Project 3 (50 points)
Due Friday, September 23 by midnight

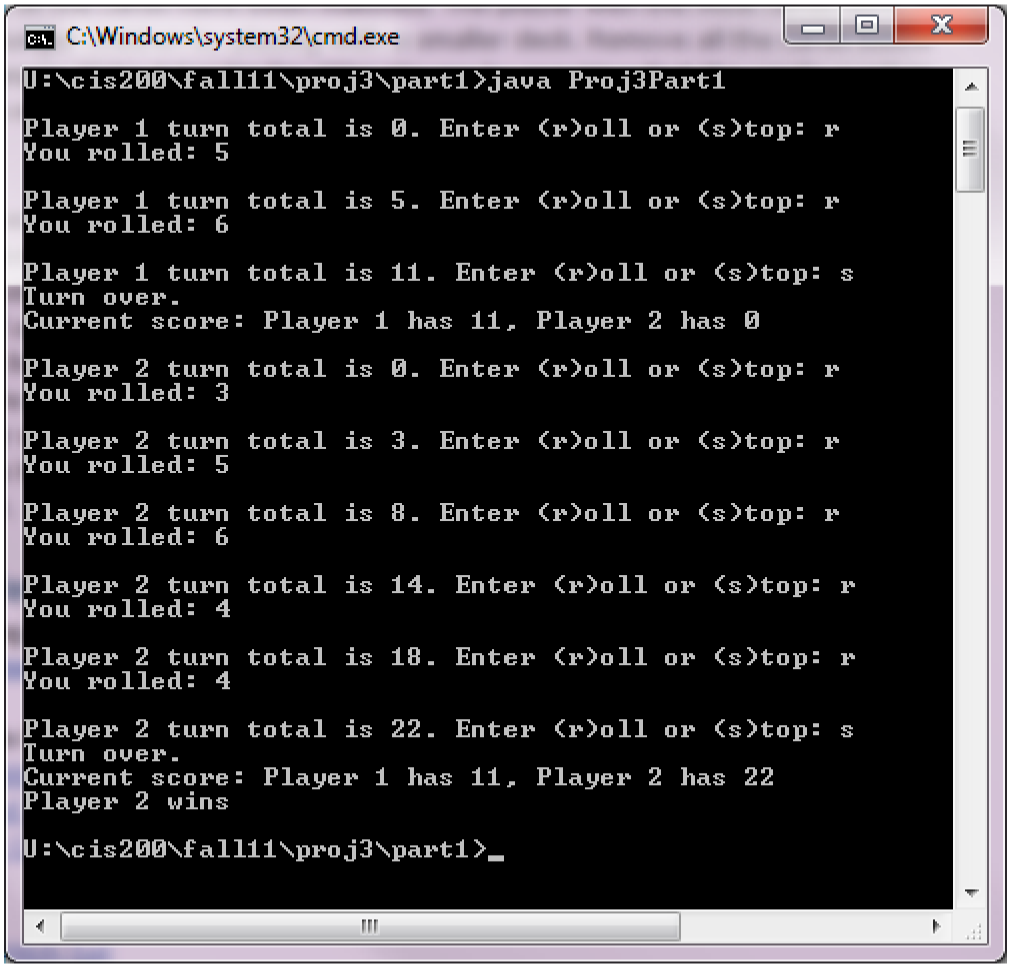
Background
Pig is a dice game between two players. Here are the rules:

1) Players alternate turns, and the first to reach or exceed a set number of points (say, 20) wins
2) On a player's turn, he can choose whether to *keep rolling* or to *stop*. If they want to keep rolling, they roll a die (6 sides, 1-6). If that value is not 1 (2-6), the value is added to a "turn total". The player can then choose to roll again, and add that roll to the turn total as well.
3) If the player rolls a 1, he loses everything from his turn total (but not his overall score), and it becomes the other player's turn.
4) If the player decides to stop (BEFORE rolling a 1), his turn total is added to his overall score, and it becomes the other player's turn.

For example, suppose John and Bob are playing. John goes first and rolls 4, 6, 2, and then decides to stop. John's overall score is now 12. It is now Bob's turn, and Bob rolls 6, 3, 4, 1. When he rolls that last 1, he loses his turn total (which was 6+3+4 = 13). Bob's overall score is still 0, and it is now John's turn again.

Assignment Description:
This project is made up of three pieces. First, you will implement a two-player game of Pig that goes to 20 points. Second, you will write another game of Pig where the user plays the computer. You will have to come up with the logic of when the computer will stop rolling on each turn (obviously you don't want the computer to keep rolling until they get a 1 each time!). Finally, you will run a simulation that shows how well your computer player works. The three parts are described in more detail below.

