CPSC 2151

Lab 4

Due Friday, September 18th at 10:00 pm

In this lab you will be working with `interface`s. You are provided with an `interface` for an `IntegerQueue`, and you will provide the implementations and specification for the `interface`. You will need this code completed for later labs.

Instructions

1. Create a new project called `Lab4` with a package called `cpsc2150.MyQueue`. Add a class called `QueueApp` that will contain our `main` function.
2. Copy the following code into your `main` function

```
IQueue q;

/*
You will add in code here to ask the user whether they want an
array implementation or a list implementation. Then use their
answer to initialize q appropriately
*/

Integer x = 42;
q.enqueue(x);
x = 17;
q.enqueue(x);
x = 37;
q.enqueue(x);
x = 36;
q.enqueue(x);
x = 12;
q.enqueue(x);

//Add the code to print the queue. After the code is finished,
the queue should still contain all its values in order
```

3. Create a new `interface` file. In your project window, right click on the package name, select **New → Java Class**. In the window that pops up, name your class `IQueue` (that's an upper case i not a lower-case L) and change the "kind" field to `interface`.
4. Copy the following code and partial specification into your `interface` file. You should not make any changes to this code itself, but you should add contracts and Javadoc comments for each method and complete the `interface` specification (i.e. defines, constraints and initialization ensures).

```
/**
 * A queue containing integers.
 * A queue is a data structure where the first item added to the
structure is the first item removed from the structure
 * This queue is bounded by MAX_LENGTH
*/
public interface IQueue {
    public static final int MAX_LENGTH = 100;

    // Adds x to the end of the queue
    public void enqueue(Integer x);

    //removes and returns the Integer at the front of the queue
    public Integer dequeue();

    //returns the number of Integers in the Queue
    public int length();

    //clears the entire queue
    public void clear();
}
```

5. Create a new class in your package called `ArrayQueue`.
6. Copy the following code into the `ArrayQueue` file

```
public class ArrayQueue implements IQueue {

    // where the data is stored. myQ[0] is the front of the queue
    private Integer[] myQ;
    // tracks how many items in the queue
    // also used to find the end of the queue
    private int myLength;

    // complete the class
}
```

7. Finish implementing the `ArrayQueue` class.
    a. Add a constructor to the class that creates an "empty" queue with a maximum length of 100. The constructor should not have any parameters.
    b. You will need to add any necessary contracts and Javadoc comments to the constructor
    c. Add the methods needed to meet the `interface` specification
    d. Do not add any additional methods other than the ones provided in the `interface` specification
    e. Do not add any additional `private` data fields
    f. Add your correspondences and invariants to the class
8. Create a new class in your package called `ListQueue`.
9. Copy the following code into the `ListQueue` file

```
import java.util.*;
```

```
public class ListQueue implements IQueue {

    // this time store the queue in a list
    // myQ.get(0) is the front of the queue
    private List<Integer> myQ;

    // complete the class
}
```

10. Finish implementing the `ListQueue` class.
    a. Add a constructor to the class that creates an "empty" queue. The constructor should not have any parameters.
    b. You will need to add any necessary contracts and Javadoc comments to the constructor
    c. Add the methods needed to meet the `interface` specification
    d. Do not add any additional methods other than the ones provided in the `interface` specification
    e. Do not add any additional `private` data fields
    f. Add your correspondences and invariants to the class.
11. Return to your `main` function in `QueueApp.java`. Add the code to ask the user which implementation they would like to use and to initialize the `IQueue`. Then initialize the `IQueue` object according to the option they chose. Only the constructor call should be in the `if` statement.
12. Add the code to print the queue to the screen. After the queue has been printed, it should have the same values in the same order as it did before it was printed.
13. Create a `makefile` and test your code on SoC Unix machines before submitting.

Partners

You may work with one partner on this lab assignment and you are encouraged to do so. Make sure you include both partners' names on the submission. You only need to submit one copy. Remember that working with a partner means working *with* a partner, not dividing up the work. You will need this code for a later lab, so make sure both partners have a copy of it.

Before Submitting

You need to make sure your code will run on SoC Unix machines and create a `makefile`.

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 submit a zipped directory with your package directory and your `makefile`. The TA should be able to unzip your directory and type `make` compile your code, `make run` to run it and `make clean` to delete any `.class` files.

NOTE: Make sure you zipped up your files correctly and didn't forget something! Always check your submissions on handin to ensure you uploaded the correct zip file.