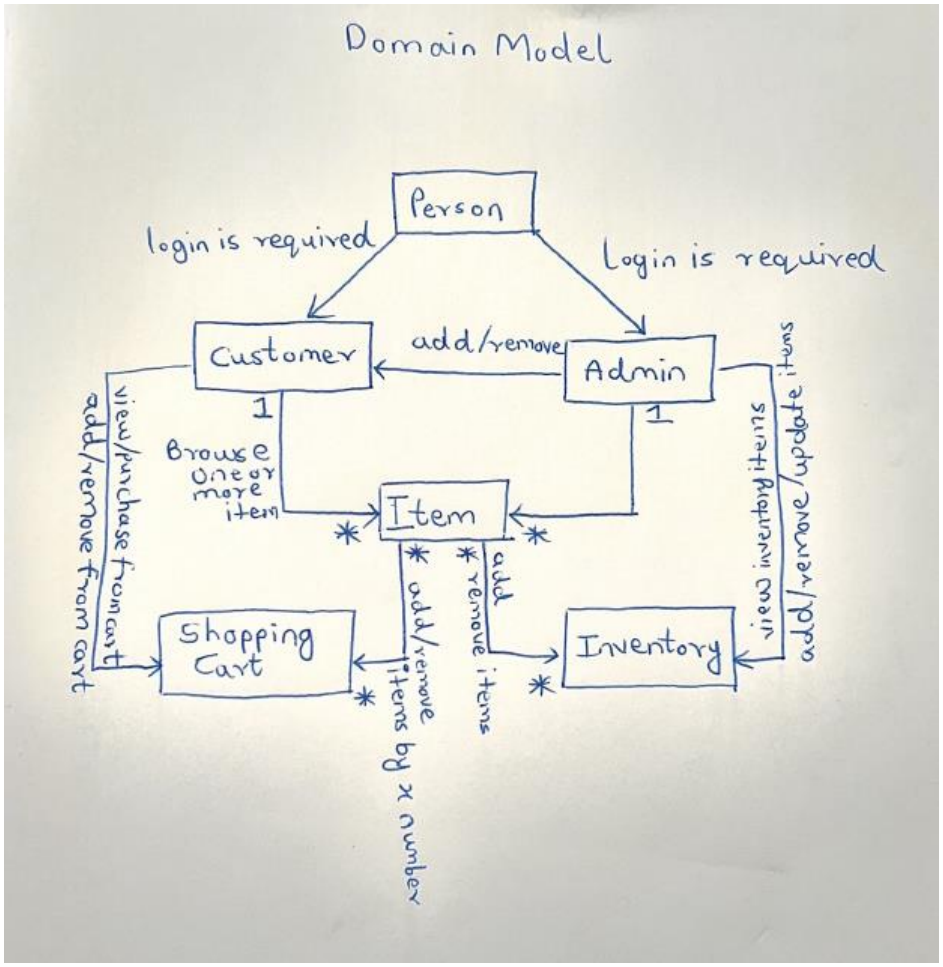
OOPS – Report Assignment 3

-AARYA DESHMUKH

Purpose and Overview:

