COMP122 Assessment 1

Worth 25% of the final course mark.

Submission Deadline

Friday, 9 March 2018, 5:00pm (Friday of Week 6)

Learning Outcomes. This assessment addresses the following learning outcomes of COMP122:

- Describe object hierarchy structure and how to design such a hierarchy of related classes.
- Describe the concept of object polymorphism in theory and demonstrate this concept in practice.
- Identify and describe the task and issues involved in the process of developing interactive products for people, and the techniques used to perform these tasks.

Part I (50% of assessment mark)

Introduction

Download the file `ThreeDice.java` from the COMP122 module website (the link is right below that of this assignment). The first part of 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 (initially) 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.)

Note that the way the class is written, the three numbers given to the constructor do not have to be the integers 1-6, but can be any integers.

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 of three
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 5, 1, 4	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

As you can see, 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. These methods are relying on the fact that the three dice rolls are stored in (non-decreasing) order in the class variables.)

The `printResult()` method is provided mainly for testing this class as it prints the values of the three dice, and calls the methods to determine the type of outcome.

Requirements for Part I

Part I of the assignment builds on the `ThreeDice.java` class described above. The aim is to design, implement, and test a Java program to simulate a "game" between 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 for each round:

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

After all the rounds have been displayed, the following information 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 (i.e. total points divided by the number of rounds played); 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 1, 1, 1	9 + 60 = 69 3 + 60 = 63
A run	sumOfDice + 40	4, 5, 6 3, 1, 2 4, 3, 5	15 + 40 = 55 6 + 40 = 46 12 + 40 = 52
A pair	sumOfDice + 20	1, 1, 4 5, 1, 5	6 + 20 = 26 11 + 20 =31
All different	sumOfDice	2, 3, 6 5, 1, 4	11 10

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

(a) 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.

(b) Design a separate application program that will take as user input a number of rounds to play. The number of rounds should be a positive integer or zero. Negative values must be disallowed. Your application must store the dice rolls for the pair of players as *an array of objects*. This application should display the desired output specified earlier (there is some example output below).

Call your application class `Game.java`.

Note: If desired, you can design other classes (in addition to `ThreeDiceScorer.java`) that are used by your application. Obviously, you should include all classes in your class diagram as appropriate. Make certain that the application class (the Java program with the `main` method) is called `Game.java`.

Rather than asking for user input for each dice roll, use randomly generated numbers (integers between 1 and 6, inclusive).

Recall from previous discussion/examples/practicals that a single random integer in the range 1-6 can be generated using `Math.random()` as follows:

```
1 + (int)(6 * Math.random())
```

Alternatively, you could use the methods from the `Random` class that have been mentioned in the lab practicals to generate these random integers (not forgetting to import the library to use these methods).

(c) Write a separate program, called `Average.java`, that will compute the average number of points over all possible rolls of three six-sided (fair) dice. (How many such combinations of the three dice are possible? See your COMP109 notes to remind yourself.) `Average.java` should (of course) make use of your `ThreeDiceScorer.java` class that you designed above.

(d) Suppose that instead of three fair six-sided dice, two of the dice are normal (fair) six-sided dice, and that the third six-sided die has the numbers $2, 3, 4, 5, 6, 6$ (where each of the possible six sides of this die occurs with equal probability). What is the average number of points over all possible rolls of these dice (two normal six-sided dice and one with a repeated 6 on it, in place of the 1)?

You only need to state the average here, and how it compares to the average you have obtained in Part (c). (I am assuming that you will slightly alter your program in Part (c) to answer this question here.)

Example output for Game.java

Below, output is provided for a game with five rounds. (In the actual ouput from my program, each round summary appears on one line. It is wrapped here in order to fit on the page.)

```
Input the number of rounds to play (min 0):  5
Round 1   Player 1: 1 2 2  Points: 25  Player 2: 1 1 5  Points: 27
      Round winner is player 2.
Round 2   Player 1: 2 3 3  Points: 28  Player 2: 1 3 3  Points: 27
      Round winner is player 1.
Round 3   Player 1: 3 5 5  Points: 33  Player 2: 3 4 5  Points: 52
      Round winner is player 2.
Round 4   Player 1: 1 3 6  Points: 10  Player 2: 1 4 5  Points: 10
      Round is tied!
Round 5   Player 1: 1 1 3  Points: 25  Player 2: 1 5 6  Points: 12
      Round winner is player 1.
```

```
Total wins:
     Player 1: 2     Player 2: 2
Total points:
     Player 1: 121     Player 2: 128
Average points per round:
     Player 1: 24.2     Player 2: 25.6

Overall points winner is player 2.
```

Submission

See the section "Submission Instructions" towards the end of the assessment for exactly what you should submit. (It's similar for each of the two parts of the assessment.)

