Object Oriented Design and Programming
Assignment 2

- **Intuition**

User Class :
- There are two kinds of users : Customer and Authenticator.
- Since users have some common attributes like firstName, LastName and other login details, we define a common class for them titled “User”.
- **Customer** and **Authenticator** classes inherit from **User** and define their own attributes and methods based on their business requirements.

Customer Class :
- Inherits from User
- Customers should have access to a Cart wherein he/she can store can add items.
- Customers should be able to add or remove items from the Cart.
- Each Customer should also have a payment history about the persons previous purchases.
- If items are added into cart and they are purchased, the Cart should be empty and purchase_history has to be updated.
- If no purchase is made, then items have to remain in the customer's cart unless he or she removes them.
-

Administrator Class :
- Inherits from User
- Authenticator should be able to update the details of each item.
- He/ She should also be able to add other authenticators. So, he/She should have an option to create new users.

Item Class :
- Each item should have **type, description** and **price**.

- Along with above attributes, each item should have a **“Count”** attribute which decreases if a customer adds an item to cart. If an item is removed from Cart, the Count attribute should increase.

Cart Class :

- Each Customer has his/her own cart.
- The Cart class should contain a list of items a person added.
- If a person adds or removes items, the Cart should be updated.
- Changes to cart should also reflect changes to item.

Purchase Class :

- Each customer should be able to purchase items in his cart.
- Once a purchase is made, items have to be removed from the customer's cart.
- Each purchase should also be recorded in the purchase_history of the customer.

Store Class :

- The `Store` class acts as a central entity to coordinate interactions between users, items, purchases, and authenticators within the system.

Relationships between Classes :

- Customer, Authenticator inherit from User
- Each Customer has one Cart
- Each Cart can have multiple Items
- Each Payment corresponds to One Cart belonging to One Customer
- Each Customer can add/remove multiple items to Cart
- Each Authenticator can make changes to multiple items
- Each Authenticator can add multiple other authenticators

—---
—---

Implementation :

- **Use Cases**

  Actors :

  - Customer -
    1. He can either login if the user already exists (verified by username) or register if the username does not exist.
    2. Once logged in, he can add items to his cart (provided count is not less than 1), remove items from his cart, make a purchase (buys all items in his cart)

  - Administrator -
    1. An administrator is defined initially and he can login.
    2. A new administrator can be added using the register option, it can only be done using an existing administrator.
    3. The administrator can also make changes to items in the store.

  Use Cases :

  User :

  1. Can login/ register as a customer.
  2. Can login as administrator.
  3. An administrator can add other administrators.

  Item :

  1. Can be Added to Store - Done by Administrator.
  2. Can be Added to Cart - Done by Customer.
  3. Can be removed from Store - Done by Administrator if seeks to remove.
  4. Can be Removed from Cart - Done by Customer.
  5. Edit Item - Can be done by Administrator to alter details of item.
  6. Purchase Item - Can be done by Customer, through purchasing **cart**.
  7. Browse items - Can be done by both customer and administrator.

  Cart :

  All the following use-cases for carts are done by customers.

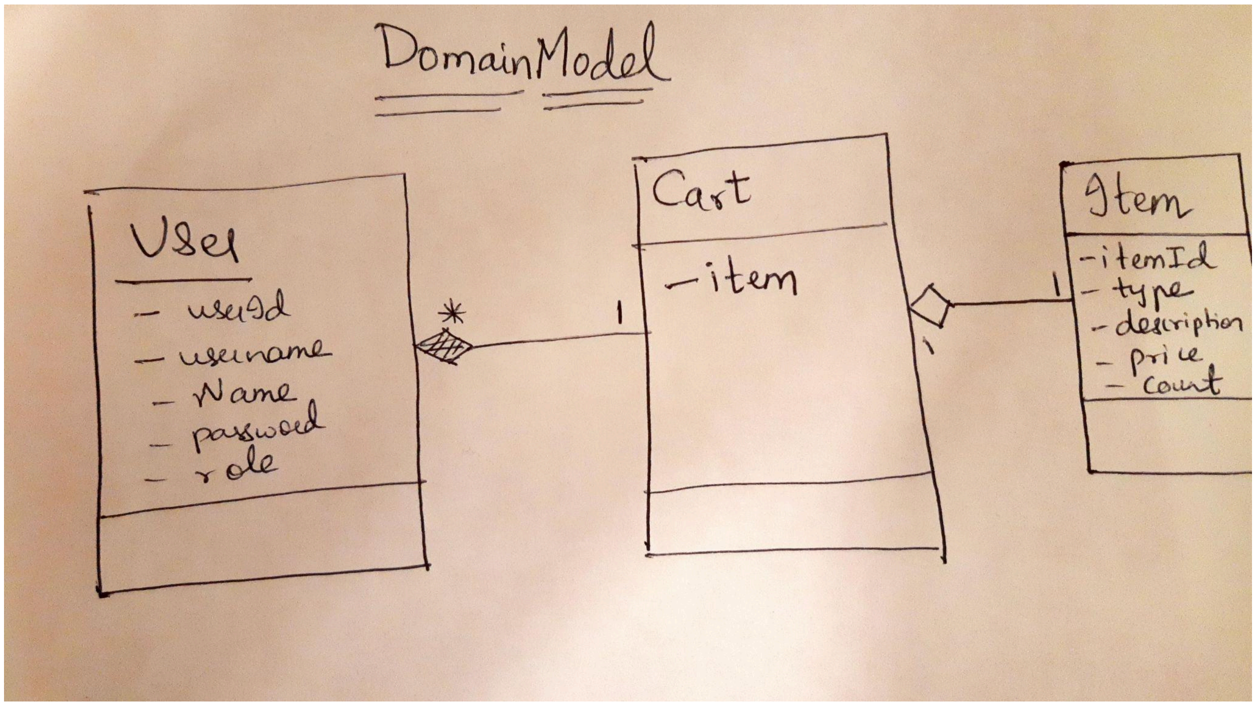
1. Add Items to Cart
2. Remove Items from Cart
3. Purchase Cart - Once a cart is purchased, remove all items from the cart and record the purchase.