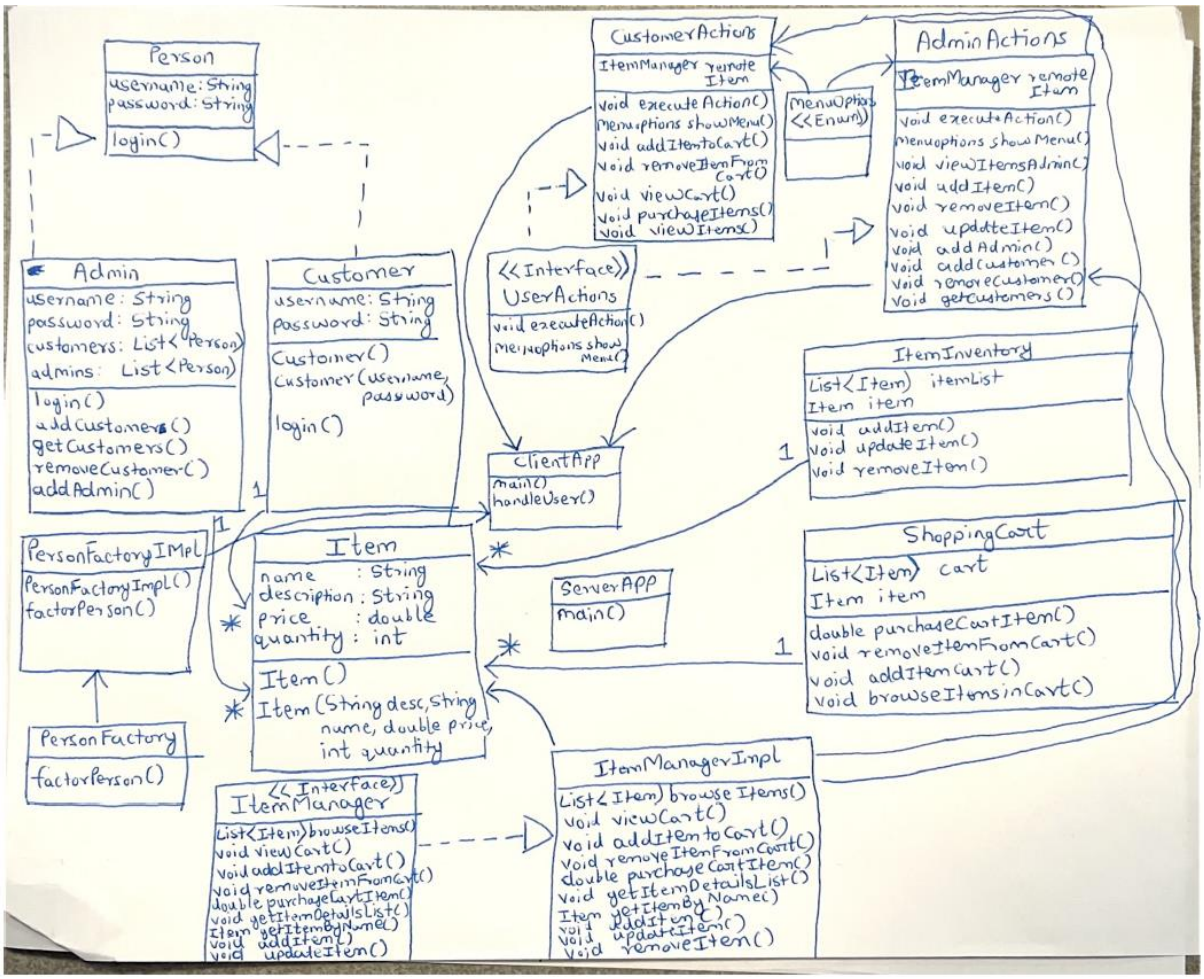
In this assignment, I have enhanced my domain model by integrating Application Control Patterns, focusing particularly on implementing robust control structures through the Model-View-Controller architecture previously developed in assignment 2. The core of Assignment 3 involves the adoption of the **Front Controller** and **Authorization** patterns to establish secure and efficient login mechanisms for both administrators and customers. I have also applied the **Command, Factory/Abstract Factory**, and **Template** design patterns to strengthen the functionality and security of my online store's operations, which include item browsing, updating, adding, and purchasing, as well as user registration and login. Constructed using JavaRMI, the system provides tailored interfaces for customers and administrators, ensuring clear separation of concerns and streamlined management of user interactions. This project aims to deepen my understanding of how to translate customer requirements into a well-organized domain model with clearly defined class responsibilities and operations.

Domain Diagram:



The domain model diagram is a visual representation of the classes that exist within a specific domain or subject area which in this case is the Online store. It is a key component in object-oriented analysis and design as it helps to map out the structure and relationships between the various classes within a system. It also includes multiplicity and how various classes are associated with each other.