Part II (50% of assessment mark)

This part of the assessment is about designing a hierarchy of Java classes. Please carefully read the entire description of the assessment before beginning to work on it.

Introduction

The idea here is that you are writing a program to display information to a potential client for a company that supplies combined broadband, TV, and mobile phone packages.

This company (currently) provides three different broadband, TV, and mobile phone packages which are summarised below. Each account type has a basic flat cost (per month), and additional charges that depend upon the number of minutes used on the phone and broadband usage charges. (Happily for all customers, text messages are free...)

	Bronze	Silver	Gold
Package Costs (monthly)	£36.00	£46.00	£61.00
Daytime phone costs (per minute)	12p	12p	0p
Evening/Weekend phone costs (per minute)	5p	0p	0p
Number of TV Channels	60	130	230
Broadband (included per month)	500Mb	1000Mb	1520Mb
Broadband (per Mb above included amount)	2p	1p	1p

Additionally, the Silver and Gold accounts provide a free Spotify account, and Gold accounts include music on demand.

This information can be used by potential customers to decide which package is the most suitable for them given their recent phone call and broadband usage.

The idea for this part of the assessment is that you are going to use Java's inheritance mechanism to implement a set of classes to store information about the company's packages, and display costs to a potential client.

Requirements for Part II

Make certain to read the "Hints/Suggestions" subsection below before starting your implementation.

(a) A user should be able to input the number of daytime phone minutes that she/he uses, the number of evening/weekend minutes that she/he uses, and the amount of broadband usage, expressed in Mb (megabytes), for a given month. These values should all be integer values, greater than or equal to zero (i.e. negative values should be disallowed and the user asked to re-input these values).

(b) For each account (Bronze, Silver, and Gold), the account information should be printed, the total cost of each type of call should be calculated and printed, the total cost for (extra) broadband usage, and the total cost of that account (made up from the call costs, broadband costs, plus the package costs) should be calculated and printed.

(c) Additionally, the program should output which account has the total cost that is the cheapest. If two (or more) accounts have the same cost, which would you recommend to the customer?

See an example of output below.

Call your application program `AccountUser.java`.

(d) This question requires no code to be written, just a short description in your report for the assessment.

Suppose that the company wants to introduce a "Platinum Account", which has a monthly cost of £75, increases the number of TV channels to 275, and has unlimited broadband (i.e. no set limit on broadband included, so no "additional" costs on a per Mb basis). Otherwise, all other features about the Platimum Account are the same as the Gold Account.

How could you implement this additional account, in terms of your class hierarchy and the way you have implemented your application program?

Describe clearly what you would do to add this new account type. What would you do in terms of Java programming, e.g. modifying/overriding methods, adding new constants, etc?

Example program output

```
$ java AccountUser
Please enter the number of daytime minutes used per month: 100
Please enter the number of nighttime minutes used per month: 60
Please enter the number of Megabytes used per month: 1100

Account Summary for Bronze Account
Package Cost: 36.00
Cost of daytime calls: 0.12/min
Cost of evening and weekend calls: 0.05/min
Number of Channels: 60
Broadband Included: 500Mb
Broadband Cost (above included limit): 0.02/Mb
Total daytime calls cost: 12.00
Total evening calls cost: 3.00
Total (extra) broadband cost: 12.00
Total cost: 63.00

Account Summary for Silver Account
Package Cost: 46.00
Cost of daytime calls: 0.12/min
Cost of evening and weekend calls: 0.00/min
Number of Channels: 130
Broadband Included: 1000Mb
Broadband Cost (above included limit): 0.01/Mb
Total daytime calls cost: 12.00
Total evening calls cost: 0.00
Total (extra) broadband cost: 1.00
Total cost: 59.00
Spotify Account provided
```

```
Account Summary for Gold Account
Package Cost: 61.00
Cost of daytime calls: 0.00/min
Cost of evening and weekend calls: 0.00/min
Number of Channels: 230
Broadband Included: 1520Mb
Broadband Cost (above included limit): 0.01/Mb
Total daytime calls cost: 0.00
Total evening calls cost: 0.00
Total (extra) broadband cost: 0.00
Total cost: 61.00
Spotify Account provided
Music on Demand provided

Silver Account is cheapest cost.
```

Hints/Suggestions

1. You will need (at least) four Java classes: `BronzeAccount`, `SilverAccount`, and `GoldAccount`, and an application class.