Purchase History :

1. A customer can view all his previous purchases.

—---
—---

- **Domain Model**



—---
—---

- **Class Diagrams**

1. **User :**

Attributes :
1. UserID
2. Firstname
3. Lastname
4. Username
5. Password
6. Role

Methods :
1. Getusername
2. Getpassword
3. getrole

Relationships :
1. it's inherited by Customer and Administrator.

—---

2. **Customer :**

Attributes :
1. cart: Cart
2. purchaseHistory: List<Purchase>

Methods :
1. getCart()
2. setCart()
3. getPurchaseHistory()
4. setPurchaseHistory()
5. addItemtoCart()
6. removeItemFromCart()
7. makePurchase()

Relationships :

1. Inherits from User (inheritance)
2. Has-one relationship with Cart (**Composition**)
3. Has one-many relationship with Purchase (**Composition**)

—---

3. **Administrator :**

Attributes :

1. —-

Methods :

1. updateItemDetails(Item item)
2. addAuthenticator(Authenticator authenticator)

Relationships :

1. Inherits from User (inheritance)

—---

4. **Cart :**

Attributes :

1. items: List<Item>

Methods :

1. addItem(Item item)
2. removeItem(Item item)
3. getitems()

Relationships :

1. Has-one relationship with Customer
2. Has one-many relationship with Item

—---

5. **Item :**

Attributes :

1. ItemID
2. Type
3. Description
4. Price
5. Count

Methods :

1. getType()
2. getItemID()
3. setType()
4. getDescription()
5. setDescription()
6. getPrice()
7. Increase_count()
8. get_count()
9. setCount()

Relationships :

1. Many to One relationship with Cart
2. Has Many to One Relationship with Store

—--

6. **Purchase :**

Attributes :

1. Purchased_Items

Methods :

1. Purchased bill

Relationships :

