CSC9A3 Practical 5

Queues and more queues!!!

Intro

This practical will introduce you to the Java documentation and some BlueJ/Eclipse shortcuts. You will then build a shop with various queues, continuously running the unit tests to ensure that your code is correct.

IN THIS PRACTICAL, THERE ARE 3 CHECKPOINTS, EACH ONE ASSOCIATED WITH A UNIT TEST CLASS. WHEN ADJUSTING/CHANGING THE CODE FOR ONE CHECKPOINT, YOU WILL PROBABLY BREAK THE UNIT TEST CLASSES FOR THE OTHER CHECKPOINTS. IF YOU WANT TO MAKE ALL UNIT TEST CLASSES WORK AT THE SAME TIME, YOU HAVE TWO OPTIONS:

(1) CHANGE THE ORIGINAL CODE TO MAKE IT WORK FOR ALL UNIT CLASSES AT THE SAME TIME, or

(2) REPLICATE THE ORIGINAL CODE 3 TIMES, ONE FOR EACH CHECKPOINT/UNIT TEST CLASS.

Javadocs

Documentation is boring to write, but very handy if you're a programmer needing to find out what code does. Let's make use of the Javadocs, documentation to describe Java classes, to learn what methods are available to us for the ADTs; Stacks and Queues.

Check the online documentation for Java Stack and Queues:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Queue.html

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Stack.html

Have a look around the page to get an idea for what is going on. Half way down you will see a 'Method Summary' list, showing the methods that we would expect for a Stack (peek, pop and push) and Queue (add, peek, poll).

Task

Find out the method names used in Java for dequeueing and enqueueing an item from a queue.

BlueJ/Eclipse Shortcuts

Here are some shortcuts to make your life a little easier (if you don't already know them).

Compiling

To compile in BlueJ (from any window) press CTRL+K.

To compile in Eclipse press CTRL+F11

Magic Fix Layout

This is a wonderful feature of BlueJ (only because it is sorely required). When you have a class open, press CTRL+Shift+I and watch your code tidy itself. (Now you have no excuse for showing demonstrators/submitting assignments with poorly laid out code!).

The same outcome can be achieved in Eclipse by pressing CTRL+Shift+F.

Viewing Documentation

In BlueJ, when you have a class open there is a pull-down menu at the top right of the screen, you can toggle it between source code and documentation. For classes which have good documentation this can be a handy way to view this documentation.

In Eclipse, when you hover your mouse over the Class import and press F2, the documentation is presented.

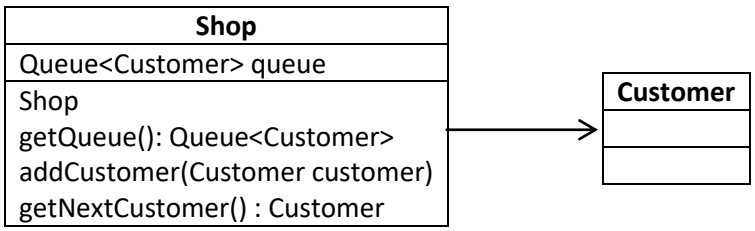
Method Name Auto-Completion

For Eclipse and BlueJ, at any stage when you are coding you can access a list of available methods to you. Hit CTRL+space to see them, navigate using the cursor keys and select using enter.

If you want to access a method within another class you need to access it via an object. This would be objectName.methodName(parameters), for example `myInstanceVariable.methodInsideClass()`. This shortcut works well in this situation, as you can type "myInstanceVariable." then hit CTRL+space to get a list of all the available (public) methods within that class.

The Problem: Coffee Shop

We have a coffee shop, which has a single queue of customers. We can add a customer to the queue, get the next customer in the queue, and get the entire queue.



Note

You will notice that the methods have all been made public, and you'd be right to question this. The methods are public to allow the unit tests to access them.

Task 1 – Warm Up

Download the code on Canvas.

Open the code for this practical (note, ShopTestTask2, 3 and 4 will not compile – this is OK) and fill in the method bodies `addCustomer` and `getNextCustomer`. Run the `ShopTestTask1` tests to ensure that you have correct code.

Task 2 – Multiple Tills

Our lovely boss in the coffee shop bought some new tills! Instead of having one queue, we now have multiple queues, one for each open till. The number of tills available in our shop is passed in as a parameter to our constructor. Observe the changes made to our shop class:

Shop
Queue<Customer>**[]** queue**s**
Shop(**int numberOfTills**) getQueue(**int tillNumber**): Queue<Customer> addCustomer(Customer customer**, int tillNumber**) getNextCustomer(**int tillNumber**) : Customer

Make the necessary changes to your Shop class to reflect the addition of new tills. (Note: `tillNumber` starts at 1). Run the `ShopTestTask2` tests to ensure that you have correct code.

Hint

To initialise the array of Queues you will need the following code:

```
new Queue[numberOfTills]
```

Instead of one queue, we now have an array of queues. For each queue in the array you will need to initialise it. (Look at the code from Task 1 to see how to declare and initialise one queue).

[Checkpoint]

When you finish programming and all the tests pass in `ShopTestTask2` you can be happy that your solution is correct. Call a demonstrator and show your code.

Task 3 – Check Point: Shortest Queue

Instead of adding a customer to a specific queue, let's add the customer to the shortest queue. Add a new **method** to the Shop class called `getShortestQueue` that returns a reference to the shortest queue in the array of queues. This new method will be used by `ShopTestTask3` to check that we can now add a customer to the shortest queue. When you think you have it working, run the `ShopTestTask3` tests to ensure that your code is correct.

Hints

Avoid looking this up – there is no magical solution. Consider the steps you take when choosing a till at a supermarket. Next, put this on paper, and finally pseudocode. Only at this stage when you are comfortable with the logic move should you move onto code.

[Checkpoint]

When you finish programming and all the tests pass in `ShopTestTask3` you can be happy that your solution is correct. Call a demonstrator and show your code.

Task 4 – New System

That boss of ours decided to stop throwing money around on new tills, and instead he got smart! He noticed that it was all the fancy-coffee buyers that were slowing things down for the simple-coffee buyers, and this was unfair. So instead he decided that his shop would have only two tills, the first till was for

simple coffee customers, and the second till for the fancy coffee customers. This allowed the simple coffee buyers a nice speedy way of buying their coffee.

Make the changes to the Customer class to store what kind of coffee buyer they are.

Customer
buyingFancyCoffee: boolean
Customer(boolean buyingFancyCoffee) isBuyingFancyCoffee(): boolean

Make the necessary changes to your Shop class to reflect our boss's new idea (note that you will need a new default constructor for Shop that initialises the shop with **2 tills** – this is same as having **numberOfTills = 2**, in task 2). Run the `ShopTestTask4` tests to ensure that your code is correct.

[Checkpoint]

When you finish programming and all the tests pass in `ShopTestTask4` you can be happy that your solution is correct. Call a demonstrator and show your code.