2. I strongly suggest making an **`abstract`** class called (for example) `StandardAccount` that will be the base class of a hierarchy of classes. The classes `BronzeAccount`, `SilverAccount`, and `GoldAccount` can each extend this `StandardAccount` class.

   Make the class attributes in `StandardAccount` to be **`protected`** (not **`private`**) because then subclasses that extend `StandardAccount` can easily modify these class attributes as appropriate.

   With careful thought and planning, most of the Java code for the program operations (such as computing the costs of mobile phone charges, broadband charges, etc) can be placed in the `StandardAccount` class. These methods can be overridden (if needed) in the classes that extend it. So the code for the `BronzeAccount` class can be quite short (similarly for the other classes that extend `StandardAccount`).

   You could also include, for example, an **`abstract`** method in `StandardAccount` called `accountType()` which returns a `String` for the type of account (e.g. "Bronze"). Since that method is **`abstract`**, it must be overridden (implemented) in the classes that extend `StandardAccount`.

3. If you find yourself repeating code, or having very similar code, in (for example) your `BronzeAccount` class and `SilverAccount` class, you're probably going about making your classes in the wrong way. You shouldn't need to repeat lots of code across classes. Repeating lots of code goes against the idea of "code re-use" and the "write once, test once, re-use often" idea. As I said above, put most of your calculations (and class attributes) into a `StandardAccount` class that each other class will extend.

4. I also suggest it can be useful to create a Java **`interface`** that will contain various constants that describe the charges per minute, (extra) broadband charges per Mb, number of channels included in the package, etc. Each class can implement this interface and then have access to the constants which are all in one place. (If account charges were to change, then you would only need to edit this one interface, and re-compile the interface, instead of having to modify several individual class files to update package charges.)

5. You must check the number of minutes input (for daytime and evening/weekend calls) and the amount of broadband usage, and disallow the input of negative values and ask the user to re-input these. You can assume that each of these values is an integer.

Submission Instructions

Your submission should consist of a report (a PDF file) and implementation (source code) files. Be certain to include all Java source files (i.e. the ".java" files) needed to run your application.

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

  Requirements: Summary of the above requirements statement in your own words. Do this for each part of the assessment.

  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. Again, do this for each part of the assessment.

  For Part I you must have (at least) four class files, the (unaltered) `ThreeDice.java` file, your `ThreeDiceScorer.java` file, the `Game.java` application file, and `Average.java` for computing the average possible score for a single throw of the three dice.

  For Part II, your solution should comprise (at least) four classes: an application class called `AccountUser.java`, and one class file for each of the Bronze, Silver, and Gold accounts. If you make the suggested `StandardAccount.java` file, you'll have (at least) five classes. If you also have an **`interface`** as suggested, you'll have (at least) six classes.

  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

Friday, 9 March 2018, 5:00pm (Friday of Week 6)

Notes/Penalties

- Because submission is handled electronically, ANY FILE submitted past the deadline will be considered at least ONE DAY late (penalty days include weekends since assessment is performed electronically). Late submissions are subject to the University Policy (see Section 6 of the Code of Practice on Assessment).
- Please make sure your Java classes successfully compile and run on ANY departmental computer system. For this reason, the use of IDEs like NetBeans and Eclipse is discouraged. If your Java source files do not successfully compile/run, you could suffer a penalty of 5 marks from your grade. This penalty could be applied for Part I and/or Part II of the assessment, but only once per part. (However, this will not turn a passing grade into a failing grade.)
- If your report is not a PDF file, you will lose 5 marks from your final grade. (However, this will not turn a passing grade into a failing grade.)

- If you use any form of compression other than "zip", you risk your files not being read by the assessors, resulting in a mark of 0! If your files can be read, you will still lose 5 marks from your final grade. (As before, this will not turn a passing grade into a failing grade.)
- Note this is an individual piece of work. Please note the University Guidelines on Academic Integrity (see Appendix L of the Code of Practice on Assessment). You should expect that plagiarism detection software will be run on your submission to compare your work to that of other students. (Obviously in this case, the `ThreeDice.java` class file will be ignored as it has been supplied to you.)

Mark Scheme

Marks for each part of the assessment will be awarded for:

- Analysis and Design 10% (This includes things like UML diagrams for your classes/application, justification for how/why you structure your classes, appropriate pseudocode for methods, etc.)
- Implementation 25% (This includes correctness of your programs, appropriate level of comments, correct indentation so your code is readable, having useful identifiers, etc.)
- Testing 10% (Have you done suitable testing in terms of checking different paths through your code, checking what you do with "unexpected" inputs, a sufficient number of tests, etc.?
- Extra questions (such as (c) and (d) in Part I or (d) in Part II) 5%

Please see the module web page for the feedback form.