Grocery Store Simulator 2014 - Part 1

Backstory

After success with their latest simulation game “The Jims 3”, BA Games is at it again. You’ve been hired to develop a prototype simulator for this year's most anticipated Grocery Store Simulation game creatively named “Grocery Store Simulator 2014”. Will you rise to the challenge?

Overview

For this assignment you will be familiarizing yourself with the Queue ADT. In addition to this, you will begin developing a framework to be used to simulate a Grocery Store.
Due: March 14th, 2014 @ 5pm

Files to complete

You are expected write an implementation for each of the interfaces listed in the classes presented in the **config** package privided. As with the last assignment, you must specify which implementation you would like us to grade in this file.

In addition to this, you **MUST** include a String field in the **config.Configuration** file with your UMass student ID number. This is the 8 digit value found on your student ID card and is used to identify you in moodle. This will allow us to get your grade and feedback to you quickly. We recommend that you enter this ID value immediately after importing the project.

It will almost certainly be useful for you to write additional class files. Any class file you write that is used by your solution **MUST** be in the provided **src** folder. When we test your assignment, all files not included in the **src** folder will be ignored.

Note: When you submit your solution, be sure to remove **ALL** compilation errors from your project. Any compilation errors in your project may cause the autograder to fail and you will receive a zero for your submission.

Test files

In the test folder, you are provided with several JUnit test cases that will help you keep on track while completing this assignment. We recommend you run the tests often and use them as a checklist of things to do next. You are not allowed to remove anything from these files. If you have errors in these files, it means the structure of the files found in the **src** folder have been altered in a way that will cause your submission to lose points. We highly recommend that you add new @Test cases to these files. However, before submitting, make sure that your program compiles with the original test folder provided.

Support Code API

The Support Code’s comments have been generated into a nicely formatted API that can be found here: http://people.cs.umass.edu/~jcollard/cs187/s14/assignments/6/doc/

It is highly recommended that you spend a day simply reading over the comments in each of the interfaces and classes provided.

Part One: Importing Project into Eclipse

Begin by downloading the provided project from Moodle and importing it into your workspace.

You should now have a project called **grocery-simulator-student** it is very important that you do not rename this project as it is used during the grading process. If the project is renamed, your assignment may not be graded.

By default, your project should have no errors and contain the following root items:
src - The source folder where all code you are submitting must go. You can change anything you want in this folder, you can add new files, etc...
support - This folder contains support code that we encourage you to use (and must be used to pass certain tests). You are not allowed to change or add anything in this folder
test - The test folder where all of the public unit tests are available
JUnit 4 - A library that is used to run the test programs
JRE System Library - This is what allows java to run

If you are missing any of the above or errors are present in the project, seek help immediately so you can get started on the project right away.

Once you have your project imported, add your 8 digit student ID number to the **config.Configuration** class.

Part Two: Implement the QueueInterface

Due: March 14th 2014 @ 5pm

Create a class that implements **structures.QueueInterface**. The Queue you are implementing should be unbounded and meet the big-O time complexities specified in the interface.

Tip: Don't try to implement your entire Queue class all at once. Start by looking at the the tests provided in **structures.QueueInterfaceTest**. Do these tests look like they cover all the cases presented by the interface? If not, can you come up with additional tests? Add a method to the test file and add the @Test annotation before it. When you run your unit tests, this method will also be executed. Be careful not to change any of the provided unit tests!

Once you feel like the tests accurately represent what you would like to test, choose the test that looks the least complex. Implement the parts that will allow that test to pass. Once those tests pass. Look for another test to implement.

Note: For this portion of the assignment, you may not use any of the provided Collection classes provided by the java standard API. A list of them can be seen in the "All Known Subinterfaces" and "All Known Implementing Classes" of the following URL:
http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html

Part Three: Implement a NormalLine and an ExpressLine

Due: March 14th 2014 @ 5pm

The **simulator.checkout.CheckoutLineInterface** extends **QueueInterface<Shopper>**. That is, it is just a queue for shoppers to stand in while they wait to be processed. However, there are a few additional requirements for this interface.

First, it defines a **canEnterLine**. This method determines if a shopper is allowed to enter a particular line. This can be used to represent special lines such as “15 items or less”.

Second, the definition for **enqueue**, is slightly different in that if the **canEnterLine** method returns false on the input, this method should throw an exception.

For this part, you must implement two versions of CheckoutLineInterface. The Normal Line class should always return true with a call to canEnterLine. The ExpressLine should return false if the specified Shopper has more than 15 items.

Tip: You should be able easily implement this class by extending the class you wrote in part two and @Override enqueue.

Tip: Write NormalLine first then extend it when you implement ExpressLine and @Override canEnterLine.

Part Four: Create Groceries

Due: March 14th 2014 @ 5pm
In the **config.Groceries** class, return valid implementations for each of the static methods defined.

Tip: You should be able to accomplish this by implementing a single class called Grocery.

Part Five: Implement a Receipt Class

Due: March 14th 2014 @ 5pm

For this part of the assignment, you will be extending the abstract class **simulator.AbstractReceipt**. If you look at the definition, you will see there are two abstract methods left for you to implement: **getSubTotal()** and **getSaleValue()**.

Tip: Take advantage of the supplied methods. You shouldn’t need to have fields in your subclass to hold the list or discount. You can access these in your subclass by using **super.getDiscount()** and **super.getGroceries()**, **super**.

Tip: Take a look at the List<T> interface in the Java API to see how it works. You will need to use it to calculate the subtotal. Notice that List<T> implements Iterable<T>. This means that it can be used in the enhanced for loop!

List API: http://docs.oracle.com/javase/7/docs/api/java/util/List.html
Iterable API: http://docs.oracle.com/javase/7/docs/api/java/lang/Iterable.html

Using Enhanced For Loop w/ List:
Assume there is some List<String> called strings. Then we can write:

```
for(String s : string){
  System.out.println(s);
}
```

This will print out all of the elements in strings.

Another Example:
Assume there is some List<GroceryInterface> called groceries. Then we can write:

```
for(GroceryInterface g : groceries){
  System.out.println(g.getCost());
}
```

This will print out the cost of all GroceryInterface items in groceries.

Part Six: Export and Submit for Deadline #1

When you have finished and are ready to submit, export the entire project. Be sure that the project is named **grocery-simulator-student**. Save the exported file with the zip extension (any name is fine).

Log into Moodle and submit the exported zip file.