Group 3 Nozama Milestone 3:Final Report

Eyan Eubanks, Jamar Andrade, Jordan Diaz

Project Title: Nozama Shopping App
Java Swing Application

Specific Use cases for Option #1: The Shopping Cart

Use case - User Logs in:

1. User enters Username and Password into the system.
2. System responds by Loading Customer Dashboard.

Variant #1 - User fails to log-in:

1. User enters a username or password that does not exist.
2. System gives an “UserName and Password is Invalid” and prompts the user to enter credentials again.

Variant #2 - User logs in with a Seller Account:

1. User enters Username and Password into the system.
2. System responds by Loading Seller Dashboard.

Use case - Customer Adds Items to Shopping Cart:

1. System loads the Customer Dashboard.
2. Customer highlights a product from the list of available products and hits “add to cart”.
3. System Prompts User of what items have been added to their cart.
4. User hits “OK” to confirm that the items were added.
5. System updates cart.

Variant #1 - Customer Adds item but no inventory:

1. Customer Carries out **Customer Adds Items to Shopping Cart** to step 2.
2. System prompts customer “Cannot Add more than given quantity” and Returns current capacity and item name.
3. Uer hits “OK” to confirm that items could not be added.
4. Customer closes the prompt.

Use case - Customer Reviews Item Details:

1. System loads the customer window.
2. Customer highlights a item from the list of available items and clicks it.
3. System responds by updating the window to show a more detailed screen to show item name, price, Description of item, quantity, and the vendor selling the item.

Variant #1 - Customer Reviews Bundle Details:

1. System loads the customer window.

2. Customer highlights a bundle from the list of available bundles and clicks it.
3. System responds by updating the window to show a more detailed screen to show the Bundles name, the vendor selling the bundle, the bundled price of all the Items in the bundle, and the quantity of the Bundle.

Use case - Customer Reviews / Updates Shopping Cart:

1. System loads the customer window.
2. The User executes Usecase **Customer Adds Items to Shopping Cart.**
3. Customer hits the “cart” button.
4. System Loads Checkout window to show everything in the Customer’s cart.
5. Customer reviews cart.

***Variant* #1** - Customer Deletes Item From Cart

1. Customer executes Usecase **Customer Reviews / Updates Shopping Cart**.
2. Customer clicks on item or bundle.
3. Customer selects determines how many items to delete by typing in inputbox.
4. Customer clicks the delete button.
5. The system updates cart.

Variant #2 - Customer Applies Coupons

1. Customer executes Usecase **Customer Reviews / Updates Shopping Cart**.
2. Customer clicks on “Apply 5% Coupon” Checkbox button.
3. System Updates total price.
4. Customer clicks on “Apply 10% Coupon” Checkbox button.
5. System Updates total price.

***Variant* #3** - Customer exits reviewing shopping cart details:

1. Customer hits “return to previous page”.
2. System returns to shopping window

Use case - Customer Checks Out:

1. Customer carries out **Customer Reviews / Update Shopping Cart:**
2. Customer selects checkout.
3. System displays a checkout window.
4. Customer enters First Name, Last Name, Billing Address, Street, Card Number, and other important information.
5. User clicks on submit button.
6. The system loads window prompt“Transaction complete, Thank you for shopping at NOZAMA!”.
7. The system loads the User Login page.

***Variants* #1** - Customer exits checkout before payment:

1. System throws a completion message.
2. Customer exits the page before entering the payment method.
3. The system is not updated and does not remove inventory.

Use case - Seller Views Revenues, Sales, Profit:

1. User executes **User logs in with a Seller Account.**
2. The System loads the Seller Dashboard.
3. Customer/Seller has access to Total Profit, Revenues, and Cost at top of window.

Use case - Seller Reviews/ Adds Item:

1. User executes **Seller Views Revenues, Sales, Profit.**
2. Seller clicks on “Add New Item Listing” button.
3. The System loads a “Add Item” Window.
4. The User inputs Name, Invoice Price, Sell Price, Description, and quantity.
5. User confirms their input by clicking the “Add item” button at the bottom of the page.
6. The System appends Item to Item list.
7. The System updates inventory for the Item.

***Variants* #1** - Seller exits before confirming updates

1. Seller carries out **Seller Review/Adds Item** and stops at step 4.
2. Seller exits page before confirming changes.
3. The system is not updated and does not add inventory.

Variants #2 - Seller Reviews/ Adds Bundle:

1. User executes **Seller Reviews/ Adds Item** and stops at step 3.
2. The User inputs the Name of the bundle and the quantity.
3. The User then executes **Seller Reviews/Adds Item** without step 6 and 7.
4. The User then clicks the submit button.
5. The system saves the bundle to the item list.
6. The system updates inventory for the bundle.
7. The system loads the Seller Dashboard.

Use case - Seller Edits an existing Item:

1. The User executes **Seller Views Revenues, Sales, Profit.**
2. Seller clicks on the Item or Bundle.
3. The System then executes **Seller Reviews/ Adds Item** without step 1, 2, and 3.

Variant #1- Seller updates Bundle Items**:**

1. The user executes **Seller Views Revenues, Sales, Profit.**
2. The user then double clicks on A Bundle.
3. The System responds with a dropdown of all items within the bundle.
4. The user clicks on any item then executes **Seller Edits an existing Item** without step 1.

Glossary

User: Individual who is interacting with the system.

Account: A registered user, either seller or customer.

System: An application which the user interacts with.

Customer: The user who buys items through the system.

Seller: The user who sells products through the system.

Inventory: The sellers list of products avaliable to be sold.

Item: Item(s) to be purchased or sold.

Sales: Items that were sold by the seller.

Revenue: Sum of sales price for all sold items.

Profit: Difference between the revenue and cost of goods sold.

Cost: The invoice price for all items brought in inventory(bought).

Checkout: Where the customer goes to pay for the items in the cart.

Window: Specific screen displayed from the user's interaction.

Login: User's use their credentials to sign-in into the system.

Prompt: A message from the system to the user.

Cart: Container where the customer stores items for purchase.

Bundle: A unordered set of item(s)

Functional Specification

Context

Nozama Is a Online store where users can buy Items and where Sellers, can sell their items and not worry about the cost associated with a business.

Product Requirements

The user requirements for customers are as follows:

- Able to login to Dashboard to see items available for purchase.
- Able to buy Items in bulk.
- Able to determine how much of the Item they want to buy in one transaction.
- Able to store Items users want to buy in a cart while continuing to shop.
- Able to View Items in cart before purchasing.
- Able to Delete, items before checkout.
- Able to apply multiple coupons with purchase.
- Able to cancel order when checking out.
- Able to check out all the items/bundles in a users cart.

For sellers the requirements are:

- Able to login to SellerDashboard.
- Able to see Total Profits of sold items, Revenues, and cost.
- Able to edit the name of a item, invoice Price, sellPrice, Description, and Quanity.
- Able to delete an item/bundle.
- Able to add new Item(s) and new bundle(s).

These requirements are essential for a confident user experience and suports basic CRUD(Create, Read, Update, Delete)funcitons.

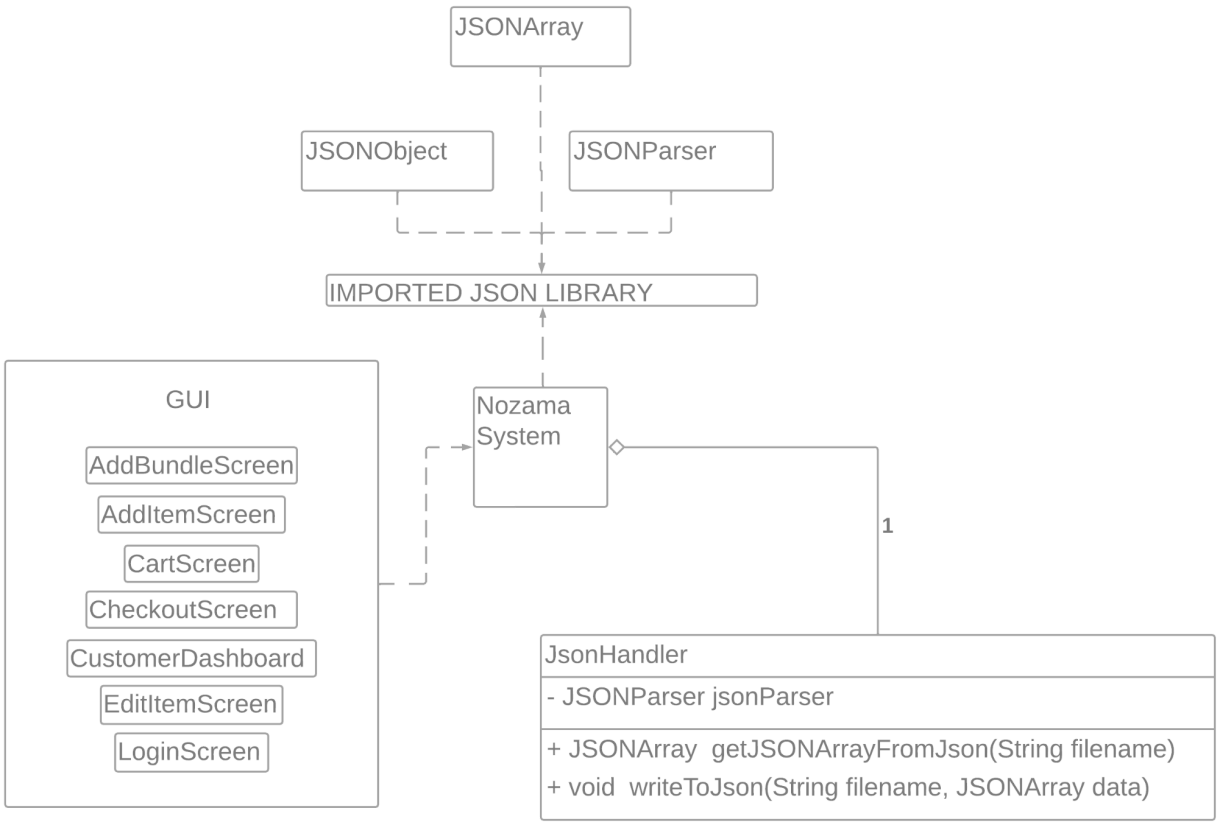
Non-Functional Requirements

- For application to meet User standards, Reading and writing functionality will be needed to aggregate data from different files. This includes files that contain the user's information and password, item's list for sellers, and each sellers profits, cost, and revenue.
- There will need to be a user interface for the user to interact with the Nozama system.
- Files that store User information, Bundles, Items, and Seller information.

