Homework One for CSE 373 Spring 2015

Due 11:59 PM Wednesday, April 8

Introduction

The purpose of this assignment is to get you programming as soon as possible and to try out the queue structure. The application is a set of oracles (oracle_0, oracle_1, …, oracle_{n-1}) whose job it is to answer questions. Each oracle always answers in the same way no matter the question (answer[0], answer[1], …, answer[n-1]). The questions and answers are both read from files. Each oracle has a queue for questions, creating n queues total. The questions are then randomly assigned to an oracle and placed in that oracles queue. After all questions are assigned, the oracles start answering the questions in a round robin fashion. This means that oracle0 answers its first question from its queue, then oracle1 answers its first question from its queue, and so on until every oracle has answered its first question (or not answered if the queue is empty). This process is repeated with every oracle answering their second question in the same order, and so on until all the queues are empty.

However, you like to get multiple answers to all of your questions, so you decide to resubmit all of your questions to the oracles to see if you get a different answer. The assignment of questions and reading of the answers should be done in the same way. The first time you submit your questions, you should use your ArrayQueue implementation and the second time you should use your ListQueue implementation.

Problem Definition/Examples

This homework assignment is an application of queues and is designed to help you familiarize you with basic data structures. The assignment revolves around the list of questions read in form a file that need to be assigned, at random, to an oracle. After the questions are assigned, the oracles should each take turns answering questions as described above.

Each of the oracles will answer all the questions in the same way (i.e. the oracle will give the same response no matter the question posed to it.) The program should output the oracle's response and the original question. For example, if the question "What is your quest?" is submitted to the oracle who always responds "To seek the holy grail," the program should output

```
What is your quest?: To seek the holy grail
```

Your program should successfully submit all questions to an oracle and print the answers in order.

Given Material

There are four java files and two text files provided as part of the assignment. They are commented for the Eclipse IDE to explain the behavior of the provided code.

Utility.java

The Utility file is provided to deal with the file reading and random number generation because they are not part of the course. This file should not be modified.

The init methods initializes utility and sets up the file reader and random number generator. This method is sufficient initialization and calling a constructor on Utility is unnecessary.

The readQuestions/readAnswers methods return the files questions.txt and answers.txt as arrays of strings.

The random function will return a random number between 0 and the parameter number. This will be useful for assigning questions to oracles. You will call it to find out to which oracle a questions should be assigned.

Executor.java

Executor is the main program that you will execute to run your code. It will involve all the code that is not written in the queue implementations. As provided, this file only initializes Utility and reads the input files into string arrays. There are comments to tell you what to do.

ArrayQueue.java

The ArrayQueue file needs student work as it only provides the structure of the data structure. All functions need to be written by the student and class variables may be added if needed. Some functions currently have temporary return values, which can be removed when you write your code.

We will be using a circular array. This means that you **do not move the elements in the array**. You should enqueue and dequeue by moving the corresponding front and back pointers which each track of where the front and back of the array is. The front pointer should point at the front of the queue (the next element to be dequeued) and the back pointer should point at the back of the queue (the most recently added item). Because these pointers may over lap and move from the back of the end of the array to the front of the array, array queue are also known as rotating queues.

Array queues can become completely full if the array becomes full and no element is dequeued. Make sure that your code can recognize when the queue is full, especially when enqueueing additional elements. There is no need in this assignment to create an array queue which increases the size of its array, a simple error message when adding to a full queue is sufficient.

ListQueue.java

The ListQueue file needs student work as it only provides the structure of the data structure. All functions need to be written by the student and class variables may be added if needed. Some functions currently have temporary return values, which can be removed when you write your code. As this implementation is using a list, there should be a pointer keeping track of the front of the list and the back of the list. The front pointer should point at the front of the queue (the next element to be dequeued) and the back pointer should point at the back of the queue (the most recently added item).

Currently, there is no file for ListQueueNode. You need to either create a class in a separate file or you can create and inner class inside ListQueue (in which case you will not have a separate file).

Input Files (questions.txt and answers.txt)

These are the questions and answers. The number of oracles should be the number of answers in the text file. These files are read into string arrays by the Utility class, so you don't have to do it.

ArrayQueueTest.java

This is a test program to test the circular nature of your ArrayQueue. You will need to run this program to help determine if your ArrayQueue class is coded correctly. This only checks the circular nature of the queue, not any other functionality.

Testing Your Queue Implementation

Just showing that your program is able to print the questions and answers correctly is not proof that your program is working correctly, nor is being able to pass the ArrayQueueTest proof that all functions are working correctly. You will need to test it more thoroughly. We will test your queue implementations more generally and so should you. We suggest creating another small client program of your own to help test other portions of your code.

Expected Results

As a standard for this course, please make sure your code is thoroughly commented. Not only does this help the course staff when grading, but it will help you to understand the flow of the data structures. Concepts in this course can be very abstract, getting into the habit of commenting early will help later in the course.

Write Up:

Please submit a README.pdf file, answering the following questions.

1. How did you test your queue implementations were correct?

2. One option with array stacks and queues is to resize when they become full. Assuming that the initial size is 128, how many times would the array need to resize if you added 1 million items? What about with 1 billion items or 1 trillion items (assuming the computer had enough memory)? Explain how you got your answer.

3. Suppose that instead of using an array or list as the organization structure to implement a queue, you were provided a functional Stack. How would you implement dequeue using one or more stacks? Provide the psuedocode below. (Reminder Stack provides the methods pop, push, peek, and isEmpty)

4. Suppose that you have enqueued ***n*** items. Then you dequeue a single item. For that dequeue, what is the time complexity for the list implementation? What about the array implementation? And the Stack implementation? Additionally, what is the space needed for the each of the three implementations?

5. Include a description of how your project goes "above and beyond" the basic requirements (if it does).

6. What did you enjoy about this assignment? What did you not enjoy? What could you have done better?

7. What else, if anything, would you like to include related to this homework?

Going Above and Beyond

The following list of suggestions are meant for you to try if you finish the requirements early. Recall that extra-credit points earned for "above and beyond" are kept separate and will be used to adjust your grade at the end of the quarter, as detailed in the course grading policy. Please do any extra credit in a separate file (you can copy the class and make the modifications in the extra credit version).

- (1 point) Modify your array implementation so that when the array is full, the queue resizes to use an array of twice the size.

Logistics

Turn-in: You should turn in the following files. Each should have your name, id number, and UW username at the top of the file. As we are not collecting the Utility.java, you should make sure you do not change this file and your program will run with the file as it is given to you.

- Executor.java
- ArrayQueue.java
- ListQueue.java
- ListQueueNode.java (If you wrote separate file for this class. If you wrote this as an inner class, that is fine and it will already be turned in with the ListQueue class)
- README.pdf
- Any additional files for extra credit in a zip file named extracredit.zip. Please make sure that this zip file decompresses its contents into a folder called extracredit and *not* into a bunch of individual files.