Milestone 5 Report

Tawseef Patel (101145333)

Gilles Myny (101145477)

Question:

1. What is the difference between an interface and abstract class?
   The key differences between an interface and an abstract class are as follows. Methods inside an interface are implicitly abstract, they can't have implementations. Compared to an abstract class which can have methods that implement a default behaviour. The classes which extend the abstract class can then incorporate and override these methods to their liking. Methods in an interface are public, compared to the abstract class which allow for public, private, etc. In conclusion, an abstract class permits the programmer to create methods that subclasses which extend the abstract method can implement or override. Compared to the interface class where the programmer can define functionality of the classes that implement it.

2. Why we should "code against"/"program to" an interface (in relation to OOP)?
   The reason we should program to an interface is because we focus our design on what the code is doing rather than how it does it. This allows for a design which is very flexible. In an interface we define all the methods we want and then create classes which implement the interface.

Changelog:

Version 5.0 (2020-04-14)
Release Highlights
- StoreView.java
- StoreManager.java
- ShoppingCart.java
- Inventory.java
- Product.java
- ProductStockContainer.java
- StoreUML_M5.pdf

Features

- ProductStockContainer.java:

New Additions: An interface implementation was chosen for the ProductStockContainer as an interface is mostly used for future enhancement of a code project. Since Inventory and ShoppingCart share many near identical methods such as: getProductQuantity, addProductQuantity, removeProductQuantity and getNumOfProducts it made the most sense to choose an interface. If future development calls for the creation of a class that shares many attributes with a ShoppingCart or Inventory class, this interface makes it easier. The goal in our code is to define functionality rather than implement it into the code.

- StoreView.java:

New Additions: None

Milestone 4 Description: Completely revamped the StoreView into a class which can produce a GUI. The brand-new GUI can add and remove an item to/from the cart using a JComboBox dropdown menu. If there is no quantity of the item in the cart, then the remove from cart button is not able to be clicked. If there is no more quantity of the product available in the inventory, then the add to cart button is not able to be clicked. When an item is added to cart, the user can choose to press checkout which prompts the user for confirmation to checkout. Once checked out, the browse frame is then closed, and a new frame is opened where the items that were in the users' cart are displayed along with the total for the items in cart. For easier understanding, we added a "leave store" button which upon being clicked, exits the program. Also, the program exits if the user closes the browse frame. The browse frame and checkout frame are not resizable and have the dimensions set to 1280x710.

Milestone 3 Description: Added exception handling for InputMismatchException. This exception occurs when the user input is not an integer. Fixed a bug where a user could remove products that did not exist from cart.

- StoreManager.java:

New Additions: None

Milestone 3 Description: Added the ability to retrieve a shopping cart object that is stored and controlled by the manager using a given cartID.

Milestone 2 Description: Completely revamped the StoreManager class such that upon creation of the manager, there is an inventory that is created along with a HashMap of carts and the associated cartID. From there, multiple methods were implemented such as getters to call on from the Shoreview's manager to access the inventory and cart that are both controlled by the StoreManager. Further, the StoreManager now can add/remove items from the cart.

Milestone 1 Description: creates new Inventory and a variety of products given their name, ID, and price. Tests adding the created products to the inventory with stock amount; restocking products given their IDs with amount to restock; removing stock of products given their ID's with amount to remove. Returns a Boolean value if all tests are successful.

- ShoppingCart.java:

New Additions: Refactored and renamed some methods to fit with the implemented class. Added some helper functions as well.

Milestone 4 Description: : Changed the viewCart() and checkOut() methods to return String representations rather than printing. These functions were used in the GUI.

Milestone 3 Description: Added the ability to retrieve the quantity of a product that is in the cart using a Product object. Also added a method which creates and returns an ArrayList of all the products in the shopping cart.

- Inventory.java:

New Additions: Refactored and renamed some methods to fit with the implemented class. Added some helper functions as well. Deleted the getInventory() class which was no longer used after Milestone 3 due to the GUI rather than the terminal outputs.

Milestone 3 Description: Bug fixes.

Milestone 2 Description: Javadoc Format, added the ability to use Hashmap instead of two ArrayLists. Renamed methods and created new ones to better accommodate the move to Hashmap.

Milestone 1 Description: Inventory class allows StoreManager to create a new inventory. Contains two ArrayList's for products and stock information. Setters allow StoreManager to create (if the product does not exist) or add and remove product stock given the product name or the product ID. Getters allow StoreManager to get product info with the product ID, returns the price of a quantity of items given the product ID, or get stock amount with the product ID.

- Product.java:

New Additions: None

Milestone 2 Description: Javadoc formatting, added and implemented much more efficient equals methods which now could check given a product object attributes if it equals to the attributes of a product object.

Milestone 1 Description: Allows StoreManager to create and set a name, ID, and price for the product. Includes getters and setters for required public information. Creates a print override for proper printing of each product to console. Checks for duplicate products.