We began by crafting UML diagrams for each class, which visually depicted how different parts of a store's system for managing inventory and shopping activities are set up and connected:

SalableProduct Class:

- Represents a product available for sale in the store.
- Contains attributes like productId, name, description, price, and quantity.
- Includes a constructor to initialize product details and methods to access and modify them.

InventoryManager Class**:**

- Oversees the store's inventory of products for sale.
- Maintains a list named "products" to store instances of SalableProduct.
- Provides functions to add, remove, and retrieve products from the inventory.

StoreFront Class:

- Acts as the interface for the store's front end.
- Contains an attribute called "inventoryManager" of InventoryManager type to manage stock operations.
- Implements methods to start the store, purchase products, and cancel orders.

ShoppingCart Class:

- Models a shopping cart for customers to manage their selections.
- Keeps track of SalableProduct instances in a list named "items".
- Provides functions to add or remove items, calculate the total cost, and simulate the checkout process.

The UML diagram illustrates the relationships among these classes, such as:

- StoreFront relies on InventoryManager for inventory tasks.
- ShoppingCart interacts with SalableProduct to adjust items and compute costs.
- StoreFront communicates with SalableProduct for purchasing and canceling orders, utilizing InventoryManager's features.

In essence, the UML diagram acts as a guide for understanding the structure and interactions within the store's management system, aiding developers in effectively building and managing the application.

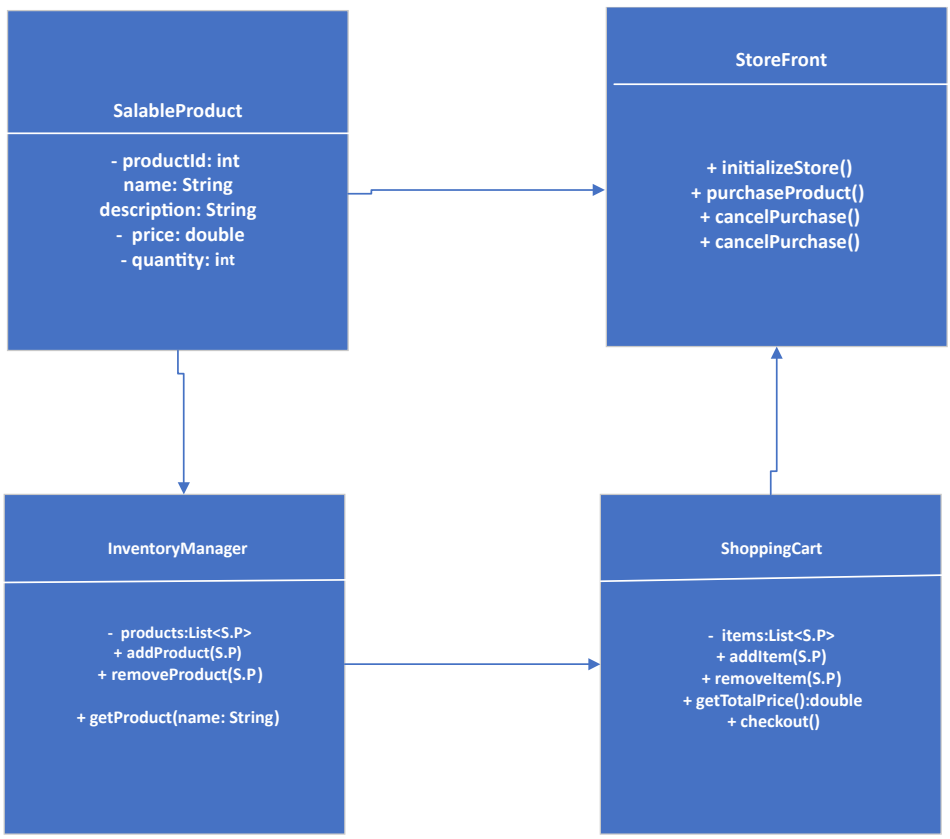
As for the main class:

- It simulates various store actions by creating instances of SalableProduct, InventoryManager, and ShoppingCart.
- Products are added to the inventory, and its contents are displayed.

- Products are added to the shopping cart, and its contents are displayed before removing a product.
- The updated contents of the shopping cart are displayed, and it is confirmed to be empty by emptying it.

