Group 5

Software Engineering Report 3

Stock Market Fantasy Game

Presented by:

Alex Sood, John Grun, Kevin Folinus, & Chris Zalewski

Sunday, May 3rd, 2009

	Kevin Folinus	John Grun	Alex Sood	Chris Zalewski
Cover Page	%100			
Table of Contents	%50			%50
Summary of Changes	%50			%50
Customer Statement of Requirements	%100			
Glossary of Terms	%100			
Functional Requirements Specifications		%50	%50	
Nonfunctional Requirements				%100
Domain Analysis			%100	
Interaction Diagrams		%100		
Class Diagram and Interface Specifications		%50	%50	
System Architecture and System Design		%50	%50	
Algorithms and Data Structures	%25			%75
User Interface Design and Implementation	%60			%40
History of Work & Current Status of Imp.	%25	%25	%25	%25
Conclusions and Future Work	%25	%25	%25	%25
References	%25	%25	%25	%25

Table of Contents

3. Summary of Changes

Domain Model

User Interface
- -Screen Mockups
- -User Effort Estimation

Use Case Diagram (Made visible)

Interaction Diagrams

Class Diagram and Interface Specification
- -Class Diagram

System Architecture and System Design
- -Identifying subsystems / Architectural Styles

4. Customer Statement of Requirements

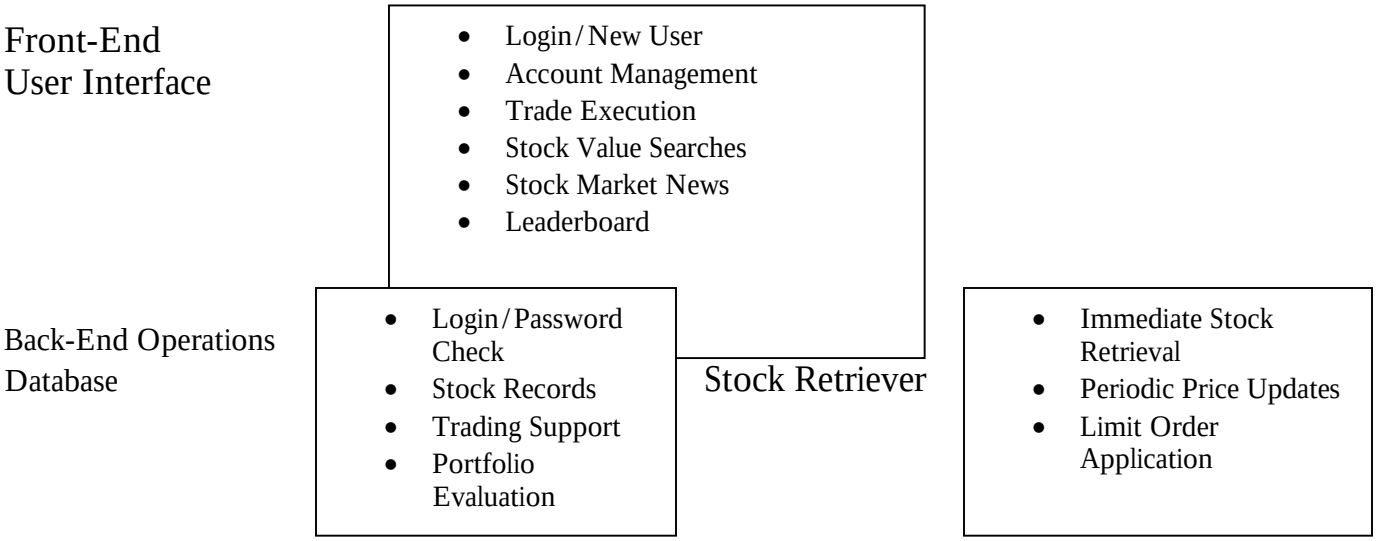
Kenneth Lay
Enron Corporation

Dear Group 5 Programmers,

The current state of the economy has brought it to our attention the public's need for a tool that will teach the fundamentals of investing. We are excited here at Enron Corp. to be working with your software engineering team to create an application that will satisfy this need. The application to be developed should create a virtual stock market. This virtual stock market will be a fantasy league in which users can compete against each other to improve their investing prowess. The final design will mimic real stock brokerage services for the self-directed investor, such as E-Trader. It will, however, not deal with real money and instead be a platform for individuals to test their investing capabilities in the stock market against one another without risk.

In order to create an easy to use application it should not be a website and instead run off an individual's desktop. This will enable them to virtually participate against users on other stations. Users who participate in the stock market league should have accounts that are specific to them, with a logon and password for security. The users should be given a specific amount of virtual money when they first join. They will then use this money and make real time investments in the stock markets in an attempt to out invest their competition. A real time investment means that any stock purchased or sold will be done so at the current market value.

To create a program that satisfies these user specifications we believe there will be three main components. These components are, one front-end and two back-end items. The front-end specification is the user interface. This will enable the user to control his account and the investment decisions he wants to make. The projects two back-end items are the back-end database and the code to access the real time stock prices. With these three items in place users will be able to buy and sell securities via your electronic trading platform.



The simplest aggregate of means by which users interact with a piece of software is the user interface. The user interface is the front-end to your application. In your program the first form this will take, when individuals fist access the stock market fantasy league, will be a login screen. They will be faced with two options, log on using an existing account or to create a new account. When the user types in an incorrect account name or password an error will be returned and the user will get to try again. If they choose to create a new account, their new account will be added to the database. The second user interface gives the users the ability to make stock interactions. The current plan is to give the user the ability to check a ticker price, buy, sell, view leader board, and view the current stocks they own.

A primary aspect to your application will be the back-end database. This relational database will use a schema in which two are connected through a primary key. The primary key will be the username.

This database schema satisfies the user requirements by enabling individuals to invest with their personal account, keeping it separate from other users. With a primary key that is checked against a password we eliminate the possibility that your code has an error causing multiple users to accounts to get modified by the incorrect user. This functionality is inherently critical in a stock market virtualization for security purposes. The back-end database is a crucial user requirement that will never be visible to the user themselves. A key tool in implementing the user requirements is the programs ability to retrieve real time stock prices, at intervals and upon request. This is your second back-end item in your application. Without this the program would not be able to run a realistic stock market virtualization. When a user enters a stock ticker they would like to buy, sell, or view, your application will return the real value from yahoo finance. In addition to this stock ticker pull from the internet, later builds of your code will attempt to incorporate RSS news feeds related to the ticker inquired to give additional information to the user.

These three items come together to create an environment that satisfies all of the customer requirements. When a user logs in they will have access to their specific stock history and account balance. When a user goes to use an option from the user interface both the database and stock price updater code will be used to create a seamless runtime virtual stock market environment satisfying all of the customer needs.

Congratulations again on being chosen to develop this ground breaking software and strengthening the economic future alongside Enron Corporation.

5. Glossary of Terms

Login Database – This is the database containing the users log in and password.

Stock Database – This is the database containing the stocks owned by individual investors.

Login Interface – The screen users use to log in or create a new account.

Functional Interface – The primary interface used to interact with the stock market.

Yahoo Finance Library – Library which parses Yahoo Finance with Tickers for stock values.

Portfolio - All of the assets held by the current user.

Timer Interval - Time between updates to stock ticker prices on database.

Transaction – A transaction is when a security is either purchased or sold.

Security – In our program this will only be used to represent stocks.

Buy – Purchasing a stock at the current market value.

Sell – Selling a stock at the current market value.

Trade – A securities transaction.

Ticker – The symbol used to represent a company in the stock market.

Quantity – The amount of shares of a stock bought or sold.

Virtual Wallet – The amount of virtual money a participant has to spend on securities.

Thread- fork of a computer program into two or more concurrently running tasks.
.

6. Functional Requirements Specification

6a Stakeholders

Potential stakeholders in this project include people who enjoy online fantasy games, people who want to test various investment strategies before assuming the actual risk of investing, as well as anyone who wants to learn the basics of how investing works. Additionally, educators are potential stakeholders as they may wish to use this product to teach students about the stock market. The nature of this product also makes it an excellent spot for investment firms to place advertisements, so they hold a stake in this as well.

6b Actors and Goals

Actor 1: New Player

Goal: To create a new account and start playing the game.

Use Case: Initiates Create Account (UC-1).

Actor 2: Finished Player

Goal: To quit playing the game.

Use Case: Initiates Remove Account (UC-2).

Actor 3: Player

Goal 1: To login to an existing player account.

Use Case: Initiates Login (UC-3).

Goal 2: To view the stocks and cash currently owned by this player.

Use Case: Initiates View Portfolio (UC-4).

Goal 3: To purchase shares of a stock.

Use Case: Initiates Buy/Sell Stock (UC-5).

Goal 4: To sell currently owned shares of a stock.

Use Case: Initiates UC-5.

Goal 5: To view the current price of a stock.

Use Case: Initiates View Stock Price (UC-7).

Goal 6: To view additional information regarding a stock.

Use Case: Initiates View Stock Details (UC-8).

Goal 7: To compare the value of this player's portfolio to the highest valued portfolios in the game.

Use Case: Initiates View Leaderboard (UC-9).

Goal 8: To set price thresholds for the player to be automatically notified when a certain stock exceeds them.

Use Case: Initiates Set Alert Thresholds (UC-10).

Goal 9: To view an alert regarding a stock price.

Use Case: Participates in Send Alert (UC-11).

Goal 10: To logout of the current account.

Use Case: Initiates Logout (UC-12).

Actor 4: Account Database

Goal 1: To insure that a new account does not share a name with an existing account and add the new account to the database.

Use Case: Participates in UC-1.

Goal 2: To completely remove all information of the account being deleted.

Use Case: Participates in UC-2.

Goal 3: To verify that the username-password combination matches an existing account and grant access to that account.

Use Case: Participates in UC-3.

Goal 4: To provide accurate information regarding the contents and/or value of player portfolios.

Use Cases: Participates in UC-4 and UC-9.

Goal 5: To update information regarding the contents and/or value of a player's portfolio.

Use Cases: Participates in UC-5 and UC-14.

Goal 6: To provide accurate information regarding a stock price.

Use Case: Initiates Update Price (UC-13) and UC-11. Participates in UC-5, UC-14, and UC-7.

Actor 5: Cell Phone

Goal: To receive an alert regarding a stock price.

Use Case: Participates in UC-11.

Actor 6: Yahoo! Finance

Goal: To provide up-to-date information regarding stock prices.

Use Cases: Participates in UC-8, and UC-13.

6c. Use Cases

The final product will implement the following 13 use cases:

Create Account (UC-1): This use case allows a new user to create an account with a selected username and password. A new portfolio containing only cash is created for that account. UC-3 is also carried out automatically after the account has been created.

