CPSC 2151

Lab 5 – Generics, Makefiles, and Default Functions

Assigned: Tuesday 10-9 && Thursday 10-11

Due: Tuesday 10-24 && Thursday 10-26 (**Before lab starts**)

In this lab, you'll be continuing lab 4. If you forgot, lab 4 contained four files:

- IDoubleQueue.java – A now finished interface for a Double queue.
- ArrayDoubleQueue.java – A now finished implementation of a Double queue with an array backend structure.
- ListDoubleQueue.java – A now finished implementation of a Double queue with an arrayList backend structure.
- DoubleQueueDriver.java – A main method file. Mostly used for verifying your solution.

For lab 4, these queues held Doubles, the wrapper/object form for the primitive double type. However, these queues only work for Doubles. This means that while we can make multiple different back-ends for these queues that hold Doubles thanks to the interface, we're still limited to just *Doubles*. Let's fix that.

For lab 5, your job is to convert the IDoubleQueue to be a **generic** queue, practice **default functions**, and add a **makefile** to the project.

Two things to note:

- 1: Do not change the names of the files. Even though they'll be made generic, meaning that they will accept more than just Doubles, you should still keep all of the file names.
- 2: If you program in Windows or MacOS and/or use a modern IDE, makefiles aren't all that necessary as the IDE handles compiling and running our code for us. That being said, this is still a valuable tool to have on the not-so-unlikely chance you'll develop code in Linux or without an IDE.
- A new drive has been provided on canvas for you to help test your code and your makefile. It's effectively the same driver but it uses the generic queue allowing us to make the Queue in the driver accept Strings instead of Doubles. It also calls the peek() function for you.

Instructions:

1. Step 1 is to make your interface and implementing classes all **generic**. This is a simple task since we already have most of the function's bodies written. Don't overthink this step.
    a. Make sure you update any contracts to reflect the fact that we're not always returning / passing in / using Doubles, if needed.
    b. **There's a function in ArrayDoubleQueue.java, right at the bottom, called getQueue(). Make sure you also convert that to be generic.**
2. Next, we should write a function, called **peek()**, that can tell me what is at the front (left-most position) of my queue **without removing that item from the queue.** Let's think about this before we write it…
    a. Functionally, regardless of whether the back-end of the queue is an array or an arrayList, peek is doing the exact same thing: getting the value at the left-most position. This should be the first hint that this is a good candidate for a **default function**.
        i. However, remember that default functions cannot refer to private data directly, since default functions are written in the interface and interfaces don't have private data of their own…
        ii. Look at the four functions we created for lab 4. `Enqueue`, `Dequeue`, and `Length` are **primary functions**, meaning that they *do* have immediate access to private data in the two implementations of the interface. We can use these to code our peek function as a **secondary function** meaning that we "get access to private data" by calling the primary functions. This allows peek to do its job without accessing the private data itself.
    b. Declare and implement the default function, `peek()`, in IDoubleQueue. This will need to be done mostly by calling a combination of your primary functions: `enqueue`, `dequeue`, and `length`.
        i. Peek accepts no parameters and returns the generic type.
        ii. Do not forget to write a contract for `peek`. Since it's default, we always put the contract in the interface.
3. Finally, we need to make a **makefile** for this program.
    a. Your makefile should exist in the src directory, which holds your com package and the Driver package.
    b. Most of you should be familiar with a makefile. Functionally, they're the same as what you'd use one for in C++.
    c. There are 3 different targets we should account for in this course:
        i. **default -** what to do when no specified target when calling `make`. This should compile my program using the **javac** command.

        ii. **run –** self-explanatory. Should run the program using the **java** command on the **.java** file that contains the main, in this case the DoubleQueueDriver.java.
        iii. **clean –** delete the **.class** files since these must be recreated every time we compile, so there's no reason to keep them after my program runs.

d. Remember, because the makefile isn't in the same package as the .java files, when we refer to them (and their .class files) we need to refer to them by their path.
    i. For example, the way to the IDoubleQueue.java dependency in the makefile is:

```
com/Gradescope/DoubleQueue/code/IDoubleQueue.java
```

e. If you wish to test your makefile before you submit it and you're not using Linux as your primary operating system, you're in a Linux lab for a reason… (or you could run a Linux subsystem).
    i. In Linux, open your project and go to the src directory.
        1. Run `make` and your code should compile, meaning that if you go into the `com/Gradescope/DoubleQueue/code` directory you should see your .java files and your .class files.
        2. Running `make run` should run your program, allowing you to run it as if it were running in your normal IDE.
        3. Running `make clean` should delete all the .class files.

f.

Contracts and Formatting:

Contracts should be formally stated when possible. We have not covered much formal notation, but if a parameter named row needs to be greater than 0, then the precondition should say "x > 0" not "[x must be greater than 0]."

Everything that has been mentioned as a "best practice" in our lectures should be followed, or you risk losing points on this portion of the lab assignment.

Partners:

You must work in teams of 3-4 students. You do not need to team with the same students that you have teamed with before in previous labs. **Remember that working with a partner means working with a partner, not dividing up the work.** You don't need to run a driver-navigator set-up for this kind of assignment, but you should still make sure that everyone works on contracts to some degree, and everyone works on implementation to some degree…

Submission:

You need to submit:

- IDoubleQueue.Java
- ListDoubleQueue.java
- ArrayDoubleQueue.java
- makefile

Submit the 4 files to Gradescope. **Do not submit your entire project directory.**