CIS 200: Project 3 (40 points)
Due Friday, February 12th by midnight

Reminder: Programs submitted after the due date/time will be penalized 10% for each day the project is late (not accepted after 3 days, i.e. Midnight, Mon, Feb 15).

** For all remaining projects, you may use either ***Console-based I/O*** or **GUI-based I/O**. (Sample screenshot of executed program will always be shown as Console-based I/O.) **

Reminder: As the projects become more challenging, it is a good idea to remind all students about the *Academic Integrity Policy* for this course. **ALL projects are intended to be done *individually*. Do NOT share your code with anyone.** If it is not your work, don't submit it as yours! Refer to the policy within the course syllabus which states, ***"If two (or more) students are involved in ANY violation of this policy,* at a minimum, *ALL* students involved receive a *zero*** for the assignment and the offense is officially reported to the *KSU Honor Council*. The second offense results in a failing grade for the course and possible suspension from the university (this decision is made by the *K-State Honor Council*)."

This program requires a more complex algorithm than previous projects. I strongly recommend that you do some '*pre-planning*' before you code (i.e. I-P-O, algorithm, etc.) The difficulty lies in the logic and algorithm more than the coding.

Assignment Description:
This assignment has multiple parts (i.e. three separate *.java* files). If submitting all three parts, the three files must be submitted TOGETHER in a ***single*** zip file. **If any of the 3 parts are not completed by the due date and are submitted late, *then the entire assignment is penalized as late***. Partial credit is awarded for completion of each part.

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, they 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 (i.e. 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, they lose everything from their turn total (but not the overall score), and it becomes the other player's turn.

4) If the player decides to stop (BEFORE rolling a 1), their turn total is added to their 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** *parts* (i.e. programs or *.java* files). **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 against the computer***. You will have to come up with the strategy 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 create another program that ***runs a simulation*** that shows how well your computer player strategy works. The three parts are described in more detail below.

Part 1

Write the class **Proj3Part1** which contains a main method. In this part, you will implement a ***two-player*** game of Pig that goes to 20 points. You should alternate between players as described in the *background* description on the previous page. Once a player has at least 20 points, you will stop the game and display who won. The winning number of points required (20) must be declared as a *constant*.

Here is an example run of Part 1:

```
Player 1 turn total is 0. Enter (r)oll or (s)top: r
You rolled: 3

Player 1 turn total is 3. Enter (r)oll or (s)top: r
You rolled: 1
Turn over.
Current score: Player 1 has 0, Player 2 has 0

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

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

Player 2 turn total is 8. Enter (r)oll or (s)top: s
Turn over.
Current score: Player 1 has 0, Player 2 has 8

Player 1 turn total is 0. Enter (r)oll or (s)top: r
You rolled: 3

Player 1 turn total is 3. Enter (r)oll or (s)top: r
You rolled: 3

Player 1 turn total is 6. Enter (r)oll or (s)top: r
You rolled: 2

Player 1 turn total is 8. Enter (r)oll or (s)top: r
You rolled: 2

Player 1 turn total is 10. Enter (r)oll or (s)top: r
You rolled: 2

Player 1 turn total is 12. Enter (r)oll or (s)top: r
You rolled: 4

Player 1 turn total is 16. Enter (r)oll or (s)top: r
You rolled: 6

Player 1 turn total is 22. Enter (r)oll or (s)top: s
Turn over.
Current score: Player 1 has 22, Player 2 has 8
Player 1 wins
```

Part 2

Write the class **Proj3Part2**, which contains a main method. Start by copying your code from Part 1 and pasting it for Part 2. Then, you will make ***Player 1*** **be the** ***user*** (so you will still ask if they want to roll or stop each time) and ***Player 2*** **will be the** ***computer***. When it is the computer's turn, you will need to come up with a strategy for when you want the computer to stop rolling based on 1) How close the computer is to winning, 2) How close the opponent is to winning, and the 3) Current total for that turn. You should still display the values of each roll the computer makes and when they decide to stop. An example run of Part 2 is shown on the next page.