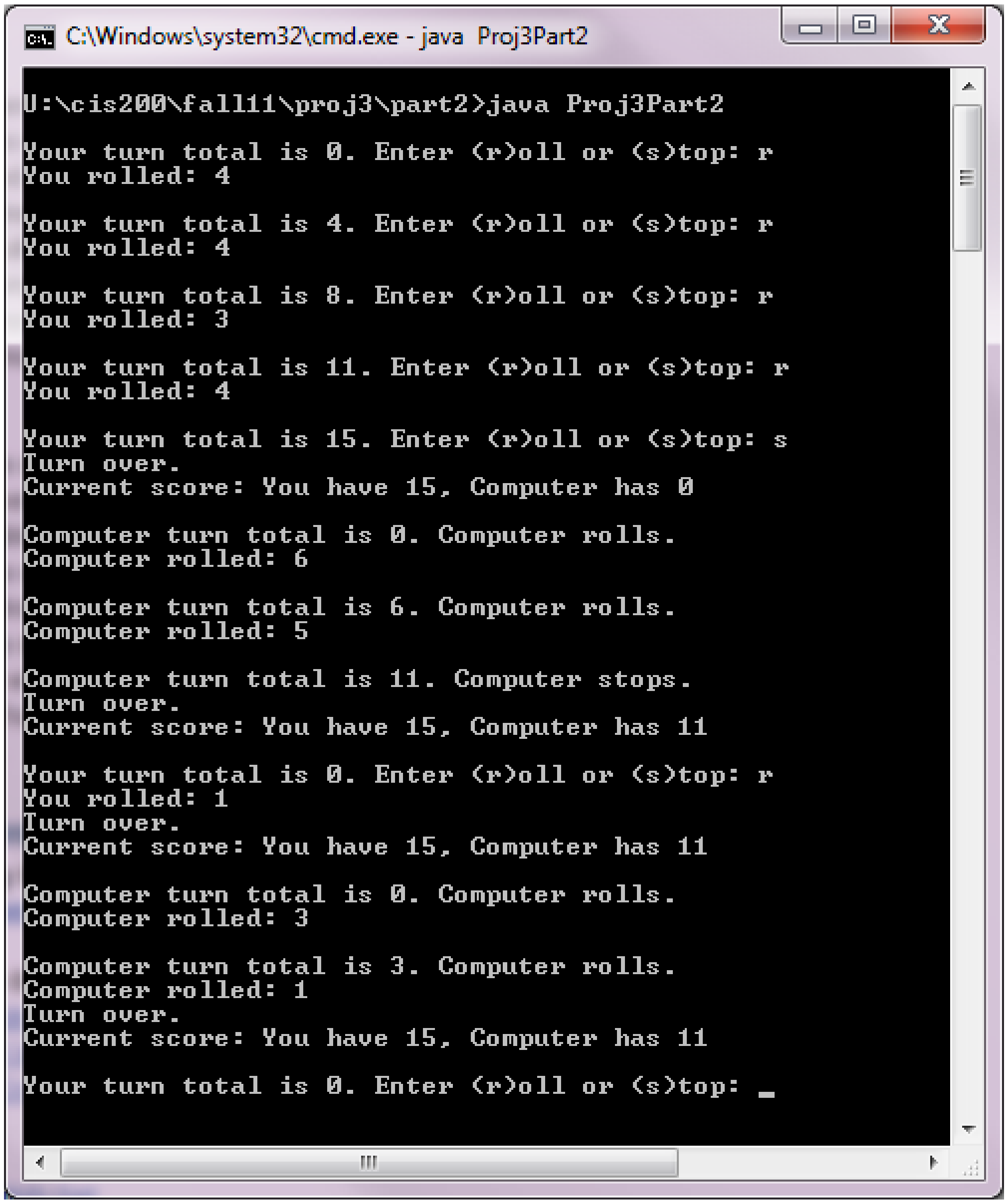
Part 1
Write the class `Proj3Part1`, which contains a single `main` method. In this part, you should implement a two-player game of Pig that goes to 20 points. You should alternate between players as described in the background above. Once a player has at least 20 points, you should stop the game and print who won. Here is an example run of Part 1:



```
U:\cis200\fall11\proj3\part1>java Proj3Part1

Player 1 turn total is 0. Enter (r)oll or (s)top: r
You rolled: 5

Player 1 turn total is 5. Enter (r)oll or (s)top: r
You rolled: 6

Player 1 turn total is 11. Enter (r)oll or (s)top: s
Turn over.
Current score: Player 1 has 11, Player 2 has 0

Player 2 turn total is 0. Enter (r)oll or (s)top: r
You rolled: 3

Player 2 turn total is 3. Enter (r)oll or (s)top: r
You rolled: 5

Player 2 turn total is 8. Enter (r)oll or (s)top: r
You rolled: 6

Player 2 turn total is 14. Enter (r)oll or (s)top: r
You rolled: 4

Player 2 turn total is 18. Enter (r)oll or (s)top: r
You rolled: 4

Player 2 turn total is 22. Enter (r)oll or (s)top: s
Turn over.
Current score: Player 1 has 11, Player 2 has 22
Player 2 wins

U:\cis200\fall11\proj3\part1>_
```