Functional Requirements

Users must be able to access their account. Alternatively, the use of localfile storage was used as a database for users settings, Seller inventory, and etc. A JSON file was used for ease of access, ability to recursively store items, and key-value pair access.

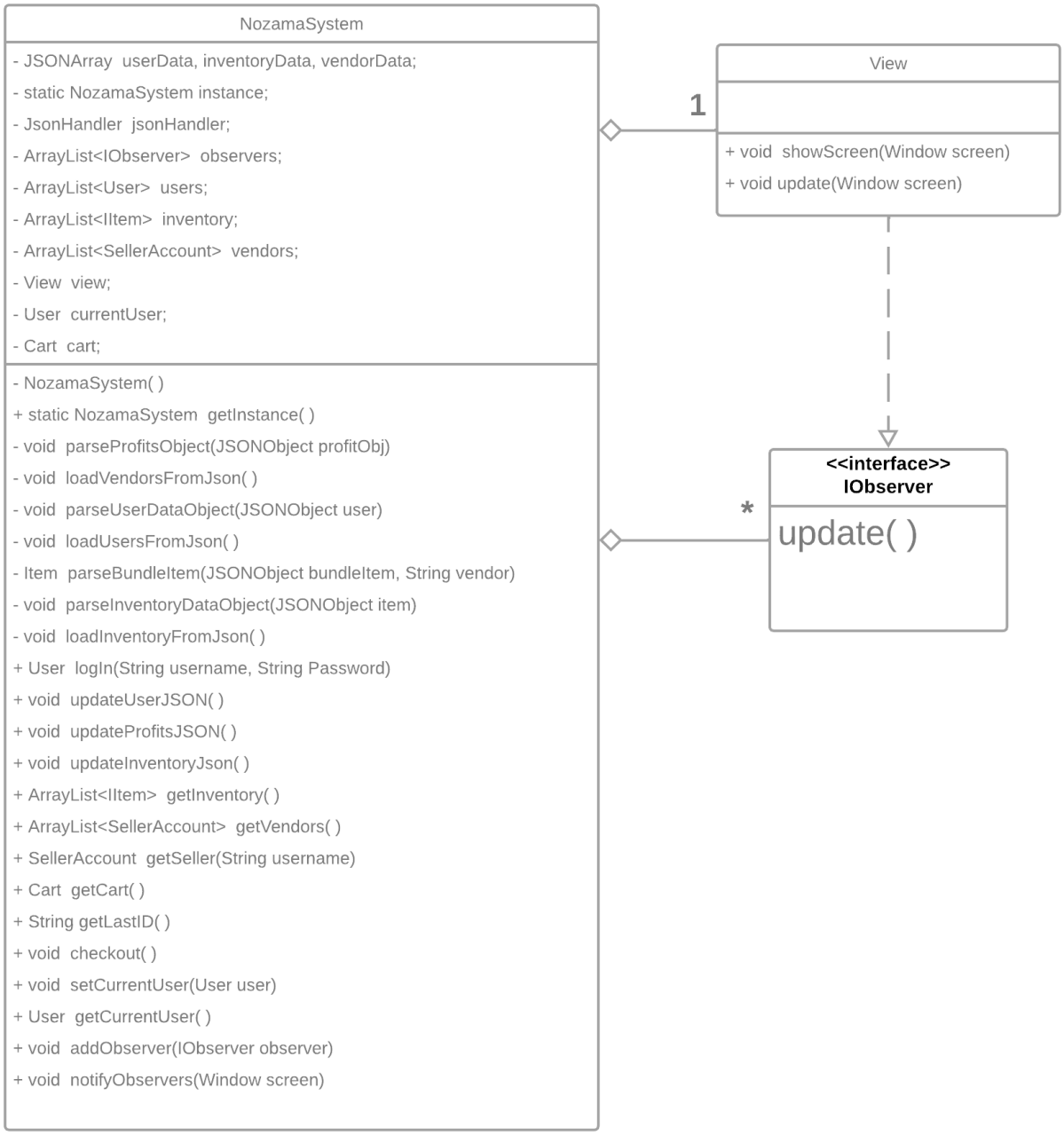
Storage



The imported JSON library Json-simple by fangyidong was used to abstract CRUD functions. These functions aggregate data from items.json, profits.json, and etc.

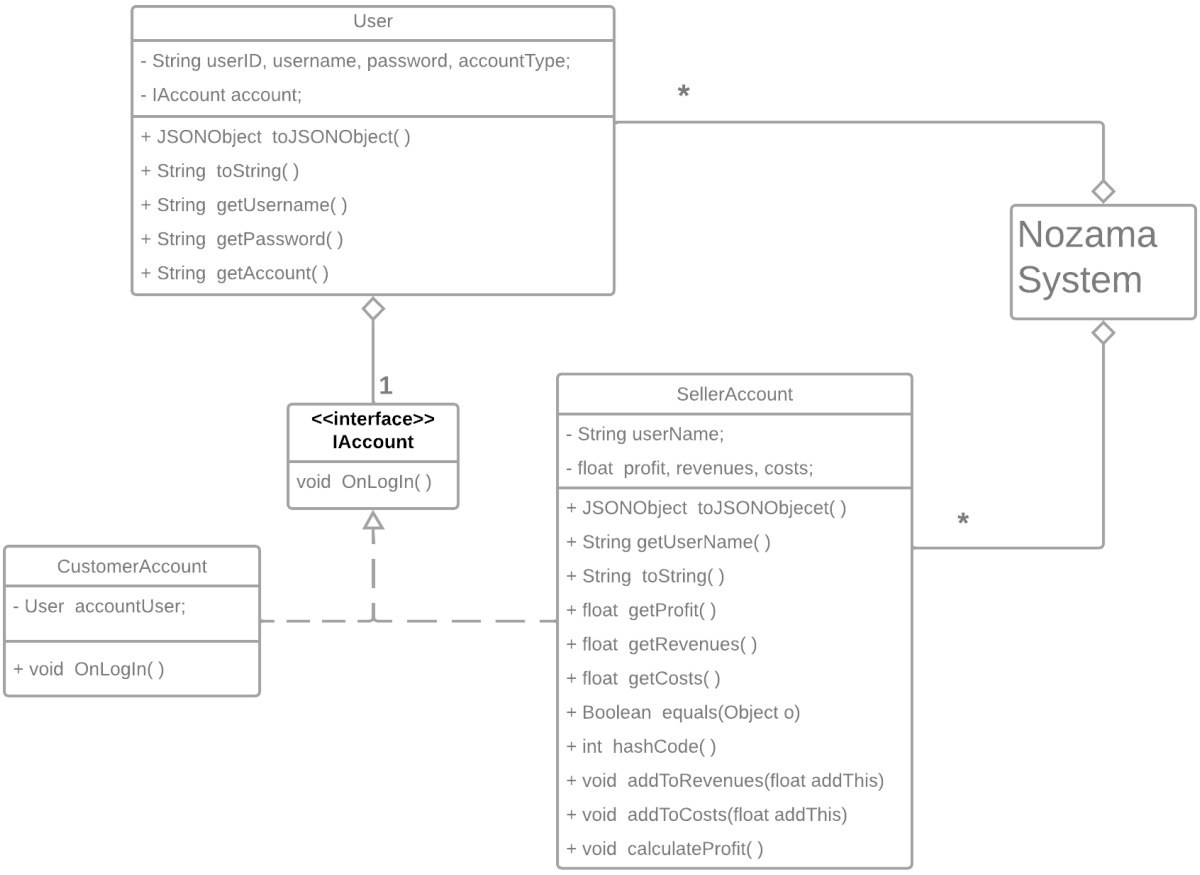
The Nozama system Is the heart of the application. It holds the logic of the application such as what to do on user login and resolves which view should be loaded and destroyed.

NozamaSystem



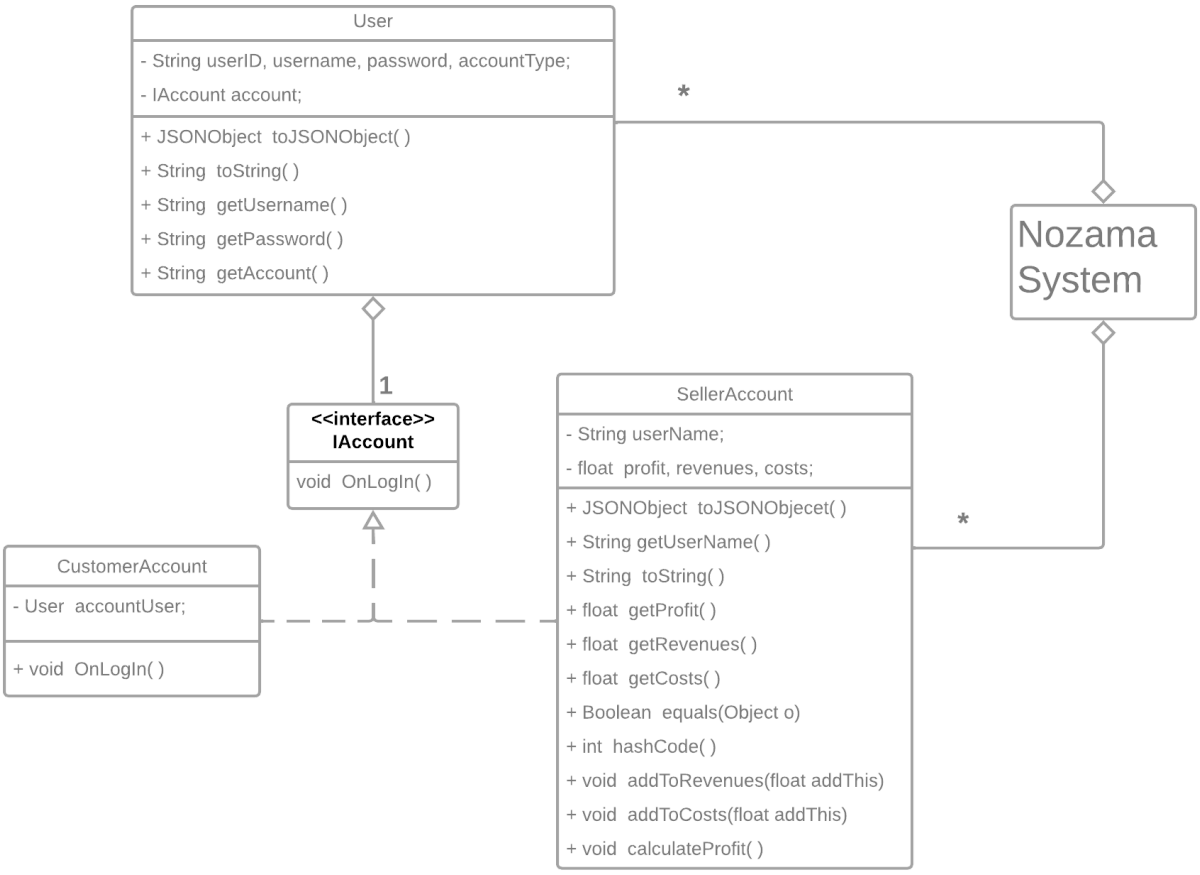
Different accounts will be needed to differentiate between a Customer account and a Seller account. The differentiate is implemented out from an interface into different classes. These classes have divergent functionalities. This interface is a IAccount, while the User aggregates, one or the other types of accounts.

