Stock Trading System

GitHub Repository URL

Group#17
Ella Li
Shiwen Fang
Cerina Yao

08/16/2023

Table of Work

(Please write x in the boxes to mention what each student achieved in this project)

	Student-1 Ella Li	Student-2 Shiwen Fang	Student-3 Cerina Yao
Project Description			x
Uses Cases Diagram(s)		x	x
Sequence Diagrams	x		x
Class diagram(s)		x	
Implementation	x	x	x
Conclusion	x		

Table of Contents

Terminology Glossary

Investor: A participant in the stock market who buys and sells stocks, places orders, and manages a portfolio.

Stock: A tradable financial instrument representing ownership in a company. It has a specific name, price, and volume.

Order: A request made by an investor to buy or sell a specific number of shares of a particular stock at a designated target price.

Order Book: A record of all buy and sell orders from investors, including details such as investor name, order type, stock name, target price, and volume.

Portfolio: The collection of stocks an investor owns and their corresponding volumes.

Market Price: The current price at which a stock trades in the market.

Target Price: The desired price an investor specifies in a buy or sell order. The order is executed when the market price reaches or surpasses this target price.

Buy Order: An order placed by an investor to purchase a specified number of shares of a stock at a target price.

Sell Order: An order placed by an investor to sell a specified number of shares of a stock at a target price.

Broker: A middleman who executes buy and sell orders on behalf of investors, facilitating trading in the stock market.

Stock Market: A virtual marketplace where stocks are bought and sold. It includes a collection of stocks available for trading.

Execute Order: The process of fulfilling a buy or sell order when the market price exceeds the target price specified by the investor.

Random Market Volatility: Simulated fluctuations in stock prices to replicate real-world market dynamics involving random adjustments within a predefined range.

ExecuteOrNot: A decision-making factor that determines whether an order is executed based on market conditions and the target price set by the investor.

System Analysis

Project Description

General Description, Goals, and Benefits:

The Stock Trading System is a software application designed to offer users a simulated yet immersive stock trading experience. Through an intuitive graphical interface, users can place buy and sell orders, manage their investment portfolios, and witness the execution of orders in a dynamic virtual stock market. The system's primary objectives are to provide interactive trading capabilities, replicate real-world market conditions with random volatility, and facilitate comprehensive portfolio management. By offering users a realistic platform to practice trading strategies, track investments, and observe the effects of market fluctuations, the Stock Trading System aims to enhance users' understanding of stock trading dynamics and investment decision-making.

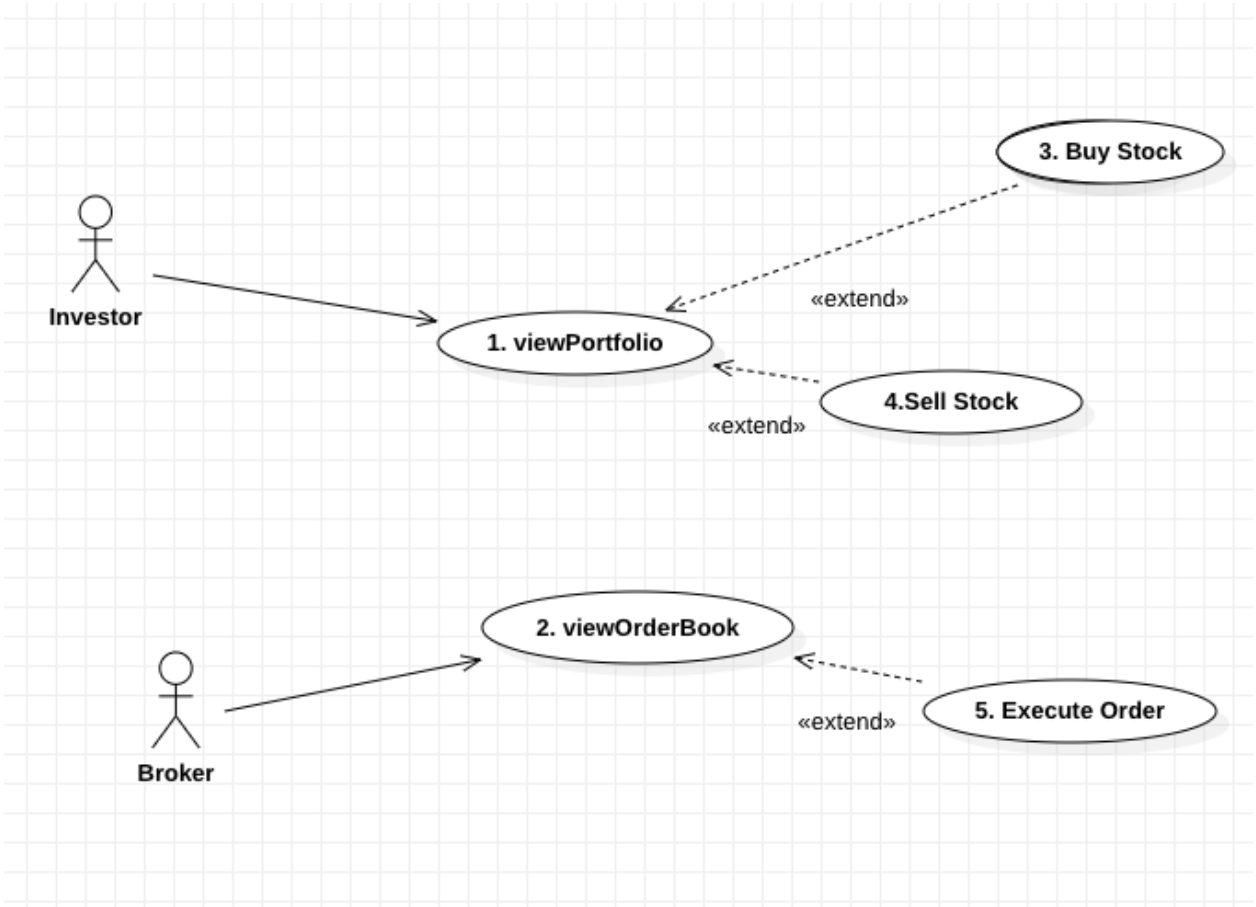
System input(s) and output(s)

