Milestone 2 Report

Tawseef Patel (101145333)

Gilles Myny (101145477)

Questions:

1. What kind of relationship is the StoreView and StoreManager relationship? Explain.

    The relationship between StoreView and StoreManager can be described as StoreView has a relationship with StoreManager. This is because StoreView creates a StoreManager and has thus has an instance of StoreManager every time a StoreView is launched which will allow for a unique cartID to identify the user of the system which is generated by the StoreManager.

2. Due to their behavioral similarity, would it make sense to have ShoppingCart extend Inventory, or the other way around? Why or why not?

    No, it does not make sense for ShoppingCart to extend inventory or the other way around because the shopping cart is controlled by the StoreManager and the StoreManager has a relationship with Inventory which in turn allows for the ShoppingCart to be able to access the Inventory via the StoreManager class.

3. What are some reasons you can think of for why composition might be preferable over inheritance? These do not have to be Java-specific.

    Composition is the ability to combine simple expressions and statements into compound expressions and statements. Composition might be preferable over inheritance because it is easier to modify and change the behaviour with getters and setters. This allows for an easier accommodation for future requirement changes in a program which in an inheritance model would require an almost complete restructure of the program. Object composition is defined at runtime through objects acquiring references to other objects. In this case the objects will never be able to reach each-other's protected data and will be forced to respect each other's interface. Composition classes should be able to achieve polymorphic behaviour and allows for code reuse because composition contains instances of other classes that implement the desired functionality whereas in inheritance it is done through a superclass. Inheritance can lead to subtle logic errors because you can run inherited code without realizing it.

4. What are some reasons you can think of for why inheritance might be preferable over composition? These do not have to be Java-specific.

    Inheritance is when one class extends another class. (i.e., a subclass (child) extends a superclass (parent)) and is the ability to define a new class that has the same instance variables and methods of an existing class. Inheritance helps with eliminating repeated code through a process called generalization.

Changelog

Version 2.0 (2020-03-08)

Release Highlights

- StoreView.java
- StoreManager.java
- ShoppingCart.java
- Inventory.java
- Product.java
- StoreUML_M2.pdf

Features

- StoreView.java:

  The StoreView is designed to be the front end of the project where it creates a static StoreManager who has control of the entire front end and allows the user to select the StoreView they wish to browse on. This class contains the main method which takes user input for the StoreView they wish to view followed by giving the option to Browse the store, go to Cart, get Help, or Quit the current StoreView. The user can add an item to cart by clicking B for browse after selecting their desired StoreView and can then enter the product id followed by the quantity they want, and the product is then added to cart. The user then has the option to click B once again for browse or C for cart which then prints out the items in cart and gives the option to Remove from cart, Checkout, or Continue back to the store. If the user selects R for remove then the user has to enter the product id and quantity they wish to remove. If the user selects CH for checkout then the transaction is processed and the option to select a new StoreView is brought up once again. If the user selects CO to continue back to the store they can then choose B for browse, C for Cart, H for help or Q for quit. Once a user selects Q, the StoreView is deactivated and cannot be used again and if all the active StoreViews are deactivated then the program finishes with exit code 0.
- StoreManager.java

  New Additions: 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 StoreViews manager to access the inventory and cart that are both controlled by the StoreManager. Further, the StoreManager now has the ability to add/remove items from the cart.

  Old 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 ID's 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

  The ShoppingCart class is designed to add and remove products from the users shopping cart in the store after they have browsed via StoreView.java. This is done by using a HashMap which has a Key as a Product and value which is an Integer representing the quantity of the product. The addToCart method is designed to add a certain quantity of a certain product which the user selects into the cart and then prints out all the items in cart along with the total. The removeFromCart method is designed to do the exact opposite. It removes a certain quantity of a product if the product exists in cart along with a quantity greater than what the user wants to remove and then prints the updated cart. The viewCart method was created so the user can access the ShoppingCart and view the items in the cart, price, and total.

- Inventory.java

  New Additions: 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.

  Old 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: Javadoc formatting, added and implemented much more efficient equals methods which now have the ability to check given a product object attributes, if it equals to the attributes of a product object.

  Old 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.