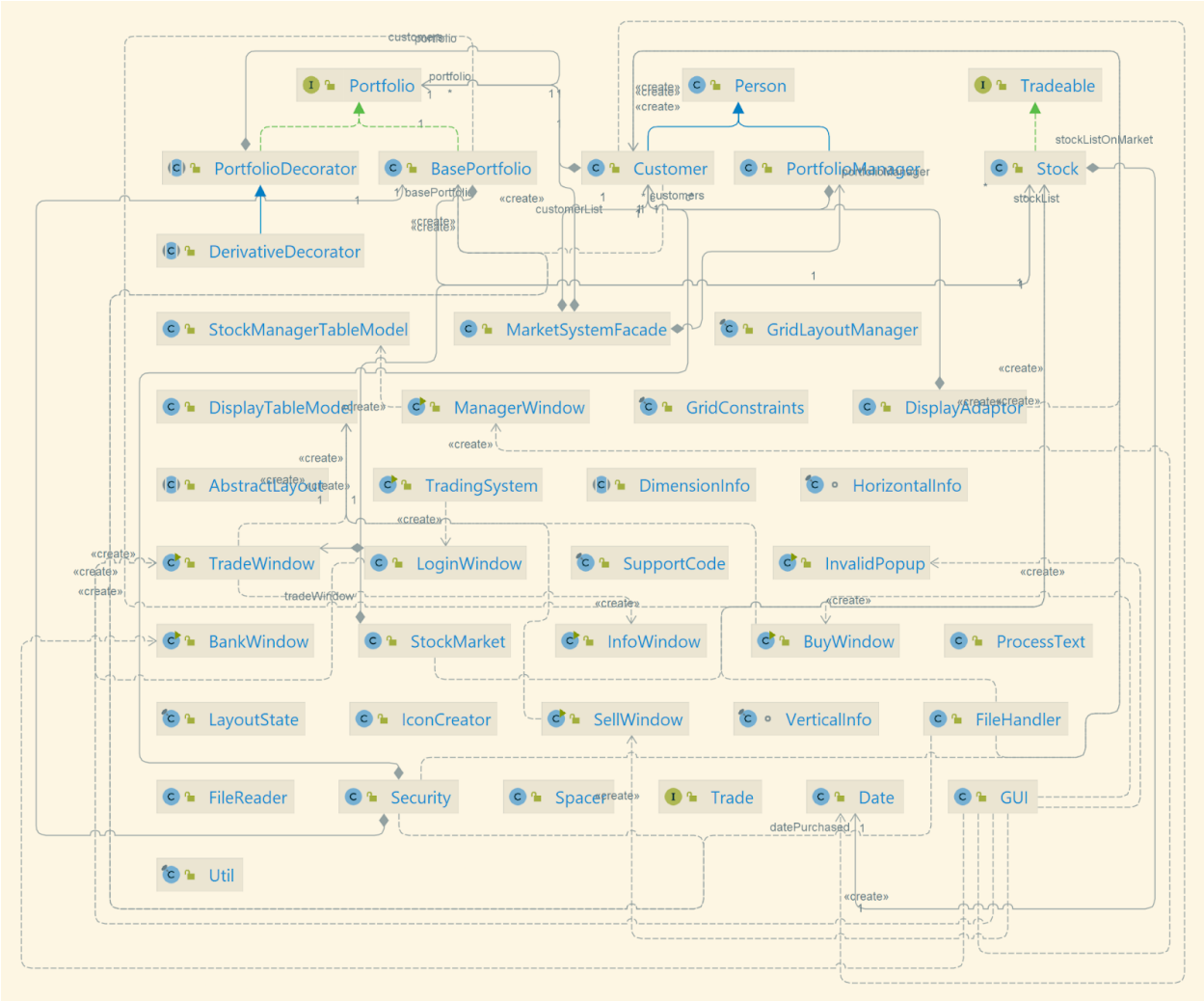
CS611-Final-Project
Trading System

Salma Alali
salali@bu.edu
U68917126

Sanjana Kasarla
skasarla@bu.edu
U44735944

Mark Yang
mdyang@bu.edu
U27016194

UML Diagram



The Trading System project is designed for both extendibility and scalability. Observing the diagram, we see TradingSystem initiates user login

and the StockMarket. All in all, the classes can be split into two categories: UI and backend with the use of design patterns to connect them. The two parts are segregated using a DisplayAdaptor and MarketSystemFacade class, meaning each part can be changed independently. Display-wise, each window is associated with its own class. This facilitates the UI to be open for modification, as adding new UI functionality is done by creating more window classes. In the backend, the mix of classes exhibit excellent object-orientated design. The responsibilities are subdivided into each class, making methods and classes very reusable and lowers redundancy.

How We Used Design Patterns

Facade Pattern:

- MarketSystem Facade
  - This class provides a simplified interface to connect the market and the functionalities of buying and selling stocks with the Portfolio manager. It decreases the overall complexity of the application.

Adaptor Pattern:

- DisplayAdaptor
  - The DisplayFacade class acts as a connector between two incompatible interfaces, the GUI and the backend, that otherwise would be difficult to be connected to directly. It contains methods that facilitate backend information being printed on the GUI.

