Project Report: Warehouse Management System

CSCI 430 Project 1 - Stage 1 Implementation

Team Members:

- Tsion Nidaw
- Nathan Nelson
- Morgan Rassatt
- Larson Triston

Project Overview:

This project involves building a basic **Warehouse Management System** that handles clients, products, orders, and wishlists. In **Stage 1**, the team focused on implementing the core features: adding clients, managing products, handling client wishlists, and creating orders. The system allows users to interact with a console-based UI, which can add and view clients, products, orders, and notices. We structured the project using object-oriented principles, with each team member responsible for specific class implementations.

Key Tasks Accomplished:

Team Member	**Role**
Tsion Nidaw	Implemented the WarehouseConsole UI, Notice class, integration testing, and managed the GitHub repository.
Nathan Nelson	Developed Client and ClientList classes, responsible for client management features.
Morgan Rassatt	Developed the Wishlist and Order classes, focusing on wishlist functionality and order creation.
Larson Triston	Developed the Product and Catalog classes, ensuring product management features (add, remove, update) are correctly implemented.

1. Class Design and Responsibilities:

Each class was designed with specific methods based on **sequence diagrams** and **UML diagrams** outlined in the planning phase.

1.1. Client Class

Designed to manage individual clients with attributes like name, address, and phone.
Key Methods:

- getID(): Returns the unique client ID.
- issueProduct(): Allows clients to add products to their wishlist.
- getWishlist(): Returns the client’s wishlist.

Client Class Diagram

Client

Sequence Diagram	Method(s) to add	Description
Add Client	constructor	Parameters are name, address, phone number; this will generate the ID and Wishlist
Issue Products	issueProduct()	Stores a reference to the given Product object inside the Wishlist for the Client; records the transaction
Remove Products	removeProduct()	Removes a reference to the given Product object inside the Wishlist for the Client; records the transaction
Get Transactions	getTransactions()	Returns a list of all the transactions of the Client.
Get Wishlist Items	getWishlist()	Returns a list of all the Products in a Wishlist.
Get Orders	getOrders()	Returns a list of all completed Orders made by the Client.

```
Client
--------------------------------------------------------------
    -   name: String
    -   address: String
    -   phone: String
    -   transactions: List
    -   myWishlist: Wishlist
    -   myOrders: Orders
--------------------------------------------------------------
+Client (name: String, address: String, phone: String): Client
+issueProduct (product: Product): Boolean
+removeProduct (product: Product): Boolean
+getID(): String
+getName(): String
+getAddress(): String
+getPhone(): String
+setID(): String
+setName(): String
+setAddress(): String
+setPhone(): String
+getTransactions(): Iterator
+getWishlist(): Iterator
+getOrders(): Iterator
```

1.2. ClientList Class

Manages a list of clients with the ability to add and search clients.
Key Methods:

- insertClient(): Adds a new client to the list.
- search(): Finds a client by their ID.

ClientList Class Diagram

ClientList

Sequence Diagram	Method(s) to add	Description
Create List	constructor	No parameters; generates the list
Add Clients	insertClient()	Stores the Client and their details to the end of the list.
Get Clients	getClients()	Return a list of all Clients within the system (one per line, and all details displayed).

ClientList
- clients: List
+search (clientID: String): Client +insertClient (client: Client): Boolean +getClients(): Iterator

(See details in [23†ClientList])