The Stock Trading System allows users to input buy and sell orders, specifying stock names, order types, target prices, and share volume. Outputs include order execution status notifications indicating market-based execution outcomes. Portfolios are accessible, showing holdings with stock names, volumes, and current market values. The system also delivers market data, enabling users to observe dynamic stock prices and fluctuations due to random volatility.

Special Requirements (Performance, Interfaces, Constraints, Reliability, if any)

The Stock Trading System is designed with performance, ensuring responsive and smooth interaction even during market volatility and concurrent order execution. The user-friendly GUI simplifies the process of placing orders, viewing portfolios, and receiving execution notifications. However, it's important to note that the system is a simplified simulation and does not reflect real-time market conditions. The system's reliability is emphasized, handling errors gracefully and providing clear error messages for invalid inputs or stock market connection issues. Data persistence is maintained through serialization and deserialization, enabling the system to retain user data, order history, and stock market information across multiple sessions.

Uses Cases Diagram(s) and use case description



UC Reference Name/Number: 1. View Portfolio	
Overview	An investor will be able to view his/her portfolio where all stocks he/she has and all stocks in the market will be shown with detailed information such as its name, price, and volume.
Related use cases	Extended by 3. Buy Stock and 4. Sell Stock.
Actors	Investor
Pre Conditions(Optional)	The user clicks the investor button and enters the investor portal.
Post Conditions(Optional)	Investor object will be read from the portfolio file and displayed

UC Reference Name/Number: 2. View OrderBook	
Overview	A Broker can view all the orders placed by the investors he/she is in charge of.

Related use cases	Extended by 5. Execute Order
Actors	Broker
Pre Conditions(Optional)	The user clicks the investor button and enters the broker portal
Post Conditions(Optional)	Investors' order books will be read from the order book and stock market file and got displayed

UC Reference Name/Number: 3. Buy Stock	
Overview	An investor will be able to view the current Stock Market and then place orders to buy a specific amount of chosen stocks at their target prices.
Related use cases	Extends 1. View Portfolio
Actors	Investor
Pre Conditions(Optional)	The user clicks the investor button, enters the investor portal, and then clicks the Buy Stock Button.
Post Conditions(Optional)	A new buy order will be created and added to the investor object and stored in the stock market file