In the code:

- The SalableProduct Class represents products available for sale, encapsulating attributes like productId and name.
- InventoryManager Class manages the store's inventory, including adding and removing products.
- StoreFront Class integrates InventoryManager and ShoppingCart components, but its method bodies are placeholders.
- ShoppingCart Class manages the shopping cart's contents yet lacks implementation details for adding and removing products.
- Weapon Class defines attributes and methods for a type of weapon product but lacks the implementation of the compareTo method required by the Comparable interface for proper functionality. Figure 1 below shows the UML Class diagram for the Store Front



Flowchart of User Interaction with the Store Front

This flowchart outlines the steps a user follows when they engage with the Store Front, including any interactions with the Inventory Manager and the Shopping Cart, if needed. It shows how the user moves through the store's interface and performs actions like buying and canceling products. To start, the user initializes the store and receives a friendly welcome message. Then, the Store Front presents a list of actions for the user to choose from. If the user decides to purchase a product, the Store Front displays the available inventory, lets the user pick a product, confirms the purchase, and gives feedback. In case the user wants to cancel a purchase, the Store Front shows the inventory again, allows the user to select the product they wish to cancel, confirms the cancellation, and provides feedback accordingly. Lastly, if the user decides to exit, the program closes.

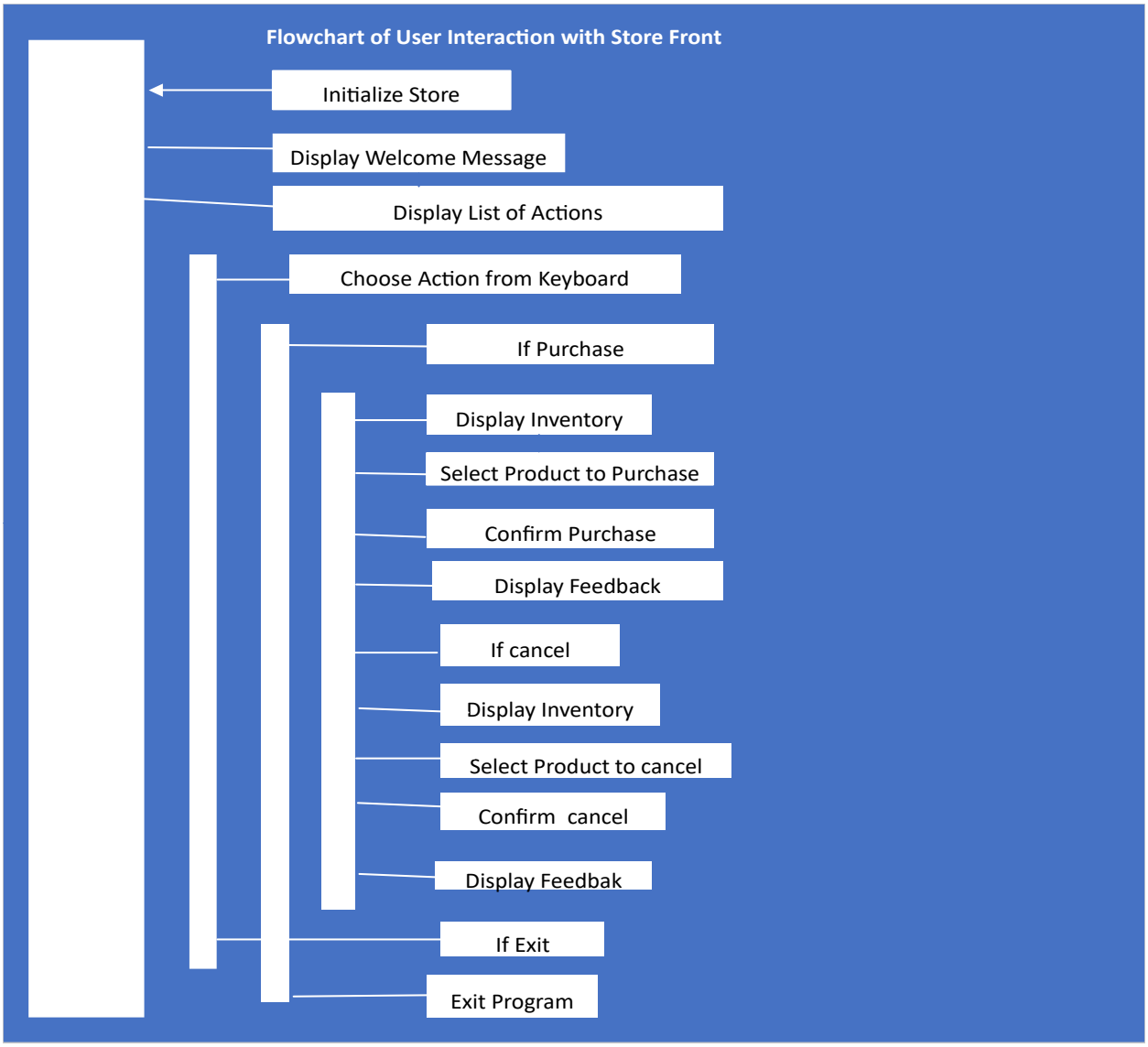


Figure 2 Shows the Flowchart of the User Interaction with a Store Front

Next, we'll proceed to put into action the provided UML class designs for SalableProduct, InventoryManager, ShoppingCart, and StoreFront. This code is equipped with features to set up the store, buy products, and cancel purchases. The Main class is responsible for trying out these functions and showing what's inside the inventory and shopping cart.