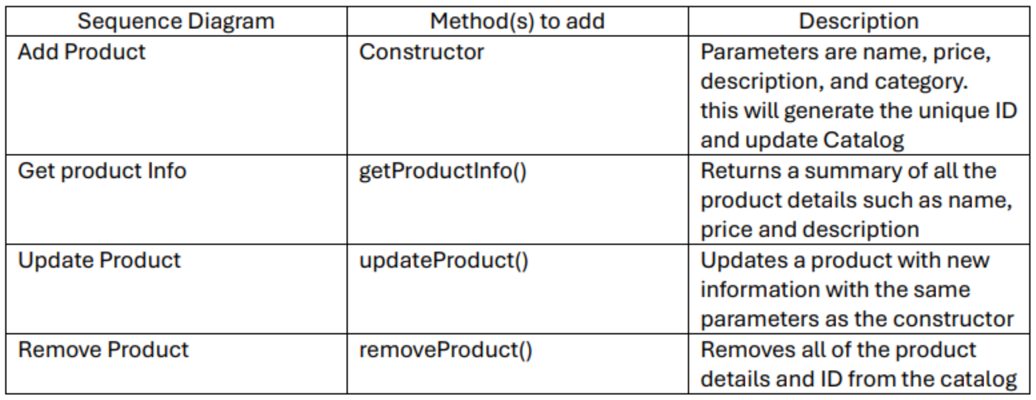
1.3. Product Class

Handles individual product attributes, such as name, price, and description.
Key Methods:

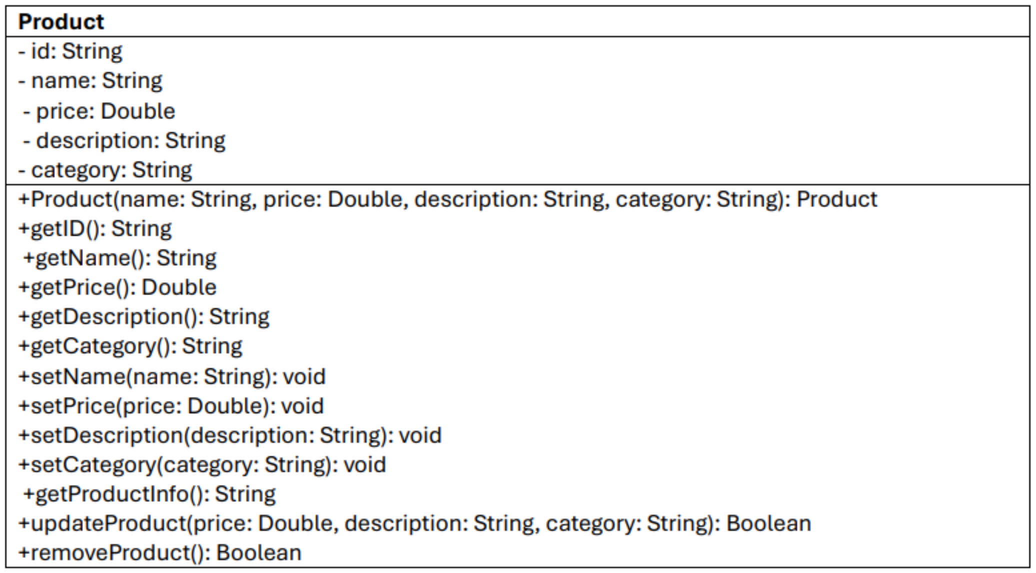
- getProductInfo(): Provides a summary of product details.
- updateProduct(): Allows updating product information.

Product Class Diagram

Product



Sequence Diagram	Method(s) to add	Description
Add Product	Constructor	Parameters are name, price, description, and category. this will generate the unique ID and update Catalog
Get product Info	getProductInfo()	Returns a summary of all the product details such as name, price and description
Update Product	updateProduct()	Updates a product with new information with the same parameters as the constructor
Remove Product	removeProduct()	Removes all of the product details and ID from the catalog