Part 2

Write the class `Proj3Part2`, which contains a single `main` method. You should start by copying your code from Part 1 and pasting it again here. Then, you should make Player 1 be the user (so you will still ask if they want to roll or stop each time) and Player 2 be the computer. **When it is the computer's turn, you will need to come up with a strategy for when you want them to stop rolling.** You should still print the values of each roll the computer makes, and when they decide to stop. Here is an example of PART of a run of Part 2:



```
U:\cis200\fall11\proj3\part2>java Proj3Part2

Your turn total is 0. Enter (r)oll or (s)top: r
You rolled: 4

Your turn total is 4. Enter (r)oll or (s)top: r
You rolled: 4

Your turn total is 8. Enter (r)oll or (s)top: r
You rolled: 3

Your turn total is 11. Enter (r)oll or (s)top: r
You rolled: 4

Your turn total is 15. Enter (r)oll or (s)top: s
Turn over.
Current score: You have 15, Computer has 0

Computer turn total is 0. Computer rolls.
Computer rolled: 6

Computer turn total is 6. Computer rolls.
Computer rolled: 5

Computer turn total is 11. Computer stops.
Turn over.
Current score: You have 15, Computer has 11

Your turn total is 0. Enter (r)oll or (s)top: r
You rolled: 1
Turn over.
Current score: You have 15, Computer has 11

Computer turn total is 0. Computer rolls.
Computer rolled: 3

Computer turn total is 3. Computer rolls.
Computer rolled: 1
Turn over.
Current score: You have 15, Computer has 11

Your turn total is 0. Enter (r)oll or (s)top: _
```

Again, this is an incomplete run – the game is not over. When the game is over (someone has 20 or more points), you should stop and print the winner (either “You won” or “Computer wins”). Notice that the only user input if for the user’s turn (“Your” turn) – the computer options are just being printed according to whatever strategy you pick.

Part 3
In the final part, you will try to see how good your strategy is for the computer player. In this part, you will write the class `Proj3Part3`, which contains a single `main` method. This part will get rid of the user player, and will no longer alternate turns. Instead, it will try to determine how many turns the computer needs on average to get to 20. Here is what you should do:

```
Repeat 10 times (run 10 simulations)
        While the computer hasn’t reached 20 points
```

Let the computer make a "turn" according to the rules you made in Part 2
Print the rolls as you go, as in Part 2.
If the computer rolls a 1, he loses his turn total and starts his next turn

Print how many turns the computer needed to reach 20 points
Print how the average of how many turns the computer needed to reach 20 points (averaged over the 10 simulations).

If your computer strategy is based on the other player, then make your simulation alternate between two computer players (who both use your strategy). Perform calculations on just one of those players, and pretend the other one is the "user".

Your output for this portion can be formatted however you like, **as long as you are printing the rolls, the turn totals and current computer score, the total number of turns in each simulation, and the average number of turns across the 10 simulations.**

In your documentation for Part 3 (described below), you must briefly describe your strategy for the computer, and why you believe it is effective.

Requirements

The three parts of your program must compile (by command-line) with the statements:

```
javac Proj3Part1.java
javac Proj3Part2.java
javac Proj3Part3.java
```

The three parts should then run with the commands:

```
java Proj3Part1
java Proj3Part2
java Proj3Part3
```

Documentation:

At the top of every class, add the following comment block:

```
/**
 * (description of the class)
 *
 * @author (your name)
 * @version (which number project this is)
 */
```

In your `Proj3Part3` comment block, you must describe your strategy for the computer, and why you believe it is effective. In addition to the comment block at the top of each file, you should include comments within each file that describe what your code does. You don't have to comment every line, but you should include enough description that it is easy for an outside reader to tell what you are doing.

Submission:
To submit your project, first create a folder called `proj3`, and move your `Proj3Part1.java`, `Proj3Part2.java`, and `Proj3Part3.java` files into that folder. Then, right-click on that folder and select "`Send To->Compressed (zipped) folder`". This will create the file `proj3.zip`.

Go to "`Files and Content->Modules->File Dropbox`" on K-State Online. Select your lab time and upload the `proj3.zip` file. **Put your name and Project 3 in the description box.**

Grading:
Programs that do not compile will receive a grade of 0. Programs that do compile will be graded according to the following point breakdown:

Requirement	**Points**
Part 1 (two-player game of Pig)	**15**
Part 2 (user vs. computer game of Pig where you write a strategy for the computer)	**10**
Part 3 (simulation of computer's strategy)	**15**
Output exactly matches examples	**5**
Documentation (including description of computer's strategy)/naming/submission	**5**
Total	**50**

You will receive full credit for Part 2 if you have a working computer vs. user game. However, your points for Part 3 will depend on how good your computer strategy is. (You don't have to have an "optimal" strategy, but you should at least put some thought into it and be able to defend your choices.)