1. Many to One Relationship with Customer

—---

7. **Store :**

Attributes :
1. Customers
2. Items
3. Administrators

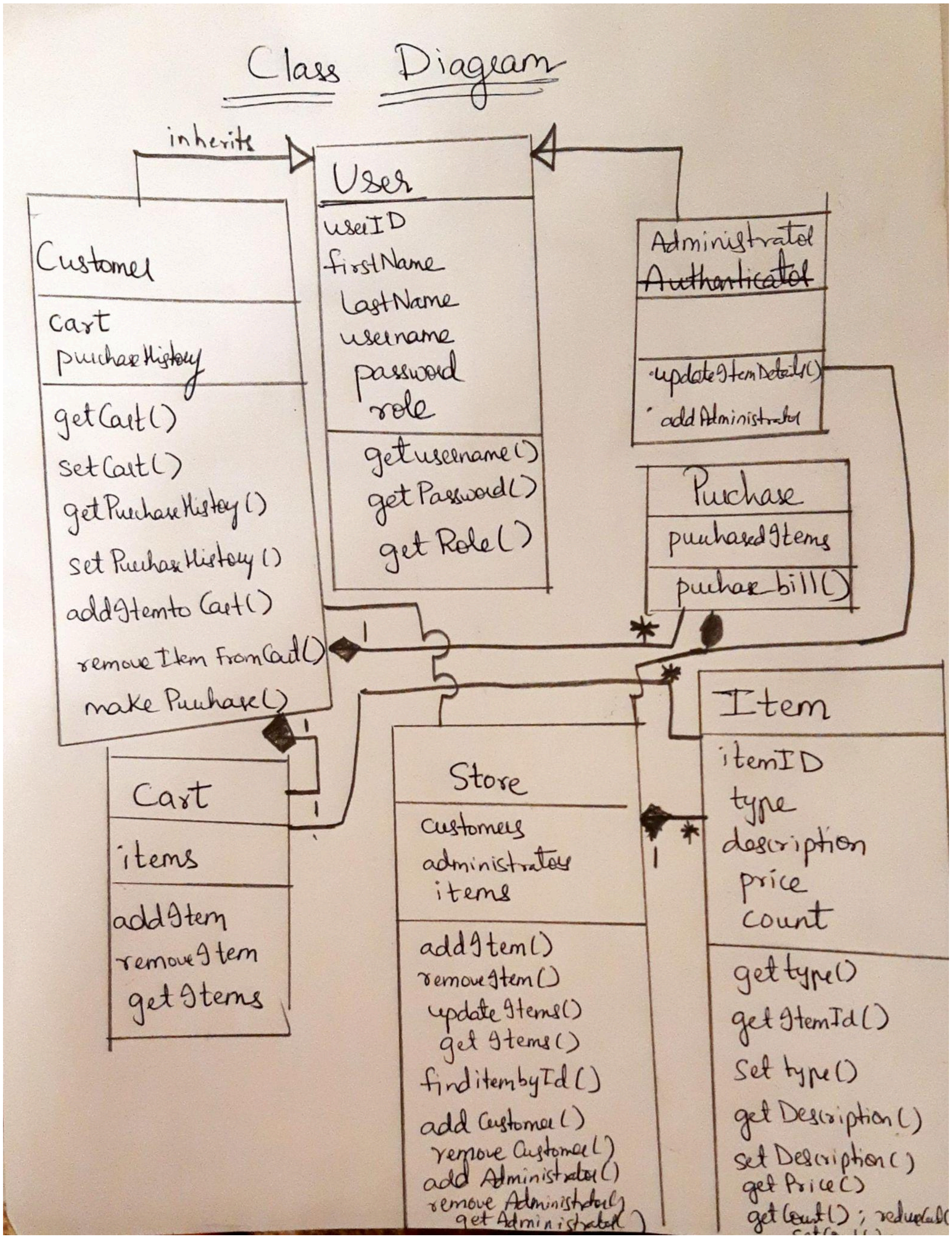
Methods :

Admin Management :
1. addAdministrator (administrator)
2. removeAdministrator()
3. getAdministrators()
4. addCustomer()
5. removeCustomer()

Item Management :

6. addItem()
7. removeItem()
8. updateItem()
9. getItem()
10. findItemByID()

Relationships :
1. `Customer`: A one-to-many relationship, where one `Store` can have many `Customer` objects.
2. `Administrator`: A one-to-many relationship, where one `Store` can have many `Administrator` objects.
3. `Item`: A one-to-many relationship, where one `Store` can have many `Item` objects.


Class Diagram
inherits
User
userID
firstName
LastName
username
password
role
getusername()
getPassword()
get Role()
Customer
cart
purchaseHistory
getCart()
setCart()
getPurchaseHistory()
set PurchaseHistory()
addItemto Cart()
remove Item FromCart()
make Purchase()
Administrator
~~Authenticator~~
·updateItemDetails()
· add Administrator
Purchase
purchasedItems
purchase_bill()
1
*
*
1
1
Cart
items
addItem
removeItem
get Items
Store
customers
administrators
items
addItem()
removeItem()
update Items()
get Items()
finditembyId()
add Customer()
remove Customer()
add Administrator()
remove Administrator()
get Administrator()
1
*
Item
itemID
type
description
price
count
get type()
get ItemId()
Set type()
get Description()
set Description()
get Price()
get Count() ; reduceCo

Interfaces :

1. AdministratorManagement
    - addAdministrator
    - getAdministrator
    - getAdministrators
    - removeAdministrator
2. ItemManagement

- addItem
- removeItem
- getItems
- findItembyId

- **Server folder implements the methods from the client and AdministartorManagementImpl contains methods related to administrators and ItemManagement contains methods related to Items.**

- **So their respective interfaces are contained in the above interfaces.**

—---

Implementation and Code :

1. Using singleton design pattern
    - We have a storage class which serves as our main class. There are many classes which operate on the attributes of storage class. So it is important to operate on the same instance.
    - To maintain a single instance of "Store" class across all implementations, we use the **Singleton design pattern**.
    - The Singleton design pattern is used when you want to ensure that only a single instance of a class exists throughout your application. It provides a globally accessible point of access to that instance.

—---

Design Patterns Used :

1. Singleton design pattern

—---