COMP30820 Main Project

Last Updated: 1st April 2021

Submission Details

Deadline: May 7th 2021, 5pm (Uploaded to Brightspace).

Late Submissions Standard UCD policy on late submissions applies; see `https://www.ucd.ie/t4cms/latesub_po.pdf`. Your submission is deemed late if at least one deliverable is submitted late.

Plagiarism: The submission must be yours and yours alone. If you are unsure what is or is not plagiarism, the following is a none exhaustive list of example activities you cannot do:

- Copy the completed files of another student and submit them as your own
- Share copies, images or print outs of your code with another student (by e-mail, FB messenger, WhatsApp etc.)
- A group of students working on a single solution and then all submitting the same work or subset of the same work (regardless of whether variable, method, class names or ordering have been changed)
- Students collaborating at too detailed a level. For example, consulting each other after each line / block / segment of code and/or sharing the results.

For more details see:

- `https://csintranet.ucd.ie/sites/default/files/cs-plagiarism-policy_sept2020.pdf`, and
- `https://www.ucd.ie/governance/resources/policypage-plagiarismpolicy/`
- `https://www.ucd.ie/secca/studentconduct/`

Instructions

For this project, you should write a program to implement a simple games system with points and a leaderboard.

Minimum Specification: At a minimum, the program should work as follows:

1. Begin with a main menu that allows the user to choose to start as a new player or quit.
2. If the user chooses to start as a new player, the program should prompt the user to enter a name for the player, and then allow the player to choose which game to play. Your system should have two or three very simple games.
3. After playing the game, the user can choose to play a different game or to finish playing and return to the main menu.
4. At the main menu, if the user chooses to quit, the program should display a of all players who have played, sorted by their points totals.

You should have at least two games, but these games can be very simple single-player games. For example, a simple version of roulette, blackjack, or betting on the outcome of a coinflip, or Rock Paper Scissors, or guessing game. Each of these games could be implemented in a single method. Your program should be console based (i.e., no GUI is needed). You might choose to start with each player having zero points, and winning points for playing a game successfully, or, to start with a number of points which they can 'bet' in the games. Any simple game that has at most a small number of interactions and involves mapping a random number to a game state/outcome would be appropriate.

Advanced Specification For full marks, consider the following extensions to the above specification:

1. You will need to define a Player class to store information associated with each Player. For full marks, extend the Player class to allow different kinds of players, for example VIPPlayer and LimitedPlayer that have bonuses or protections that affect how they play, for example limiting the amount they can bet in a turn, or changing how their name is displayed on the leaderboard.

2. The leaderboard will store the player names and scores while the system is running. For extra marks, store this information in a text file so that it will persist between runs of the system. For this, you only need to make the leaderboard persistent, not the players. I.e., you don't need to allow the same player to resume playing after another player has finished or exited.

3. For full marks, each game should have multiple interactions (i.e. not finish after a single user action).

4. Here is an example.

```
Please choose an option:
1. New Player
2. Quit
1
Please enter a name:
Alice
Hello Alice. Please choose a game, or -1 to quit:
1: Lottery
2: Coin flip
1
[ .. Lottery game runs ]
Hello Alice. Please choose a game, or -1 to quit:
2
[... Coinflip game runs..]
Hello Alice. Please choose a game, or -1 to quit:
-1
Please choose an option:
1. New Player
2. Quit
1
Please enter a name
Bob
Hello Bob. Please choose a game, or -1 to quit:
1: Lottery
2: Coin flip
1
[... Lottery game runs..]
Hello Bob. Please choose a game, or -1 to quit:
-1
-1
Please choose an option:
1. New Player
2. Quit
2
Player :  Points
Alice (VIP) : 40
Bob        :  10
```

Deliverable

Code All .java files as a .zip archive.

Report In addition to your code, write a short report (approx 1000 words) that explains your project and outlines how you made use of programming concepts covered in the course (such as access modifiers, getters/setters, datastructures). Include a class diagram with corresponding discussion on class design choices and decisions made

Tips and Marking guidelines

- **Get the menu logic working first.** The main skill tested for this project is organising the menu and player structure. Get this logic working first, and then make the games. For example, you could use placeholder methods that award random points for the games until you have the main game and leaderboard logic working.
- **Workload** It is expected that this project requires 16-20 hours to complete. This project is worth 30% of your mark for this module.
- **Marking breakdown** The project will be marked out of 100, with 50 marks for program functionality (specification), 30 for program design and style, and 20 for the accompanying report.
- **Minimum/Advanced** The minimum specification corresponds to a B grade (75/100).

Criteria	A+, A, A-	B+, B, B-	C+, C, C-	D+, D, D-	≤ E+
Specification (50/100)	System fulfils all of the minimum and some or all of the advanced specification requirements. The program is robust to unexpected user input.	Program works perfectly on minimum specification with some appriate checking for bad user input.	Minor logic errors, or logic errors in one part of the program flow	Program compiles and runs but lacks major functionality	Program does not compile, or performs little to none of the required behaviour
Design (30/100)	The code is easy to read, appropriately commented, and uses an efficient solution. object-oriented design principles are used throughout.	Some minor faults in presentation, such as inadequate comments, poor variable naming, or a slightly awkward approach; or, design does not make proper use of object-oriented design principles.	Design is severely lacking modularity or other object-oriented principles and also does not adhere to proper style conventions.	A major problem in implementation design in logic or class design that makes the system very difficult to read or refactor.	Major problems in all of the above.
Report (20/100)	Well-written report that discusses all aspects of design and datastructures and includes UML diagram	Some aspects of design, datastructures, or UML diagram missing, or some writing issues	Report is incomplete or unclear in several areas.	Many aspects of the report incomplete.	Major problems in all of the above.