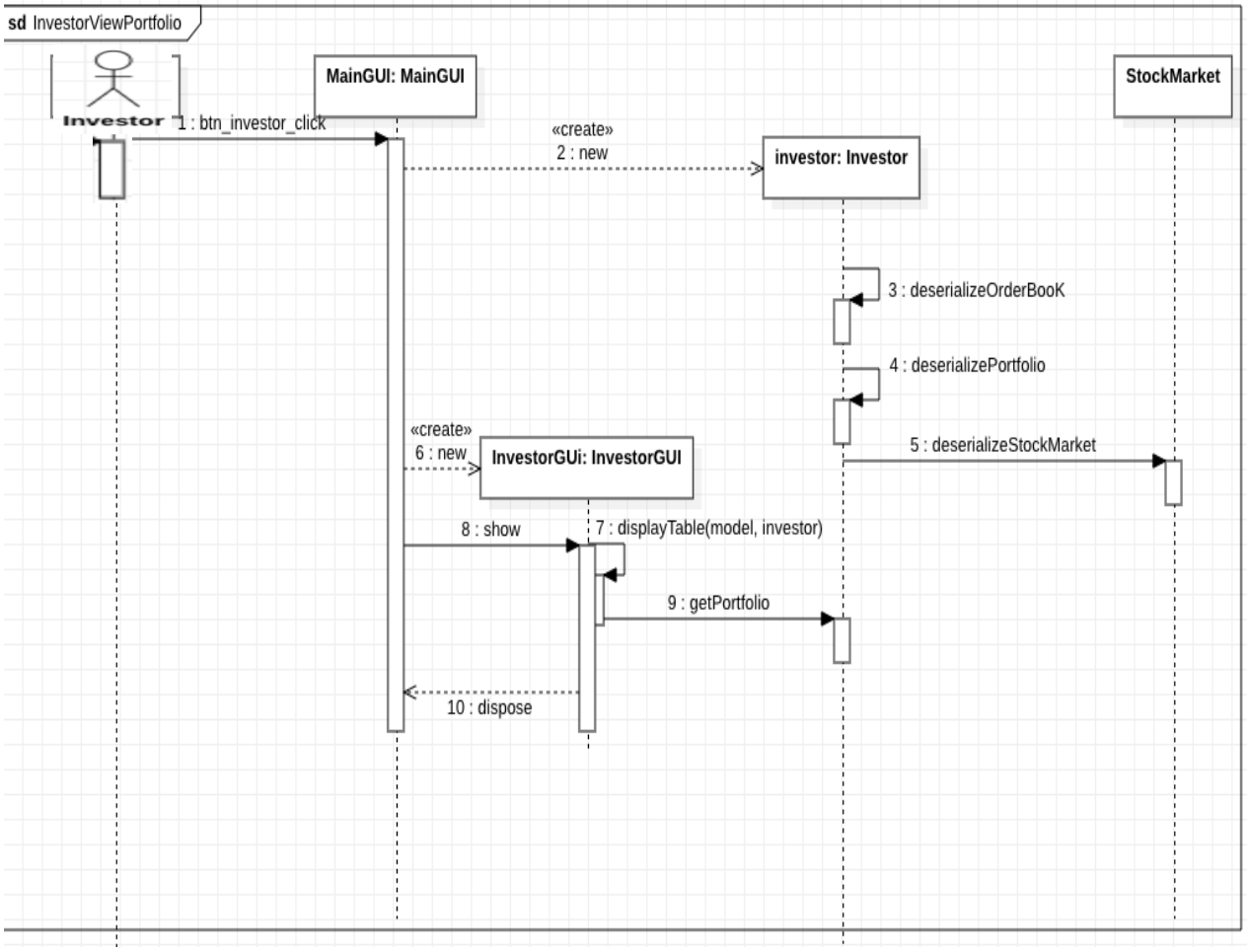
UC Reference Name/Number: 4. Sell Stock	
Overview	An investor can place orders to sell a specific amount of chosen stocks he/she owns at their target prices.
Related use cases	Extends 1. View Portfolio
Actors	Investor
Pre Conditions(Optional)	The user clicks the investor button and logs in as an investor. He/she has selected one stock he/she owns and wants to sell by clicking on that stock in a table and then clicking the Sell Stock Button.
Post Conditions(Optional)	A new sell order will be created and added to the investor object and stored in the portfolio file.