Remove Account (UC-2): This use case allows the user to delete the account to which he or she is currently logged in. All of the account information in the database is deleted, and UC-12 is automatically carried out.

Login (UC-3): This use case allows the user to access an account if he or she provides the correct username and password. In that case, UC-4 is carried out. If incorrect login information is supplied, the system displays an error message to notify the user.

View Portfolio (UC-4) (sub-use case): This use case allows the user to view the contents and value of his or her portfolio. This use case is not directly selected by the user; it is automatically carried out when the user initiates UC-3.

Buy/Sell Stock (UC-5): This use case allows the user to place an order to buy or sell stock. The user enter the stock ticker and number of shares and chooses between placing a market order, limit order, stop loss order, or a trailing loss order. If the user selects a type other than market, the systems prompts the user for the price/percentage drop at which to execute the order. If the user enters invalid information, an error message is displayed.

View Stock Price (UC-7): This use case allows the user to view the current price of a given stock. The user enters the company name or ticker symbol for the stock they wish to track, and the price of the stock is displayed along with how much the price has changed in the day's trading. These numbers are automatically updated. If the user enters an invalid company name of ticker symbol, an error message is displayed.

View Stock Details (UC-8): This use case allows the user to view additional information about a stock. The user enters the company name or stock ticker symbol and is provided with an interactive display which the history of the stock price and or trade volume over a selected period of time. If the user enters an invalid company name of ticker symbol, an error message is displayed.

View Leaderboard (UC-9): This use case allows the user to see who has the highest valued portfolios. The user selects an integer, n, and the usernames and values corresponding to the highest n valued portfolios are displayed. If the user is not among the leaders, his or her rank will be at the bottom of the displayed.

Set Alert Thresholds (UC-10): This use case allows the user to set thresholds for selected stocks, so that he or she will be notified if the stock price crosses that threshold. The user enters a stock name along with a lower and/or upper threshold. The user must also enter a cell phone number for the alert to be sent to.

Send Alert (UC-11): This use case allows the user to be notified when alert thresholds are crossed. When the system finds that a threshold has been crossed, it sends an alert to the cell phone number provided by the user.

Logout (UC-12): This use case allows the user to log out of the account to which he or she is currently logged in. The user clicks a logout button, and the system returns to the login screen. However, screens opened with UC-7 and UC-8 will remain open.

Update Price (UC-13): This use case is initiated periodically by the Account Database in order to keep its list of stock prices up-to-date. Current stock prices are retrieved from Yahoo! Finance.

Fill Orders (UC-14): This use case allows for the completion of buy/sell orders placed by the user and is indirectly initiated when such orders are placed via UC-5. If it is not possible to complete an order, the system displays an error message informing the user, and the order is removed from the list.

Why are Login and Logout use cases?: The action of logging in is not ordinarily considered a use case because the user typically has some other goal in mind and does not plan on just logging in. However, that is not necessarily the case with this system. A competitive player who knows how quickly stock prices can change may wish to login just to ensure quick access to the account later. In this case, logging in is achieving a goal of the user by itself and should therefore be considered a use case. Similarly, logging out achieves the goal of a security-concious player to protect his or her account from unauthorized access.

We will now proceed with a more detailed description of all use cases.

Use Case UC-1: Create Account

Related Requirements:

Initiating Actor: New Player

Actor's Goal: To create a new account with a new portfolio.

Participating Actors: Account Database

Preconditions: None.

Postconditions: 1. A new account is created with the username and password supplied by the New Player.

2. A new portfolio containing only cash is created for the new account.

3. All postconditions of UC-3.

Flow of Events for Main Success Scenario:

1. System prompts the New Player for a username and password.

2. New Player enters a username and password.

3. System sends username to the Account Database.

4. Account Database confirms that the username is not taken.

5. System enters username and password into the Account Database.

6. System enters the starting amount of cash into username's account in the Account Database.

7. System displays account screen with buttons to initiate other use cases.

8. include:: View Portfolio (UC-4).

Flow of Events for Extensions:

2. New Player fails to enter either username or password.

1. System displays error message telling New Player enter the missing information.

2. Start over from Step 1.

4. Account Database finds that the username is taken.

1. System displays an error message telling New Player that the chosen username is already taken.

2. Start over from Step 1.

Use Case UC-2: Remove Account

Related Requirements:

Initiating Actor: Finished Player

Actor's Goal: To delete his or her current account.

Participating Actors: Account Database

Preconditions: 1. The Finished Player is logged in to an existing account.

Postconditions: 1. All information regarding the account is question is deleted from the Account Database.

2. All postconditions of UC-12.

Flow of Events for Main Success Scenario:

1. Player select remove account.

2. System removes all account information from Account Database.

3. include:: Logout (UC-12).

Flow of Events for Extensions: None

Use Case UC-3: Login

Related Requirements:

Initiating Actor: Player

Actor's Goal: To access an existing account.

Participating Actors: Account Database

Preconditions: 1. The Account Database is non-empty.

Postconditions: 1. The System displays the account screen which provides buttons for Player to initiate all other use cases except UC-1 and UC-4.

2. Also displayed on the account screen are the current contents and value of Player's portfolio.

Flow of Events for Main Success Scenario:

1. System prompts Player for a username and password.

2. Player enters a correct username and password.

3. System sends the username and password to Account Database.

4. Account Database confirms that the username exists and the password is correct.

5. System retrieves the contents of Player's portfolio from Account Database.

6. System displays account screen with buttons to initiate other use cases.

7. include:: View Portfolio (UC-4).

Flow of Events for Extensions:

2a. Player fails to enter either username or password.

1. System displays an error message telling Player to enter the missing information.

2. Start over from Step 1.

2b. Player enters nonexistent username or incorrect password.

1. System sends the username and password to Account Database.

2. Account Database finds that the username is nonexistent or the password is incorrect.

3. System displays an error message telling Player that the login information was incorrect.

4. Start over from Step 1.

Use Case UC-4: View Portfolio (sub-use case)

Related Requirements:

Initiating Actor: Player

Actor's Goal: To view the contents and/or value of his or her portfolio.

Participating Actors: Account Database

Preconditions: 1. Player has initiated either UC-3, UC-5.

Postconditions: 1. The contents and value of the portfolio is displayed on the account screen.

Flow of Events for Main Success Scenario:

1. System prepares a list of stocks owned and the number of shares owned of each stock.

2. System retrieves the current price of each stock from Account Database and calculates the value of the portfolio.

3. System displays the list of stocks, cash balance, and total value of the portfolio on the account screen.

Flow of Events for Extensions: None.

Use Case UC-5: Buy/Sell Stock

Related Requirements:

Initiating Actor: Player

Actor's Goal: To purchase shares of a stock.

Participating Actors: Account Database

Preconditions: 1. Player is logged into an existing account.

2. Player's cash balance is non-zero.

Postconditions: 1. Buy/Sell order is placed in Player's order list.

Flow of Events for Main Success Scenario:

1. System prompts Player for a stock name, number of shares to buy and order type.

2. Player enters a valid ticker symbol, an affordable number of shares, and selects order type.

3. System prompts Player for additional information required by order type.

4. Player enters additional information.

5. System places order in order list.

Flow of Events for Extensions:

2a. Player fails to enter either stock name or number of shares.

1. System displays an error message telling Player to enter the missing information.

2. Start over from Step 1.

2b. Player enters invalid company name or ticker symbol.

1. System attempts to retrieve the stock price from Account Database.

2. Account Database cannot find a match for the stock name.

3. System displays an error message telling Player that the stock name is invalid.

4. Start over from Step 1.

4a. Player fails to enter additional information.

1.System displays an error message telling Player that the additional information is required.

2. Start over from Step 4.

4b. Player enter invalid information.

1. System displays an error message telling Player that the information is invalid.

Use Case UC-7: View Stock Price

Related Requirements:

Initiating Actor: Player

Actor's Goal: To view the current price of a stock.

Participating Actors: Account Database

Preconditions: 1. The window for viewing stock prices has been opened.

Postconditions: 2. The current price of the stock and the change in the price for the current day are displayed in the viewing window.

Flow of Events for Main Success Scenario:

1. System prompts Player for a stock name.

2. Player provides a valid stock name.

3. System asks Account Database for the current price and change for the entered stock.

4. Account Database finds the stock on its list and returns the current price and change.

5. System displays the stock ticker symbol along with its price and change.

6. Steps 3-5 are repeated periodically to keep the displayed price and change current.

Flow of Events for Extensions:

2a. Player fails to enter a stock name.

1. System displays an error message telling Player to enter a stock name.

2. Start over from Step 1.

2b. Player enters an invalid stock name.

1. System asks Account Database for the current price and change for the entered stock.

2. Account Database fails to find the stock on its list.

3. System displays an error message telling Player that the entered stock name is invalid.

4. Start over from Step 1.

Use Case UC-8: View Stock Details

Related Requirements:

Initiating Actor: Player

Actor's Goal: To view additional information regarding a particular stock.

Participating Actors: Yahoo! Finance, Account Database

Preconditions: 1. Player is logged into an existing account.

Postconditions: 2. A graph showing how the stock price has changed over a period of time is displayed.

Flow of Events for Main Success Scenario:

1. Player selects View Stock Details.

2. System prompts Player for stock ticker and time period.

3. Player enters stock ticker and time period.

4. System asks Account Database if stock ticker is valid.

5. Account Database confirms that stock ticker is valid.

6. System retrieves graph from Yahoo! Finance.

7. System displays graph.

Flow of Events for Extensions:

3a. Player enters invalid stock ticker.

1. System asks Account Database if stock ticker is valid.

2. Account Database fails to find stock ticker in list.

3. System displays an error message telling Player that the entered stock ticker is invalid.

4. Start over from Step 3.

Use Case UC-9: View Leaderboard

Related Requirements:

Initiating Actor: Player

Actor's Goal: To compare the value of his/her own portfolio to others.

Participating Actors: Account Database.

Preconditions: 1. Player is logged into an existing account.

Postconditions: 1. A list of the top 10 valued portfolios is displayed along with Player's rank.

Flow of Events for Main Success Scenario:

1. Player selects View Leaderboard.

2. System asks Account Database for complete player standings.

3. Account Database returns complete player standings.

4. System displays list of top 10 valued portfolios.

5. System displays Player's rank at the bottom of the list.

Flow of Events for Extensions: None.

Use Case UC-10: Set Alert Thresholds

Related Requirements:

Initiating Actor: Player

Actor's Goal: To make sure he/she is notified when a stock price exceeds a certain threshold.

Participating Actors: Account Database, Cell Phone.

Preconditions: 1. Player is logged into an existing account.

Postconditions: 1. Stock ticker, threshold price, upper or lower limit, and phone number are entered into Account Database.

Flow of Events for Main Success Scenario:

1. Player selects Set Alert.

2. System prompts Player for stock ticker, threshold price, whether the threshold is an upper or lower limit, and phone number.