Class Diagram:



Implementation of different Application Control Patterns:

- **Implementing MVC (Model, View, Controller) architecture:**

The above class diagram represents the various classes, interfaces and enums etc and the relationships between them. All these classes are divided into 3 packages namely: server, client and common.

Server: The server encapsulates all the business logic of the online store and provides a way for the clients to remotely access methods. The server makes use of RMI registry to bring about this remote communication with the client.

In our case, the classes:

- ➔ ServerApp (has the main method and contains logic to bind the ItemManager and PersonFactory object to a registry at a configured port, which in my case is 2302)
- ➔ PersonFactoryImpl which implements the PersonFactory interface in the common package is used to create a person so as to hide the details of how the person is created from the client
- ➔ Same as above, an ItemManagerImpl which implements the ItemManager interface again in the common package is used.

Client: The client talks with the remote server and gets access to a list of items for them to go through all the items, add them to cart, purchase from cart etc.

I have further segregated the Customer and Admin action into different classes which implement the interface UserActions. The different actions are stored in an Enum called MenuOptions.

Classes used:

- ➔ UserActions
- ➔ CustomerActions
- ➔ AdminActions
- ➔ MenuOptions
- ➔ ClientApp

Common: The classes common to both server and client are in this package:

- ➔ PersonFactory
- ➔ Person
- ➔ Item
- ➔ ItemManager
- ➔ Customer
- ➔ Admin
- ➔ ItemInventory
- ➔ ShoppingCart

- ➢ **Implementing Authorization Pattern:**

In this assignment, I have modified the previous source code by completing the authorization (login and register) functionalities of the system. I have implemented the Authorization pattern to enhance security for the online store to ensure that only authenticated users can access appropriate resources. I have established specific roles and permissions:

Admins are granted full access to the system, allowing them to manage products and customer accounts.

On the other hand, **customers** are limited to browsing products, adding items to their carts, and making purchases, ensuring that they do not get access to the functionalities provided to admins.

- **Implementing Command Pattern:**

The Command Pattern turns a request into an object, which lets clients manage operations, requests, and queues more flexibly. This approach helps keep the part of the system that makes requests separate from the part that processes them. For managing a shopping cart in a client-side application:

An **ItemCommand** interface defines all the operations needed for managing items. Specific commands like "**AddItemInventory**", "**UpdateItemInventory**" and "**RemoveItemInventory**" are created following this interface. These are used by an "**ItemManager**" that takes care of all the inventory activities, ensuring that the system is organized and that different parts of the system do not depend directly on each other.

- **Implementing Front Controller Pattern:**

The **ClientApp** class Java code implements a simplified version of the Front Controller pattern by centralizing the handling of initial user requests related to login, registration, and subsequent action delegation based on user roles (either "Customer" or "Admin"). It manages all user inputs and routes these inputs to the appropriate processes or actions. All requests (user actions like login, registration, and subsequent tasks) go through the main method first. This unified approach is characteristic of the Front Controller pattern, which aims to provide a single point of control for managing request handling. After processing initial user inputs for role selection and authentication, the application delegates further actions to either **CustomerActions** or **AdminActions** via **handleUser** method. By separating the initial interaction (role selection, login, registration) from the specific actions (executeAction within CustomerActions and AdminActions), the program loosely applies the principle of separating control logic from business logic, which is central to the Front Controller pattern.

- **Implementing Abstract Factory Pattern:**

The Factory Pattern is a creational design pattern that abstracts the instantiation process by allowing objects to be created without specifying their concrete classes. It delegates the creation of objects to a factory class that determines the type of object to produce based on provided input. In our implementation, we utilized this pattern on the server side through the **ServerApp** class by developing a remote object that creates the Person and Item objects through a Factory class **PersonFactory**. Clients interact with this remote object, invoking its method to generate specific types of objects as needed.