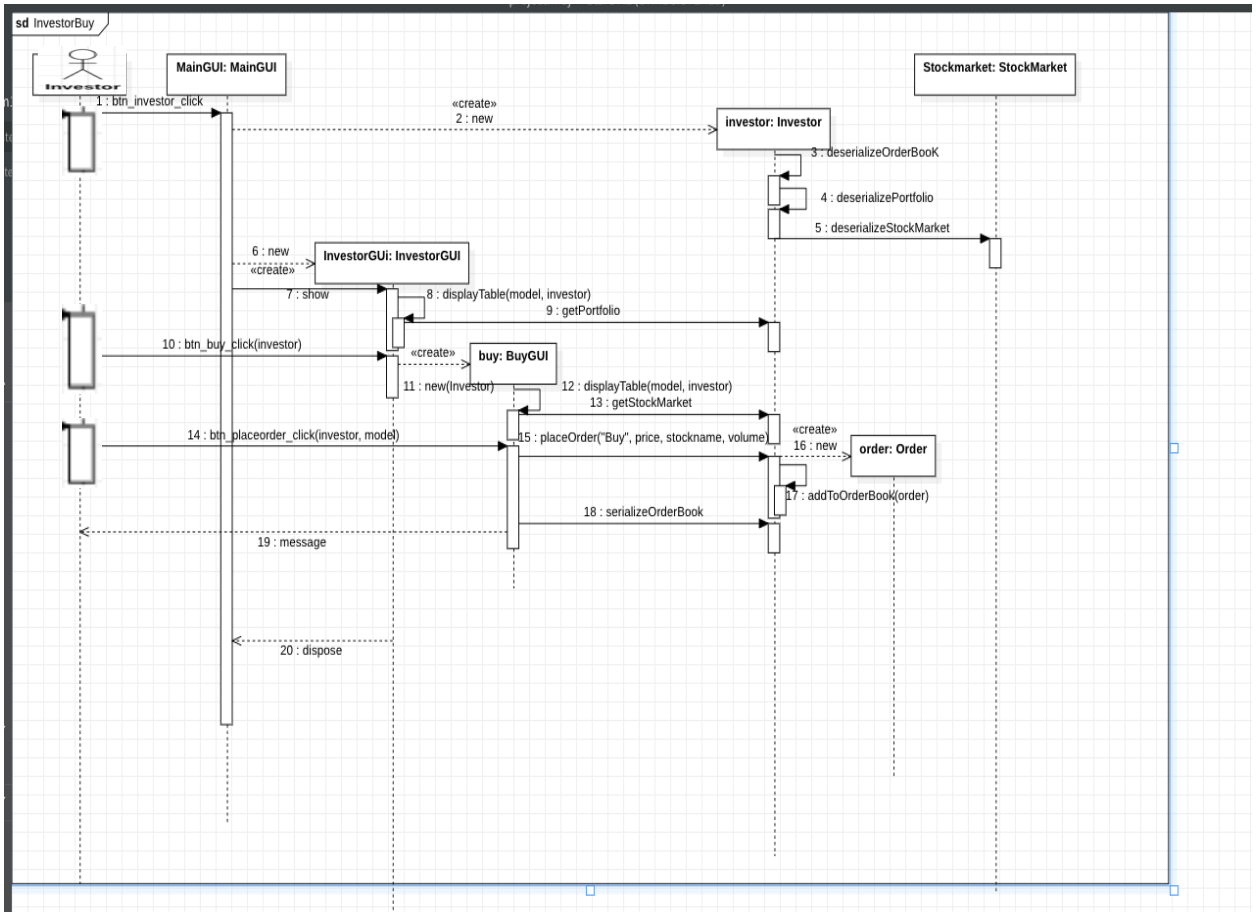
UC Reference Name/Number: 5. Execute Order	
Overview	A Broker will be able to execute the orders placed by the investors he/she is in charge of when they satisfy their price condition and refresh the market information to keep it up to date (randomize price of each stock in the market at a reasonable scale to simulate the fluctuation of the market)
Related use cases	Extends 2. View Order Book
Actors	Investor
Pre Conditions(Optional)	The user clicks the investor button and enters the broker portal, and clicks the Execute Order Button. Also, there should be at least one order not yet executed in the orderbook

