OnlineStore

I started with creating a simple class-responsibility-collaborators table to visualize how the program is going to function.

|  |  |  |
| --- | --- | --- |
| Class | Responsibility | Collaborators |
| Client.Java | A class that houses the main method for the client interface. Entry point for the customer | ClientFrontController.java |
| AccountFactoryImplementation.java | A concrete implementation of the AccountsFactoryInterface.java. creates and returns the AccountsContainer | AccountsContainer.java |
| AccountsFactoryInterface.java | An interface. Defines the functionality of the AccountsFactoryInterface.java | AccountFactoryImplementation.java |
| AccountsObserver.java | Listens to changes in the AccountsContainer and puts it in the registry | AccountsConatiner.java |
| AddItemcommand.java | Used by the customer side to add items to the cart. | ProductInterface.java, Command.java |
| AddNewAccountCommand.java | Used by the admin for the creation of a new account | Command, AccountsConatiner, Account |
| AddproductCommand | Used by the admin to add product to the inventory list | ProductsInterface, ProductInterface |
| Authorization | Checks current user permissions | ClientFrontController, ServerFrontController |
| ClientDispatcher | Handles and redirects instructions taken by the Front controller and allocates the appropriate command | Command, ProductsInterface, |
| ClientFrontController | Controls and manipulate user request. Calls on the dispatcher to handles the jobs | ClientDispatcher, Client.java, Account, Authorization. |
| Command | An interface that defines the functionality of which all the implementing command types. |  |
| CreateAdminCommand | Creates the admin Account after signing by the admin | Account, AccountsContainer |
| CreateCustomerCommand | Creates the customer Account after signing by the customer | Account, AccountsContainer |
| Dispatcher | Used by the ServerFrontController to dispatch and assign appropriate commands | Command, product, Account, ProductsContainer, AccountsContainer |
| LoginCommand | Logs in a user | AccountsContainer |
| Observer | An interface. Uses the observer pattern |  |
| PurchaseCommand | Implements the Command interface |  |
| RemoveAccountCommand | Removes an account from the AccountsContainer | AccountsContainer |
| RemoveProductCommand | Removes an account from the productsContainer | productsContainer |
| ServerFrontController | An interface for the server side | Command, Authorization, Account, AccountsFactoryInterface, AccountsContainer, ProductsContainer |
| SimilarAccFactoryImpl | Uses the Abstract factory pattern to create Accounts | AdminAccount, CustomerAccount |
| SimilarAccFactoryInfc | An interface for the SimilarAccFactoryImpl | SimilarAccFactoryImpl |
| UpdateProductCommand | Used by the server side to update a product | Product, ProductsContainer |
| Account. | An interface for types of account |  |
| AdminAccount | A concrete implementation of the Account interface. Holds admin information | Account |
| CustomerAccount | A concrete implementation of the Account interface. Holds customer information | Account |
| AccountsContainerInterface | An interface for the Accounts container interface to be sent to the rmi registry | Account, CustomerAccount, AdminAccount, Observer, |
| ProductInterface | An interface for products type |  |
| Product | A concrete implementation of ProductInterface | ProductInterface |
| StoreServer | An interface for the server. Admin interacts with the system mostly here. | ServerFrontController |
| AccountsConatiner | A concrete implementation of the AccountsContainerInterface | Account, observer |
| ProductsInterface | An interface for ProductsContainer |  |
| ProductsContainer | Holds and houses the product | Product |

Design patterns.

**Command Pattern:** In the ClientDispatcher and Dispatcher class. I parameterize the processCommand function to take in a Command Object as an input. Inside that function, the execute of the individual command types are called and executed. I also used it in an arraylist when customer add items to the cart, the commands are pushed in an arraylist and when the customer is ready to checkout. The arraylist gets looped through and the command objects are called. The model objects are acting as the receive while the two dispatcher objects of both sides know how to invoke the necessary the commands. The dispatchers are the invokers.

Factory/Abstract factory: I used this pattern to create and return account types and the AccountsContainer object.

**Template Pattern:** This pattern was the most used. Most of the classes used template pattern except few such as the controllers. Account, product, Accounts, ProductsInterface etc. all design with template pattern in mind.

**Authorization:** The details of this are in the Authorization.java file. Whenever a user manages to get in with a bad username and password. If they try to access important of the system, such as updating an item. Their authorization is checked. At the beginning, each logged in user authorization is set.

**Front Controller Pattern:** Each user interface (Client and server side) used this pattern. Whenever the program is started, the main program uses the controller class which then uses the dispatcher object. The dispatcher object then calls on the appropriate command that handles a certain request sent in by the front controller. Details of this are in the corresponding classes.

**Singleton Pattern:** There can only be one instance of both the controllers and the dispatchers’ objects, so I used the singleton object.

**Observer Pattern:** An observer object that put the container in the registry when a new account is added.

Below is the diagram for domain modeling, UML diagram, and use case diagram.

Diagram

Description automatically generated

Diagram, engineering drawing

Description automatically generated

Diagram

Description automatically generated

Below are diagrams of the sample runs.

