Software Development 1, Coursework 3

This is individual assessed coursework. You are allowed to discuss this assessment with other students, but you should not copy their code, and you should not share your own code with other students. We will carry out plagiarism checks on submissions.

Objectives

This coursework contributes 25% of your overall mark for the course. It assesses your understanding of topics covered in Weeks 7, 8, 9, 10, and 11:

- Using inheritance and polymorphism in OOP
- Using methods, passing variables, and dealing with return types
- Overriding and overloading methods
- Using Java Collections and Exceptions
- String manipulation
- Loading from a file

These topics are all covered in the lectures and Live Coding sessions on Canvas. It is also recommended that you work through the tutorial exercises from these weeks before attempting the coursework, since these help build the understanding you need to complete the coursework.

Submission

The **deadline** for completing this coursework is **Monday 28th November (Week 12)**. Make sure you do these two things:

1) **Commit your work to GitLab** before this deadline. If you don't do this, a late penalty may be applied. There are instructions on using GitLab in the Week 1 familiarisation tutorial.
2) **For students in Edinburgh, attend a lab session and ask someone to mark your work**. After they've done this, they will give you feedback on your work. You may get your work marked before the deadline or in the week following the deadline without a penalty, so long as you committed it to GitLab before the deadline.

Do This First: Fork and Import the GitLab Project

You will be adding your code to an existing project in GitLab. The first thing you need to do is fork this project, so that you'll be working on your own copy, and import it into Eclipse:

1. Visit https://gitlab-student.macs.hw.ac.uk/F27SA_2022-23/f27sa_2022-23_coursework-3
2. Fork the project
3. Import your forked project into Eclipse

! ***Don't forget to fork the project before you import it***
if you don't do this, you won't be able to save your changes back to Gitlab.

If you don't know how to do any of this, then make sure you first work through the Week 1 tutorial, which will take you through the process step by step. If you're having trouble getting it to work, then come to a timetabled lab session and talk to a lab helper, well before the submission deadline.

(continues on next page…)

The Behaviour of Your Programme

The GitLab project contains two classes: **MainClass.java** and **GameSuperclass.java**. These will form the basic structure of your code, but you will be expected to make new classes to complete this coursework. If you fail to create the correct classes, you will not get full marks.

The programme is a simple suite of text-based games or game-helping tools for players to use. When executed, it will ask the user which of three options they wish to pick from, and then allow them to interact with the option selected.

The final programme will have the following three games:

1. Dice Roller
2. Higher or Lower
3. Hangman

Each of these 'games' will have its behaviour in a separate class which should inherit from the **GameSuperclass.java** file. You should notice this is an abstract class, and as such you will be expected to fill in the method functionality for each game. You should start by adding a correctly encapsulated **gameName** String and setting it in the constructor.

The **MainClass.java** file will act as the interface class to the user, and deal with all their input. We'll start with the behaviour of **MainClass.java**. It should follow the same basic structure for each game:

1. Ask the user which game they want to access (1, 2, or 3)
2. Create an instance of the correct subclass of **GameSuperclass.java** and storing it as a variable of type **GameSuperclass**.
3. Call the **.printGameIntro()** method for said instance
4. Enter a loop in which it should:
    a. Accept an input from the user
    b. Pass this to the game using the **.processInput()** method
    c. Loop if the method returns true, or continue if it returns false
5. Thank the user for playing, and quit the programme

An example of a full interaction with your programme would look like this:

```
Welcome to the Game Lounge! Please pick your game:
1. Dice Roller
2. Higher or Lower
3. Hangman
1
Welcome to Dice Roller
Please input how many D6s you want to roll:
4
Dice rolled:
Die 1 rolled 6 | Die 2 rolled 4 | Die 3 rolled 2 | Die 4 rolled 6
All dice rolled!
Thank you for playing, goodbye!
```

(continues on next page…)

Game #1: Dice Roller

This is the first game which you should implement and is also the simplest. You should create a subclass of **GameSuperclass.java** called **DiceRoller.java** where all the 'game specific' code for this game would reside.

In this 'game', the user will be asked to input a number of D6 (or six-sided) dice in the **.printGameIntro()** method. Once a valid integer is input, it would then simulate rolling that many dice. The rolling should be done using the **Random** class in Java. You may need to consult the API for how to use this.

For each die rolled, your code should print "Die x rolled y" where x is which number die it was, and y is the value rolled. These should all be printed on a single line and separated by | symbols. This should look like the following if done correctly:

```
Please input how many D6s you want to roll:
4
Dice rolled:
Die 1 rolled 6 | Die 2 rolled 4 | Die 3 rolled 2 | Die 4 rolled 6
All dice rolled!
```

Note that for full marks you should ensure you only put the | symbols between the dice rolls, and not at the end.

Game #2: Higher or Lower

This is the second game which you should implement and is slightly more complex than the Dice Roller. You should create another subclass of **GameSuperclass.java** called **HigherOrLower.java** where all the 'game specific' code for this game would reside.

In this game, the user will be asked to guess a number between 0 and 100 (inclusive) which your programme has randomly selected. You should use the **Random** class in Java again. Your code will then compare the user's guess against its random number and one of the following three things should happen:

1. If the guess is too low, the programme should inform the user and let them guess again
2. If the guess is too high, the programme should inform the user and let them guess again
3. If the guess is exactly right, the programme should inform the user how many guesses they took, before exiting.

