Patrick Buhmbi CST-239 Milestone 1 Report 2/03/2024

What does each Class do?

SalableProduct Class;

This class stores all the important details about a product, making it easy for other sections of the application to work with and control product data efficiently. The SalableProduct class serves as a representation of an item available for sale in the store. It includes private details like productId, name, description, price, and quantity. Moreover, it has a constructor that sets up these details when a new product instance is created. Furthermore, it offers methods like getters and setters to retrieve and update the product's attributes as needed.

InventoryManager Class:

The InventoryManager class oversees the stock of products that are available for sale in the store. It holds a private list named products, which stores the SalableProducts. Upon the creation of an InventoryManager object, the constructor sets up an empty ArrayList of products. The addProduct() method facilitates the addition of a new SalableProduct to the inventory, while removeProduct() removes a specific SalableProduct from the inventory. The getProduct() method retrieves a SalableProduct from the inventory by searching for its name. If the product with the specified name cannot be found, the method returns null.

ShoppingCart Class:

The ShoppingCart class functions as a digital representation of a shopping cart, allowing customers to add and remove items they wish to purchase. It incorporates a private list named items to hold the SalableProducts selected by customers. Upon creating a new ShoppingCart object, the constructor initializes an empty ArrayList of items to start the cart. The addItem() method facilitates the addition of a new SalableProduct to the shopping cart, enabling customers to expand their selections. Conversely, the removeItem() method allows customers to remove a specific SalableProduct from their shopping cart, providing flexibility in their choices. The getTotalPrice() method computes the overall cost of all items currently present in the shopping cart by summing up their individual prices, aiding customers in budgeting their purchases. Lastly, the checkout() method emulates the checkout procedure. Presently, it simply displays a message confirming that the checkout process was successful, though in a real-world application, it would handle payment and order finalization.

StoreFront Class:

The StoreFront class serves as the interface customers interact with at the front of the store. It oversees both inventory management and purchase processes. To manage inventory effectively, it includes a private attribute named `inventoryManager`. Upon the creation of a new StoreFront object, the constructor sets up the `inventoryManager`. The `initializeStore()` method is responsible for setting up the store initially by adding some example products to the inventory. For purchasing items, the `purchaseProduct()` method is employed. This method verifies if the desired product is available in the inventory and if there's enough quantity before reducing it after the purchase. Conversely, if a customer decides to cancel their purchase, the `cancelPurchase()` method is utilized. This method ensures that the product being returned exists in the inventory before increasing the available quantity after the cancellation.

StoreFrontApplication Class:

The StoreFrontApplication class showcases how the front-end operations of the store work. It starts by setting up the store, creating a StoreFront object, and using the initializeStore() method to add some sample products. A ShoppingCart object, known as cart, is then made to simulate shopping activities. Next, two sample products, namely product1 and product2, are created and placed into the shopping cart. After that, the total price of all items in the cart is shown. The application then proceeds to test the purchaseProduct() and cancelPurchase() methods of the StoreFront class to handle product purchases and cancellations. Following any changes, the new total price of the items in the cart is displayed. Lastly, to mimic the checkout process, the checkout() method of the cart is invoked.