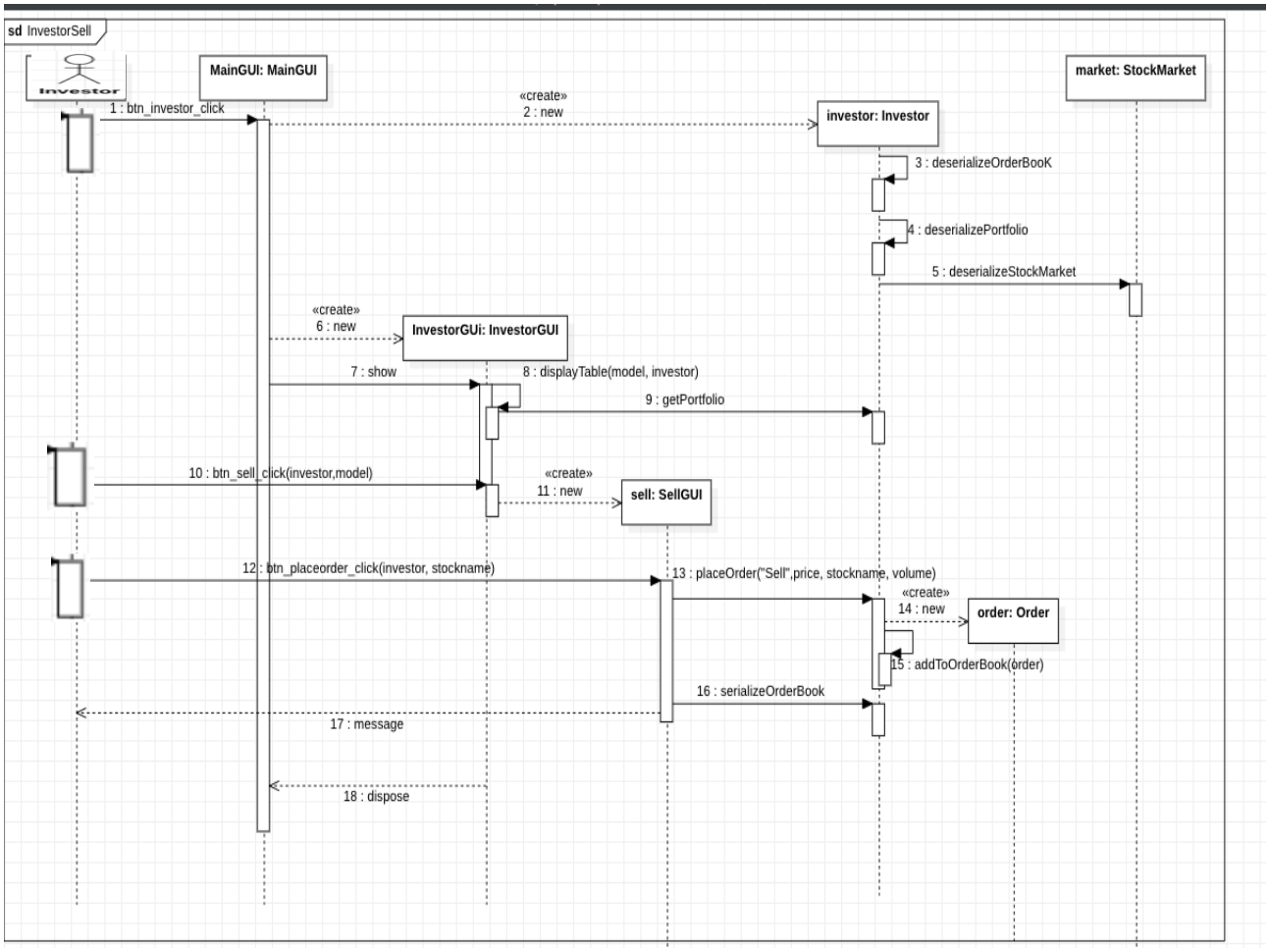
Post Conditions(Optional)	The orders that satisfy the price conditions will be executed. The execution of an order will, depending on its type (“BUY” OR “SELL”), add/remove an amount of stock to/from the investor’s portfolio and remove/add that amount of stock from/to the Stock Market. The market price of that stock will fluctuate. Then the order will be deleted from the order book. At last, all changes will be stored in the storage files.