3. Player enters valid information.

4. System asks Account Database for current price of stock.

5. Account Database returns current stock price.

6. System verifies that the threshold has not already been exceeded.

7. System sends randomly generated code to Player's Cell Phone.

8. System prompts Player for code.

9. Player enters the correct code.

10. System enters stock ticker, threshold price, upper or lower limit, and phone number into Account Database.

Flow of Events for Extensions:

3a. Player enters invalid stock ticker.

1. System asks Account Database for current stock price.

2. Account Database fails to find stock ticker in list.

3. System displays an error message telling Player stock ticker is invalid.

4. Start over from Step 3.

3b. Player enters negative threshold price.

1. System displays an error message telling Player that the threshold price must be positive.

2. Start over from Step 3.

6a. System finds that the threshold has already been exceeded.

1. System displays an error message telling Player that the threshold has already been exceeded.

2. Start over from Step 3.

9a. Player enters incorrect code.

1. System displays an error message telling Player the code is incorrect and to make sure the cell phone number entered was correct.

2. Start over from Step 3.

Use Case UC-11: Send Alert

Related Requirements:

Initiating Actor: Account Database

Actor's Goal: To provide information about a stock price.

Participating Actors: Cell Phone, Player.

Preconditions: 1. Account Database contains at least one alert threshold which has been exceeded.

Postconditions: 1. Cell Phone displays a message telling Player the threshold has been exceeded.

2. The alert threshold is removed from Account Database.

Flow of Events for Main Success Scenario:

1. Account Database notifies System that a threshold has been exceeded and provides System with phone number, stock ticker, threshold price, and whether it was an upper or lower limit.

2. System sends message to Cell Phone saying that the threshold has been exceeded.

3. System removes alert threshold from Account Database.

4. Cell Phone displays message to Player.

Flow of Events for Extensions: None.

Use Case UC-12: Logout

Related Requirements:

Initiating Actor: Player

Actor's Goal: To log out of the current account.

Participating Actors: None.

Preconditions: 1. Player is logged into an existing account.

Postconditions: 1. The account screen is closed and the login screen is reopened.

Flow of Events for Main Success Scenario:

1. Player clicks logout button.

2. System closes the account screen.

3. System opens login screen prompting Player for username and password.

Flow of Events for Extensions: None

Use Case UC-13: Update Price

Related Requirements:

Initiating Actor: Account Database

Actor's Goal: To keep its list of stock prices up-to-date.

Participating Actors: Yahoo! Finance

Preconditions: 1. Account Database contains a non-empty list of stocks and their prices.

Postconditions: 1. The price of each stock is replaced with a new value from Yahoo! Finance.

Flow of Events for Main Success Scenario:

1. Account Database asks for the current price of the first stock on its list.

2. Sys retrieves the current price of that stock from Yahoo! Finance.

3. Account Database replaces its price for that stock with the new price.

4. Steps 1-3 are repeated for each stock on the list.

Flow of Events for Extensions: None

Use Case UC-14: Fill Orders

Related Requirements:

Initiating Actor: Player

Actor's Goal: To complete buy or sell order.

Participating Actors: Account Database

Preconditions: 1. Account Database contains a non-empty list of stocks and their prices.

2. The list of pending orders is non-empty.

Postconditions: 1. If it is possible to complete the order, Player's portfolio is updated with the results of the transaction.

Flow of Events for Main Success Scenario:

1. System asks Account Database price and change in price of the stock for the first order on its list.

2. Account Database returns the price and change in price.

3. System determines the conditions for executing the order have been met.

4. System verifies that Player has enough shares/cash for the transaction.

5. System calculates the total income/payment and subtracts/adds the commission.

6. System subtracts/adds the shares to Player's portfolio and adds/subtracts the income/payment to Player's cash balance.

7. System updates Account Database with the new values.

8. System removes order from list.

9. Steps 1-9 are repeated for each order on the list.

Flow of Events for Extensions:

2a. Account Database fails to find stock in list.

1. System displays error message telling Player that the stock ticker for this order is invalid.

2. System removes this order from the list.

3. System moves to next order on list.

3a. System determines the conditions for executing the order have not been met.

1. System moves to next order on list.

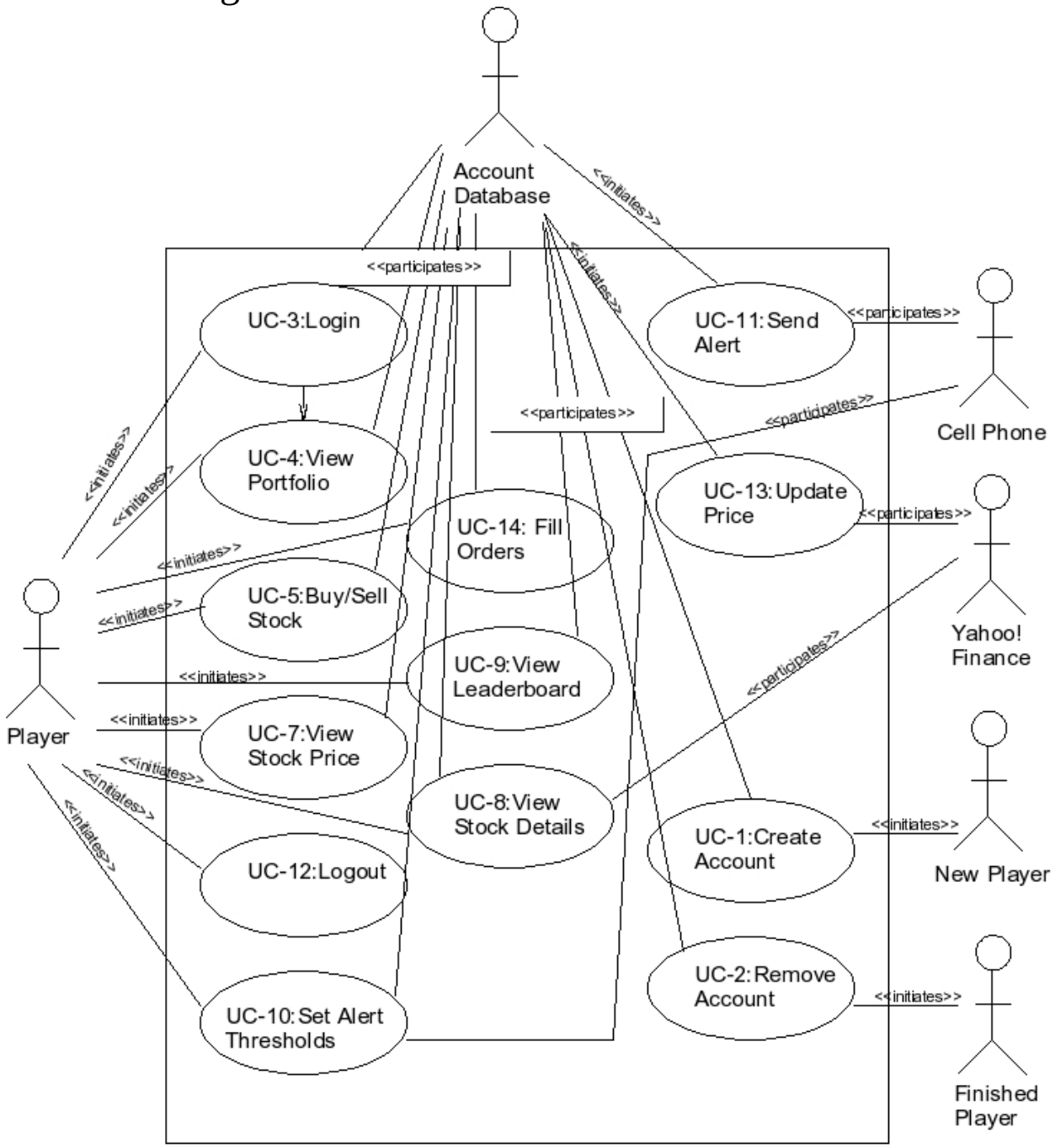
4a. System finds that player does not have enough shares/cash for the transaction.

1. System displays an error message telling Player that he/she does not have the resources for this transaction.

2. System removes this order from the list.

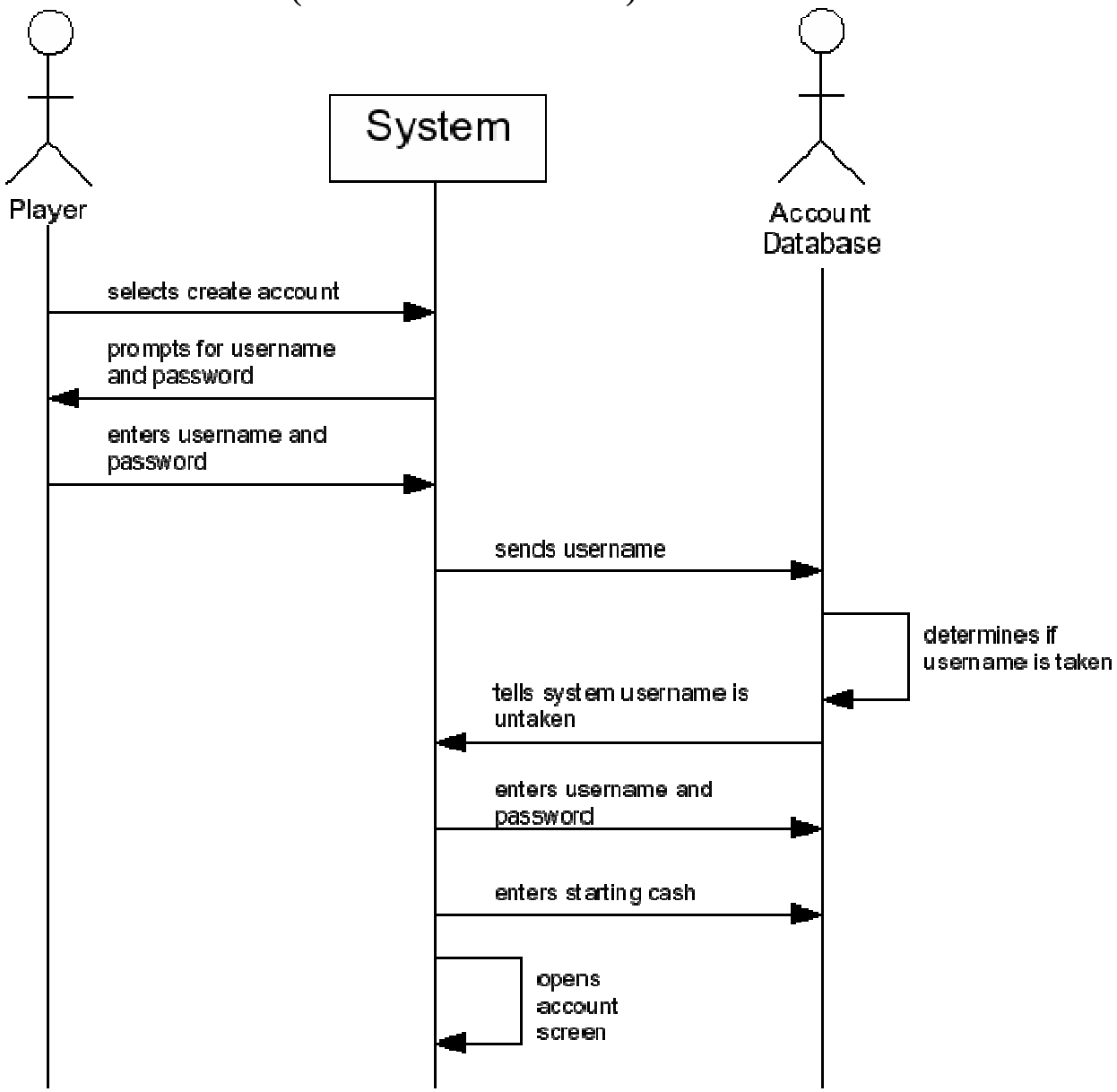
3. System moves to next order on list.