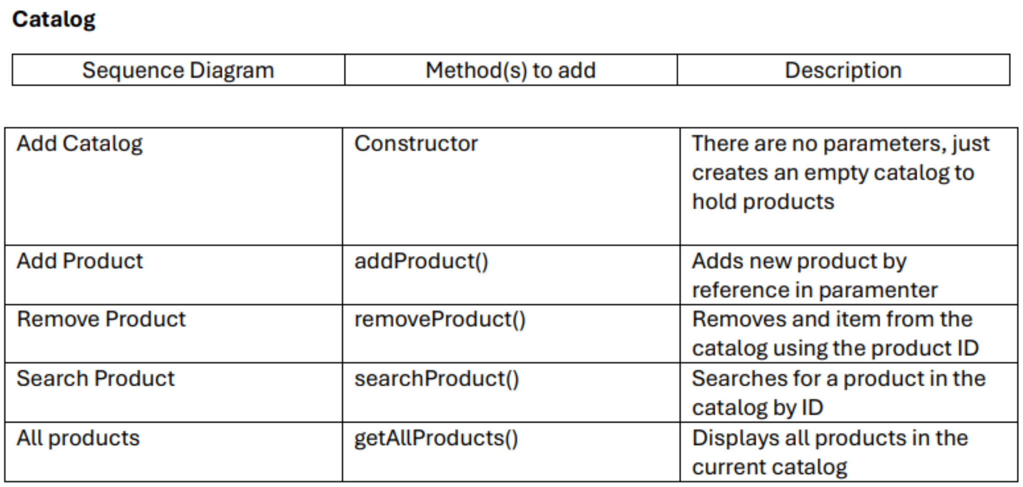
Product
- id: String - name: String - price: Double - description: String - category: String
+Product(name: String, price: Double, description: String, category: String): Product +getID(): String +getName(): String +getPrice(): Double +getDescription(): String +getCategory(): String +setName(name: String): void +setPrice(price: Double): void +setDescription(description: String): void +setCategory(category: String): void +getProductInfo(): String +updateProduct(price: Double, description: String, category: String): Boolean +removeProduct(): Boolean

1.4. Catalog Class

A singleton class responsible for managing all products in the warehouse.
Key Methods:

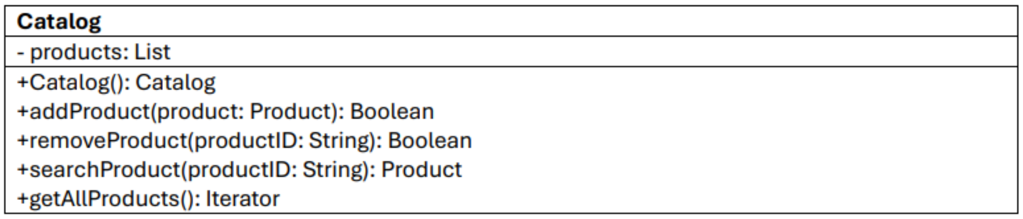
- addProduct(): Adds a product to the catalog.
- removeProduct(): Removes a product by ID.
- searchProduct(): Searches for a product in the catalog by its ID.

Catalog Class Diagram



Catalog

Sequence Diagram	Method(s) to add	Description
Add Catalog	Constructor	There are no parameters, just creates an empty catalog to hold products
Add Product	addProduct()	Adds new product by reference in paramenter
Remove Product	removeProduct()	Removes and item from the catalog using the product ID
Search Product	searchProduct()	Searches for a product in the catalog by ID
All products	getAllProducts()	Displays all products in the current catalog



Catalog

- products: List

+Catalog(): Catalog
+addProduct(product: Product): Boolean
+removeProduct(productID: String): Boolean
+searchProduct(productID: String): Product
+getAllProducts(): Iterator

1.5. Order and Wishlist Classes

The **Order** class is responsible for managing customer orders, including the addition, removal, and processing of items. The **Wishlist** class allows clients to add products they are interested in purchasing later.
Key Methods:

- addItem(): Adds a product to an order or wishlist.
- processOrder(): Finalizes an order.

Whishlist class diagram

Wishlist

Sequence Diagram	Method(s) to add	Description
Constructor	Constructor()	Initialize a wishlist for a specific client.
Add Product	AddProduct()	Verifies product ID and checks to see if the product can be added to wishlist. If it can add it and it already isn't in wishlist, then return true and add it. Otherwise return false.
Get Wishlist Info	GetWishListInfo()	Returns a string containing all the products in the wishlist. It includes name, price, description for each.
Remove Product	RemoveProduct()	Uses product ID to remove it from the wishlist. Returns true of it was removed and false if the product isn't in wishlist.