Remember, that the **.processInput()** method returns a boolean value which should tell **MainClass.java** whether to look for another input value from the user, or to quit the programme. You should use this, alongside the correct type of loop, to allow the user to guess as many times as necessary. The guess tracking, however, should be done within **HigherOrLower.java**.

Once working, the game output should look like the following:

(continues on next page…)

```
Welcome to Higher or Lower
I'm thinking of a number between 0 and 100 (inclusive)
Please input your first guess:
50
Sorry, my number is HIGHER than that! Guess again:
70
Sorry, my number is LOWER than that! Guess again:
65
Sorry, my number is LOWER than that! Guess again:
58
Sorry, my number is LOWER than that! Guess again:
51
Correct! My number was: 51
You got it right in 5 guesses!
Thank you for playing, goodbye!
```

Game #3: Hangman

This is the third and final game which you should implement, and is more complex that the other two, as it will require loading from a file, and some more complex String manipulation. You should create a third subclass of **GameSuperclass.java** called **Hangman.java** where all the 'game specific' code for this game would reside.

In this game, the user will be asked to guess letters in a word which your programme has randomly selected from a provided list of words. Your code should keep track of all letters guessed, and after each guess should show the user a String which contains any correctly guessed letters in situ. Assume the word to guess is 'programming' and the user has guessed 'g', 'm', 'p', and 't' you would show a String in the following format: p__g__mm__g

Currently, however, the superclass you've been using cannot accept anything but an int value into the **.processInput()** method. You should overload this method in the superclass to include a method which can accept a char. Note: you will have to then include this new version of the method in both **DiceRoller.java** and **HigherOrLower.java**. In both cases, they should tell the user to enter a letter and not a number, and allow them to input a new value.

To help you determine whether the input value is an integer or not, there is a skeleton method provided in **MainClass.java**, which you should use before calling the different versions of the **.processInput()** method. You will need to complete this before it can be used.

You have also been provided a text file called **wordList.txt** which you should use to pick a random word for the user to guess. This should be placed in root folder of your Java project so you can load it. There is more information about this, and how to process the resultant String in the Week 11 Live Coding Sessions.

(continues on next page…)

Your **Hangman.java** code should work in the following way:

1. When **.printGameIntro()** is called, your programme should load the list of words from the provided file. You can use a **BufferedReader** and its **.readLine()** method to do this
2. You will then need to split the comma separated String into separate words, and store them in a word list. We would suggest storing these in an **ArrayList<String>** as you don't know before loading how many words it will contain.
3. Your code should then randomly select a single **String** from the stored word list. This is the target word for this game
4. In your new, overloaded .processInput() method your programme should:
    a. Check if the user has already guessed that letter – you will need to keep track of this. We would suggest using an **ArrayList<Character>** variable. If the user has already guessed that letter, the programme should inform the user of such
    b. See if the target word contains the letter guessed. Either way the programme should inform the user
    c. Check if the word has been entirely guessed: if it has, congratulate the user and exit the programme, if the programme should show the user the current state of the guessed word and let them guess again.

Once you have it working, it should look something like this to the user (this has been shortened):

```
Welcome to Hangman
I'm thinking of a word!
Please input your first guess at a letter:
e
Yes, my word does have that letter!
The word now looks like this: ___e_________
Please guess another letter:
...
f
Sorry, the word I'm thinking of doesn't have that letter!
The word now looks like this: i__esti__tio_
Please guess another letter:
...
n
Yes, my word does have that letter!
The word now looks like this: investi__tion
Please guess another letter:
n
You already guessed that letter. Pick a different one!
The word now looks like this: investi__tion
Please guess another letter:
g
...
Yes, my word does have that letter!
You've guessed my word! It was: investigation
You needed to guess 11 letters.
```

(continues on next page…)

Note: you should not count repeated letters in the number of guesses for the user. You should only count the number of **unique letters guessed** to get full marks for this game.

Marks

You will get a mark out of 25 for your work. Here is a guide to how your work will be marked, though the final mark is up to the person marking your work, and will reflect the understanding of the course materials shown in your work**:**

- 5 marks for the getting the Main Class structure basics working:
  - 1 mark for getting the game selection working (with any number of games)
  - 1 mark for calling the correct methods in the instantiated class
  - 1 mark for updating the GameSuperclass correctly
  - 2 marks for writing and using the isInteger method correctly
- 5 marks for the Dice Roller game split as follows:
  - 1 mark for correctly inheriting the class
  - 1 mark for rolling the correct number of dice
  - 1 mark for ensuring you roll between 1 and 6 inclusive
  - 1 mark for printing out the results
  - 1 mark for ensuring the separator is used correctly
- 6 marks for the Higher or Lower game split as follows:
  - 1 mark for correctly inheriting the class
  - 1 mark for selecting a number between 0 and 100 inclusive
  - 2 marks for allowing the user to guess as many times as necessary
  - 1 mark for informing the user to guess higher or lower correctly
  - 1 mark for counting the number of guesses correctly
- 9 marks for the Hangman game split as follows:
  - 1 mark for adding the correct overloaded method
  - 2 marks for loading in the word list, including catching the correct exceptions
  - 1 mark for splitting the String into separate words in an ArrayList and picking one
  - 5 marks for allowing players to guess letters and responding as defined

Late submissions will be marked according to the university's late submissions policy, i.e. a 30% deduction if submitted within 5 working days of the original deadline, and no mark after that.

If you have mitigating circumstances (e.g. illness), please submit a mitigating circumstances application, as described at:

https://www.hw.ac.uk/students/studies/examinations/mitigating-circumstances.htm