Use Case Diagram:

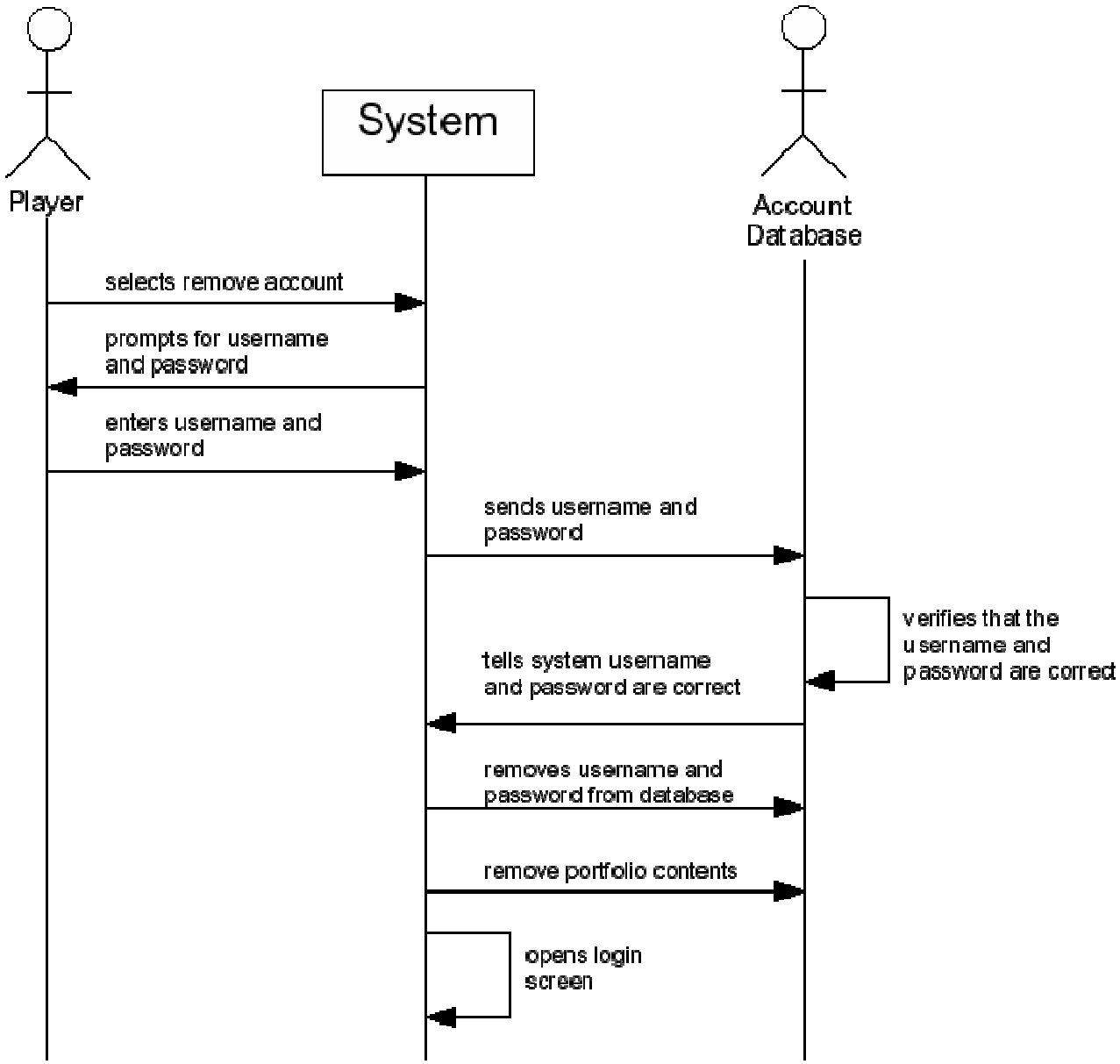


6d. System Sequence Diagrams

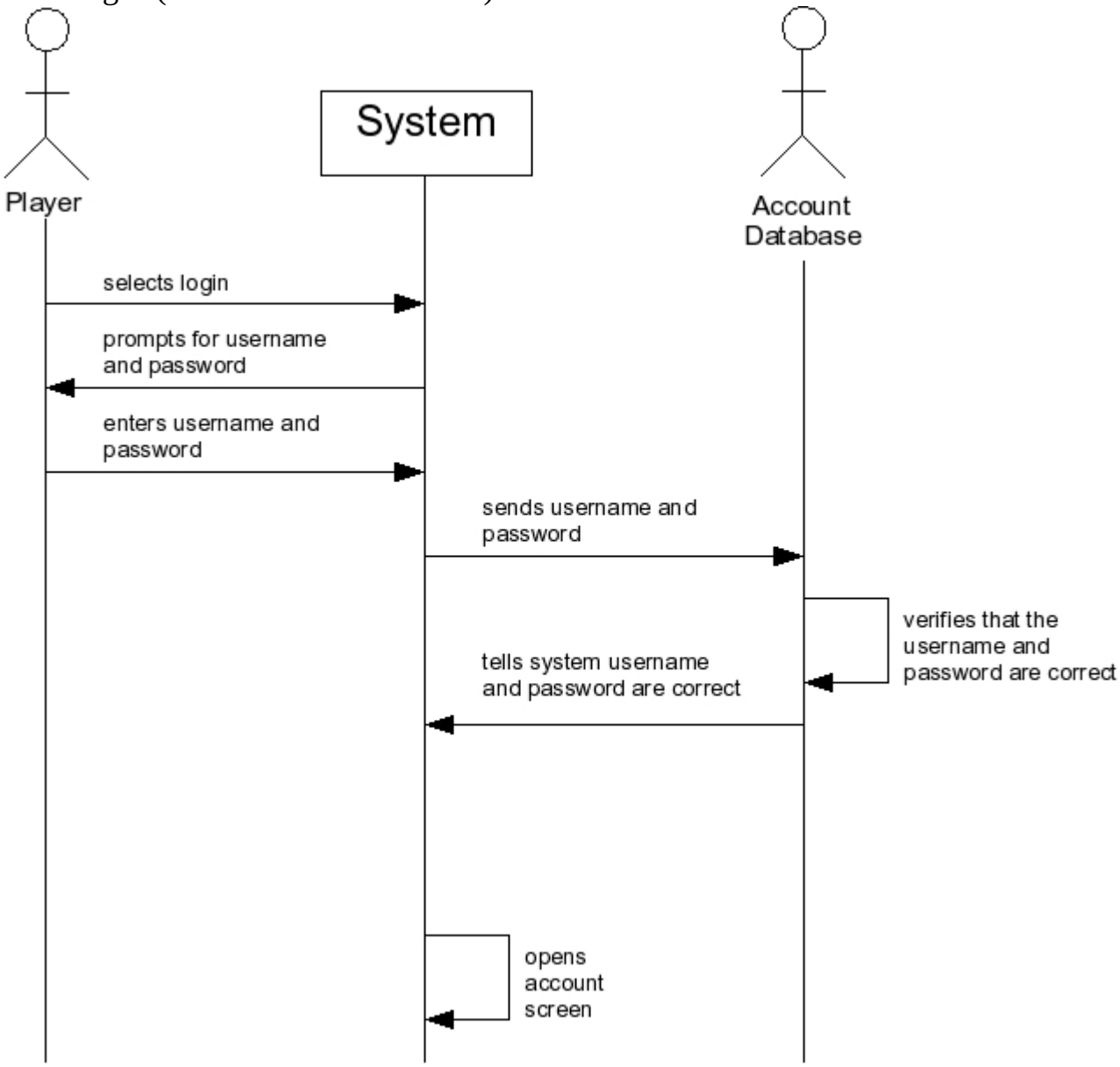
UC-1: Create Account (Main Success Scenario)



