ASSIGNMENT 2

COMP-202, Fall 2017, All Sections

Due: Friday, October 13^{th} 2017, (23:59)

Please read the entire PDF before starting. You must do this assignment individually.

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

To get full marks, you must:

- Follow all directions below
- Make sure that your code compiles
  - Non-compiling code will receive a very low mark
- Write your name and student ID as a comment in all .java files you hand in

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Implement a working version of the warm-up Question 5 from Assignment 1. That is, write a method `swap` which takes as input two int values x and y. Your method should do 3 things:

1. Print the value of x and y
2. Swap the values of the variables x and y, so that whatever was in x is now in y and whatever was in y is now in x
3. Print the value of x and y again.

For example, if your method is called as follows: `swap(3,4)` the effect of calling your method should be the following printing

```
inside swap:  x is:3 y is:4
inside swap:  x is:4 y is:3
```

Warm-up Question 2 (0 points)

Consider the program you have just written. Create 2 int (integer) variables in the main method. Call them x and y. Assign values to them and call the swap method you wrote in the previous part using x and y as input parameters.

After calling the `swap()` method—inside the main method— print the values of x and y. Are they different than before? Why or why not?

Warm-up Question 3 (0 points)

Write a method that takes three integers x, y, and z as input. This method returns true if z is equal to 3 or if z is equal to the sum of x and y, and false otherwise.

Warm-up Question 4 (0 points)

Let's write a method incrementally:

1. Start by writing a method called `getRandomNumber` that takes no inputs, and returns a random `double` between 0 (included) and 10 (excluded).
2. Now, modify it so that it returns a random `int` between 0 and 10 (still excluded).
3. Finally, let the method take two integers `min` and `max` as inputs, and return a random integer greater than or equal to `min` and less than `max`.

Warm-up Question 5 (0 points)

Create a file called `Counting.java`, and in this file, declare a class called `Counting`. This program takes as input from the user (using `args`) a positive integer and counts up until that number. eg:

```
> run Counting 10
I am counting until 10:  1 2 3 4 5 6 7 8 9 10
```

Warm-up Question 6 (0 points)

For this question you have to generalize the last question. The user will give you the number they want the computer to count up to and the step size by which it will do so.

```
> run Counting 25 3
I am counting to 25 with a step size of 3:
1 4 7 10 13 16 19 22 25
```

Part 2

The question in this part of the assignment will be graded.

Question 1: Mexico (100 points)

Mexico is an elimination dice game composed by different rounds. To play, all that is required is two dice. At the start of the game, all players accept to gamble a fixed amount of money (this is similar to a "buy in" in poker). At the end of each round, the player with the lowest score puts a predetermined portion of that money into the pot. For instance, players might start out with $25 each, having agreed in advance that each round will cost the loser of that round $5. The game ends when enough rounds have been played that only one player with any money remains, at which point the pot is his. Thus, following the above example, three players with five betting units of five dollars each might play a minimum of ten and a maximum of fourteen rounds (excluding ties) before a winner emerges.[1]

The goal of this question is to write several methods in order to create a program that simulates two players, Giulia and David, playing a game of Mexico. All the code for this question must be placed in a file named Mexico.java.

A round of Mexico

Mexico is a game of rounds. For the purpose of this question we will assume that each of the two players is allowed to roll the two dice only once per round. On each roll, the numerical values of the two dice are combined into a two-digit number, assigning a tens-column value to the higher of the two dice and a ones-column value to the lower. Thus, a roll of 4-2 would translate into a result of forty-two, a roll of 4-5 would be fifty-four, a roll of 6-5 would be sixty-five, and so on. The results of the rolls are ranked from highest to lowest, based on the following scoring system:

1. The 2-1 roll. This is the highest score, the "Mexico" roll after which the game is named. This score is unbeatable, ranking above all others.
2. Doubles are next. They are ranked numerically (6-6 ranks highest, followed by 5-5, and so on down to 1-1). Doubles are worth more than any mixed roll.
3. Finally the mixed rolls are ranked based on their numerical value. Thus, the highest mixed roll is a sixty-five, while the lowest possible mixed roll is a thirty-one.

The player ending the round with the lowest score has to put into the pot an amount of money equal to the predetermined base bet. Players will keep playing until only one player is left with money from the buy-in made at the start of the game.

Let's see an example of 2 players playing a game of Mexico with a buy in of $10 and a base bet of $5:

- Round 1:
  - Player 1 rolls a 5 and a 2, scoring a 52.
  - Player 2 rolls a 1 and a 2, scoring a Mexico!
  - Player 2 wins the round, and Player 1 has to pay $5 into the pot. At this point, Player 1 is left with $5, while Player 2 has still $10.
- Round 2:
  - Player 1 rolls a 4 and a 4, scoring a 44.
  - Player 2 rolls a 5 and a 6, scoring a 65.
  - Player 1 wins the round, and Player 2 has to pay $5 into the pot. At this point, both Player 1 and Player 2 are left with $5.

[1] From wikipedia: $https://en.wikipedia.org/wiki/Mexico_(game)$.

- Round 3:
  - Player 1 rolls a 5 and a 5, scoring a 55.
  - Player 2 rolls a 1 and a 1, scoring an 11 .
  - Player 1 wins the round, and Player 2 has to pay $5 into the pot. At this point, Player 2 is left with no money, and Player 1 wins the game.

Now that we know how the game works, let's look at the methods we need in order to simulate two players, Giulia and David, playing the game of Mexico.

