CPSC 2151

Lab 6

Due: Friday October 12th at 11:59 pm

In this lab you will be converting our IntegerQueue interface and implementations to a Generic interface and implementation

Instructions

1. Copy your code from the previous lab into a new project. Name your package cpsc2150.myQueue
2. Rename your IntgerQueueI Interface to be IQueue. Make sure to change file names as well
3. Change your implementations to implement IQueue. They can keep the same file name
4. Convert the interface and it's implementations to be generic, so they can create a Queue of any data type. Update any contracts to no longer refer to Integers, but refer to some generic concept like an object instead.
5. Convert your QueueApp code to use the generic IQueue interface and implementations instead of the old IntegerQueueI interface and implementations. QueueApp will still only create a Queue of Integers QueueApp itself will not be generic.
6. Create a new class called StringQueueApp. This class will be the same as QueueApp, except instead of a Queue of Integers it will create a Queue of Strings. Some of your helper methods that interact with the Queue will need to change as well. Most of the code will be unaffected.
7. You can add a new run configuration to point to your StringQueueApp code instead of QueueApp so you can test out your code. If you don't rememeber how to do that, please see the video on Canvas about creating a new program in IntelliJ. It shows how to do this at around the 6:50 mark in the video. IntelliJ will allow you to have multiple run configurations, so you should be able to run both QueueApp and StringQueueApp from the same project.

TIPS and additional Requirements

- If you were using nextInt() to read in input in your previous assignment, you may have some issues when you are also reading in strings. Scanner.nextInt() reads in the integer value, but leaves the End of Line character on the buffer. When you call Scanner.nextLine() to get a String, it will first read the End of Line character, which is the character it looks for to stop reading in input. It will return that EoL character as the string, but leave the desired input on the buffer. An easy way to solve this is to either always use nextLine and parse the string to an Integer, or after you call Scanner.nextInt() call Scanner.nextLine() to clear the EoL character from the buffer.
- Remember that casting will be necessary to create your array
- You will also need to include a make file. Note: we now have two possible entry points and two ways to run our program because there is a main in QueueApp and a main in StringQueueApp. This means to will need to have 2 different targets in your make file to run the program. The target that will run the QueueApp Main will be called runInt, and the target that will run the StringQueueApp Main will be called runString.
-

Groups

You may, but are not required to, work with a partner on this lab. Your partner must be in the same lab section as you, not just the same lecture section. If you work with a partner, only one person should submit the assignment. You should put the names of both partners in a comment at the top of the file in order for both partners to get credit. This assignment may take more than just the lab time. Make sure you are able to meet outside of class to work on the assignment before you decide to work with someone else. Remember to actively collaborate and communicate with your partner. Trying to just divide up the work evenly will be problematic.

Before Submitting

You need to make sure your code will run on Unix. You will also need to include a make file. Note: we now have two possible entry points and two ways to run our program because there is a main in QueueApp and a main in StringQueueApp. This means to will need to have 2 different targets in your make file to run the program. The target that will run the QueueApp Main will be called runInt, and the target that will run the StringQueueApp Main will be called runString.

Submitting your file

You will submit your files using handin in the lab section you are enrolled in. If you are unfamiliar with handin, more information is available at https://handin.cs.clemson.edu/help/students/

You should zip up a folder that contains your package directory and your make file. This will allow the grader to unzip your folder, type make to compile your code, and make runInt to run the integer queue version and make runString to run the string version.

No late submissions are accepted for labs

Check List:

- Have you converted your interface file to now be Generic?
- Are the contracts in your interface file still applicable to a Generic Queue? If they refer to integers, then they are not generic
- Have you converted both of your implementations to be Generic?
- Have you Converted the QueueApp code file to use the Generic version of the Queue to create a Queue of Integers?
- Have you created a StringQueueApp code file that has all the functionality of the QueueApp, but uses a Queue of Strings instead of a Queue of Integers?
- Have you converted all the helper methods (that get and return input from the user) in StringQueueApp to work with Strings as opposed to Integers?
- Have you followed all best practices we have discussed in class. Some examples include:
    - Avoiding magic numbers
    - Programming to the Interface
    - Properly hiding information
    - Commenting your code
    - Factoring out common code to reuse it (creating functions, using default implementations, etc)

    - Limiting input and output with the user to one class (note: for this one we have two classes QueueApp and StringQueueApp)
- Have you created a make file for your project
    - Does that make file have a runInt and a runString target?
- Can your run your code on unix using the make file
- Have you properly structured your submission so the TA can unzip your submission, and then type make to compile your program?
- Have you followed Design by contract
    - Specified invariants, preconditions and postconditions
    - Followed the responsibilities outlined by design by contract (ex. The method does not check it's own preconditions)
    - Are contracts written as formally as they can be?
        - i.e. "0 <= x <= 100" is formal, while "x is between 0 and 100" is not
- If working with a partner, did you include both partners' names in the submission?
- Have you thoroughly tested your program to ensure everything is functional?