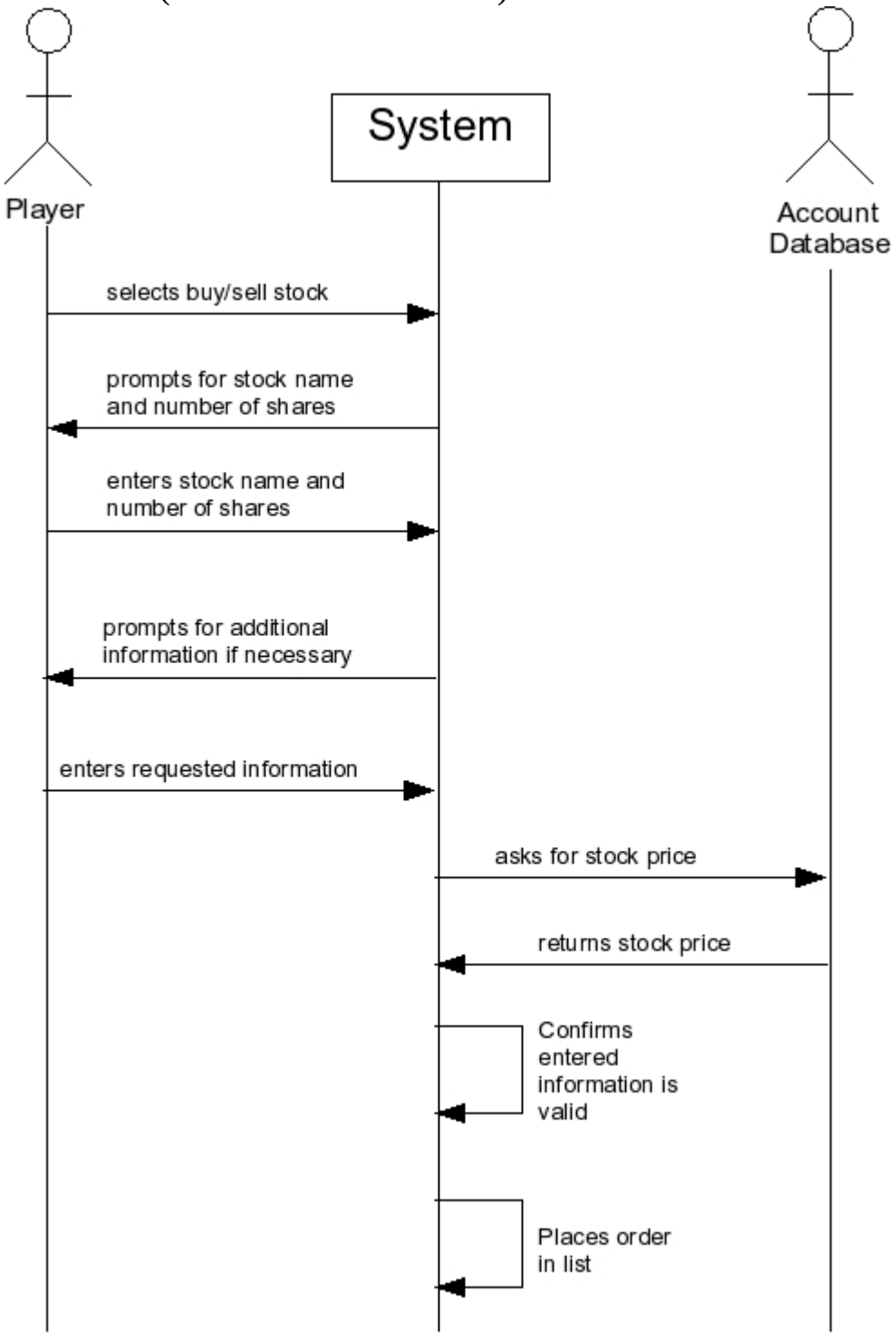
UC-2: Remove Account (Main Success Scenario)



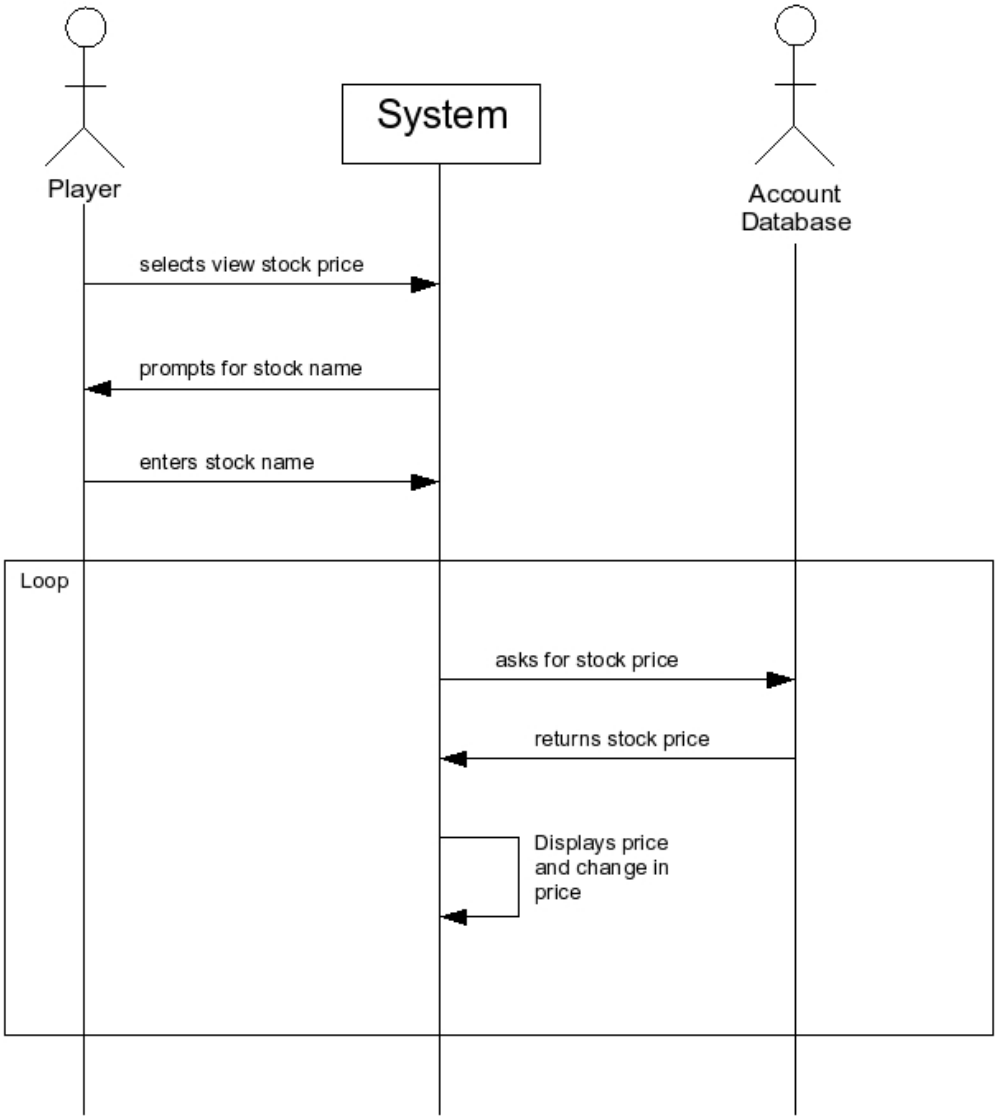
UC-3: Login (Main Success Scenario)



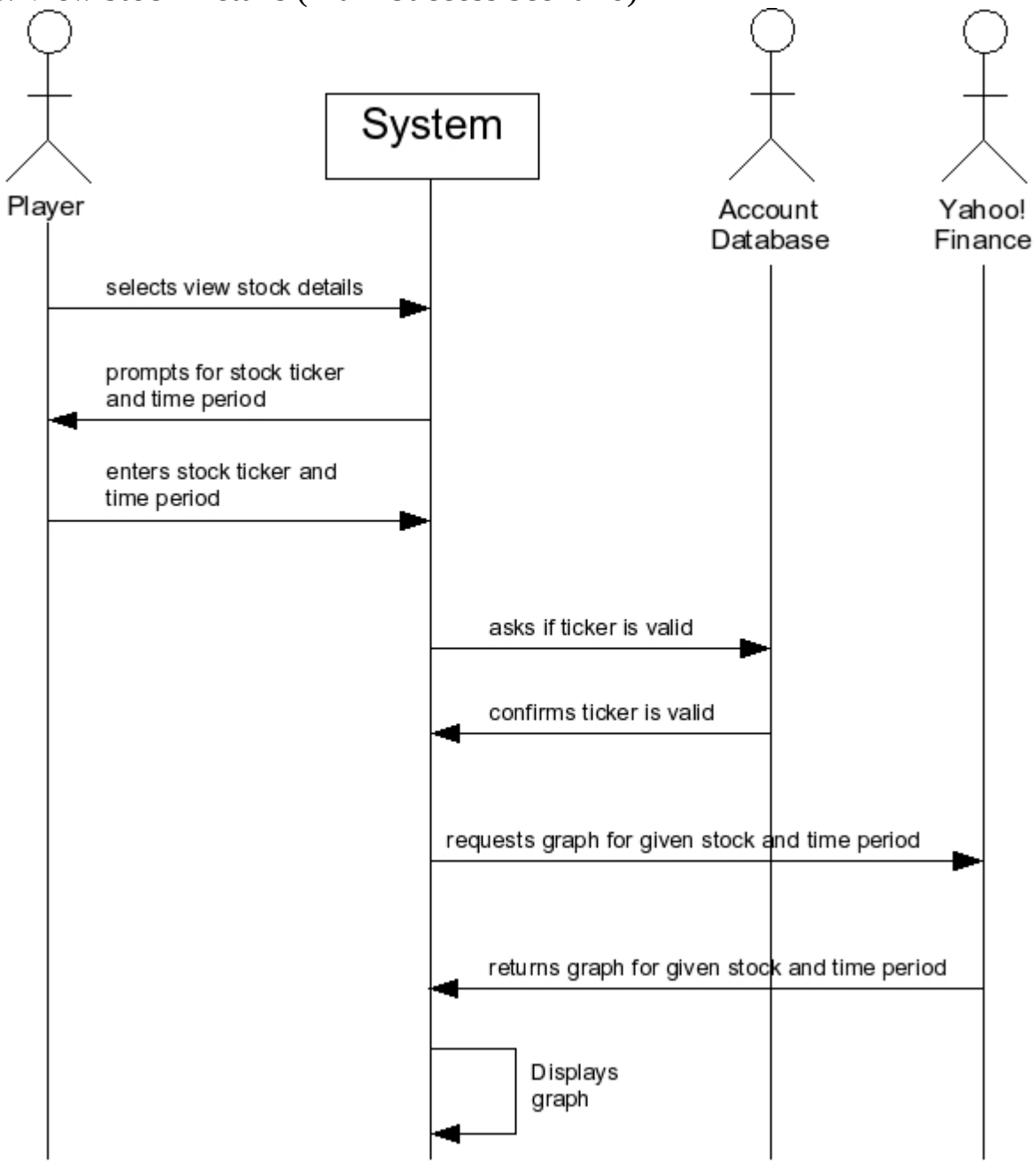
UC-5: Buy / Sell Stock (Main Success Scenario)