UserAccounts



Items and bundles coincide with each other.The relationship between Items, and Bundles are dependent on the IItem interface. This insures that bundles and items behave like each other. Bundles are items that contain other items; Hence, additional methods to compensate.

Users


User
- String userID, username, password, accountType;
- IAccount account;
+ JSONObject toJSONObject( )
+ String toString( )
+ String getUsername( )
+ String getPassword( )
+ String getAccount( )
*
Nozama System
1
<<interface>>
IAccount
void OnLogIn( )
SellerAccount
- String userName;
- float profit, revenues, costs;
+ JSONObject toJSONObjecet( )
+ String getUserName( )
+ String toString( )
+ float getProfit( )
+ float getRevenues( )
+ float getCosts( )
+ Boolean equals(Object o)
+ int hashCode( )
+ void addToRevenues(float addThis)
+ void addToCosts(float addThis)
+ void calculateProfit( )
*
CustomerAccount
- User accountUser;
+ void OnLogIn( )

CRC Diagram

Class Name	Responsibility	Collaborators
Item	Converts to String Retrieves ID Retrieves Name Converts to JSON Retrieves Sell Price Answers “is Bundle?” Retrieves Quantity Retrieves Vendor Retrieves invoice price Retrieves Description Compares with objects Retrieves hashcode Changes ID Changes Name Change Sell Price Changes Description Changes Quantity Changes Vendor	Cart System Bundle
CustomerAccount	Perform Action on Login	User
SellerAccount	Converts to JSON Retrieves username Converts to String Retrieves Profit Retrieves Revenues Retrieves Costs Compares with objects Retrieves hashcode Adds to Revenues Adds to Costs Calculates profit	System User
User	Converts to JSON Converts to String Retrieves username Retrieves password Retrieves account	System
System	Logs user in Retrieves Inventory Retrieves Vendors Finds seller given a username Retrieves Cart Retrieves the last Items ID	User SellerAccount CustomerAccount View Cart CouponDecorator

	Checkout the cart Set current user Retrieve current user Add observers Notify observers	Bundle Item JsonHandler
View	Gets updated by System On update, show screen	System
Cart	Retrives Cart Retrives the quantity for each Item in the cart Removes Item from the cart Print the Items in the cart Calculates the total and retrieves it Calculates the total with coupons Changes the total with coupons Retrives quantity of a specific item in the cart	Item Bundle System CouponDecorator
JsonHandler	Gets JSON Array from JSON file Writes data to JSON	System
Bundle	Convert to String Retrieve sell price Answers “is Bundle?” Retrieves quantity Retrieves instance of vendor Retrives name Converts to JSON Adds item to bundle Retrieve the list of Items Compares with other objects Retrieves hashcode Retrieves ID Changes quantity	Item System Cart

Class Diagram

Design Patterns

Singleton Pattern

Requirements:

1.Define a class with a private constructor.

2. The class constructs a single instance of itself

3. Supply a static method that returns a reference to the single instance

NozamaSystem.java

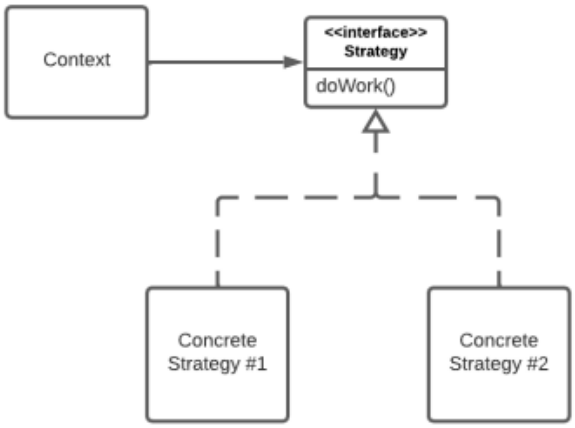
```
private NozamaSystem(){}


public static NozamaSystem getInstance()
{
    if (instance == null)
        instance = new NozamaSystem();

    return instance;
}
```

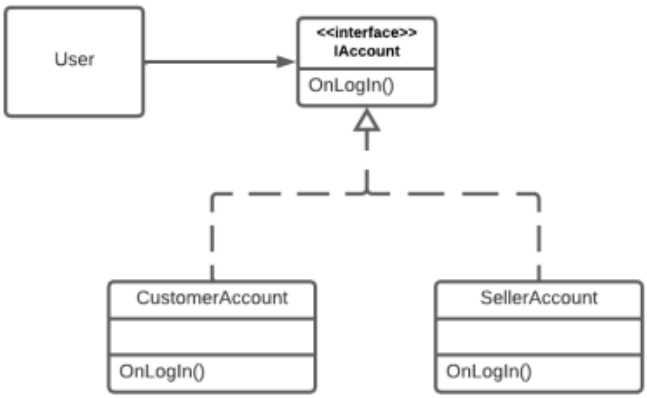
Strategy Pattern

Requirements:



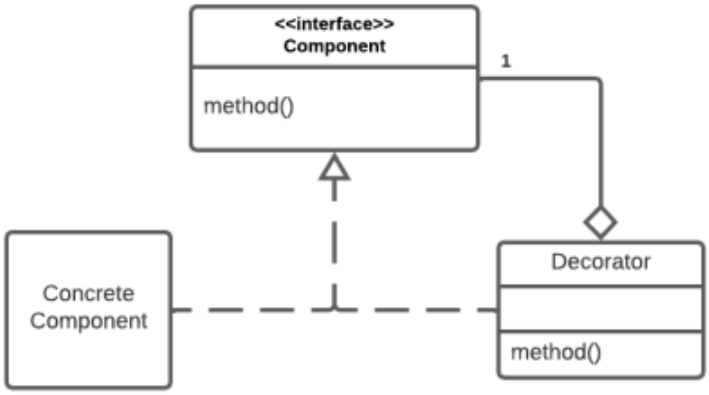
Implementation:

Name in Design Pattern	**Actual Name (layout management)**
Context	User
Strategy	IAccount
Concrete Strategy #1	CustomerAccount
Concrete Strategy #2	SellerAccount
doWork()	OnLogIn()



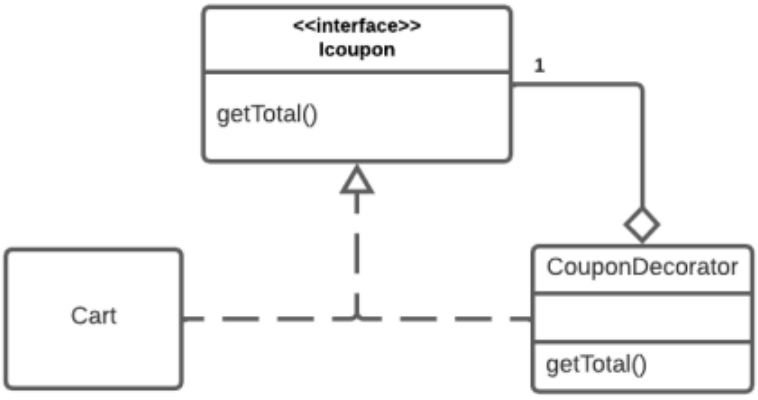
Decorator Pattern

Requirements:

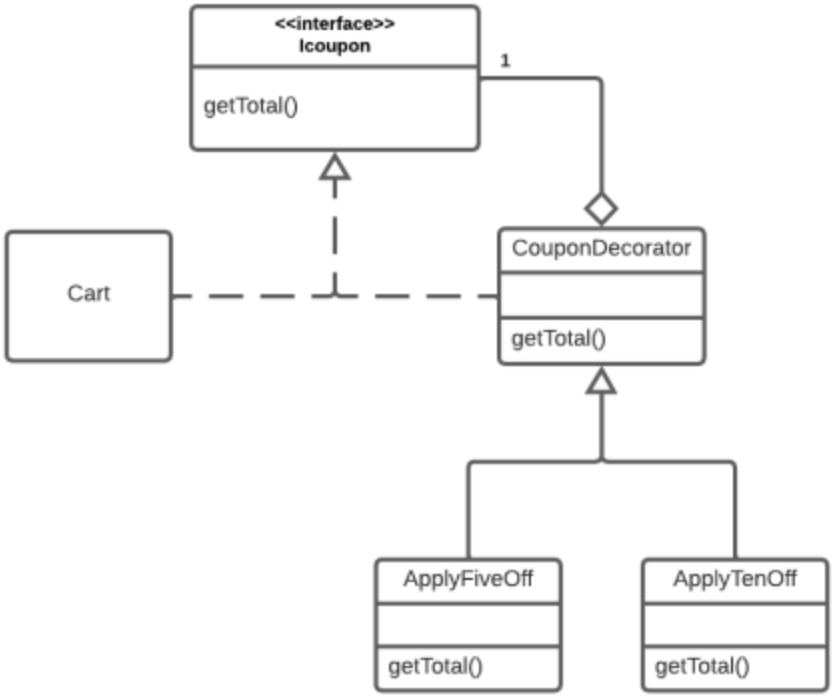


Implementation:

Name in Design Pattern	**Actual Name (layout management)**
Component	ICoupon
Concrete Component	Cart
Decorator	CouponDecorator
method()	getTotal()

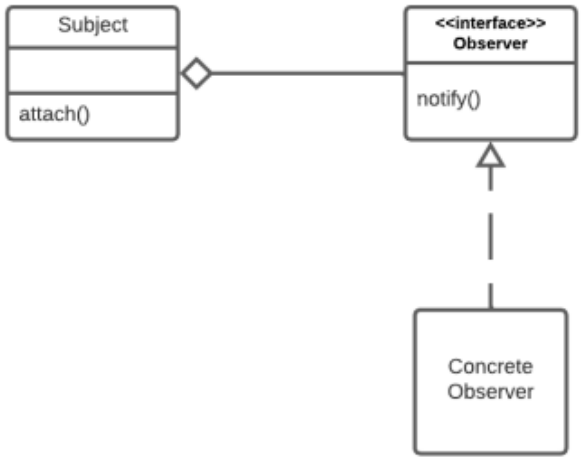


Additional:


<<interface>>
Icoupon
getTotal()
1
Cart
CouponDecorator
getTotal()
ApplyFiveOff
getTotal()
ApplyTenOff
getTotal()

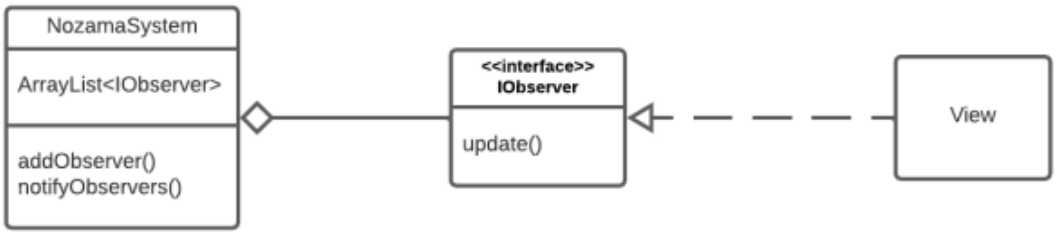
Observer Pattern

Requirements:



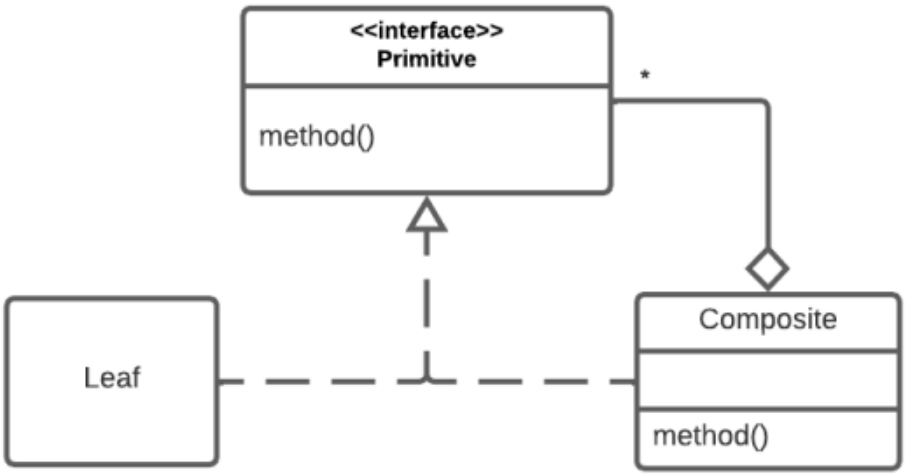
Implementation:

Name in Design Pattern	Actual Name (layout management)
Subject	NozamaSystem
Observer	IObserver
Concrete Observer	View
Attach()	addObserver()
Notify()	notifyObservers()



Composite Pattern

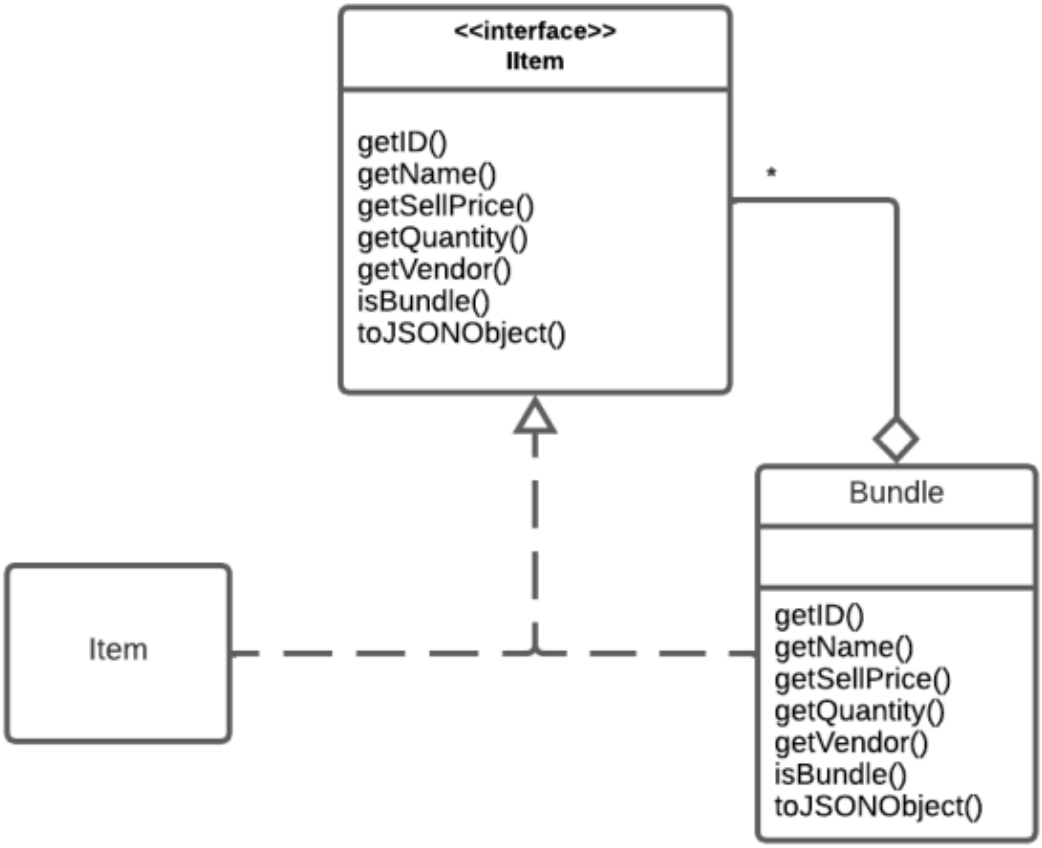
Requirements:



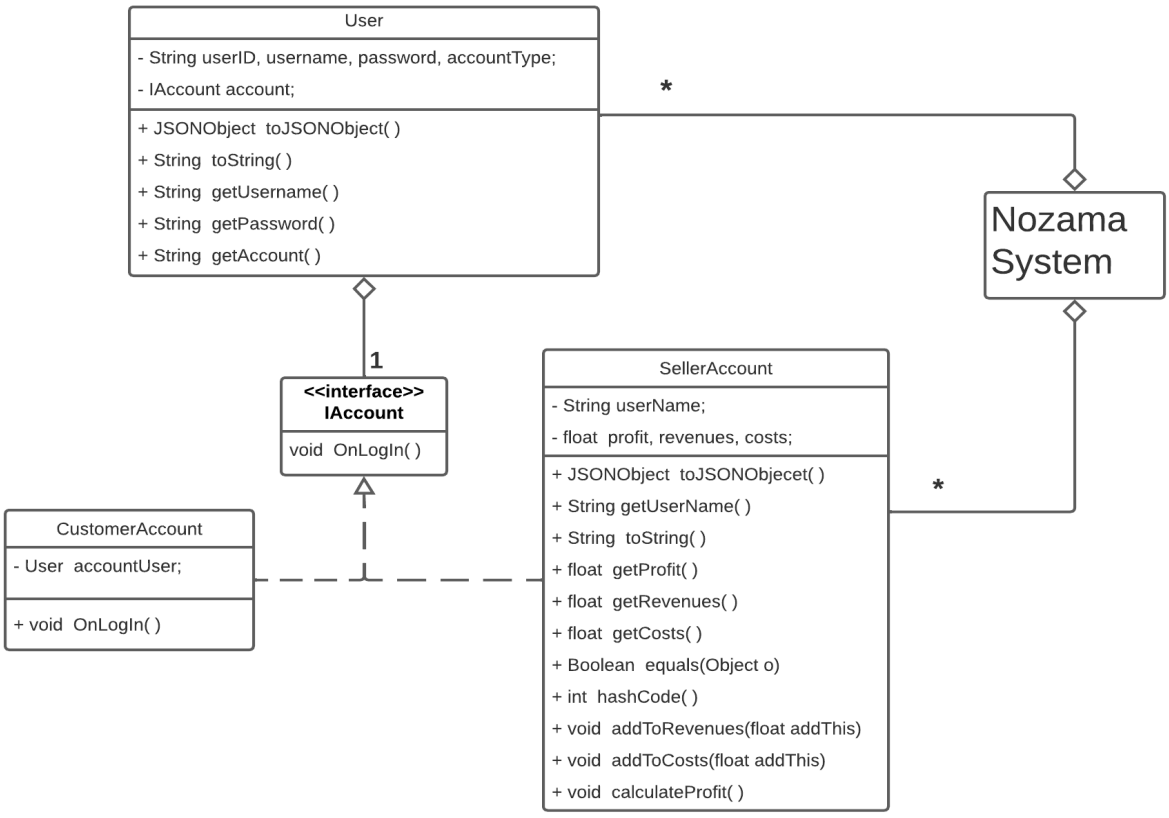
Implementation:

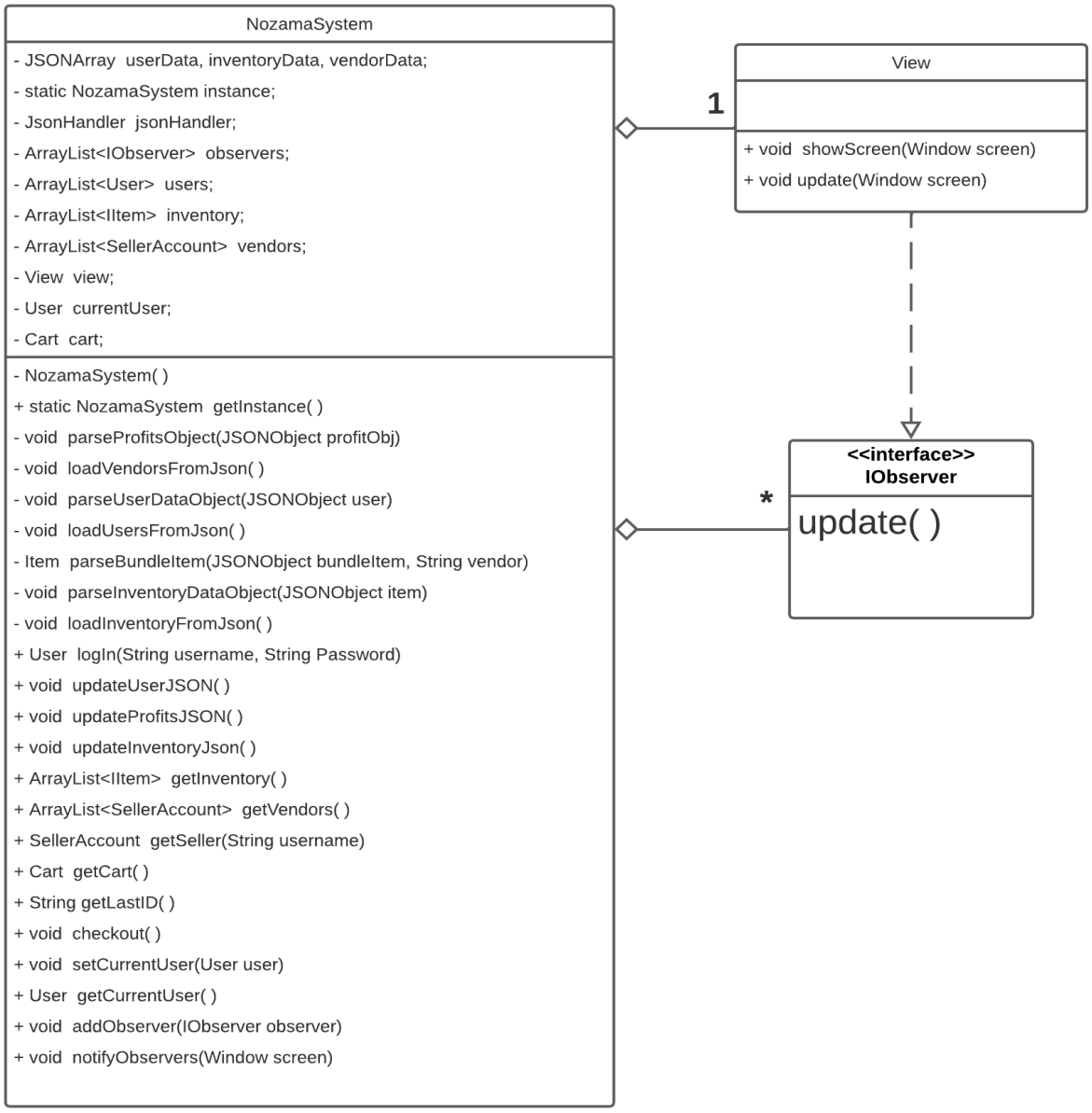
Name in Design Pattern	Actual Name (layout management)
Primitive	Iitem
Composite	Bundle
Leaf	Item
method()	getID(), getName(), getSellPrice(), getQuantity(), getVendor(), isBundle(), toJSONObject()

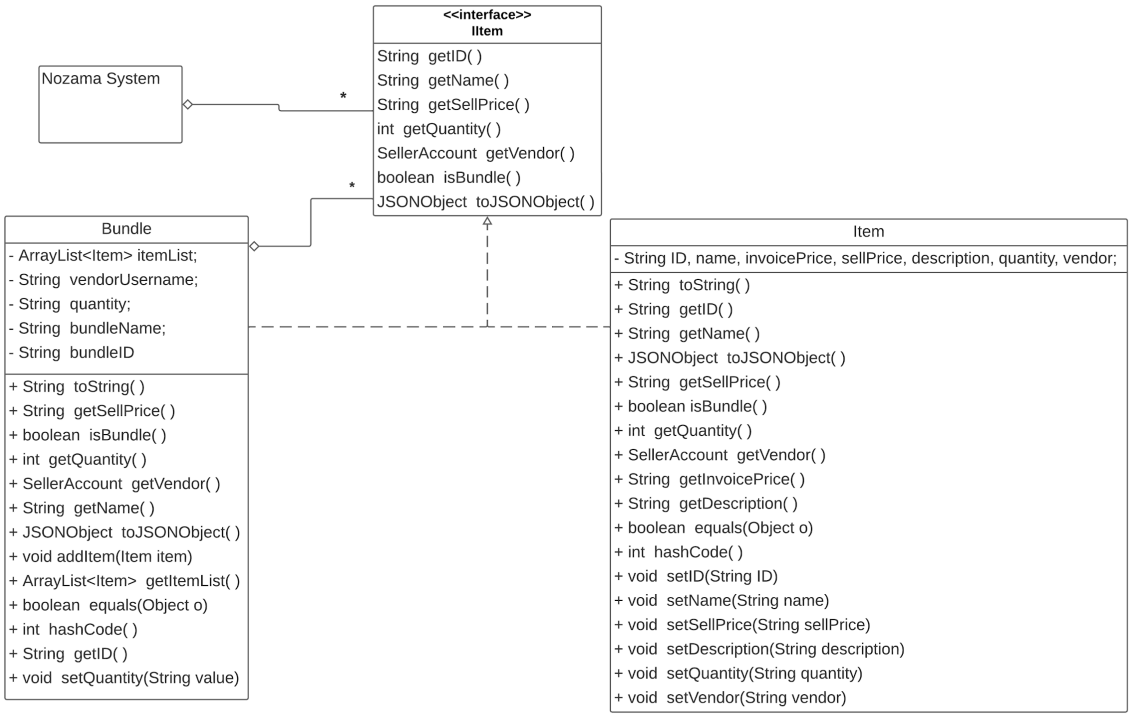
Note: Interfaces have an I before their name. so, the Interface IItem is not the class Item