Here are the links to the Github and Loom websites

1) CST 239 Milestone 1_Javadoc

https://github.com/Buokwifoy/MY_GCU_COURSES/tree/main/CST%20239%20Mileston%201_JavaDoc

2) CST-239 Milestone Zip File

https://github.com/Buokwifoy/MY_GCU_COURSES/blob/main/CST-239Milestone1.zip

3) Video Link CST 239 Milestone 1

https://www.loom.com/share/6e4eb9b454d64654baebc9db16943a17

Milestone 1 add on

4) **List your computer specs (type of computer, OS, memory, etc)**

Device name	Buonsen
Processor	Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz 2.40 GHz
Installed RAM	6.00 GB (5.89 GB usable)
Device ID	AC1132CB-E0F5-4D0B-9CE2-4DEB53E0C493
Product ID	00325-80000-00000-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

5) **Create 3 test Cases**

Below are three examples of how we could test logging into a GCU website for example:

Test Case 1: Making Sure Logins Work
Test Case ID: TC1
Test Objective: To check if a user can log in successfully using the right details.
Test Steps:
Open the GCU website.
Go to the login page.
Type in your correct username and password.
Click the "Login" button.
Wait for the system to check your details.
Make sure you're taken to your personal dashboard or the homepage.
Expected Result:

You should be logged in successfully.

Your dashboard or the homepage should appear.
You should be able to use all the features of the website.
Test Case 2: Dealing with Wrong Passwords
Test Case ID: TC2
Test Objective: To see if the system shows an error message when you put in the wrong password.

Test Steps:

Open the website such as the GCU website.
Go to the login page.
Enter your correct username.
Put in an incorrect password.
Click the "Login" button.
Wait for the system to check your details.
Check if the system shows an error message for the wrong password.
Expected Result:

The system should show a message saying the password is wrong.
You shouldn't be logged in.
You should stay on the login page.
Test Case 3: Handling Non-existent Users
Test Case ID: TC3
Test Objective: To make sure the system handles accounts that don't exist.

Test Steps:

Open the website.
Go to the login page.
Enter a username that doesn't exist.
Use a valid password.
Click the "Login" button.
Wait for the system to check your details.
Check if the system shows an error message for a non-existent user.
Expected Result:

The system should show a message saying the username doesn't exist.
You shouldn't be logged in.
You should stay on the login page.

6) **List 3 or more Programming conventions that will be used all milestones**

Naming Rules: When naming things like variables, functions, and classes in your code, it's important to be both consistent and descriptive. For example:

Give variables and functions names that clearly tell what they do.

Stick to a style like camelCase or snake_case for naming variables and functions.
When naming classes, start each word with a capital letter (UpperCamelCase).
Formatting and Spacing: How you organize your code on the page makes a big difference in how easy it is to read and work with. For example:

Always use the same number of spaces or tabs for making indentations.
Keep your code style consistent, whether you put braces on the same line or the next line.
Make sure your lines aren't too long so you don't have to scroll sideways to read them.
Adding Comments and Explanations: Good comments and documentation help everyone understand what your code does and how it works. Here are some tips:

Use comments to explain tricky parts of your code or important decisions you made.
Write down what public parts of your code (like functions and classes) do so others know how to use them.
Stick to a single format for comments, like Javadoc or Doxygen, so they look the same throughout your code.
Dealing with Errors and Problems: When things go wrong in your code, it's important to handle them gracefully to keep your program running smoothly. Here's what you should do:

Always handle errors and exceptions properly to prevent your program from crashing or behaving unexpectedly.
Write error messages that make sense so that when something goes wrong, it's easier to figure out what happened.
Follow the best practices for dealing with different kinds of errors, like mistakes in user input, unexpected problems during runtime, or failures in the system.

7) Create Use case diagram

Let's think about a simple online shopping system(amazon.com) as an example for a use case diagram.
Customer: This is someone like you or me, just a regular person using the website. They can do lots of things like looking at products, searching for items, seeing more about a product, putting stuff in their shopping cart, buying things, and looking after their account.

Admin: This is like another user, but with extra powers. They're the ones who can add new products to the website, change details about stuff that's already there, take things off the website, and see what orders people have made.

So, what does all this mean?

User: These are the people who are using the website to do things like buying stuff and keeping their accounts in order. They're the ones doing the main stuff.

Admin: This is someone who helps look after the website, like making sure all the products are up-to-date and seeing what orders have come in.