**Shopping Center Progress Report**

Kyle Kaminski

Progress as of **12/3/2018**

**\*WIP = Work in Progress**

**There will be 5 classes that will be used to efficiently and easily run this project. There is also 2 versions of my own data structures, and 5 exception classes.**

1. **Driver class (WIP, currently testing with methods of Shopping Center class)**
2. **Shopping Center class (Methods fully tested and working, currently testing compatibility with Driver class)**
3. **Customer class (Complete)**
4. **Items class (Complete)**
5. **NonUniqueCustomerException (Complete)**
6. **DEQ (Complete)**
7. **ExtendedQueueException (Complete)**
8. **ExtendedQueueInterface (Complete)**
9. **InvalidException (Complete)**
10. **Queue (Complete)**
11. **QueueException (Complete)**
12. **QueueInterface (Complete)**

**Driver Class (Currently WIP)**

* This class will have the 10 menu options that are necessary to run the program.
* Before printing the menu to the user, they are prompted to enter all of the available items that the store carries, and the respective item quantity. (WIP to get the items to add to the inventory array list)
  + When the user is done entering items, they input the word “Done”. This will end the loop and redirect them to the menu system.
* Option 0 will close the shopping center (terminate the program). By default, everything will re-initialize when the program is rerun. (Complete)
* Option 1 will add a customer to thelist of customer’s shopping in the shopping center and accepts their name. This option will be prepared to catch a NonUniqueCustomerException. If a NonUniqueCustomerException is caught, that means the name of the customer that the user was trying to add matches the name of a customer that is already in the shopping center. The user will be prompted to enter a different name that is unique. (Complete)
* Option 2 will ask the user which customer took an item, and which specific item they took and decreases the stock for that item by 1. (Complete in Shopping Center, still needs to be optimized with Driver)
* Option 3 will ask the user which customer wants to remove an item, and which specific item they want to remove. (Complete in Shopping Center, still needs to be optimized with Driver)
* Option 4 takes the least recently added customer (oldest) and will automatically push the customer into the most efficient checkout Line. However, if the customer that has been shopping the longest has no items in the cart, the customer will not be added to any checkout line and the user will be prompted to say Yes or No to whether or not the customer would like to continue shopping. If yes, time spent in the shopping center will be reset. (Complete in Shopping Center, still needs to be optimized with Driver)
* Option 5 will call the checkout() method from the shoppingCenter to checkout the customers in a fair manner. (Complete in Shopping Center, still needs to be optimized with Driver)
  + The user will then be prompted to input whether the customer wants to leave the shopping center, or if they want to go back to shopping.
  + If the customer wants to leave the store, then they are removed from their respective checkout line.
  + If the customer wants to go back to shopping, then they are removed from their respective checkout line and added back to the list of customers shopping in the store.
    - The customer’s amount of items in their shopping cart is reset
    - The customer’s time spent in the store is reset
* Option 6 will print info for all customers who are still shopping including: (Complete in Shopping Center, still needs to be optimized with Driver)
  + The customer names
  + The number of items that are in each customer’s respective shopping cart
  + The amount of time each individual customer has spent in the store
* Option 7 will print info for all customers who are in checkout lines including: (Complete in Shopping Center, still needs to be optimized with Driver)
  + The customer names
  + The number of items that are in each customer’s respective shopping cart
  + The amount of time each individual customer has spent in the store
* Option 8 will print the list of all items that need to be reshelved (items that were added from option 3). (Complete in Shopping Center, still needs to be optimized with Driver)
  + The specific item will be at “restocking level” when the item is less than or equal to the restock item warning (we will declare this as an inventory of 3 for that item). If an item is at “restocking level”, then the user is given a message that they will need to reorder more of that item (option 9).
  + When the item has an amount of restock item amount (will be declared as 0, meaning no more of the item is in stock or inventory), the customer cannot take any more of that item.
* Option 9 will prompt the user to to input the specific item they want to reorder, and the amount of that item they want. (Complete in Shopping Center, still needs to be optimized with Driver)
  + When reordering successfully, then the size of that item’s list is updated with the new size (original stock + reordered amount).
  + When reordering unsuccessfully (an item that we do not carry was inputted), the item is denied a reorder.
  + When too much of an item we *do* carry is reordered, the user is notified that the reorder will overstock that item.
    - They will be prompted if they want to still reorder that item.
      * If the user does want to reorder that item, then the size of that item’s list is updated with the new size (original stock + reordered amount)
      * If the user does not want to reorder that amount of the item, then they will be denied a reorder of that item.