Use Cases implemented are:

For Customers:

- Viewing Cart Items
- Viewing Inventory Items
- Add items to cart

- Remove items from cart
- Purchase Cart Items.

For Admins:

- View Inventory Items
- Add item
- Remove item
- Update item
- Add customers
- Add other admins
- Remove customer
- Get the customers list.

Results

After a JAVA RMI connection is established between the ServerApp and the ClientApp, the client is presented with an interface from where the desired options can be selected, almost as if shopping from an online store!

For implementing above, following are the client machine screenshots where a client logs into the Online Store and carries out the various use cases:

----------CUSTOMER----------

Login:

```
aardesh@in-csci-rrpc01:~/Assignment_3$ make run_client
javac common/*.java
javac server/*.java
javac client/*.java
java -cp . client.ClientApp
Choose a role: 1. Customer, 2. Admin, 3. Exit
1
Press 1 for Login and 2 for Register
1
Enter your Username:
cust1
Enter your Password:
cust1@00
You are logged in!
```

Register:

```
aardesh@in-csci-rrpc01:~/Assignment_3$ make run_client
javac common/*.java
javac server/*.java
javac client/*.java
java -cp . client.ClientApp
Choose a role: 1. Customer, 2. Admin, 3. Exit
1
Press 1 for Login and 2 for Register
2
Enter your Username:
cust6
Enter your Password:
cust6@00
You are successfully registered!
```

View Items in Inventory

Add Item to cart

```
Customer Menu:
1. Add Item to Cart
2. Remove Item from Cart
3. View Cart Items
4. Purchase Items from Cart
5. View Items in Inventory
6. Logout
5
Items are:
Name: Notebook   | Description: Stationery          | Price: $3.50 | Qty: 10
Name: Coffee Mug | Description: Kithenware          | Price: $6.00 | Qty: 20
Name: Yoga Mat   | Description: Fitness Accessory   | Price: $50.00 | Qty: 25
Customer Menu:
1. Add Item to Cart
2. Remove Item from Cart
3. View Cart Items
4. Purchase Items from Cart
5. View Items in Inventory
6. Logout
1
The list of available items is:
Name: Notebook   | Description: Stationery          | Price: $3.50 | Qty: 10
Name: Coffee Mug | Description: Kithenware          | Price: $6.00 | Qty: 20
Name: Yoga Mat   | Description: Fitness Accessory   | Price: $50.00 | Qty: 25
Enter item name to add in cart or type STOP to finish:
Notebook
Notebook is added to your cart.
Coffee Mug
Coffee Mug is added to your cart.
STOP
The total cost is: $9.5
```

Remove Item from Cart

View Cart Items

Purchase Items from Cart

```
Customer Menu:
1. Add Item to Cart
2. Remove Item from Cart
3. View Cart Items
4. Purchase Items from Cart
5. View Items in Inventory
6. Logout
2
Enter the name of the item you want to remove or type STOP to finish:
Item is not found in cart.
Coffee Mug
Item is removed from cart.
STOP
The total cost is: $3.5
Customer Menu:
1. Add Item to Cart
2. Remove Item from Cart
3. View Cart Items
4. Purchase Items from Cart
5. View Items in Inventory
6. Logout
3
Items in your cart are:
Notebook - Stationery - 3.5
Customer Menu:
1. Add Item to Cart
2. Remove Item from Cart
3. View Cart Items
4. Purchase Items from Cart
5. View Items in Inventory
6. Logout
4
The total cost is: $3.5
```

Logout

```
Customer Menu:
1. Add Item to Cart
2. Remove Item from Cart
3. View Cart Items
4. Purchase Items from Cart
5. View Items in Inventory
6. Logout
6
You are logged out!
aardesh@in-csci-rrpc01:~/Assignment_3$
```