Wishlist
-clientID: String -products: List
+Wishlist(clientID: String) +getClientID(): String +getProducs(): List +addProduct(productID: String): Boolean +getProductInfo(productID: String): String +updateProduct(productID: String, newProduct: Product): Boolean +removeProduct(productID: String): Boolean

Order Class Diagram

Order

Sequence diagram	Methods to add	Description
Constructor	Constructor	Initializes a new order with a unique order ID, associated client ID, and the date it was placed.
Add Item	addItem()	Add a product and the quantity to the order, check to see if there is stock.
Update Item	updateItem()	Checks and updates the quantity of an item in the order.
Remove Item	removeItem()	Removes a product from the order.
Get Order Details	getOrderDetails()	Gives the user a summary of the order that includes all the items, the quantities, and the costs.
Process Order	processOrder()	Processes payment and adjusts the stock.

Order
id: String clientID: String date: Current date items: List
+order(clientID: String, date: Current Date) +getID(): String +getClientID(): String +getDate(): Current Date +addItem(product: Product, quantity: int): Boolean +updateItem(productID: String, quantity: int): Boolean +removeItem(productID: String): Boolean +getOrderDetails(): String +processOrder(): Boolean +orderItem(product: Product, quantity: Int) +getProduct(): product +getQuantity(): Int +setQuantity(quantity: Int): +getProductInfo(): String

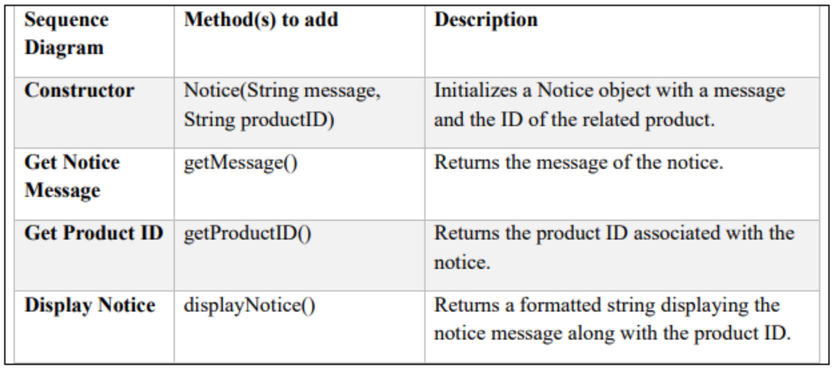
1.6. Notice Class

Handles the notifications for actions related to products, such as when a product is added.
Key Methods:

- displayNotice(): Displays formatted notice information.

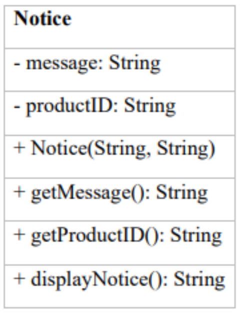
Notice Class Diagram

Notice Class



Sequence Diagram	Method(s) to add	Description
Constructor	Notice(String message, String productID)	Initializes a Notice object with a message and the ID of the related product.
Get Notice Message	getMessage()	Returns the message of the notice.
Get Product ID	getProductID()	Returns the product ID associated with the notice.
Display Notice	displayNotice()	Returns a formatted string displaying the notice message along with the product ID.