Here is an example run of Part 2:

```
Your turn total is 0. Enter (r)oll or (s)top: r
You rolled: 6

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

Your turn total is 8. Enter (r)oll or (s)top: s
Turn over.
Current score: You have 8, Computer has 0

Computer turn total is 0. Computer rolls.
Computer rolled: 5

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

Computer turn total is 11. Computer stops.
Turn over.
Current score: You have 8, Computer has 11

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

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

Your turn total is 7. Enter (r)oll or (s)top: s
Turn over.
Current score: You have 15, Computer has 11

Computer turn total is 0. Computer rolls.
Computer rolled: 5

Computer turn total is 5. Computer rolls.
Computer rolled: 2

Computer turn total is 7. Computer rolls.
Computer rolled: 6

Computer turn total is 13. Computer stops.
Turn over.
Current score: You have 15, Computer has 24
Computer wins
```

When the game is over (someone has 20 or more points), you should stop and display the winner (either "*You win!*" or "*Computer wins*"). Notice that the only user input is for the user's turn (i.e. "*Your*" turn). The computer rolls are just being displayed according to whatever strategy you develop.

Part 3

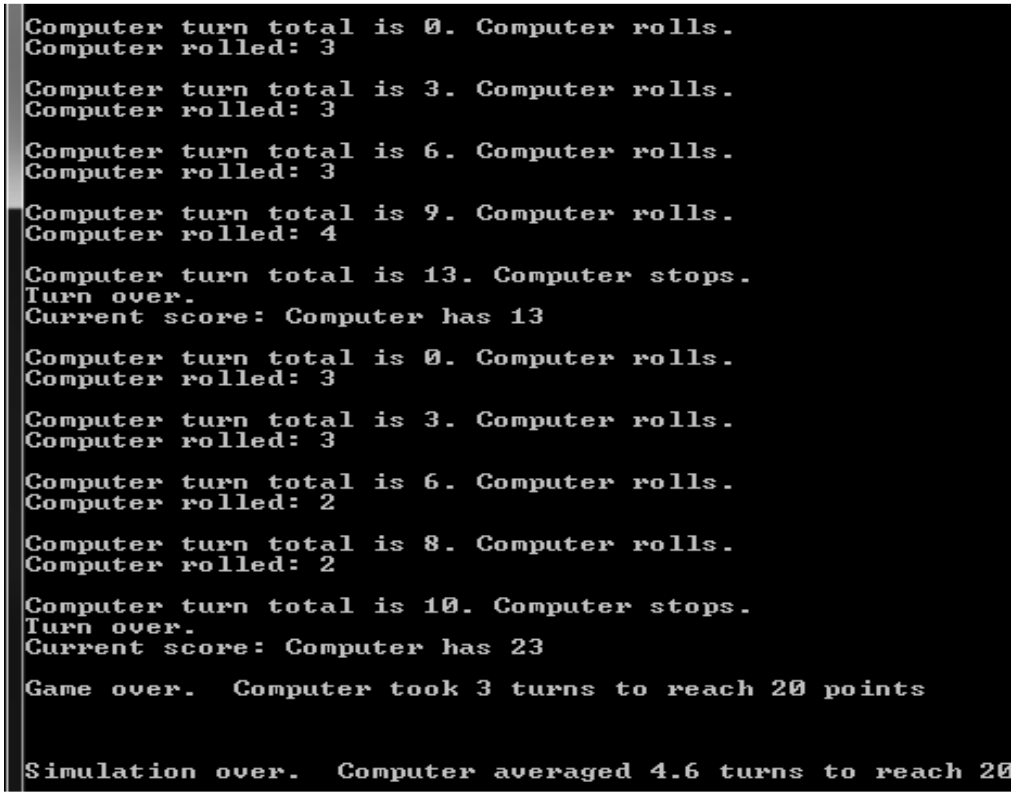
In the final part, you will **test how good your strategy is for the *computer player***. In this part, you will write the class **Proj3Part3**, which contains a main method. This part will get rid of the *user player.* Since your computer strategy should be partially based on the other player, make 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*".