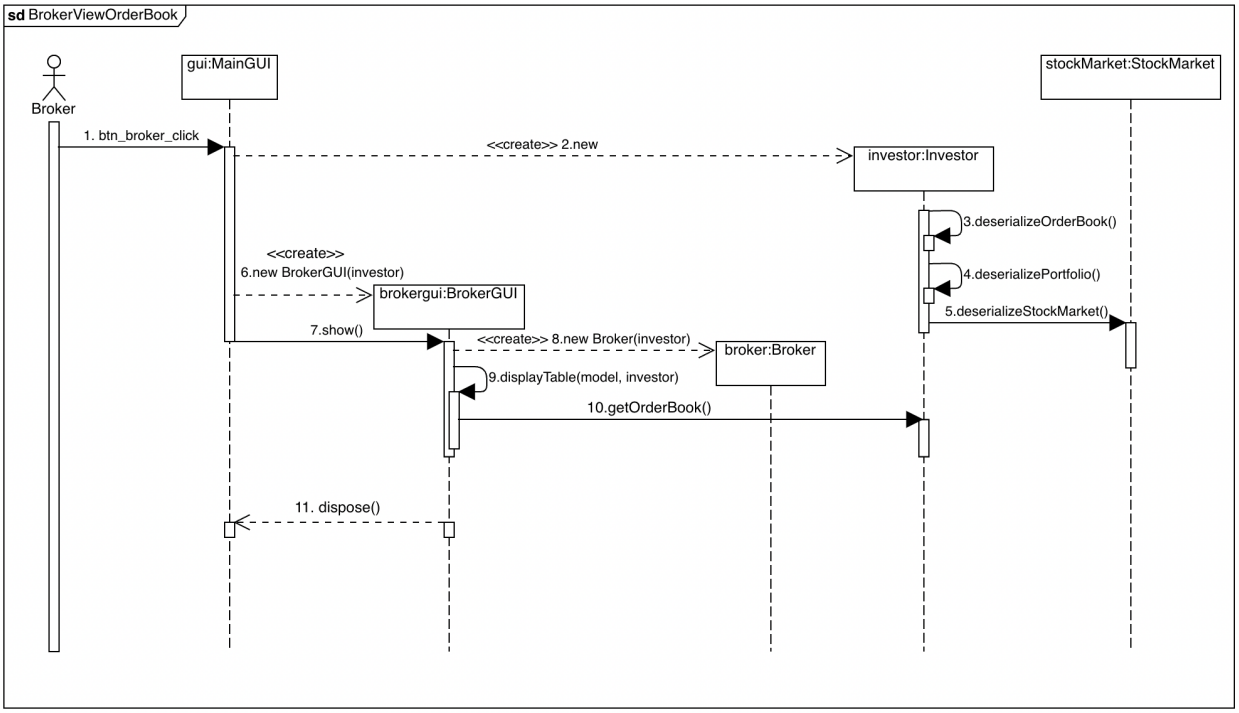
System Design

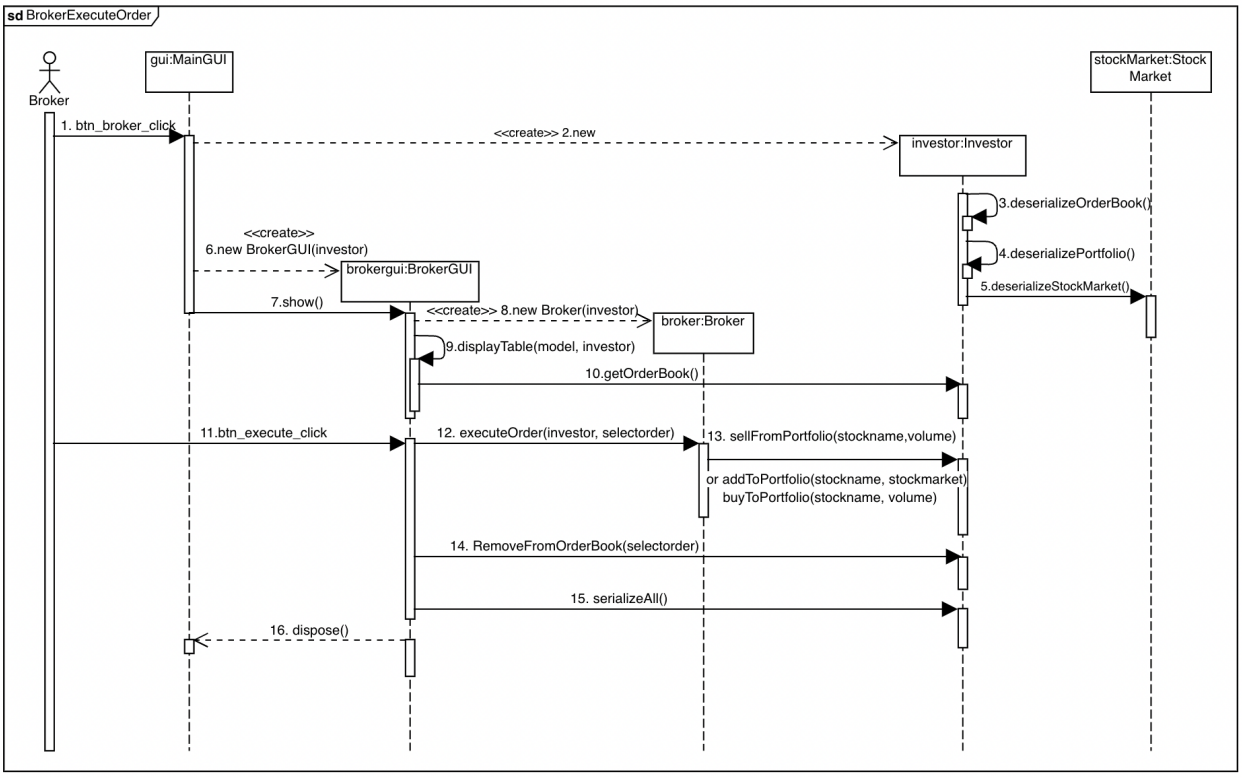
Sequence Diagrams



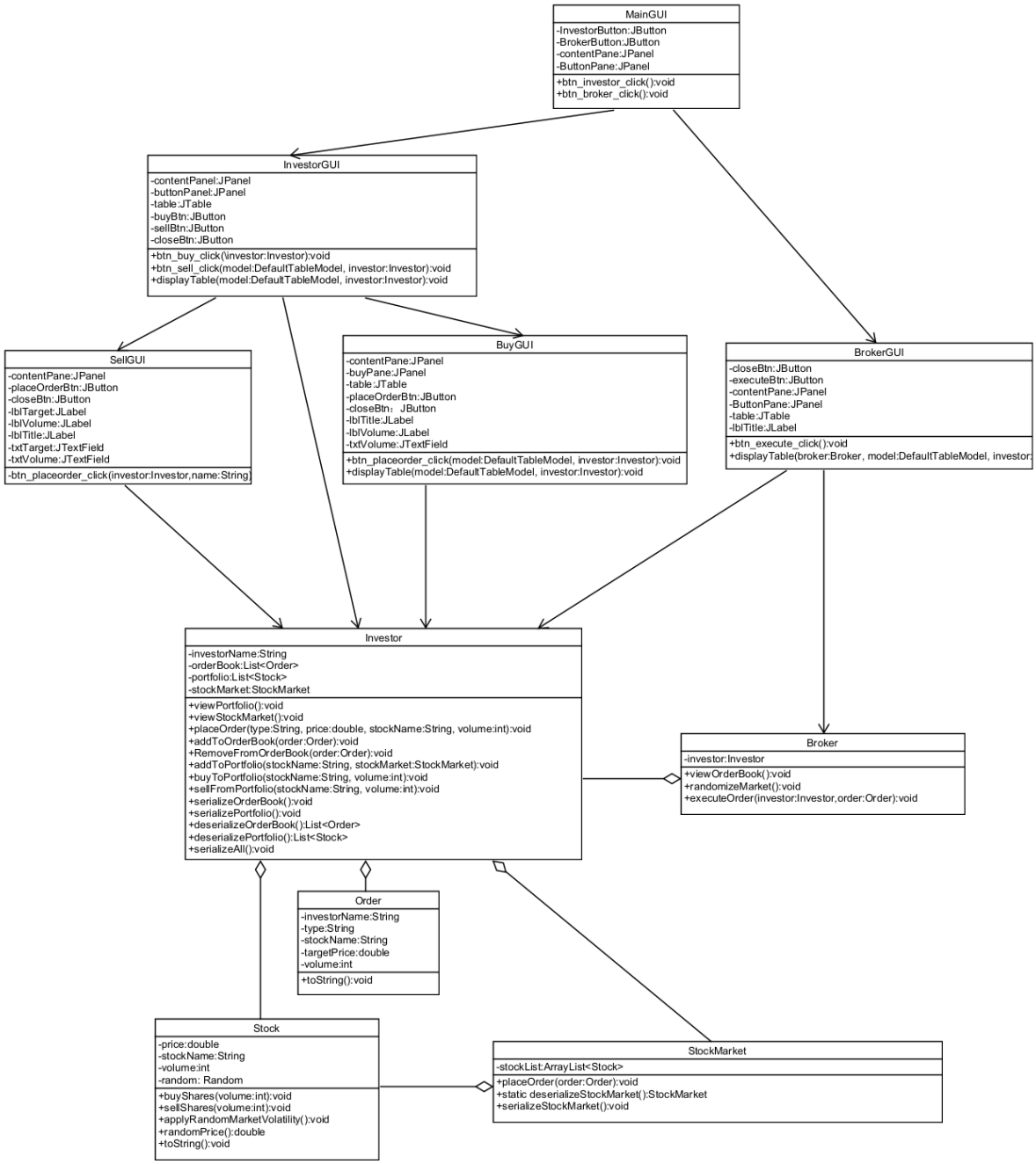

sd InvestorBuy
Investor
MainGUI: MainGUI
Stockmarket: StockMarket
1 : btn_investor_click
«create»
2 : new
investor: Investor
3 : deserializeOrderBooK
4 : deserializePortfolio
5 : deserializeStockMarket
6 : new
«create»
InvestorGUi: InvestorGUI
7 : show
8 : displayTable(model, investor)
9 : getPortfolio
10 : btn_buy_click(investor)
«create»
buy: BuyGUI
11 : new(Investor)
12 : displayTable(model, investor)
13 : getStockMarket
14 : btn_placeorder_click(investor, model)
15 : placeOrder("Buy", price, stockname, volume)
«create»
16 : new
order: Order
17 : addToOrderBook(order)
18 : serializeOrderBook
19 : message
20 : dispose


sd InvestorSell
Investor
MainGUI: MainGUI
market: StockMarket
1 : btn_investor_click
«create»
2 : new
investor: Investor
3 : deserializeOrderBooK
4 : deserializePortfolio
5 : deserializeStockMarket
«create»
6 : new
InvestorGUi: InvestorGUI
7 : show
8 : displayTable(model, investor)
9 : getPortfolio
10 : btn_sell_click(investor,model)
«create»
11 : new
sell: SellGUI
12 : btn_placeorder_click(investor, stockname)
