Module 4 – Assignment – HighOrLowGame

HighOrLowGame Specification:
In the High or Low game, the player begins with a score of 1000. The player is prompted for the number of points to risk and a second prompt asks the player to choose either high or low.

The player rolls two dice and the outcome is compared to the player's choice of high or low.
- If the dice total is between 2 and 6 inclusive, then it is considered "low".
- A total between 8 and 12 inclusive is "high".
- A total of 7 is neither high nor low, and the player loses the points at risk.

If the player had called correctly, the points at risk are doubled and added to the total points.

For a wrong call, the player loses the points at risk.

The HighOrLow client code has been provided for you, and must work with your program.

The client code uses two classes that you will need to create:
- A HLPlayer class, which allows you to create a HRPlayer object with two Die member variables that represent the dice.
- A Die class should use a random number generator to determine the outcome in a roll() method.

Application output should look like:

```
You have 1000 points.
How many points do you want to risk? (-1 to quit) 100
Make a call (0 for low, 1 for high): 1
You rolled: 3
You now have 900 points.
How many points do you want to risk? (-1 to quit) 100
Make a call (0 for low, 1 for high): 0
You rolled: 5
You now have 1100 points.
How many points do you want to risk? (-1 to quit) 100
Make a call (0 for low, 1 for high): 1
You rolled: 8
You now have 1300 points.
How many points do you want to risk? (-1 to quit) -1
```

Client Code provided:

```
/*

 * HighOrLowGame.java
 * Module 4 Assignment - ICS4U
 */
 import java.util.Scanner;

 /**
  * A HighOrLow game is played.
  */
 public class HighOrLowGame {

      public static void main(String[] args) {
            final int QUIT = -1;
            final int LOW = 0, HIGH = 1;
            HLPlayer player = new HLPlayer();
            int pointsToRisk, call;
            Scanner input = new Scanner(System.in);

            /* play High or Low game */
            System.out.println("You have " + player.showPoints() + "
points.");
            System.out.print("How many points do you want to risk? (-1 to
quit) ");
            pointsToRisk = input.nextInt();
            while (pointsToRisk != QUIT) {

                  player.riskPoints(pointsToRisk);
                  do {
                        System.out.print("Make a call (0 for low, 1 for
high): ");
                        call = input.nextInt();
                  } while (call != LOW && call != HIGH);
                  player.makeCall(call);
                  player.rollDice();
                  System.out.println("You rolled: " + player.showRoll());
                  System.out.println("You now have " + player.showPoints() +
" points.");
                  System.out.print("How many points do you want to risk? (-1
to quit) ");
                  pointsToRisk = input.nextInt();
            }
      }
}
```

a) Write out your code design for the HLPlayer and Die classes (refer to the RPS2 Case Study for an example of what code design should look like). Refer to the provided client code for hints regarding parameters and function. Copy and paste your Code Design to the Google Doc Submission Document

HLPlayer

variables:

methods:

- constructor -
- makeCall -
- riskPoints -
- rollDice -
- showRoll -
- showPoints -

Die

variables:

methods:

- constructor -
- roll -

b) Create the two classes

c) Test your classes with the provided client code

d) Write down how the application was tested. Add this to the Google Submission Doc.

e) BONUS: In order to achieve a mark of level 4 on this project, you must <u>improve upon the basics</u> of this program in a significant way by correctly implementing come different concepts.

Reminders:

- A mark of level 4 will only be achieved if you use <u>many</u> of the programming concepts from this module and improve upon a basic implementation
- Project must be submitted to the dropbox, and your project folder name must be "Last Name, First Name – Mod 4 Asn – HighOrLowGame".

Module 4 – Assignment – HighOrLowGame Rubric

Categories	**Level 1 (50 – 59%)**	**Level 2 (60 – 69%)**	**Level 3 (70 – 79%)**	**Level 4 (80 – 100%)**
Code Design (weight: 4)	Variable names and method names along with a description of methods and any required parameters are missing crucial elements and do not communicate a clear plan	Variable names and method names along with a description of methods and any required parameters is communicated with limited clarity	Variable names and method names along with a description of methods and any required parameters is communicated with only minor omissions or errors	Variable names and method names along with a description of methods and any required parameters is communicated clearly
Program Testing (weight: 1)	Program testing was missing crucial elements.	Program testing was insufficient to conclude that the program runs properly	Program testing missed covered a considerable number of possible cases and was summarized in a logical way	Program testing was thorough, and summarized in a logical and succinct way
Program Submission (weight: 1)	Program was not submitted properly	Serious or multiple omissions with assignment submission (such as incorrectly placed in dropbox, incorrect naming convention, late)	Small omission with assignment submission (such as incorrectly placed in dropbox, incorrect naming convention, late)	Program was submitted to the dropbox correctly, using proper naming conventions, on time
Program Execution (weight: 2)	Program doesn't run properly, or the output has serious errors.	Program runs properly. Output has errors. The provided client code will not run properly, but student provided client does run properly	Program runs properly. Output is correct with only minor errors. The provided client code will run properly	Program runs properly as is. The output is identical or superior to the sample provided. The provided client code will run properly.
Source code (weight: 4)	Coding conventions are missing or incomplete.	Coding conventions are followed with some attention to detail	Coding conventions are followed with considerable attention to detail	Coding conventions are followed with thorough attention to detail
Programming Concepts (weight: 5) Focus: methods	Few of the programming concepts from the unit are used properly	Some of the programming concepts from the unit are used properly	Most of the programming concepts from the unit are used properly	Many or all of the programming concepts from the unit are used properly to maximize the efficiency of the code