Class Diagram

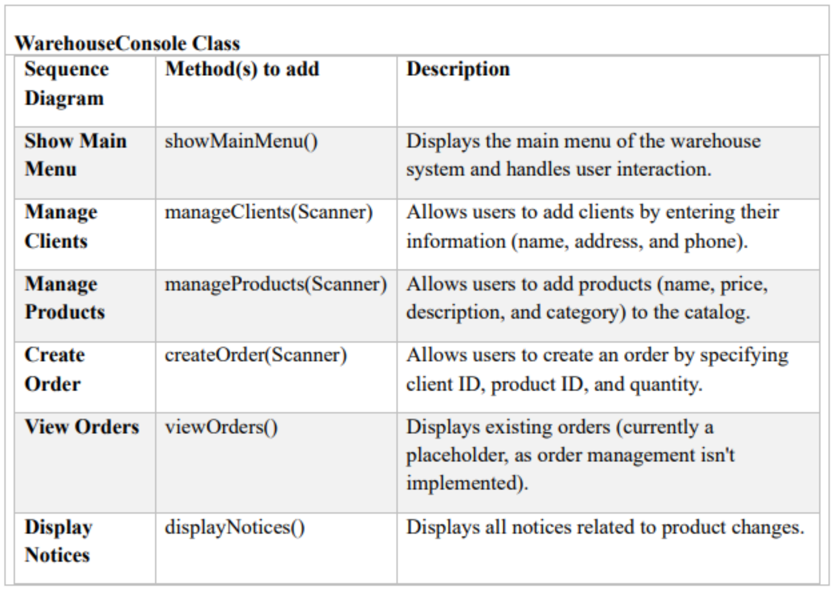


Notice
- message: String
- productID: String
+ Notice(String, String)
+ getMessage(): String
+ getProductID(): String
+ displayNotice(): String

2. System Implementation:

The **WarehouseConsole** serves as the main user interface for the system. It allows users to:

- Add clients and products.
- View clients and products.
- Create orders and wishlists.
- Display notices about product-related actions.



WarehouseConsole Class

Sequence Diagram	Method(s) to add	Description
Show Main Menu	showMainMenu()	Displays the main menu of the warehouse system and handles user interaction.
Manage Clients	manageClients(Scanner)	Allows users to add clients by entering their information (name, address, and phone).
Manage Products	manageProducts(Scanner)	Allows users to add products (name, price, description, and category) to the catalog.
Create Order	createOrder(Scanner)	Allows users to create an order by specifying client ID, product ID, and quantity.
View Orders	viewOrders()	Displays existing orders (currently a placeholder, as order management isn't implemented).
Display Notices	displayNotices()	Displays all notices related to product changes.

Main Console Menu Example:

Warehouse Management System:

1. Manage Clients

2. Manage Products

3. Create Order

4. View Products

5. View Orders

6. Exit

The **Order Creation** process involves selecting a client, adding a product, and specifying a quantity. The system generates unique IDs for each client, product, and order.

Example Output:

```
Clients:

ID: ab08f835-ef56-4dd5-8abd-2a3507372a86, Name: Tsion

Enter Client ID: ab08f835-ef56-4dd5-8abd-2a3507372a86

Enter Product ID: 28e836b7-aac7-4be1-9a7d-05a1ad9f19d1

Enter Quantity: 2

Order created successfully with ID: 2e7a276a-1b5d-4983-8398-d238aeb8f1fd
```

3. Challenges and Solutions:

Challenge	**Solution**
Code Integration	Some integration issues arose when combining individual classes. The team debugged errors through collaborative efforts.
GitHub Version Control	Each member committed their code to a shared GitHub repository after testing. Tsion Nidaw managed the repository.
Class Responsibilities Overlap	Clear communication was maintained to avoid duplication of efforts during class implementation.

---

4. Testing:

Each team member performed unit tests for their classes, followed by full **integration testing**. For instance:

- **Product and Catalog Testing:** Triston ensured the Catalog correctly manages products, allowing adding, removing, and searching of products.
- **Client and ClientList Testing:** Nathan tested the ability to add clients, search by client ID, and iterate through the list of clients.
- **Order and Wishlist Testing:** Morgan verified the functionality of orders and wishlists, ensuring products can be added and updated as needed.
- **UI and Integration Testing:** Tsion was responsible for Notice class, the final system integration and testing the UI. This involved ensuring that the **WarehouseConsole** correctly interacted with the **ClientList** and **Catalog** components. Tsion managed the GitHub repository, ran integration tests, and debugged any errors in the unified codebase.

5. Next Steps:

In the next stage, we will expand the system by implementing full order processing and adding inventory management features. This will involve handling stock, processing payments, and refining the system's UI.

Conclusion:

The Stage 1 implementation of the Warehouse Management System was successful. The core functionality is complete, and the system has passed all basic tests without critical issues. The project has set a solid foundation for further enhancements in the next stages.

6. Group Work Distribution:

Team Member	**Contribution**
Larson Triston	25% (Product & Catalog classes, testing)
Morgan Rassatt	25% (Order & Wishlist classes, testing)
Nathan Nelson	25% (Client & ClientList classes, integration test)
Tsion Nidaw	25% (UI, Notice class, GitHub management, testing)