ICS 45J: Programming in Java – Winter 2017
Lab 2
Deadline: 2/3/2017

For this lab, you will write a small simulator on the simplest version of the dice game called Craps: http://casinogambling.about.com/od/craps/a/craps101.htm

Specifically, you will implement the "Passline Bet" version of craps. Using two six-sided die, the rules are as follows:

"You place your bet on the passline before a new shooter begins his roll. This is known as the come out roll. If the shooter rolls a 7 or 11 you win. If the shooter rolls a 2, 3 or 12, you lose. If the shooter rolls any other number, that number becomes the point number.
The shooter must roll that number again before a seven is rolled. If that happens, you win even money for your passline bet. If a seven is rolled before the point number is rolled again, you lose."

When the user rolls a 7 or 11 on the first roll, this is known as "Natural." When the user rolls a 2, 3, or 12 on the first roll, this is known as "Craps."

Your simulation will prompt the user for their name, the amount of money the user brings to the table, and the amount of money the user wants to bet. The simulation will continuously run by placing the bet amount continuously until the user's balance is $0. A sample output (with some error cases) for this is as follows (bold text indicates user input):

```
Welcome to SimCraps! Enter your user name: Mustafa
Hello Mustafa!
Enter the amount of money you will bring to the table: 1000
Enter the bet amount between $1 and $1000: 1001
Invalid bet! Please enter a bet between $1 and $1000: -2
Invalid bet! Please enter a bet between $1 and $1000: 10000
```

The username can be any String (even an empty one). You can assume that the user will only enter a valid Integer value for the initial balance brought to the table. However, your program must check that the bet amount is in a valid range between $1 and the user's balance. You must also check and make sure that the bet amount throughout the simulation never exceeds the user's remaining balance. For example, if the original bet was $100 and you only have a balance $50, then the bet for that game will be $50. If you win, and your balance is back up to $100, the next bet will be your original bet of $100.

During the games played in the simulation, you will need to print out the actions to the console (i.e. what the user is rolling, if the user won, if the user lost, what the user is betting, and what the user's current balance is). A sample output for a few games of craps is as follows:

```
Richert bets $100
Rolled a 5
Rolled a 7
*****Crap out! You loose.*****
Richert's balance: 900. Playing a new game...
Richert bets $100
Rolled a 8
Rolled a 6
Rolled a 8
*****Rolled the point! You win!***** Richert's
balance: 1000. Playing a new game... Richert
bets $100
Rolled a 3
*****Craps! You loose.*****
Richert's balance: 900. Playing a new game...
Richert bets $100
Rolled a 7
*****Natural! You win!*****
```

Once the simulation is over, your program will print out statistics of the entire simulation. The stats you need to keep track of are:

- Number of games played
- Number of games won
- Number of games lost
- Maximum number of rolls in a single game
- "Natural" roll count
- "Craps" roll count
- Maximum winning streak
- Maximum losing streak
- Maximum balance throughout simulation
- The game number when the Maximum balance was obtained

After the statistics are printed out, the program will prompt the user to run another simulation or not. If the user chooses not to, your program should terminate. If the user wants to run another, then the simulation is rerun and starts by asking the user for their name, balance, and bet. Your program must check to see if the user input is valid (lower-case / upper-case responses are valid).

A sample output of what the statistics should look like starting at the last game in the simulation is:

```
Richert's balance: 100. Playing a new game...
Richert bets $100
Rolled a 9
Rolled a 8
Rolled a 6
Rolled a 7
*****Crap out! You loose.*****
Richert's balance: $0

*****************************
*** SIMULATION STATISTICS ***
****************************
Games played: 230
Games won: 110
Games lost: 120
Maximum Rolls in a single game: 29
Natural Count: 44
Craps Count: 25
Maximum Winning Streak: 9
Maximum Loosing Streak: 7
Maximum balance: 2100 during game 97

Replay? Enter 'y' or 'n': n
```

There are many ways to structure / organize your code and I will provide some requirements for you to follow. Some objects to implement are:

- Lab2.java
    - Contains the main method. All it does is construct a CrapsSimulation object and calls .start() on it.

- CrapsSimulation.java
    - A class representing all the information for the Simulation. This includes:
        - A CrapsGame object
        - A CrapsMetricsMonitor object
        - The user's name
        - The user's balance
        - The user's bet
        - The current win streak
        - The current lose streak

- Public methods that this class must implement are:
  - CrapsSimulation()
    - Constructor that initializes all fields to default values and constructs any objects used (i.e. Scanner, CrapsMetricsMonitor, …)
  - void start()
    - Main loop of a single simulation run. This is where the user inputs their name, balance, and bet, runs the simulation, and continues to do so if the user wants to run it again.

- CrapsGame.java
  - A class representing all the information for a single craps game. This includes:
    - Number of times a roll happened
    - A CrapsMetricsMonitor object
  - Public methods that this class must implement are:
    - CrapsGame(CrapsMetricsMonitor monitor)
      - Constructor that initializes the class fields. A CrapsMetricsMonitor is passed into the constructor since there are specific stats within a single game that must be updated (and the same object should be used for all metrics in the simulation).
    - boolean playGame()
      - Contains the algorithm for an actual game.

- CrapsMetricsMonitor.java
  - A class representing all of the statistics gathered during a single simulation.
  - Public methods:
    - Any methods you need to update / increment / decrement the class fields. These should be called throughout your simulation.
    - void printStatistics()
      - Prints all of the statistics for the simulation (see sample output).
    - void reset()
      - A method to reset the state of the CrapsMetricsMonitor object. This will be called if the user wants to start a new simulation after one was just completed.

Program Submission Expectations:

- Comment block for each java file provide which includes the following:
  - Name(s), IDs for the developer(s)
  - A brief description of what that file contains (A minimum of 2 lines with the exception of the Lab2.java file which can be a single line)
- Uploading format:
  - To make it easier for downloading when we grading, it is expected that all your files are uploaded in a zip file.
- Files to include:
  - All ".java" files to successfully run your program successfully
- Testing Report:
  - A word document that includes screens shots of the console output where you run through your program with at least 1 player playing your game. You will need to ensure that your output shows that you ran into all the validation rules expected as well (for example, you need to put values outs side of the range to show that your exception handling is working).
- Paired Programming Evaluation Form
  - Uploaded to EEE ONLY IF you did paired programming in this lab.