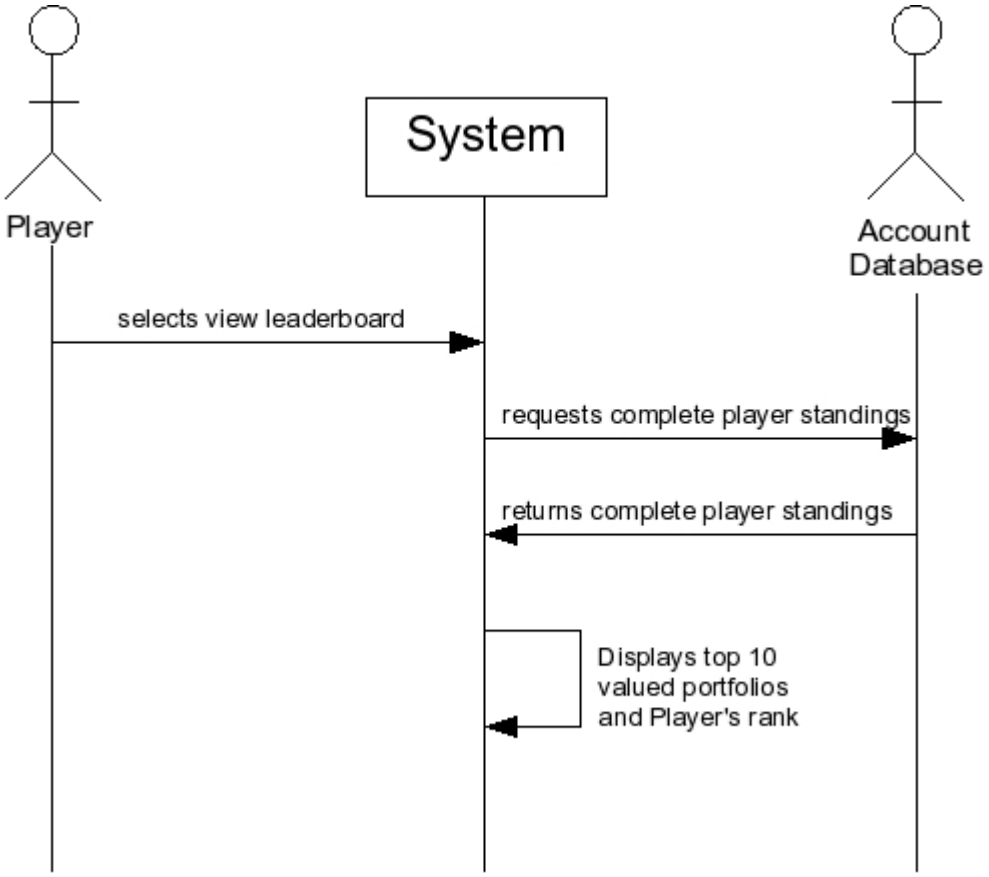
UC-7: View Stock Price (Main Success Scenario)



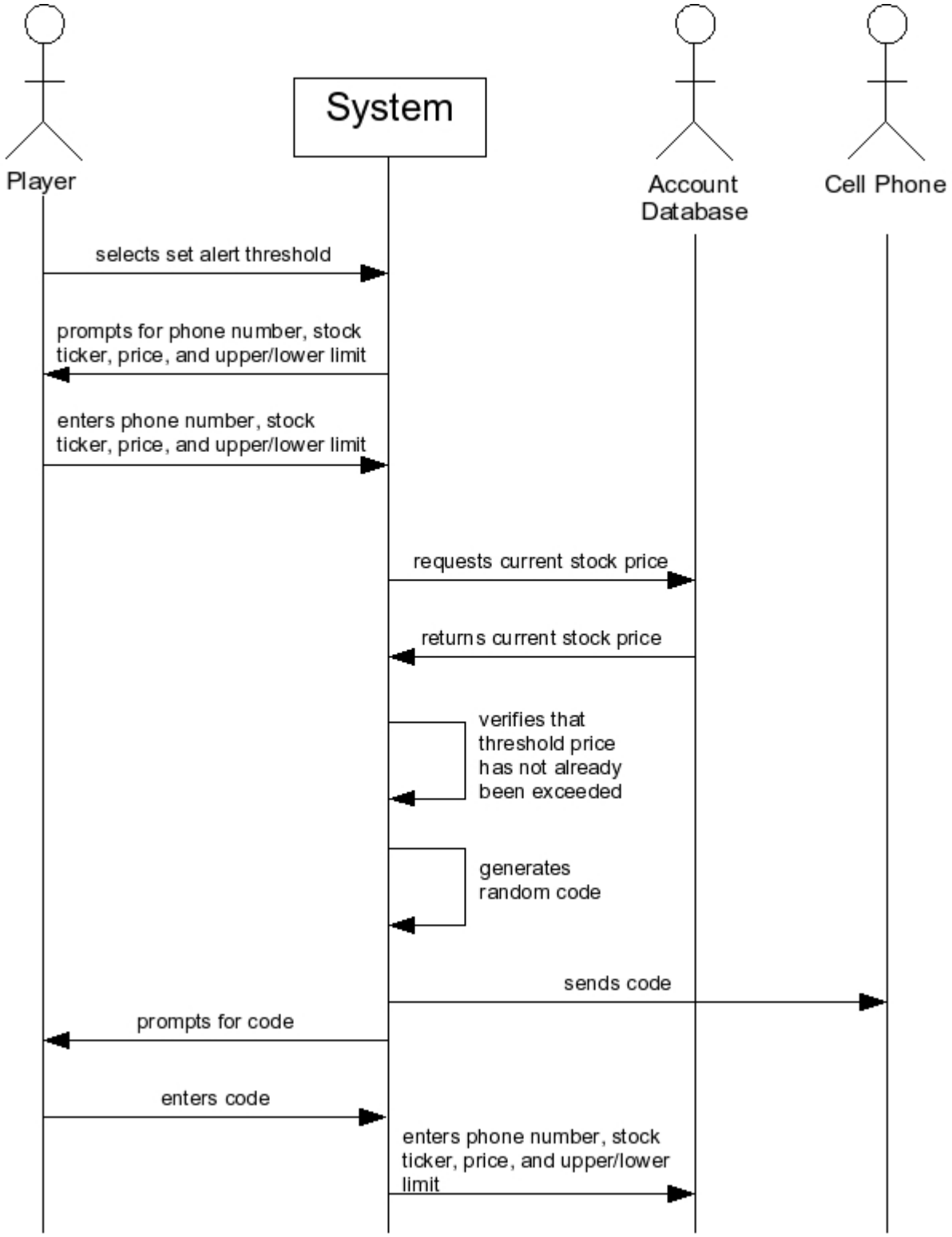
UC-8: View Stock Details (Main Success Scenario)



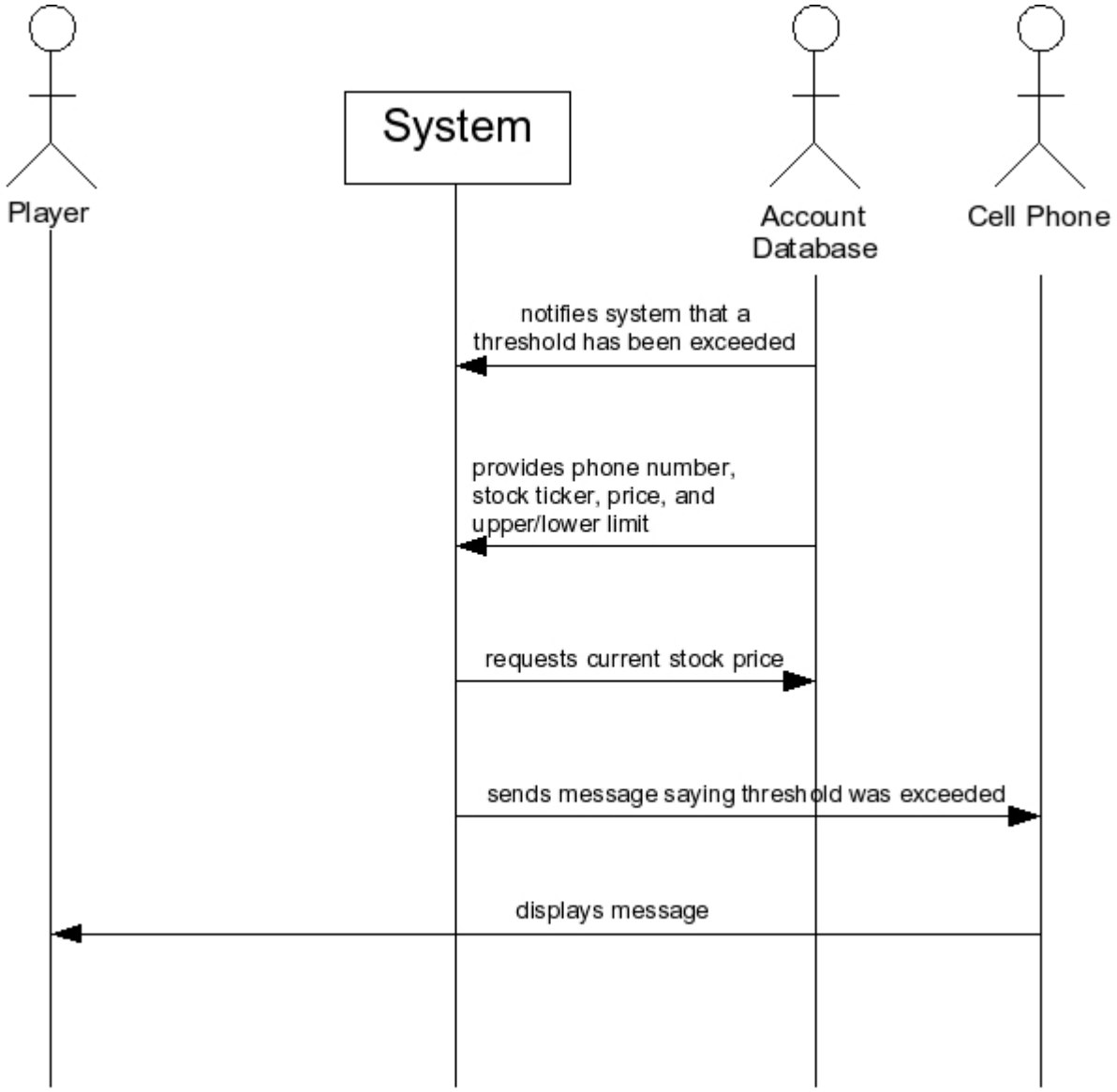
UC-9: View Leaderboard



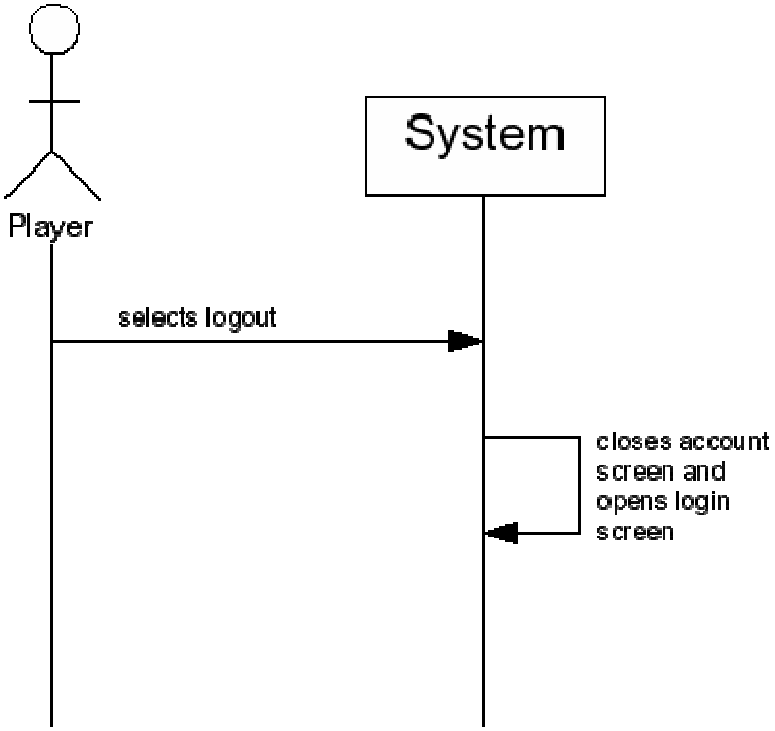
UC-10: Set Alert Thresholds:



