ITEC625 - Fundamentals of Computer Science

Assignment 2 - Worth 15%

Due Date: 4th November, 11:45pm

Introduction

This goal of this assignment is for you to implement a simple dice rolling game called "Plus and Minus", the rules can be found here
There are some limitations with the design of the game because we are limited by the scope of the unit.

The assignment is structured in 3 stages and students are required to make the submission for each stage by the dates mentioned below:

1. **Dice**: Released 15th October, **Due Sunday 21st October 23:45**. Solution released 23rd October 09:00. Expected time required to completed this task: Between 10 minutes and 2 hours.
2. **Computer**: Released 23th October, **Due Monday 29th October 11:00**. Solution released 31st October 4:00. Expected time required to completed this task: Between 1 hour and 5 hours.
3. **ScoreSheet**: Released 31st October, **Due Wednesday 7th November 23:45**. Solution released 9th November 11:00. Expected time required to completed this task: Between 1 hour and 5 hours.

Students can make submissions for each stage at any time before the deadline for that stage.

Stage 1. Dice - *Completed*

(Aside: although 'die' is traditionally considered the singular form of 'dice' modern english allows for 'dice' as both singular and plural).

Each dice has five instance variables:

1. *id* - a unique number to identify the die.
2. *faces* - How many different values can rolling the die produce.
3. *value* - The last value rolled.
4. *random* - a random number source.
5. *DEF_FACES* - a constant indicating the default number of faces.

You should also complete the following methods:

1. *Dice(int faces)* - construct a new die with an appropriate number of *faces*.
2. *Dice()* - construct a new die with the default number of *faces* (6).
3. *rng(int min, int max)* - random number generator: a wrapper method to generate random numbers.
4. *getId()* - returns the unique identifier for the die.
5. *getValue()* - returns the last value rolled.
6. *roll()* - generate a new value for the dice.
7. *toString()* - generate a string representation of the dice. For example, if your dice's has 7 faces, it's ID is 3 and you roll a 4 it's string would be '3:4/7'.
8. *compareTo(Dice other)* - compare the face value of your dice to the face value of another dice. Return -1 if your dice is less than *other*, 0 if they have the same face value, and 1 if the face value of your dice is greater than that of *other*.

You are required to complete the methods of the class based on javadoc comments.

Stage 2. Computer

This corresponds to a the computer player of the game.

The computer has only one instance variable *name* which contains the players name.

You will need to complete the following methods:

1. *Computer(String name)* - construct a new computer player with name *name*.
2. *maxFaces(ArrayList dice)* - return an *int* indicating the highest face value on a single dice from a set of dice.
3. *lowest(ArrayList dice)* - return an *int* containing the id of the lowest dice rolled in a set. (If multiple dice have an equally low value, return the first match)
4. *highest(ArrayList dice)* - return an *int* containing the id of the highest dice rolled in a set. (If multiple dice have an equally high value, return the first match)
5. *secondHighest(ArrayList dice)* - return an *int* containing the id of the *second highest* dice rolled in a set. (If multiple dice have an equally second highest value, return the first match)
6. *chooseOne(ArrayList dice)* - return the id of a single dice to be set aside. The dice should be strategically chosen in order to maximise the score for the round. Depending upon the round either the lowest or highest dice rolled should be set aside.
7. *chooseTwo(ArrayList dice)* - return the ids of two dice to be set aside. The dice should be strategically chosen in order to maximise the score for the round. Depending upon the round either the lowest or highest dice rolled should be set aside.
8. *toString()* - return the name of the player that was assigned in the constructor.

Stage 3. ScoreSheet

This class tracks the scoring of the game.

You will need to complete the following methods:

1. *commaSeparated(ArrayList l)* - return a *string* containing the elements of the list with commas between each element.
   eg. If the list is [2, 5, 7, 9, 11] the function should return "2,5,7,9,11".
2. *int tallyScore(String n)* - caclulate the total score for the specified player.
3. *ArrayList playerRound(String n, int r)* - return a list of *Turn* containing all the recorded dice rolls and dice set aside for the specified player and round.
4. *void log(String n, int r, int t, ArrayList k, ArrayList a)* - record the dice rolled and set aside for the specified player, round, and turn in the score table.

4. Marking Guide

Marking will be test-driven using the JUnit framework. There are 20 tests in all - 5 in Dice, 8 in Computer, and the remaining 6 from the final stage. Each of the tests is worth 5 marks.

5. Penalties to be applied

- 1 mark for every hour by which assignment is late. Between 1 second and 59 minutes 59 seconds: 1 mark penalty, between 1 hour and 1 hour 59 minutes 59 seconds: 2 mark penalty, and so on.
- 5 mark penalty for indentation error at one or more places. Code must be immaculately indented. Use the same style throughout the code. Any inconsistencies will be penalised. Leave a blank line between each method. Unnecessary blank lines will be penalised.
- 5 mark penalty for not enough commenting where warranted. We will be fairly forgiving on this point since we understand commenting in not something that you learn within a year.
- 4 mark penalty for not adding you name and ID as a comment on top of one or more files.