----------**ADMIN**----------

Login

View Items

```
aardesh@in-csci-rrpc01:~/Assignment_3$ make run_client
javac common/*.java
javac server/*.java
javac client/*.java
java -cp . client.ClientApp
Choose a role: 1. Customer, 2. Admin, 3. Exit
2
Press 1 for Login and 2 for Register
1
Enter your Username:
admin1
Enter your Password:
admin1@00
You are logged in!
Admin Menu:
1. View Items
2. Add Item
3. Remove Item
4. Update Item
5. Add Admin
6. Add Customer
7. Remove Customer
8. Logout
1
Items are:
Name: Notebook   | Description: Stationery          | Price: $3.50 | Qty: 10
Name: Coffee Mug | Description: Kithenware          | Price: $6.00 | Qty: 20
Name: Yoga Mat   | Description: Fitness Accessory   | Price: $50.00 | Qty: 25
```

Add Item

```
Admin Menu:
1. View Items
2. Add Item
3. Remove Item
4. Update Item
5. Add Admin
6. Add Customer
7. Remove Customer
8. Logout
2
Enter the name of item to be added:
Stapler
Enter the Description:
Stationery
Enter the price:
10
Enter the quantity:
20
The new item is added
List of products:
Name: Notebook   | Description: Stationery          | Price: $3.50 | Qty: 10
Name: Coffee Mug | Description: Kithenware          | Price: $6.00 | Qty: 20
Name: Yoga Mat   | Description: Fitness Accessory   | Price: $50.00 | Qty: 25
Name: Stapler    | Description: Stationery          | Price: $10.00 | Qty: 20
Admin Menu:
```

Remove Item

```
Items are:
Name: Notebook   | Description: Stationery          | Price: $3.50 | Qty: 10
Name: Coffee Mug | Description: Kithenware          | Price: $6.00 | Qty: 20
Name: Yoga Mat   | Description: Fitness Accessory   | Price: $50.00 | Qty: 25
Admin Menu:
1. View Items
2. Add Item
3. Remove Item
4. Update Item
5. Add Admin
6. Add Customer
7. Remove Customer
8. Logout
3
Enter the item name to be removed from inventory:
Notebook
Name: Coffee Mug | Description: Kithenware          | Price: $6.00 | Qty: 20
Name: Yoga Mat   | Description: Fitness Accessory   | Price: $50.00 | Qty: 25
The item is successfully removed!
```

Update Item

```
Admin Menu:
1. View Items
2. Add Item
3. Remove Item
4. Update Item
5. Add Admin
6. Add Customer
7. Remove Customer
8. Logout
4
Enter the item to be updated in the inventory:
Yoga Mat
Enter the new price:
30
Enter the new name:
Striped Yoga Mat
Enter the new description:
Fitness
Enter the new quantity:
30
The updated item details are:
Item is successfully updated!
```

Add Admin

```
Admin Menu:
1. View Items
2. Add Item
3. Remove Item
4. Update Item
5. Add Admin
6. Add Customer
7. Remove Customer
8. Logout
5
Enter username for new admin:
admin6
Enter the password for new admin:
admin6@00
The new admin is successfully added!
```

Add Customer

```
Admin Menu:
1. View Items
2. Add Item
3. Remove Item
4. Update Item
5. Add Admin
6. Add Customer
7. Remove Customer
8. Logout
6
Enter username for new customer:
cust6
Enter the password for new customer:
cust6@00
The new customer is successfully added!
```

Remove Customer

```
Admin Menu:
1. View Items
2. Add Item
3. Remove Item
4. Update Item
5. Add Admin
6. Add Customer
7. Remove Customer
8. Logout
7
Enter customer username you want to remove:
cust1
The customer cust1 is successfully removed!
```

Logout

```
Admin Menu:
1. View Items
2. Add Item
3. Remove Item
4. Update Item
5. Add Admin
6. Add Customer
7. Remove Customer
8. Logout
8
You are logged out!
aardesh@in-csci-rrpc01:~/Assignment_3$
```