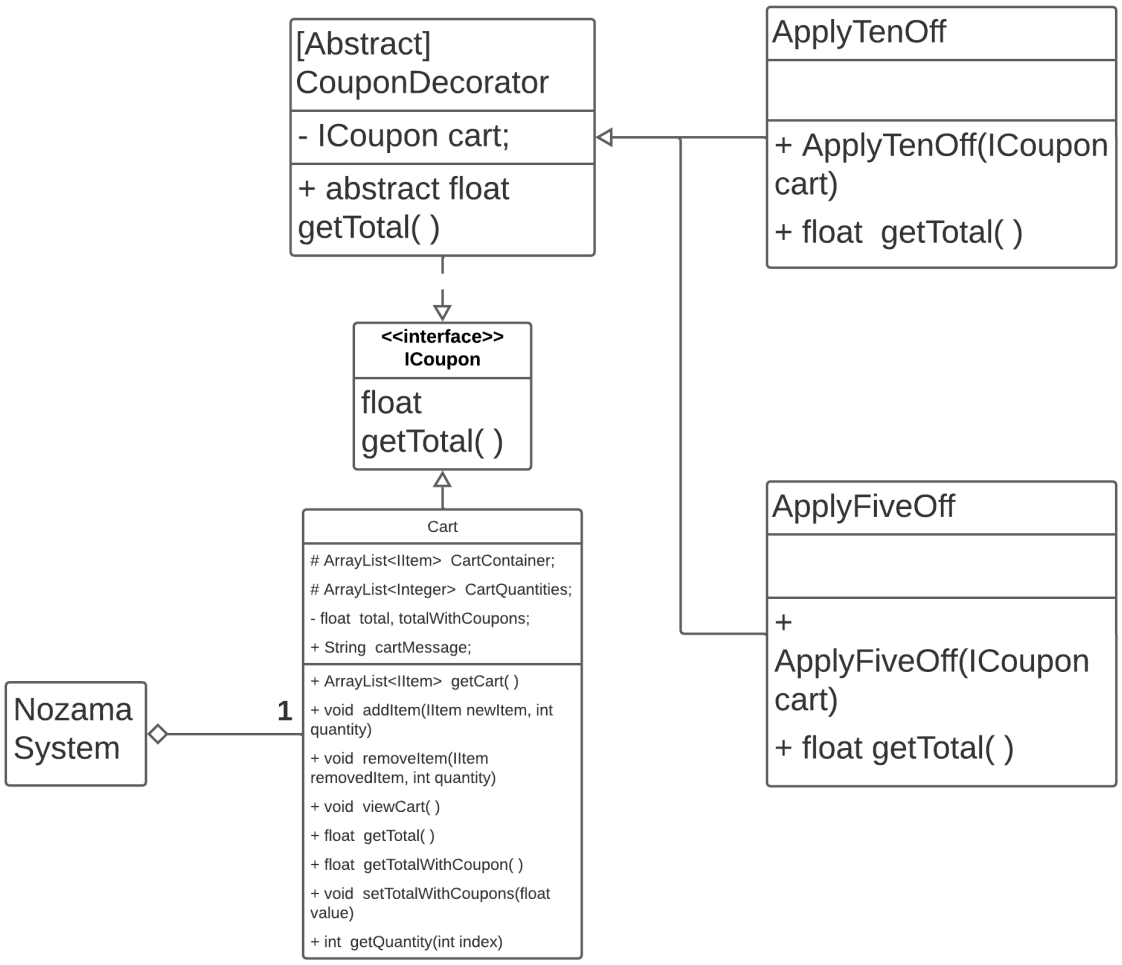


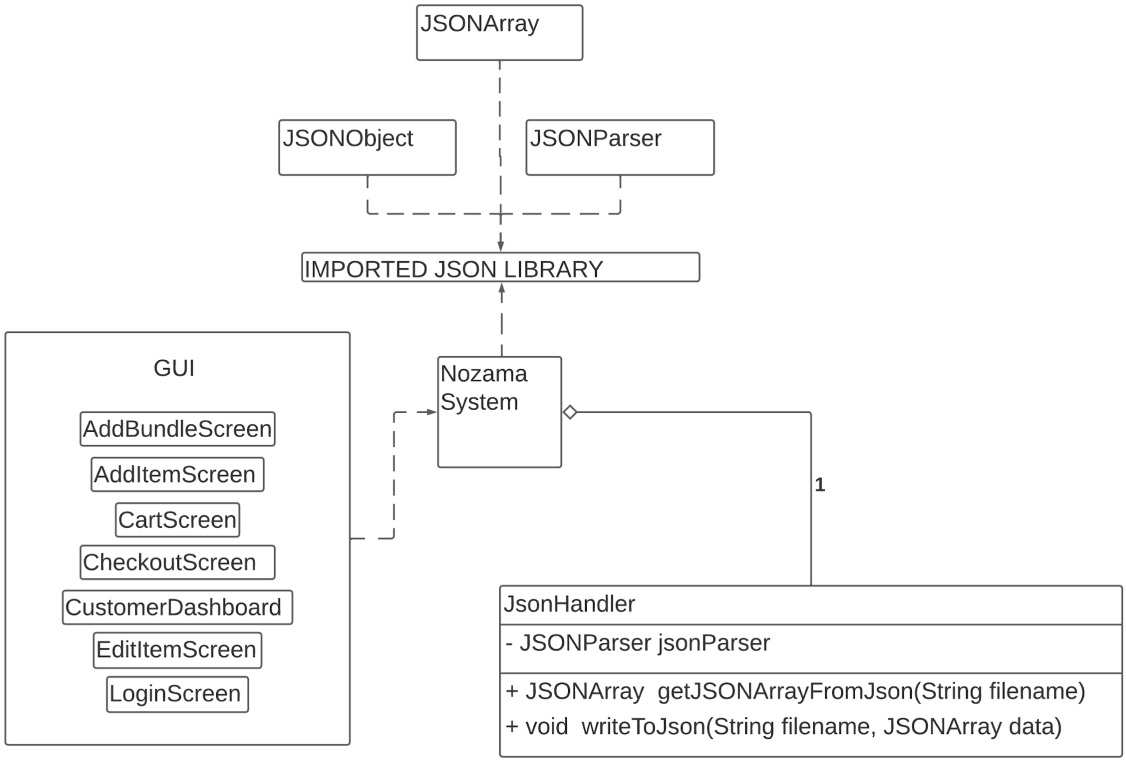
Class Diagram




NozamaSystem
- JSONArray userData, inventoryData, vendorData;
- static NozamaSystem instance;
- JsonHandler jsonHandler;
- ArrayList<IObserver> observers;
- ArrayList<User> users;
- ArrayList<IItem> inventory;
- ArrayList<SellerAccount> vendors;
- View view;
- User currentUser;
- Cart cart;
- NozamaSystem( )
+ static NozamaSystem getInstance( )
- void parseProfitsObject(JSONObject profitObj)
- void loadVendorsFromJson( )
- void parseUserDataObject(JSONObject user)
- void loadUsersFromJson( )
- Item parseBundleItem(JSONObject bundleItem, String vendor)
- void parseInventoryDataObject(JSONObject item)
- void loadInventoryFromJson( )
+ User logIn(String username, String Password)
+ void updateUserJSON( )
+ void updateProfitsJSON( )
+ void updateInventoryJson( )
+ ArrayList<IItem> getInventory( )
+ ArrayList<SellerAccount> getVendors( )
+ SellerAccount getSeller(String username)
+ Cart getCart( )
+ String getLastID( )
+ void checkout( )
+ void setCurrentUser(User user)
+ User getCurrentUser( )
+ void addObserver(IObserver observer)
+ void notifyObservers(Window screen)
1
View
+ void showScreen(Window screen)
+ void update(Window screen)
*
<<interface>>
IObserver
update( )

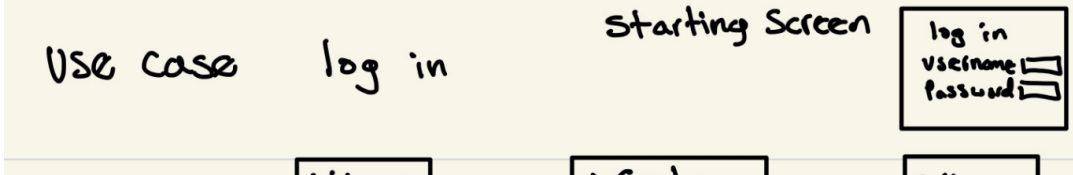

Nozama System
*
<<interface>>
IItem
String getID( )
String getName( )
String getSellPrice( )
int getQuantity( )
SellerAccount getVendor( )
boolean isBundle( )
JSONObject toJSONObject( )
*
Bundle
- ArrayList<Item> itemList;
- String vendorUsername;
- String quantity;
- String bundleName;
- String bundleID
+ String toString( )
+ String getSellPrice( )
+ boolean isBundle( )
+ int getQuantity( )
+ SellerAccount getVendor( )
+ String getName( )
+ JSONObject toJSONObject( )
+ void addItem(Item item)
+ ArrayList<Item> getItemList( )
+ boolean equals(Object o)
+ int hashCode( )
+ String getID( )
+ void setQuantity(String value)
Item
- String ID, name, invoicePrice, sellPrice, description, quantity, vendor;
+ String toString( )
+ String getID( )
+ String getName( )
+ JSONObject toJSONObject( )
+ String getSellPrice( )
+ boolean isBundle( )
+ int getQuantity( )
+ SellerAccount getVendor( )
+ String getInvoicePrice( )
+ String getDescription( )
+ boolean equals(Object o)
+ int hashCode( )
+ void setID(String ID)
+ void setName(String name)
+ void setSellPrice(String sellPrice)
+ void setDescription(String description)
+ void setQuantity(String quantity)
+ void setVendor(String vendor)

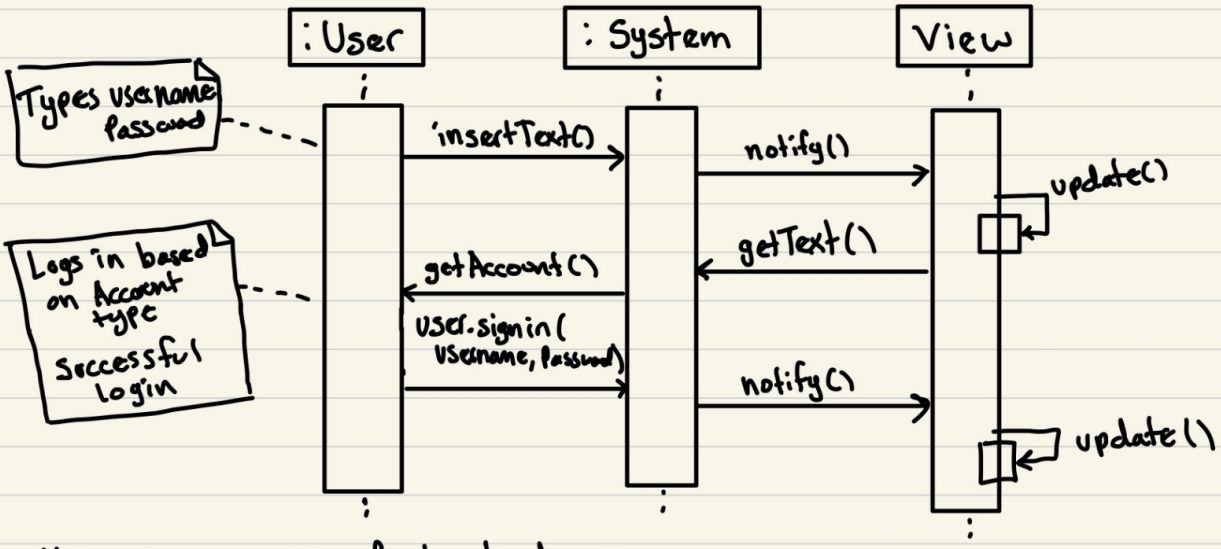

[Abstract]
CouponDecorator
- ICoupon cart;
+ abstract float getTotal( )
ApplyTenOff
+ ApplyTenOff(ICoupon cart)
+ float getTotal( )
<<interface>>
ICoupon
float getTotal( )
Cart
ArrayList<IItem> CartContainer;
ArrayList<Integer> CartQuantities;
- float total, totalWithCoupons;
+ String cartMessage;
+ ArrayList<IItem> getCart( )
+ void addItem(IItem newItem, int quantity)
+ void removeItem(IItem removedItem, int quantity)
+ void viewCart( )
+ float getTotal( )
+ float getTotalWithCoupon( )
+ void setTotalWithCoupons(float value)
+ int getQuantity(int index)
ApplyFiveOff
+ ApplyFiveOff(ICoupon cart)
+ float getTotal( )
Nozama System
1


JSONArray
JSONObject
JSONParser
IMPORTED JSON LIBRARY
GUI
AddBundleScreen
AddItemScreen
CartScreen
CheckoutScreen
CustomerDashboard
EditItemScreen
LoginScreen
Nozama System
1
JsonHandler
- JSONParser jsonParser
+ JSONArray getJSONArrayFromJson(String filename)
+ void writeToJson(String filename, JSONArray data)

