Submit Assignment (https://ccsf.instructure.com/courses/9318/assignments/68464?module_item_id=239936#)

Assignment 4: Conditionals & Loops

Due Oct 17 by 5pm **Points** 10 **Submitting** a file upload **Available** until Oct 19 at 9am

Programming Assignment 4

Game of Hog Application

In this assignment, you will create an algorithm, and then implement it as a program, that will require the use of conditionals and loops.

- **Background Thought (https://ccsf.instructure.com/courses/9318/assignments/68464?module_item_id=239936#Background_Thought)**
- **Instructions/Design (https://ccsf.instructure.com/courses/9318/assignments/68464?module_item_id=239936#InstructionsDesign)**
- **Input (https://ccsf.instructure.com/courses/9318/assignments/68464?module_item_id=239936#Input)**
- **Sample Output (https://ccsf.instructure.com/courses/9318/assignments/68464?module_item_id=239936#Sample_Output)**
- **Submitting Your Work FOR CREDIT (https://ccsf.instructure.com/courses/9318/assignments/68464?module_item_id=239936#Submitting_Your_Work_FOR_CREDIT)**

Purpose
Demonstrate your ability to use Java data and control structures to develop an algorithm using nested conditionals and loops. You will also design an aggregate class. Your program will then implement this data structure and algorithm.

Background Thought

A note on comments: self-documenting code is a code feature to be striven for. The use of identifier names that have meaning within the context of the problems being solved goes a long way in this direction. Code that is not self-documenting for whatever reasons may be made clearer by appropriate comments inserted in the code.

“The most difficult feature of any programming language to master is the comment.” - disgruntled maintenance programmer

“Where the comment and the code disagree, both should be assumed to be in error” – experienced maintenance programmer

Instructions/Design

Step 1)
Using the **Die** class defined in Chapter 4, write a class called **PairOfDice**, composed of two **Die** objects. Include methods to *set* and *get* each of the individual die values, a method to *roll* the two die, and a method that returns the current *sum* of the two die values. The constructor should initialize each die to 1.
Create a driver class called **RollingDice2** to instantiate and use a **PairOfDice** object.

Note 1: Die class does not change**.**

Note 2: Make sure **PairOfDice** is working before you proceed with the remainder of assignment.

Here is the UML:

PairOfDice

Step 2)
Using the **PairOfDice** class, design and implement a program to play a game called ***Hog***. In this game, a computer user (player 1) competes again a human user (player 2). On each turn, the current player rolls a pair of dice and accumulates points. The goal is to reach 50 points before your opponent does.
If, on any turn, the player rolls a 1 on either die, all points accumulated for that round (turn) are forfeited and control of the dice moves to the other player. The human player may voluntarily turn over the dice after each roll. Therefore, the human player must decide to either roll again and risk losing points, or relinquish control of the dice, possibly allowing the other player to win.
Implement the computer player such that it always relinquishes the dice after accumulating 20 or more points in any given round/turn.

I'll get you started by providing an algorithm (this is VERY high-level, you still have PLENTY of designing to do) and some of the variables/data you will need:

Data
char answer
PairOfDice dice
Die die1
Die die2
int computerTotalScore
humanTotalScore
computerRoundTotal //points for one turn
humanRoundTotal //points for one turn
Feel free to use more variables

High-Level Algorithm

```
while (computerTotalScore < 50 and humanTotalScore < 50)
    computerPlayer takes their turn
    if computerTotalScore < 50
        humanPlayer takes their turn
Display winner and final results
```

B) Design and implement the Computer Player (one round/turn)

1) Think about what stops Computer Player from rolling (their turn)

```
computerRoundTotal < 20 and rolls no 1
```

2) Roll the dice and accumulate computerRoundScore

3) What happens when computer's turn is over?

```
if (either die == 1)
     display “busted”
   else
     add computerRoundScore + computerTotalScore
     update computerRoundScore for next round
     display computerRoundScore and computerTotalScore
```

C) Next design and implement the Human Player (one turn)

1) Think about what stops Human Player from rolling (their turn)

```
player answers "yes" to take another turn and and rolls no 1
```

2) What happens when human's turn is over?
Think about what to display and what happens with humanRoundScore and humanTotalScore?

You may use methods to modularize your code but they are not necessary for this assignment. The only classes necessary are: **Die, PairOfDice** and the driver**: Hog**

D) Finally, implement the outer loop that controls the entire game and display who has wona and the final results

Step 3) To compile/run the program, type this (at system prompt):

javac Hog.java
java Hog

Input

There is no input for the computer user (player 1), Player 1 keeps rolling as long as turn total is less than 20 and no 1 has been rolled.

The human user (player 2) is prompted after each turn:

Take another turn (y/n)?