1a. A method to simulate a dice roll

In a game of Mexico, players are betting on the outcome of a roll of two six-sided dice. Write a method called `diceRoll` that simulates the roll of one six-sided dice. Such method takes no inputs, and it returns an integer between 1 and 6 (included). Your method must use `Math.random()` in order to achieve the desired result. If you have any doubts on how to write the method, make sure you first understand the warm-up Question 4, where you are asked to build a method called `getRandomNumbers`.

1b. A method to compute the score of a Player

Write a method `getScore` that computes the score of a player given the dice roll. This method takes as input two integer values that corresponds to the numbers the player has rolled. It should return an int indicating the score achieved. Remember that, the score is obtained by combining together the numerical values of the two dice into a two-digit number, assigning a tens-column value to the higher of the two dice and a ones-column value to the lower. For example, if the method is called with inputs 4 and 2, the value returned should be the integer 42. If the method is called with inputs 4 and 5, the value returned should be 54.

1c. A method to simulate one round of Mexico

Write a method `playOneRound` that simulates one player playing one round of Mexico. This method takes as input a String with the name of the player rolling the dice, and returns an int representing the score obtained by such player. The method must use `diceRoll` to simulate the dice roll, and `getScore` to get the score obtained by such roll. The method also prints the result of the dice roll as well as the score obtained by the player. For example, if the method is called with input `''Giulia''`, then it should print something similar to the following:

```
Giulia rolled:  1 5
Giulia's score is:  51
```

In this case, the method also returns the value 51. On the other hand, if the method is called with input `''David''`, then it should print something similar to the following:

```
David rolled:  4 4
David's score is:  44
```

In this case, the method returns the value 44.

1d. A method to determine the winner of one round

Write a method `getWinner` that determines the winner of one round. This method takes as input two int representing the scores obtained by Giulia and David, respectively. The method returns a String with the name of the winner. In case of a tie, the method returns the String `''tie''`. For example, if the method is called with inputs 44 and 53, it should return `''Giulia''` since the first score is the highest in a game of Mexico. If the method is called with inputs 42 and 42, then it returns `''tie''` indicating that there was a tie.

1e. A method to check if the buy in and the base bet are set correctly

Write a method `canPlay` which takes two doubles as input and returns a boolean value. The first input value corresponds to the value of the buy in, the second corresponds to how much money the players will have to pay each time they lose one round. A game is allowed to be played **only if** the bet is no more than the buy in, and the buy in is a positive multiple of the bet. If a game of Mexico can be played, your method should return true. The method should return false otherwise.

For Example: `canPlay(-5.0, 5)` should return `false`, while `canPlay(10.0, 2.0)` should return `true`.

1f. A method to simulate a game of Mexico

Finally, write a method `playMexico` that simulates a game of Mexico between Giulia and David. This method takes two doubles as input: the first one corresponds to the value of the buy in, the second corresponds to how much money the players will have to pay each time they lose one round. The method will not return any value.

The method should first check if a game of Mexico can be played using the method `canPlay`. If the game cannot be played, your method should print out a statement informing the user that the fund is insufficient in order to play and terminate here the execution of this method. If a game can be played, then you can go ahead and simulate Giulia and David playing a game of Mexico.

Remember that both players will start with an amount of money in their hands equal to the buy in. The method should then start to simulate Giulia and David playing rounds of Mexico. At the end of each round, the player with the lower score will lose an amount of money equal to the base bet. The method should keep simulating Giulia and David playing until a final winner can be determined. The method must use `playOneRound` and `getWinner` in order to simulate the game. The method should also print out which round is being simulated as well as who is the winner of such round. If there is a tie, the method should print out a statement declaring that there was a tie and the players should roll again. When the end of the game is reached, the method should print out the name of the winner.

1g. Setting up the main method (Optional - No extra marks)

Set up the main method so that the program receives two inputs of type double via command line arguments. The first input corresponds to the value of the buy in, the second corresponds to the base bet. From the main method, call `playMexico` with the appropriate inputs in order to simulate Giulia and David playing Mexico. These are a couple of examples of how your program should run:

```
> run Mexico 10.0 5.0
Round 1
```

```
Giulia rolled:  5 2
Giulia's score is:  52
David rolled:  3 5
David's score is:  53
David wins this round

Round 2

Giulia rolled:  3 1
Giulia's score is:  31
David rolled:  4 4
David's score is:  44
David wins this round

David won the game!


> run Mexico 3.0 2.0
Insufficient funds.  The game cannot be played.


> run Mexico 2.0 2.0
Round 1

Giulia rolled:  3 4
Giulia's score is:  43
David rolled:  4 3
David's score is:  43
It's a tie.  Roll again!

Round 2

Giulia rolled:  1 2
Giulia's score is:  21
David rolled:  2 4
David's score is:  42
Giulia wins this round

Giulia won the game!
```

What To Submit

Please put all your files in a folder called Assignment2_ID, where 'ID' should be replaced by your McGill ID number. Zip the folder (please DO NOT rar it) and submit it in MyCourses. Inside your zipped folder there must be the files listed below. **Do not submit any other files, especially .class files**.

`Mexico.java`
`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise they would not.

Marking Scheme

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or bad coding structure. **Marks will be removed as well if the class and methods names are not respected**.

Question 1

Question 1a:	5	points
Question 1b:	10	points
Question 1c:	25	points
Question 1d:	15	points
Question 1e:	7	points
Question 1f:	38	points
	100	**points**