**Shopping Center Class**

* This class will be where we initialize and store the list of customers (which are Customer objects) shopping in the store
  + There will be an addCustomer() method to add a customer to the list of customers shopping in the store
    - All the customers in the list of customers must have unique names.
    - If a name that is entered is the same as another customer’s, then a NonUniqueCustomerException is thrown
  + When a customer takes an item, the following will occur:
    - The customer’s number of items increases by 1
    - The item will be added to a list of items that need to be restocked. (will NOT automatically be added back to the shelf)
    - 1 minute passes by for each customer in the store.
    - The stock of the item is decreased by 1 (The amount field of the respective item in the inventory is decreased by 1).
  + We will have a method that can print a specific customer’s information (calls the toString() method of the Customer class).
  + We can add or remove any item that is available in the store from the customer’s cart
  + There will be a method to print the info of all of the customers still shopping in the store (calls the toString() method of each customer)
* This class will be where the items (which are Item objects) of the shopping center will be initialized
  + Items will be stored in a list of items called inventory.
  + From the menu, the user can also reorder more of a specific item for the shopping center’s inventory. That process will be handled here.
  + This class will handle the addition of an item’s inventory when more stock is ordered.
  + This class will contain a method to print all items that are at or below re-stocking level
* This class will be where we initialize and store the 3 checkout lines (which are DEQs)
  + There will be a fixed amount of checkout lanes in the ShoppingCenter class (2 regular and 1 express)
  + If the customer that is checking out has less than or equal to 4 items, they will be removed from the list of customers shopping, and added to the DEQ of the express checkout line.
  + If the express line is twice as long as regular line 1 and regular line 2, then the customer checking out will be added to the smallest regular line.
  + If the customer that is checking out has more than 4 items, they will be removed from the list of customers shopping, and added to the DEQ of the smallest regular checkout line, whether that is regular line 1 or regular line 2.
  + If the customer with the most amount of shopping time is done shopping **AND** their cart is empty, they will be prompted to leave or return shopping.
    - If the customer wants to leave the store, then they are removed from the list of customers shopping in the store (they will not be added to any of the checkout lines)
      * If the customer wants to go back to shopping, then they are removed and then re-added from the list of customers shopping in the store (they are re-initialized under the same name).
  + This class will have a method that will print out the DEQs (from front to back) of all the customers in Regular Line 1, Regular Line 2, and the Express Line
* This class will contain a checkout() method that check customers out in a fair manner.
  + This method will look throughout ALL of the three checkout lines, and will find the customer at the front of the line, with the MOST amount of time spent in the store. This customer will be the first one to be checked out (whether they were on an express or regular checkout line does not matter; for example, if Regular Line 1’s first customer has been shopping for 20 minutes, Regular Line 2’s for 10 minutes, and Express Line’s for 15, then Regular Line 1’s customer will be first).

**Customer Class**

* This class will contain a name field which is the name of the customer
* An int field for the amount of minutes the customer has been shopping will be initialized and stored in this class.
* This class will contain an int representing the number of items in the customer’s cart
* We will have a toString() method that returns a string containing the info about the customer including the customer’s name, the number of items in the customer’s cart, and the amount of time the customer has been shopping.

**Item Class**

* This class will store information about an item including the item’s name and the quantity
* This class will have a toString() method that returns a string describing the item by including its name and its quantity.

**NonUniqueCustomerException**

* A NonUniqueCustomerException is an exception that is thrown when a customer is added to a shopping center but given a name that is the same as another customer that is in the shopping center