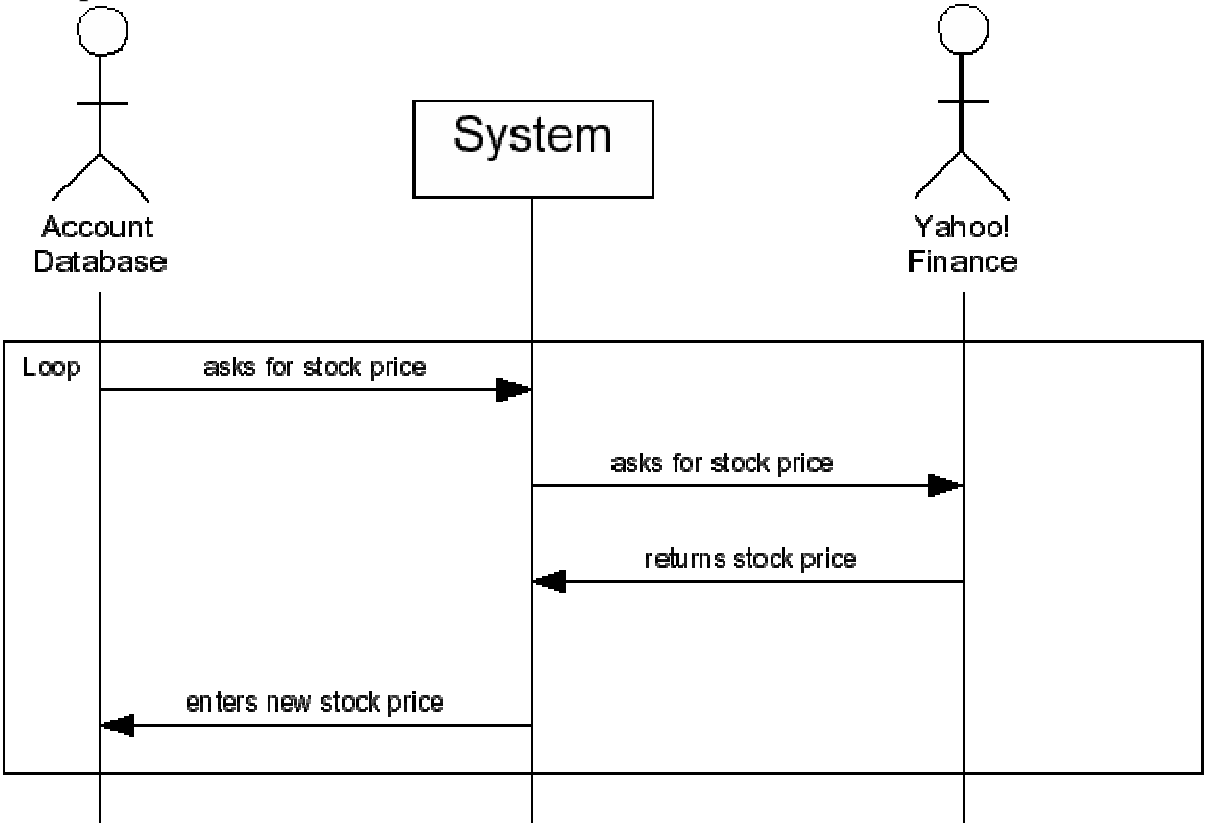
UC-11: Send Alert



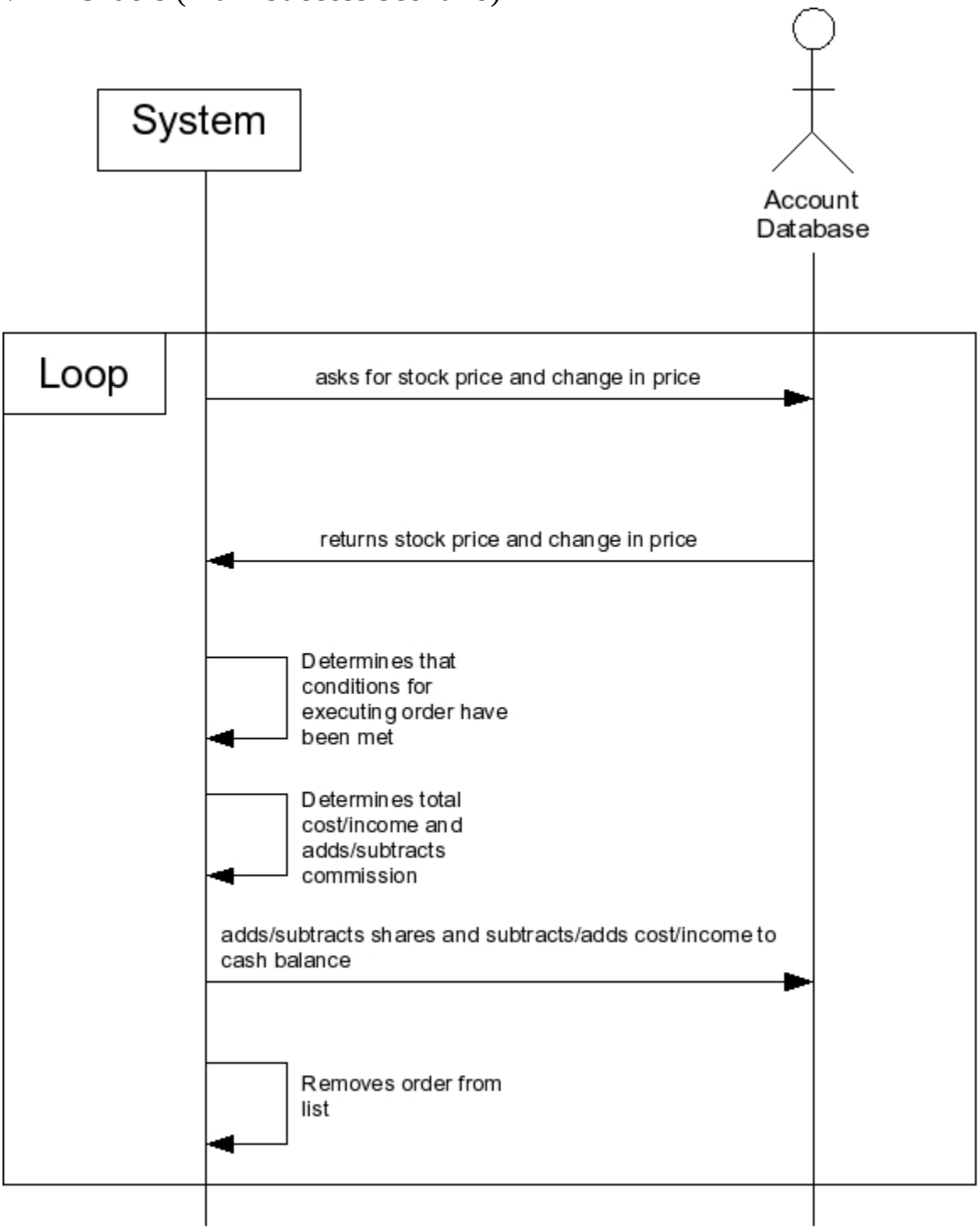
UC-12: Logout



UC-13: Update Stock Price



UC-14: Fill Orders (Main Success Scenario)



7. Nonfunctional Requirements

Functionality

FEATURE SET:

-One window with all options (provides easy access)
-Automated stock-ticker
-Automatically updated database
-Cell phone updates

CAPABILITIES

The program will be able to save and store portfolios automatically without the player requesting. Also, the stock tickers will automatically update in real time so there is no need constantly refresh it. It will be able to search any stock and send data to one's cell phone.

SECURITY

Logins with passwords are required, so one cannot simply change every player's data. Since the portfolios are stored in a data base, there is no way for other players to change information.

Usability

HUMAN FACTOR

The human factor is accounted for in our program, due to the minimal amount of clicks to get a player to their destination. There will be a very small amount of learning to figure out how to use the program. Most if not all of the buttons required for the program are on the main screen, so it will not be very hard to navigate through the program.

AESTHETICS

The program will have excellent readability and will look pleasing to the eye. It will be clear what each button's purpose is and there will only a few windows opened so there isn't much navigation required between windows.

CONSISTENCY

Since each option in the program is specific with familiar words, there will be no confusion in determining what each option does.

DOCUMENTATION

There will be instructions for the user to understand how to use the program and how the fantasy league works and the league's rules.

Reliability

FREQUENCY/SEVERITY OF FAILURE

There are a few possibilities of failures. One is if the internet connection of the player/user is interrupted or shut off. This could possibly hinder the player/user from viewing and/or changing their

stocks. Their transactions may not have gone through, which if gone unnoticed can affect the portfolio of the player.

Another possible situation where a failure may occur is if the site, Yahoo! Finance, goes down. This is where the ticker information is extracted from in real time. If the site goes down, the players cannot see their updated stocks.

RECOVERABILITY

Since the player's most up to date portfolio is in a database, it is easy to recover in case of an error. If a transaction did not go through, there would be no way to recover the desired change in stocks.

ACCURACY

Since our program will update automatically, the stock prices will be accurate. Also, since the database updates when a player changes his or her portfolio, it will also be accurate for checking standings and one's own data.

Performance

Once searched, a stock will be constantly updated. Even when another stock is searched, the first stock and each one after will be constantly updated. This will help the speed of our program as well as the response time and resource consumption, since only some stocks will be updated instead of every one. Once the stock is found, it will auto update in the window until another stock is requested or the window is closed. Our database will be updated constantly as well, which can affect performance once it gets large.

Supportability

TESTABILITY

It is easy to create new data bases and mock players, so the program will have high testability.

EXTENSIBILITY

If new extensions are added, they should not be hard to implement. Changes in the user interface will have to be made to create new buttons if desired.

ADAPTABILITY

Since the program is based on the stock market, it contains terms widely used in the market. Therefore, a user will be able to adapt to the program easily since there aren't too many aspects to learn besides the fantasy league rules.

MAINTAINABILITY

The program should not be difficult to maintain. One difficulty may occur if the databases are changed to meet a new requirement, which can affect the way the code organizes the players' portfolios.