Decorator pattern:

- Derivative Decorator
  - Used to attach additional responsibilities and functionalities to the derivative accounts
- Portfolio Decorator
  - Used to attach additional responsibilities and functionalities to the customer's Portfolio
- Base Portfolio
  - Simplest form of portfolio a customer can have, where derivative portfolios (and potentially others) can be attached using a decorator that extends Portfolio

Singleton Pattern:

- PortfolioManager class
  - Constructor set to private and can only be called by a static method getPortfolioManagerInstance() belonging to the PortfolioManager class, ensuring only one instance of PM can exist.

Observer Pattern:

- Portfolio interface **implements Observer (by Java)**
  - Ensures that portfolios can observe the changes made to Stock class (**extends Observable by Java**).
- Stock class **extends Observable:**
  - Any changes made to the current prices of stocks immediately notifies all observers, which are the portfolio objects (in this case) in order to reflect any changes in the stockList belonging to it.

Object Model (Object/Class Relationships)

Portfolio Folder:

- Base Portfolio class

    - Concrete class that implements **Portfolio** interface. It contains a mapping (using hashmap) between Stocks and their quantities (in the Portfolio). This object is a base object used by customers, which may be upgraded using Portfolio decorators (decorator pattern) to construct other types of portfolios.
- DerivativeDecorator abstract class
    - Decorator that modifies a portfolio to handle derivative portfolio responsibilities (part of extendibility to creating derivative account).
- Portfolio interface extends Observer (by Java)
    - Interface that is the basis of different decorations of portfolio objects such as BasePortfolio or DerivativeDecorator (part of extendibility).
- PortfolioDecorator abstract class
    - class that implements the decorator pattern on Portfolio objects, which return a "decoration" of those objects, where "decoration" refers to the various types of behaviors expected out of different Portfolio variants.

Display Folder:

- LoginWindow
    - UI for the login screen. Takes in a username and password that is passed to the Security class. Relies on UI such as IconLoader
- BankWindow, BuyWindow, InfoWindow, InvalidPopup, ManagerWindow, SellWindow, TradeWindow.
    - Classes for each type of window that appears in the program. Relies on a IntelliJ .form xml file. Each window additionally calls methods from DisplayAdaptor in order to display information, make buy and sell commands, and more.
- GUI
    - The GUI class contains a variety of methods for instantiating the Window classes. Furthermore, it keeps reference to existing windows to prevent redundancy (like opening a window multiple times)
- IconCreator
    - A file i/o for reading images to create Icon objects for the UI.
- StockManagerTableModel, DisplayTableModel
    - TableModels that are used by ManagerWindow and TradeWindow UI.

TXT Files Folder (Database):

- customer.txt:
    - contains a list of customers with contents being in the following order: Name, Portfolio StockList, Money, DerivativeAccount?.
        - Name: name of customer
        - Portfolio StockList: List of stocks in following format: [(stockName, quantity, datePurchased);....]
        - Money: amount of money customer has (adjustable with withdraw and deposit)
        - DerivativeAccount?: Boolean value of "Y" or "N" to indicate whether their liquid funds are sufficient (in comparison with the portfolio manager's threshold) to be able to create a derivative portfolio.
- stocks.txt:
    - contains a list of stocks with contents being in the following format: Stocks(name), Price, Count.
        - Stocks: Name of stock (written in short form)
        - Price: Current price of stock, adjustable by portfolio manager

        - Count: Number of stocks on the market, adjustable by portfolio manager
- users.txt:
    - contains a list of accounts used for login (connected with customers.txt) in the following format: name, password:
        - name: username used for a customer to login (or create an account).
        - password: password used for a customer to login (or create an account).

Customer class extends Person:
- class that maintains state necessary for a customer to conduct purchases and sells of stocks as well as cash withdrawals and deposits.
- It extends Person which has one data member as part of the common (between Customer and PortfolioManager) state: name.

Date class:
- class that is to be used as part of the extendibility of the project. This is used in stock purchases and sells, which we planned on using in generating graphs and reports of trends (to be implemented).

DisplayAdaptor class:
- The DisplayAdaptor serves as a bridge between the frontend and backend. All connections between front and back end passes through this class.

FileHandler class:
- class that handles all interactions between logic (back-end and database txt files), which includes both reads and writes.

MarketSystemFacade class:
- A facade pattern implementation that connects the display (DisplayAdaptor) with the back-end logic of the individual objects. This is used to hide/limit what the client can see/do, within our intended use.

PortfolioManager class extends Person:
- Extends person and defines the manager of the portfolio and their functionalities.

Person:
- This class is the overarching definition of a person which can be both a customer or a portfolio manager and sets the common attributes of both types of people.

Security class:
- This class defines how the user can create an account and log in. It also uses the DisplayAdaptor to send that information to the GUI.

Stock extends Observable implements Tradeable:
- class that implements the Observer pattern (its currentPrice is observed by Portfolio objects). It also maintains state about dates purchased and sold to be used when implementing the generation of reports or graphs that display trends (part of extendibility).

StockMarket class:
- This class holds a list of all the stocks and defines methods with the functionalities associated with the stocks such as purchasing and selling.

Tradeable interface:
- Interface that is implemented by Stock class, and it ensures that it has and reflects a currentPrice that can change (change is set by Portfolio manager).

TradingSystem class:
- Main class that is called to run the Trading system program. Calles FileHandler to load data from the DB (txt files) and then runs the display.