Lab 02 Specification – Implementing Queue ADT and its applications.
Due (via your git repo) no later than 8 a.m., Thursday, 14th Feb 2019.
50 points
[An individual lab assignment]

Lab Goals

- Getting familiar with Queue ADT's
- Propose solutions to highly regarded algorithmic problems with the use of ADT's.
- Implement Java programming solutions by translating Algorithms into Code.

Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at: `https://www.cs.allegheny.edu/sites/amohan/resources/suggestions.pdf`
- Often times, the lab specification is purposefully developed to include open ended tasks and not exposing too low level details. The intention is to trigger you towards further refining your thinking skills. Thinking is more important to solve any algorithmic problem provided. Thinking through, may lead you to ask further clarifying questions, which is what my expectation is. So, there will be one or more open ended parts added to the lab. Attending labs regularly and steering discussions with the Professor, TA's and your classmates is going to be the success mantra for doing a great job in the labs. Waiting till the last minute, will restrict a student from both learning the content discussed and to have a negative effect to the effectiveness of the lab work produced.

Learning Assignment

If you have not done so already, please read all of the relevant "GitHub Guides", available at https://guides.github.com/, that explain how to use many of the features that GitHub provides. In particular, please make sure that you have read guides such as "Mastering Markdown" and "Documenting Your Projects on GitHub"; each of them will help you to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, you should also read

- Sedgwick chapter 01, section (1.1 - 1.3)

Assignment Details

Firstly getting familiar with Queue ADT. Practice the source code related to Queue ADT discussed in class. Applying a solution to an algorithmic problem that is a variation of another classic problem. Finally using Java, implement a program that handles other requirements to further solidify the core solution to the problem.

In our Lab 01, there were tasks added so that you could get refamiliarized with compiling and executing programs in Java environment.

So how to compile the program

At this point we should all be familiar with the compilation of Java programs using **javac**. Since, there is a particular directory structure associated with the lab and to make the compilation process simple, there is a compile bash script shipped with the lab repository. In order to compile your programs, make sure to place all your Java files under the src directory, Then, open the terminal and navigate to the lab repository and type in:
”./compile.sh”
This should compile all your files and automatically place their class files in the classes sub directory located in the lab repository.

So how to execute the program

At this point we should all be familiar with executing Java programs using **java**. Since, there is a particular directory structure associated with the lab and to make the execution process simple, there is a run bash script shipped with the lab repository. In order to execute your programs, make sure to open the terminal and navigate to the lab repository and type in:
”./run.sh TestQueue”

The Hot Potato Childrens Game Simulator

Problem Definition: Children line up in a circle and pass an item from neighbour to neighbour as fast as they can. At a certain point in the game, the action is stopped and the child who has the item (the potato) is removed from the circle. Play continues until only one child is left. How do you find the winning position for a child, if there are N number of children’s?

Approach: We discussed this in detail during our lecture through the classic Josephus problem. You should take the similar approach for implementing this simulator.

Requirement:

1. It is required to do the implementation using Java programming language,
2. Create a separate java file called HotPotato.java and provide the implementation of your solution to the algorithmic problem given above. We had discussed some source code for this part during our lectures, so it is acceptable to reuse the code from lecture discussions. The copy of the code discussed during lecture including the Queue ADT’s are included in the repository. These code files can be accessed from the src sub-directory inside the lab repository.
3. It is required to use the custom Queue ADT that we had discussed during our lecture session. You can either reuse the code from class discussion or recreate your own Queue ADT from scratch. But it is not acceptable to use the Java builtin Queue ADT. There will be points deducted if the builtin Queue ADT is used in a student submission. The reason for asking not to use Queue builtin is to help further strengthen your understanding on Queue ADTs. The idea is that if a student is able to either build the Queue ADT from scratch or resuse the code from class discussions, it will give an opportunity to familiarize with Queue and its operations. This is part of the learning objective for this lab.
4. Input: The input for the program is a text file that has the list of children names. A sample text file is provided for you to run your program in the data sub directory.
5. Output: The output of the program is the name of the child who is the winner. Print out the winner name in console.
6. In your program’s main method, it is required to read the file input and store the file contents (child names) into an array. It is worth to note that you may have done similar process in Lab 01. So it is acceptable to reuse your code from Lab 01 as appropriate.
7. Call the playGame() method by passing the array from the previous step as an argument.

8. In order to setup the calling from previous step, create a method called playGame() that takes an array and the number of steps as input, and should include the implementation details of the HotPotato game. All the implementation details of building the game should be based on the Josephus problem. So a key aspect of your implementation, is your understanding of Josephus problem and the algorithm discussed in class to solve the problem using Queue. Finally, the method should print out the final output of the program, which is the name of the child who is a winner.
9. The child removal process in the game is done through a configurable variable (number of steps) the potato is passed before getting to the child who will be removed. The number of steps should be user driven.

   Additionally, inside the playGame() method, prompt the user to provide the number of steps before initiating the game.
   ”Hey, provide the number of steps to be used in the game:”
10. It is worth to note that in our original Josephus algorithm discussed in class there were two inputs namely m and n. In the current context, ”n” is the number of children and ”m” is the number of steps.

Submission Details

For this assignment, please submit the following to your GiHub repository by using the link shared to you by the course instructor:

1. Commented source code from the “HotPotato” program.
2. A readme file explaining how to compile and run your program. In addition, include a detailed self reflection of the lab exercise in the readme file. The self reflection may include answers to questions such as:
   - ”What challenges did you face in this lab?”
   - ”What did you like in this lab?”
   - ”What did you not like in this lab?”
3. It is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllabus. Make sure to add the statement ”This work is mine, unless otherwise cited.” in all your deliverables such as source code and other textual documents.

Grading Rubric

1. The HotPotato game simulator is worth 50 points. It is expected that the submission has the complete solution to the problem above. The details regarding point deductions will be provided through the grading report file.
2. There will be full points awarded for the lab, if all the requirements in the lab specification are correctly implemented. Partial credits will be awarded if deemed appropriate.
3. Failure to upload the lab assignment code to your git repo, will lead you to receive ”0” points given for the lab submission. In this case, there is no solid base to grade the work.
4. There will be no partial credit awarded if your code doesn’t compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student’s responsibility to let the professor know about it after the final submission in github. In this way, an updated version of student’s submission will be used for grading. If the student did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the student had not submitted any code, then in this case, there is 0 points to the student automatically for the lab work.

5. If you needed any clarification on your lab grade, talk to the instructor. The lab grade may be changed if deemed appropriate.