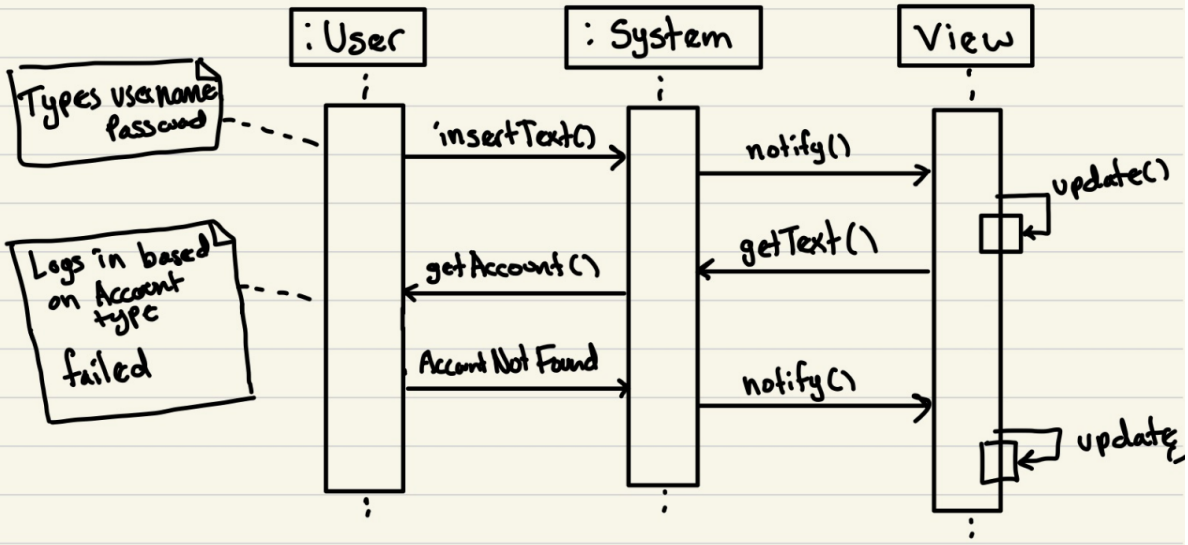
Use case log in

Starting Screen

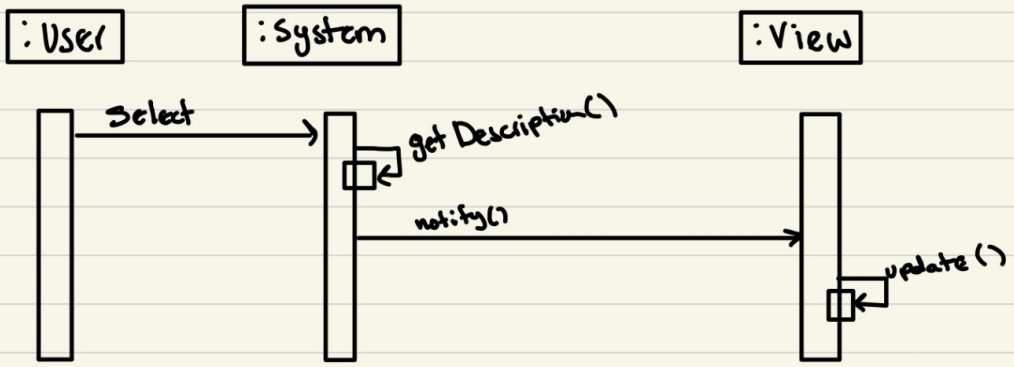




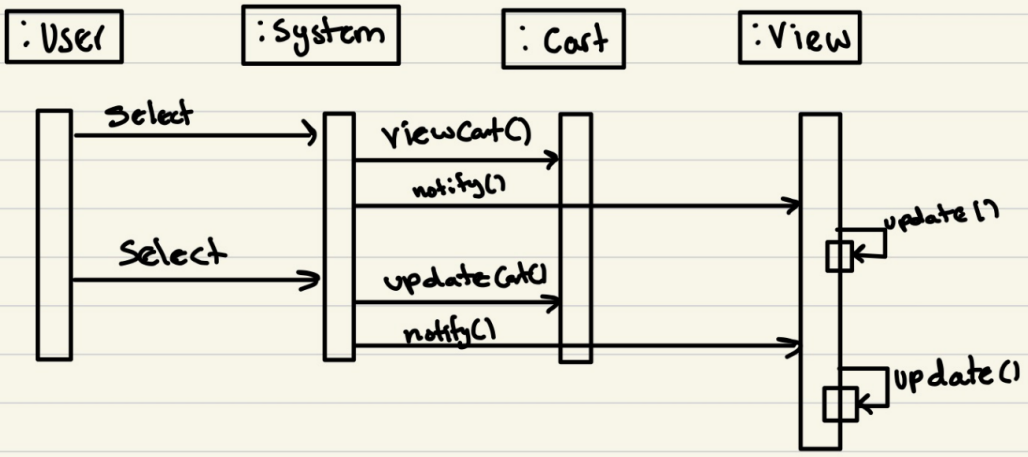
Use case User fails to log in



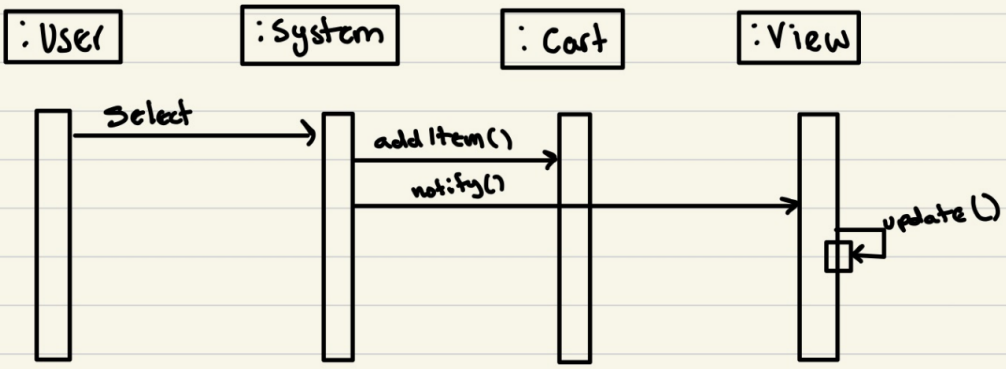
Use case Customer reviews Product Details



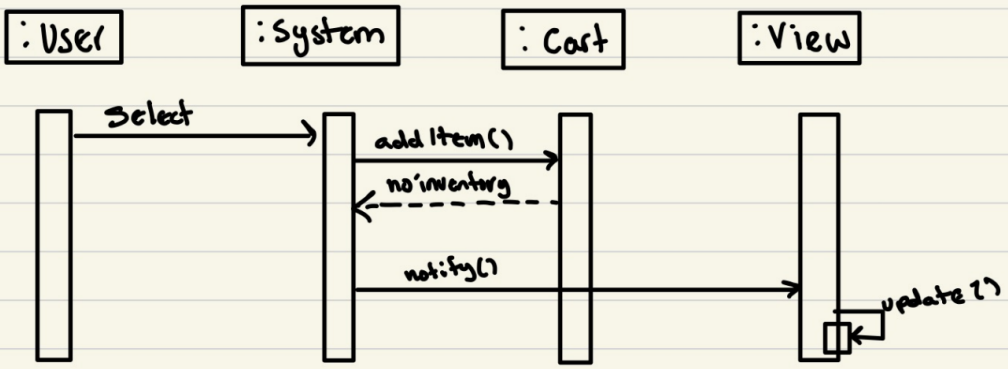
Use case Customer reviews / updates shopping cart



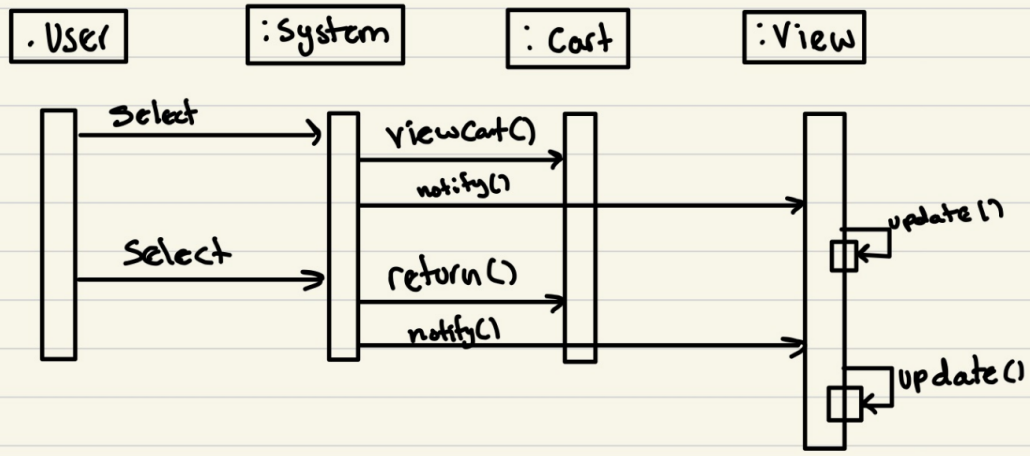
Use Case Customer Adds items to shopping Cart



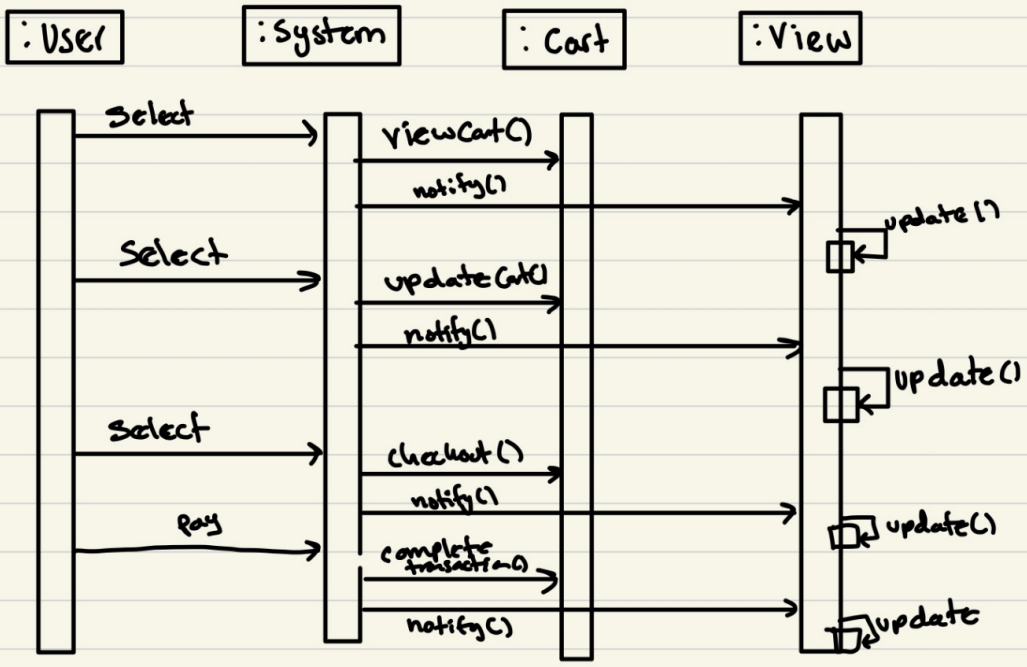
Use Case Customer Adds Item but no inventory



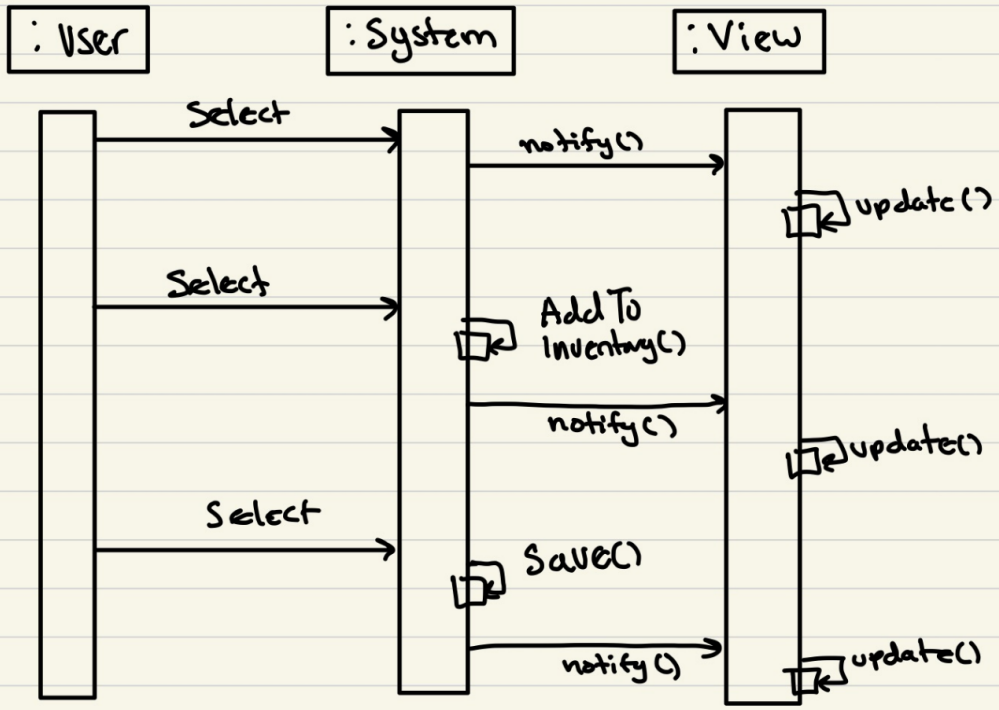
Use case Customer exits review shopping Cart



