COSC 4301 – MODERN PROGRAMMING
Project 3 – Object–Oriented and Functional Programming

Write a public class named ***StoreItem*** that holds data about an item in a retail store. The class should store the following data in attributes:

- Item Number
- Item Description
- Units in Inventory
- Price

Create another public class named ***CashRegister*** that can be used with the ***StoreItem*** class. The *CashRegister* class should be able to internally keep a list of ***StoreItem*** objects. The class should have, but not be limited to only these methods:

- **purchaseItem** — A method that accepts a *StoreItem* object as an argument. Each time the purchaseItem method is called, the *StoreItem* that is passed as an argument should be added to the list.
- **getTotal** — A method that returns the total price of all the *StoreItem's* objects stored in the *CashRegister's* internal list.
- **showItems** — A method that displays data about the *StoreItem* objects stored in the *CashRegister* object's internal list. These are the items the user has selected to purchase so far.
- **clearRegister** — A method that clears the *CashRegister* object's internal list.
- **showInventory** — A method that displays the snapshot of inventory in the store.
- **checkOut** — A method to check out the user. Include a tax of 8.25% to the total price.
- **displayMenu** — A method that displays a menu to the user. The program menu should be created from the second column in the Inventory.txt file (Item Description). The menu should also include the following: **"Show Cash Register"**, **"Clear Cash Register"**, **"Show Inventory"**, and **"Check Out"**

Write a public class name ***Project3*** that uses **collections** to store *StoreItem* objects. Each object in the list should hold data about an item in the retail store (Item Number, Item Description, Units in Inventory, and Price) read from the **Inventory.txt** file.

The test class should also use the *CashRegister* class to allow the user to select several items for purchase, using a menu **(Note: Only one menu is allowed)**.

When the user is ready to check out, the program should display a list of all the items he or she has selected for purchase (sorted by item description) as well as the total price, taxes, and the final price. **Use Lambda and Streams to perform this task**. Ask the user for confirmation. If the user agrees with the purchase, send the output to a file, **Project3-Output.txt,** as a receipt. Also print the time and date along with the cashier's name on the receipt. The cashier's name must be selected at random from the file, **Cashier.txt.** **(Note: This program might be tested with a different file so allow the**

user to enter the name of the files. Do not hardcode the file name in the program) . If the user does not agree with the invoice, clear the *CashRegister* and start over. The data for the retail store is available in the file, **Inventory.txt**. **(Note: This program might be tested with a different file so allow the user to enter the name of the file. Do not hardcode the file name in the program).**

Allow the user to run the program as many times as possible until a sentinel value, **less than zero (0)**, has been entered for the selected item. **No input, processing, or output should happen in the main method. All work should be delegated to other non-static methods.**

All classes in this project must be public, non-static and not nested in other classes.

Every method in your program should be limited to performing a single, well-defined task, and the name of the method should express that task effectively.

Run the program with your own data. Create a folder named, **YourFullName_Project3**. Copy your source codes and the output file to the folder. **Zip the folder, as a ".zip" file, and upload it to Blackboard**.

Before you upload your project to Blackboard:

- Ensure that your code conforms to the style expectations set out in class and briefly discussed below.

- Make sure your variable names and methods are descriptive and follow standard capitalization conventions.

- Put comments wherever necessary. Comments at the top of each module should include your name, file name, and a description of the module. Comments at the beginning of methods describe what the method does, what the parameters are, and what the return value is. Use comments elsewhere to help your reader follow the flow of your code. See the Orientation Project for more details.

- *Program readability and elegance are as important as correctness.* After you have written your method, read and re-read it to eliminate any redundant lines of code, and to make sure variables and methods names are intuitive and relevant.

Read the assignment very carefully to ensure that you have followed all instructions and satisfied all requirements. **You will not get full credit for this project if it is not written as instructed even if it works as expected.**