the class.)

Any outcome of the roll of three 6-sided dice can be classfied into exactly one of four cases below:

Description	Examples	Name
All the same value	3, 3, 3 1, 1, 1	Three the same
A sequence of values that are all different	4, 5, 6 3, 1, 2 4, 3, 5	A run
Two values are the same and one different	1, 1, 4 5, 1, 5	A pair
All values are different and it isn't a run	2, 3, 6	All different

The ThreeDice class includes methods that can be used to classify each of these outcomes, as can be seen in the class diagram below:

ThreeDice
die1: int # die2: int # die3: int
+ getDie1(): int + getDie2(): int + getDie3(): int + threeSame(): boolean + runOfThree(): boolean + pair(): boolean + allDifferent(): boolean + printResult(): void

Each of the methods `threeSame()`, `runOfThree()`, `pair()`, and `allDifferent()` returns a boolean value (i.e. true or false) that represents whether or not the three dice values can be classified into that category (true) or not (false). (Have a look at the code for these methods, and convince yourself that they will return the correct results.)

Requirements. This assignment builds on the `ThreeDice.java` class. The aim is to design, implement, and test a Java program to simulate two players each rolling three dice for a number of rounds (input by the user). Each dice throw is given points (see below), and the following should be displayed each round:

- the dice values, and number of points for each player for those values;
- the winner of each round (the player with the highest points for that round, or no-one if they are the same).

After all the rounds have been displayed, the following should be calculated and displayed:

- the total number of rounds won for each player;
- the total points for each player;
- the average points for each player; and
- the player with the highest points total.

The number of points for a particular dice roll is given below (with examples):

Description	Points Calculation	Example	Points
Three the same	sumOfDice + 60	3, 3, 3	9 + 60 = 69
		1, 1, 1	3 + 60 = 63
A run	sumOfDice +40	4, 5, 6	15 + 40 = 55
		3, 1, 2	6 + 40 = 46
		4, 3, 5	12 + 40 = 52
A pair	sumOfDice + 20	1, 1, 4	6 + 20 = 26
		5, 1, 5	11 + 20 =31
All different	sumOfDice	2, 3, 6	11

Here is what you should do for the regular requirements for this assignment:

1. Extend the class `ThreeDice.java` by designing a subclass that includes a method that returns the points for a particular dice roll according to the points table above. Put this new class into a separate class file called `ThreeDiceScorer.java`. Note that you need to specify a constructor for your new subclass (in this case, it will likely just call the constructor of the parent ThreeDice class), and this subclass needs at least one more method to calculate the points score of the three dice.

2. Design a separate application program that will take as user input a number of rounds to play. This should be a positive or zero integer. Negative values should be disallowed. Your solution should store the dice rolls for the pair of players as *an array of objects*. Call your application class `Game.java`.

3. Rather than asking for user input for each dice roll, use randomly generated numbers (integers between 1 and 6 inclusive). A single random integer can be calculated using `Math.random()` as follows `1 + (int)(6 * Math.random())`.

Example. Below, output is provided for a run with five rounds.

```
Input number of rounds:   5

Round 0   Player 1:  1 3 3 points: 27    Player 2:  1 4 5 points: 10    Round winner is player 1
Round 1   Player 1:  1 2 5 points: 8    Player 2:  1 3 6 points: 10    Round winner is player 2
Round 2   Player 1:  1 4 4 points: 29    Player 2:  4 5 6 points: 55    Round winner is player 2
Round 3   Player 1:  1 3 5 points: 9    Player 2:  1 5 5 points: 31    Round winner is player 2
Round 4   Player 1:  3 6 6 points: 35    Player 2:  2 2 4 points: 28    Round winner is player 1

Total wins:
      Player 1: 2    Player 2: 3
Total points:
      Player 1: 108    Player 2: 134
Average points per round:
      Player 1: 21.6    Player 2: 26.8

Overall points winner is player 2.
```

Extended Requirements. For the regular requirements we have assumed that there are always two players. For the extended requirements, re-implement your program to allow two or more players. The number of players should be input by the user (an integer greater than or equal to two). You should disallow input that is not in the correct range.

If you do the extended requirements, call your application class `GameExt.java`.

NOTE: If you do the Extended Requirements, you must still hand in source code that satisfies the Regular Requirements.

Submission Instructions. Your submission, should consist of a report (a PDF file) and implementation files.

- Submit one compressed file, using only the zip format for compression, that includes all files for your submission.
- The report (a PDF file) should consist of

  Requirements: Summary of the above requirements statement.

  Analysis and Design: A short (one paragraph) description of your analysis of the problem including a Class Diagram outlining the class structure for your proposed solution, and pseudocode for methods. Note that your solution should comprise (at least) three classes (including the unaltered `ThreeDice.java` class): an "application class" (called `Game.java`) and two (or more) "target classes" (one of which is the unaltered `ThreeDice.java` class and another which is your `ThreeDiceScorer.java` subclass).

  Include the Java source code for `ThreeDice.java` in your submission (so that the assessors will readily have it available with your other code when testing your solution).

  Testing: A set of proposed test cases presented in tabular format including expected output, and evidence of the results of testing (the simplest way of doing this is to cut and paste the result of running your test cases into your report).
- The implementation should consist of

  Your Java source files, i.e. the relevant .java files, not the class (.class) files.

Upload your files (as a single compressed zip file) to

https://sam.csc.liv.ac.uk/COMP/Submissions.pl

(you will need to log in using your Computer Science username and password).

Submission Deadline. Wednesday 14th December, 5pm.

Mark Scheme. Marks will be awarded for

- Analysis and Design 15%
- Implementation 60%

- Testing 15%
- Extended requirements 10%

Please see the module web page for the feedback form.

Note.

- Because submission is handled electronically, ANY FILE submitted past the deadline will be considered at least ONE DAY late. Late submissions are subject to the University Policy (see www.csc.liv.ac.uk/department/regulations/practical.html).
- Please make sure your Java classes successfully compile and run on ANY departmental computer system. For this reason, the use of powerful IDEs like NetBeans is discouraged.
- Note this is an individual piece of work. Please note the University Guidelines on Academic Integrity (see https://www.liverpool.ac.uk/media/livacuk/tqsd/code-of-practice-on-assessment/appendix_L_cop_assess.pdf).