For example, where *computer* below is the computer player in which you are preforming calculations:

Repeat 10 times (run 10 simulations)
    While the computer hasn't reached 20 points
        Let the computer take a "turn" according to the rules you made in Part 2
        Display the rolls as you go, as in Part 2.
        If the computer rolls a 1, the computer loses his turn total and starts his next turn
    Display how many turns the computer needed to reach 20 points
    Display the average of how many turns the computer needed to reach 20 points (averaged over the 10 simulations).

Output for this portion can be formatted however you like, **as long as you are displaying the rolls of the calculating computer, turn totals and current computer score, the total number of turns in each simulation, and the average number of turns across the 10 simulations.** An example run of Part 3 is shown on the next page.

Shown below is a **partial** screenshot of Part 3, showing **ONLY the last simulation** (#10) and the results.



Important: In addition to the regular documentation (described below), **for Part 3 you must briefly describe your strategy for the computer and why you believe it is an effective strategy**.

****Reminder: <u>Partial credit if you can get Part I, II and/or III to compile and work properly</u> …**

- This assignment has multiple parts (i.e. three .java files). The three files must be submitted TOGETHER in a ***single*** zip file. If any part is not completed by the due date and submitted late, ***then the entire assignment is penalized as late***.

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 ***EACH*** class, add the following comment block, filling in the needed information:

```
/**
  * <Full Filename>
 * <Student Name / Section Day/Time>
 *
 * <description of the project – i.e. What does the program do?>
*/
```

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 – *read these instructions carefully or you may lose points*

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***.

Log-in to Canvas and upload your ***`proj2.zip`*** file. **Only a .zip file will be accepted for this assignment in Canvas. Put your full name and Project 3 in the comments box.**

Important: It is the ***student's responsibility*** to verify that the ***correct*** file is ***properly*** submitted. If you don't properly submit the *correct* file, it will not be accepted after the 3-day late period. No exceptions.

Grading:

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 strategy.)

Again, you may submit parts of the project late. However, doing so will cause the project *as a whole* to receive a late penalty. **Please avoid submitting then changing then resubmitting**. This makes grading much more complicated. **Please wait until you are fully satisfied with your solution before submitting**.

Programs that do not compile will receive a grade of 0, so make sure you submit the *correct* file that *properly compiles*. Programs that *do* compile will be graded according to the following rubric:

	Points
Part 1 (*two-player* game of Pig) (13 pts.)	**-13-**
Correctly get all needed input from the user	**2**
Correctly tracks each users current *turn* score and *overall* score	**3**
Correctly follows the rules of the game and displays correct output	**6**
Constant correctly declared / Documentation correct	**2**
Part 2 (*user vs. computer* game of Pig; you write a strategy for the computer) (14 pts.)	**-14-**
Correctly get all needed input from the user	**2**
Correctly tracks user and computer's current *turn* score and *overall* score	**3**
Correctly follows the rules of the game and displays correct output	**5**
Code: Strategy when computer stops rolling based on 3 factors listed on bottom of p.2 of assignment sheet	**4**
Part 3 (*simulation* of computer's strategy) (11 pts.)	**-11-**
Required Output as described (displaying rolls, turn totals and current computer score, total number of turns in each simulation, and the average number of turns across the 10 simulations.)	**5**
Constant correctly declared	**1**
Documentation includes description of computer's strategy and why it is an effective strategy	**5**
Project Submission (zip file with .java file, submitted to correct folder with *Name* and *Project 3* in the description box)	**2**
Minus Late Penalty (10% per day)	