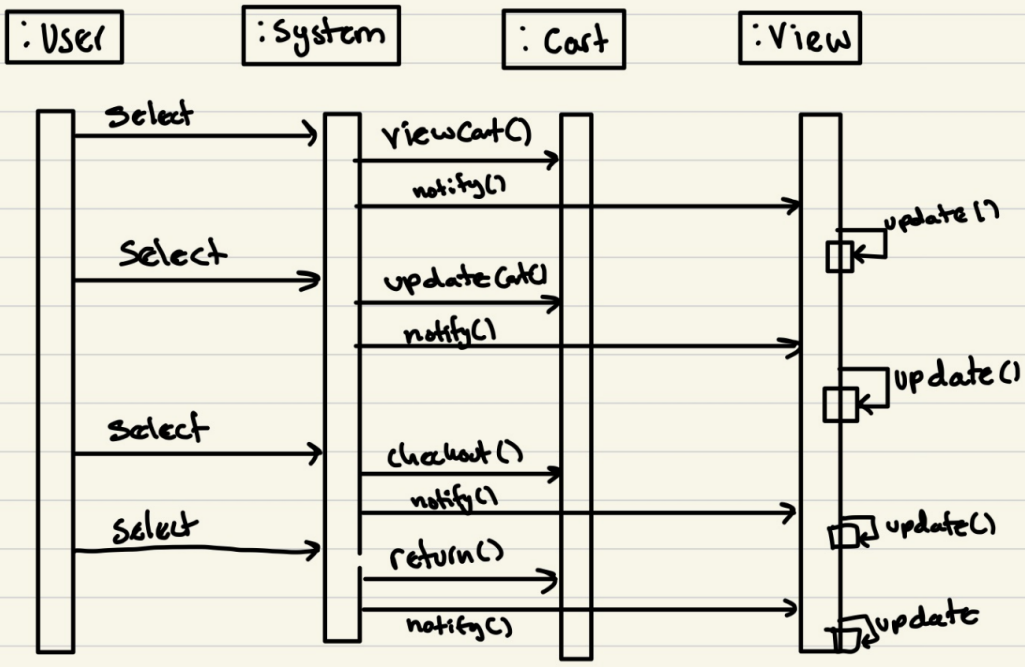
Use case Customer checks out



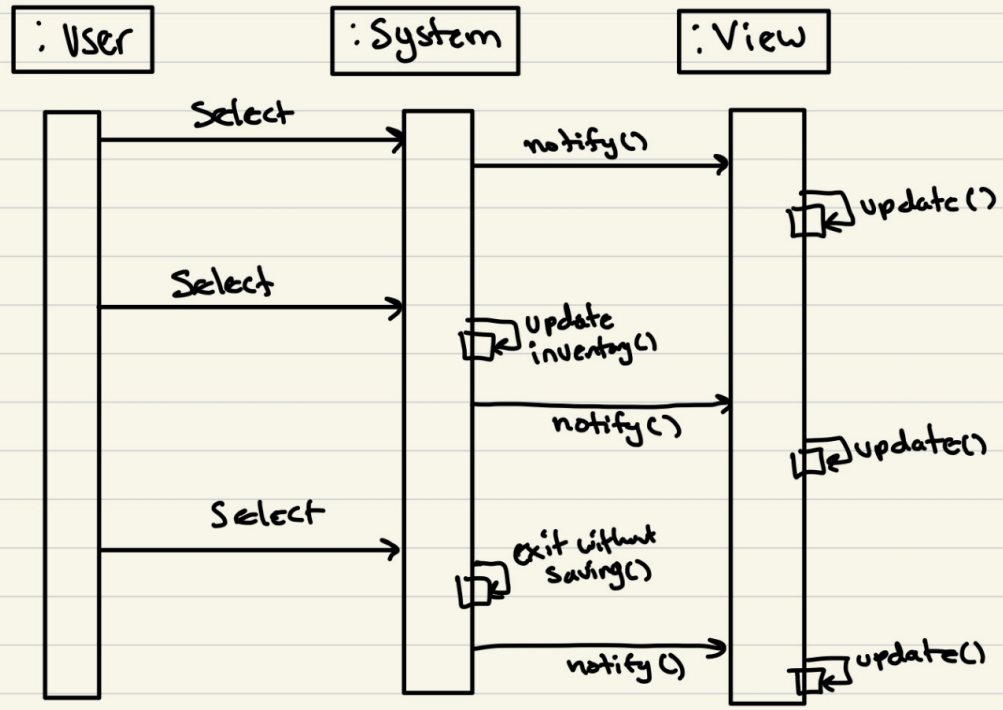
Use Case Seller Adds New Product



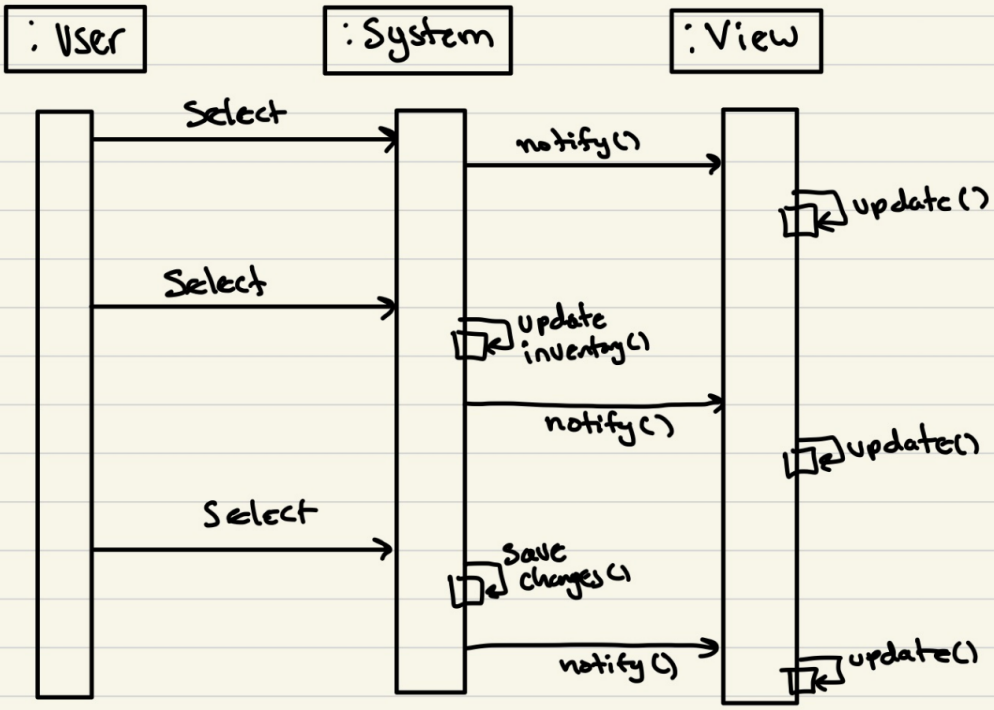
Use Case Customer exits checkout before payment



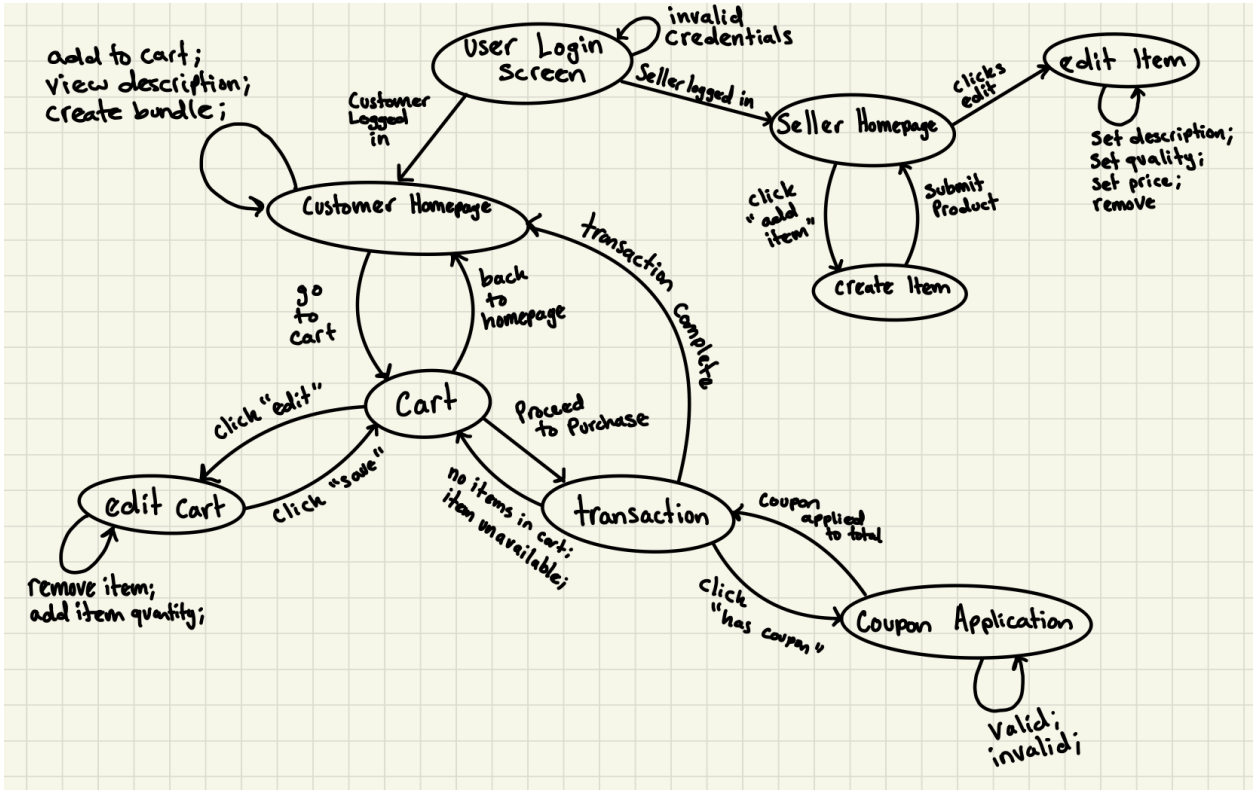
Use case Seller exits before confirming update()



Use Case Seller Review / updates Inventory



State Diagram



-seller homepage is where the seller can see revenue, sales, and profit