**Introduction**

This assignment lets you practice with writing a larger program that manipulates an ArrayList in various ways. You will write a set of Java classes to simulate a line of contestants, sort of like a spelling bee in school, though no one is ever eliminated from the game. While you aren’t actually implementing the game, you will need to implement different algorithms for moving contestants and then evaluate how fair the algorithms are and also gather some information on how long the process took. Throughout this process, you should only have one ArrayList. The point is not to be efficient, but to get a baseline set of timings. You can discuss efficiency when you reflect on the assignment.

After your program is working for each step of the process and, you should commit your changes to the repository and push the results to GitHub. This allows you to go back to a previous working version in case you decide to throw out new code and try a different approach. Doing this is a requirement and you will receive few if any points if you do not show the steps in GitHub.

**Preparation**

See the document [Getting Started with Version Control](https://canvas.park.edu/courses/74667/pages/getting-started-with-version-control) to set up your environment for coding and download the starter files.

Review the syllabus for examples of what is and is not allowed when writing these assignments.

**Directions**

Write a set of Java classes to simulate a line of Contestants.

1. You are given the class Person, and you will write a Contestant class, where Contestant extends Person. Start simple  to make sure your code compiles and runs. As you progress through the lab, you will need to add attributes and methods to the Contestant class to complete the assignment.
2. Create *n* Contestants, each with a different name, and add them all to an ArrayList of Contestants. *n* can be any value between 5 and 1,000,000, though I suggest you start small with a value like 10. Later, you will increase the size of *n*, so plan for this now.
3. Figure out how to flip a coin. You want to generate a boolean that is True about 50% of the time and False about 50% of the time, over a large number of coin flips. See java.util.Random or java.util.concurrent.ThreadlocalRandom if you need a place to get started.
4. In a real contest, each Contestant in turn is asked a question, and moves to a new spot in the line depending on whether the answer is correct or not. One simple algorithm is to move to the front of the line on a correct answer, and the back of the line on an incorrect answer. So suppose there is a line of Contestants (a,b,c,d,e) where a and d get an answer wrong, and b, c and e get their answers correct.

|  |  |
| --- | --- |
| a b c d e | Initial position |
| b c d e a | a answers incorrectly and moves to the end |
| b c d e a | b answers correctly and moves to the front |
| c b d e a | c answers correctly and moves to the front |
| c b e a d | d answers incorrectly and moves to the end |
| e c b a d | e answers correctly and moves to the front |

1. This represents one round of the competition. At the end of the round, e is at position 0, c is at position 1, ... and d is at position 4. Implement one round using this algorithm. Use your coin flip to determine whether an answer was correct or not. Write a paragraph explaining how you tested your code to know it works correctly. Include this in your final write-up.
2. Modify your program to run a series of rounds. Each round should process the Contestants starting from position 0. But first, modify the Contestant class so that it can record a Contestant's position in the list and calculate the Contestant's average position. You will have to figure out how to store this data in each Contestant and what calculation to make, and what methods to provide. In the driver program, you should record each Contestant's position at the end of each round. Run your program with a small number of Contestants (8)  and a medium number of rounds (12). Use the average position of each contestant to show that this algorithm isn't biased against certain Contestants, that is, no Contestant gets stuck at one end of the line or the other for most of the game. Write a paragraph or two with an explanation of how you tested this and show your data.
3. Add code to calculate the time it takes to run your simulation in step 5. Do not include the time to calculate the fairness aspect. Change the number of Contestants to 43 and then run your program 4 more times for 10, 100, 1000, and 10000 rounds. Record the number of rounds and run times in a table.
4. Increase the number of Contestants to 45,000, then repeat step 6. Add this data to your table and include it in your report.
5. The folks running the game don’t like the optics of moving people to the front or end of the line each time. Think about a different algorithm for moving people around in the list. You can't decrease the size of the list and the list will get rearranged in some way during each round. Write a paragraph or two on what would have to change in your program to implement the new algorithm, still using an ArrayList. Be as specific as you can with regard to your code. Consider whether it would be easy or hard to implement and estimate whether it will take more or less time to run than the current implementation. Be sure to explain your reasons.
6. Answer the following questions in your Word document:
   * In a sentence or two, what did you learn?
   * In a sentence or two, what did you like about this project?
   * In a sentence or two, what did you find confusing or would like to see done differently regarding this project?
   * In a sentence or two, if you had another hour or two, what would you like to add to the project or how would you do things differently?

**Notes**

You need to specify a message every time you commit your code to a repository. Like comments in the program, these should be meaningful. Saying "Added files" isn't going to tell you anything when you look back. Saying something like "Step 3 completed" is better, but after a week, you won't remember what step 3 was. A better comment would be "Step 3 completed - method to flip a coin".

In writing your classes, remember that instance variables should be private to the class.

Your code should also contain meaningful comments. At a minimum, each file requires a descriptive header comment with your name, date, and the purpose of the code. Methods that do more than just access/modify an instance variable should have a header comment describing the input parameters, the return value (if any), and what the method does.

When you comment your code, aim for commenting blocks of code, not individual lines. Don't explain the obvious. This is not a textbook. There is no need to write

System.out.println("The count is " + count);  // Print the count

**Submit**

Be sure to put your Word document in your repository and do one final commit. Generate a .zip file from the repository and submit that file to Canvas. Double check that what you submit to Canvas has all your .java files.

**Due Dates**

* by 11:59 p.m., Sunday, CT