13 : placeOrder("Sell",price, stockname, volume)
«create»
14 : new
order: Order
15 : addToOrderBook(order)
16 : serializeOrderBook
17 : message
18 : dispose


sd BrokerViewOrderBook
Broker
gui:MainGUI
stockMarket:StockMarket
1. btn_broker_click
<<create>> 2.new
investor:Investor
3.deserializeOrderBook()
<<create>>
6.new BrokerGUI(investor)
brokergui:BrokerGUI
4.deserializePortfolio()
5.deserializeStockMarket()
7.show()
<<create>> 8.new Broker(investor)
broker:Broker
9.displayTable(model, investor)
10.getOrderBook()
11. dispose()


sd BrokerExecuteOrder
Broker
gui:MainGUI
stockMarket:Stock Market
1. btn_broker_click
<<create>> 2.new
investor:Investor
3.deserializeOrderBook()
<<create>>
6.new BrokerGUI(investor)
brokergui:BrokerGUI
4.deserializePortfolio()
5.deserializeStockMarket()
7.show()
<<create>> 8.new Broker(investor)
broker:Broker
9.displayTable(model, investor)
10.getOrderBook()
11.btn_execute_click
12. executeOrder(investor, selectorder)
13. sellFromPortfolio(stockname,volume)
or addToPortfolio(stockname, stockmarket)
buyToPortfolio(stockname, volume)
14. RemoveFromOrderBook(selectorder)
15. serializeAll()
16. dispose()

Class Diagram



Conclusion

The stock trading system provides a good simulation of the functions available to investors and brokers in the stock trading platform. In the process of designing this system, we implemented GUI to realize the interaction with the user, carefully designed the trading interface for different characters, and provided messages guiding the user to operate smoothly. To make each part of the system independent and mutually invokable, we designed a series of classes such as Investor, Broker, Order, Stock, StockMarket, etc., which make the system flexible and maintainable, conforming to the idea of object-oriented programming effectively. While we have encountered some problems in the writing process, such as being unable to access the real-time market database. We actively looked for solutions and finally solved the problem by initializing the market data automatically and randomly generating market prices within a reasonable range. Although our system can not fully reflect real market fluctuations, it is still a reliable simulation of the basic stock trading process, dedicated to familiarizing the user with the operation of the stock trading dynamics and understanding investment decision-making.