So their answer (y/n) must be read.

Warning:
This algorithm/program is too complex to do in one chunk.
Implement computer player 1's turn first.
Then implement human player 2's turn.
Finally, implement the outer loop for the entire game.

Sample Output

Your output need not look exactly link mine but you must use 3 classes to receive credit. To see your output, I will attempt to download and then unzip your Assignment4.zip file from the file you submit onto Canvas.

I will then run your program on my local machine using the command:

```
$ java Hog
****************************************
Current Status:
   Computer: 0
   You: 0
Die 1: 2   Die 2: 1
Busted!!!
****************************************
Current Status:
   Computer: 0
   You: 0
Die 1: 4   Die 2: 6
```

```
Current Round: 10
Potential Total: 10
Take another turn (y/n)? y
Die 1: 5   Die 2: 5
Current Round: 20
Potential Total: 20
Take another turn (y/n)? y
Die 1: 5   Die 2: 5
Current Round: 30
Potential Total: 30
Take another turn (y/n)? y
Die 1: 2   Die 2: 2
Current Round: 34
Potential Total: 34
Take another turn (y/n)? y
Die 1: 4   Die 2: 1
Busted!!!
****************************************
Current Status:
   Computer: 0
   You: 0
Die 1: 1   Die 2: 5
Busted!!!
****************************************
Current Status:
   Computer: 0
   You: 0
Die 1: 3   Die 2: 4
Current Round: 7
Potential Total: 7
Take another turn (y/n)? y
Die 1: 4   Die 2: 4
Current Round: 15
Potential Total: 15
Take another turn (y/n)? y
Die 1: 1   Die 2: 2
Busted!!!
****************************************
Current Status:
   Computer: 0
   You: 0
Die 1: 5   Die 2: 5
Current Round: 10
Potential Total: 10
Die 1: 5   Die 2: 5
Current Round: 20
Potential Total: 20
****************************************
Current Status:
   Computer: 20
   You: 0
Die 1: 3   Die 2: 2
Current Round: 5
Potential Total: 5
Take another turn (y/n)? y
```

```
Die 1: 2   Die 2: 2
Current Round: 9
Potential Total: 9
Take another turn (y/n)? y
Die 1: 5   Die 2: 3
Current Round: 17
Potential Total: 17
Take another turn (y/n)? y
Die 1: 5   Die 2: 4
Current Round: 26
Potential Total: 26
Take another turn (y/n)? y
Die 1: 4   Die 2: 3
Current Round: 33
Potential Total: 33
Take another turn (y/n)? y
Die 1: 5   Die 2: 3
Current Round: 41
Potential Total: 41
Take another turn (y/n)? y
Die 1: 5   Die 2: 1
Busted!!!
****************************************
Current Status:
   Computer: 20
   You: 0
Die 1: 3   Die 2: 5
Current Round: 8
Potential Total: 28
Die 1: 3   Die 2: 6
Current Round: 17
Potential Total: 37
Die 1: 5   Die 2: 6
Current Round: 28
Potential Total: 48
****************************************
Current Status:
   Computer: 48
   You: 0
Die 1: 3   Die 2: 2
Current Round: 5
Potential Total: 5
Take another turn (y/n)? y
Die 1: 3   Die 2: 6
Current Round: 14
Potential Total: 14
Take another turn (y/n)? y
Die 1: 1   Die 2: 4
Busted!!!
****************************************
Current Status:
   Computer: 48
   You: 0
Die 1: 3   Die 2: 5
Current Round: 8
```

```
Potential Total: 56


The computer has won!


Final Results:
   Computer: 56
   You: 0
```

Note:
To receive credit for this assignment, I must be able to download your Assignment4.zip file **from Canvas and run it on my machine**
Warning: No credit given for code that does NOT compile

Submitting Your Work FOR CREDIT

Once you have the code working on your own machine, you must submit your assignment on Canvas (to avoid late penalty submission must be completed no later than 5pm on due date).
Check you have provided algorithm (in comment at top of driver file) and used javadoc style comments. See link on homepage "How Programming Assignments Are Graded".

a) ***zip*** your 3 source code and 3 byte code files into an archive named **Assignment4.zip**

- **Hog.java, Die.java,** and **PairOfDice.java**
- **Hog.class, Die.class,** and **PairOfDice.class**

b) At bottom of this page browse for and then upload **Assignment4.zip** to submit.
NOTE: All group members MUST submit the same zipped for to receive a grade.

YOU MUST USE 3 classes - Hog, Die and PairOfDice - to receive credit for this assignment.

Note: If you do not submit assignment successfully using **Canvas**, I cannot give you a grade.
To avoid late penalty, submission must be completed by 5pm on due date. GIVE YOURSELF TIME TO SUBMIT.