8. Effort Estimation using Use Case Points

Actor	Description	Complexity	Weight
Player	Interacts with system via graphical interface	Complex	3
Account Database	Interacts with system via API	Simple	1
Yahoo! Finance	Interacts with system via API	Simple	1
Cell Phone	Interacts with system via API	Simple	1

Unadjusted Actor Weight(UAW) = 3 + 1 + 1 + 1 = 6.

Use Case	Description	Complexity	Weight
Create Account	Simple user Interface. One participating actor. 8 steps in main success scenario.	Average	10
Remove Account	Simple user interface. One participating actor. 3 steps in main success scenario.	Simple	5
Login	Simple user interface. One participating actor. 7 steps in main success scenario.	Simple	5
View Portfolio	Simple user interface. One participating actor. 3 steps in main success scenario.	Simple	5
Buy/Sell	Complex user interface. One participating actor. 5 steps in main success scenario.	Average	10
View Stock Price	Moderate user interface. One participating actor. 6 steps in main success scenario.	Average	10
View Stock Details	Complex user interface. Two participating actors. 7 steps in main success scenario.	Complex	15
View Leaderboard	Moderate user interface. One participating actor. 5 steps in main success scenario.	Average	10
Set Alert Thresholds	Complex user interface. Two participating actors. 10 steps in main success scenario.	Complex	15
Send Alert	Simple user interface. Two participating actors. 4 steps in main success scenario.	Average	10
Logout	Simple user interface. No participating actors. 3 steps in main success scenario.	Simple	5
Update Price	Simple user interface. One participating actor. 4 steps in main success scenario.	Simple	5
Fill Orders	Simple user interface. One participating actor. 9 steps in main success scenario.	Average	10

Unadjusted Use Case Weight(UUCW) = 10 + 5 + 5 + 5 + 10 + 10 + 15 + 10 + 15 + 10 + 5 + 5 + 10 = 115

Uadjust Use Case Points(UUCP) = UAW + UUCW = 6 + 115 = 121

Technical Factor	Description	Weight	Perceived Complexity	Calculated Factor
T1	Distributed system running on client and server machines. Also must interact with Yahoo! Finance server and cell phone.	2	5	10
T2	User expects approximate real time stock prices.	1	4	4
T3	No outstanding demands on efficiency.	1	3	3
T4	System must update prices and player portfolio values as well as process alerts and multiple kinds of stock orders.	1	5	5
T5	No demand for reusabilty.	1	0	0
T6	Must not be exceedingly difficult to install.	0.5	3	1.5
T7	Must be easy to use.	0.5	5	2.5
T8	Ability to runon different operating systems is important.	2	4	8
T9	Expect to add new features for future versions.	1	3	3
T10	Many concurrent users expected.	1	5	5
T11	Security is important but not critical.	1	4	4
T12	System will be used from many different sites.	1	5	5
T13	No user training required.	1	0	0

Technical Complexity Factor(TCF) = 0.6 + 0.01(10 + 4 + 3 + 5 + 0 + 1.5 + 2.5 + 8 + 3 + 5 + 4 + 5 + 0) = 0.6 + 0.01 * 51 = 1.11

Environmental Factor	Description	Weight	Perceived Complexity	Calculated Factor
E1	Little familiarity with UML based development.	1.5	1	1.5
E2	Moderate application problem experience.	0.5	3	1.5
E3	Very familiar with object-oriented design.	1	5	5
E4	Beginner lead analyst.	0.5	1	0.5
E5	Team reduced to four members, remaining members higly motivated.	1	3	3
E6	Requirements are stable.	2	5	10
E7	All team members must divide their attention between several courses.	-1	5	-5
E8	Somewhat difficult programming language used.	-1	3	-3

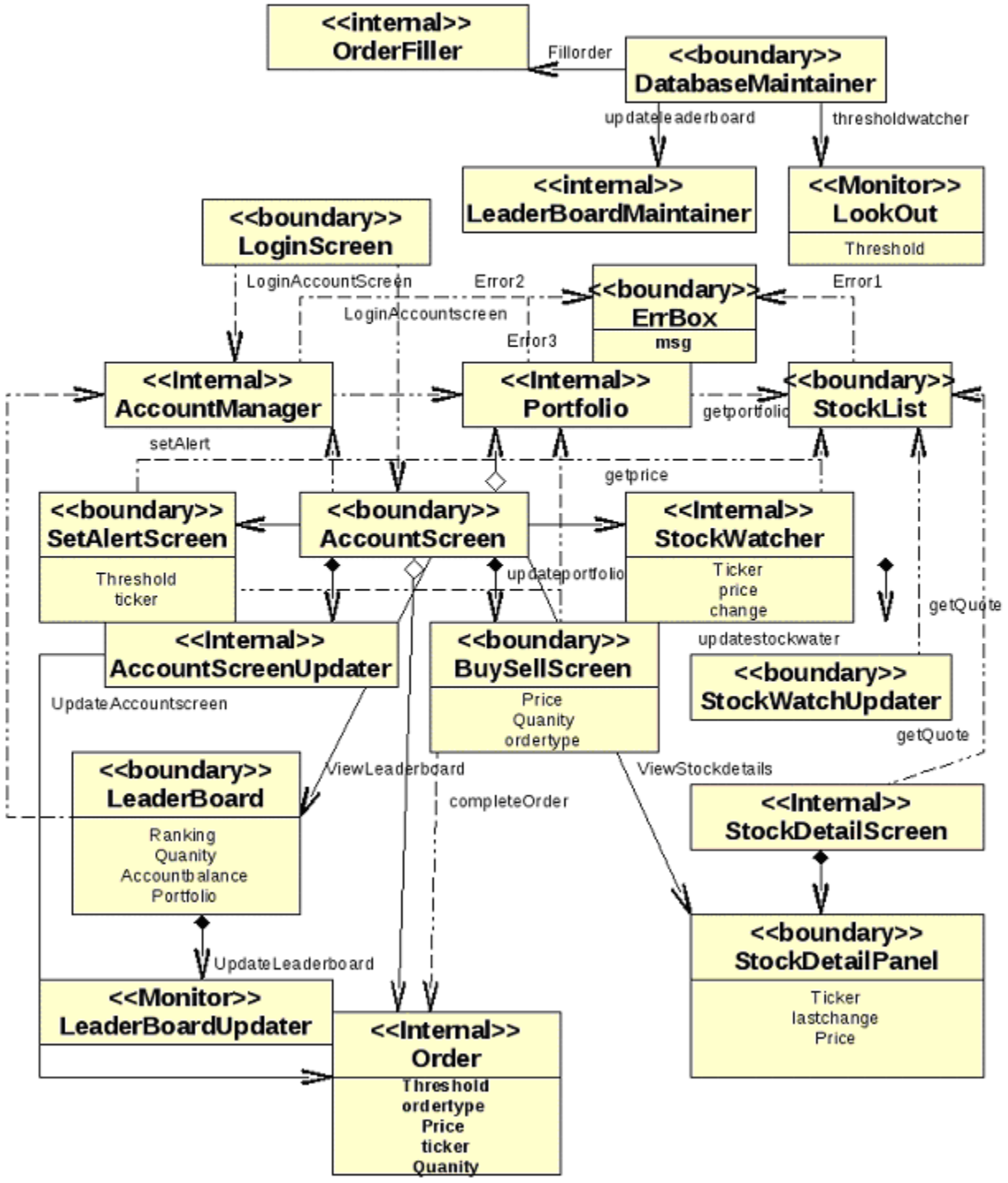
Environmental Complexity Factor(ECF) = 1.4 – 0.03(1.5 + 1.5 + 5 + 0.5 + 3 + 10 – 5 – 3) = 1.4 – 0.03 * 13.5 = 0.995

Use Case Points(UCP) = UUCP * TCF * ECF = 121 * 1.11 * 0.995 = 133.64
Expected Project Duration = UCP * PF = 133.64 * 28 = 3742 hours

This overestimates the project duration by around 50%, which leads us to conclude that a productivity factor of 28 is too high.

9. Domain Analysis

Domain Model



Domain Analysis

Concept Definitions

Concept Name	Responsibility	Type
Order	Contains all operations related to buying and selling of stock	D
LoginScreen	Controls all functions related to logging in, logging out and registering	D
Leaderboard	Displays the user's portfolio and the current top investors	D
DatabaseMaintainer	Retrieves the current stock prices from the Internet (yahoo)	D
	including connecting to the server reading from the database and	D
	updating the database.	
Lookpout	Monitors if a stock crosses a given threshold	D
Orderfiller	completes orders if a user is logged out	D
LeaderboardMaintainer	Updates the current ranking	D
ErrBox	Displays error messages to the user	D
Portfolio	stores the current user's portfolio	D
AccountManager	Manages the account	D
Stocklist	connects to the database in order to get the current price and quotes	D
SetAlertScreen	Sets thresholds for an order	D
AccountScreen	Displays Details about a user's account	D
Stockwatcher	Displays a panel for the user to watch stocks	D
Stockwatcherupdater	Updates the information to the stock watcher	D
LeaderboardUpdater	Grabs the current rankings	D
StockDetailPanel	Displays detailed information about a stock including a graph	D

Association definitions

Association name	Association Descriptions	Location
completeorder	Buys or sells a stock	UC-5
getPrice	Displays the current price of a given stock	UC-7
Login	Logs a user in if they have an account in the database.	UC-1
Logout	Logs a user out	UC-12
Sign_Up	Creates a new user account	UC-1,UC-3
ViewLeaderboard	Displays the names of the top investors, their rankings(in order) and their net worth	UC-9
Display_Top_Investors	Finds the top investors	UC-9
Update_DB	Updates the SQL database with the current price from the internet	UC-13
Download_SP	Downloads the current stock price from Yahoo!	UC-13
SetAlert	Function used to set the Threshold for a stock	UC-10
thresholdwatcher	Watches a stock for any change	UC-11
Error1	Error message	All
Error2	Error message	All
Error3	Error message	All
Attribute Name	Attribute Description	

TICKER	Stock names
QUANITY	How many of a perticular stock the user owns
price	The current price of stock
Threshold	The threshold level specified by the investor for a given stock

Operation	**Create Account**
Preconditions	Check if the account name exist.
Postconditions	Create an new user account
	Display LoginScreen

Operation	**Remove Account**
Preconditions	Check if the account name exist.
Postconditions	Remove Account
	Log User Out

Operation	**Login**
Preconditions	Check input information is valid
Postconditions	LOG user in
	Display LoginScreen

Operation	**View Portfolio**
Preconditions	Call Your_Data
Postconditions	Display LoginScreen

Operation	**Buy Stock**
Preconditions	Check if price*quantity is not less than user available funds
Postconditions	Update the database with the new values of QUANTITY, and TICKER

Operation	**Sell Stock**
Preconditions	Check if the number of shares they wish to sell is not greater QUANTITY
Postconditions	Update the database with the new values of , QUANTITY, and TICKER

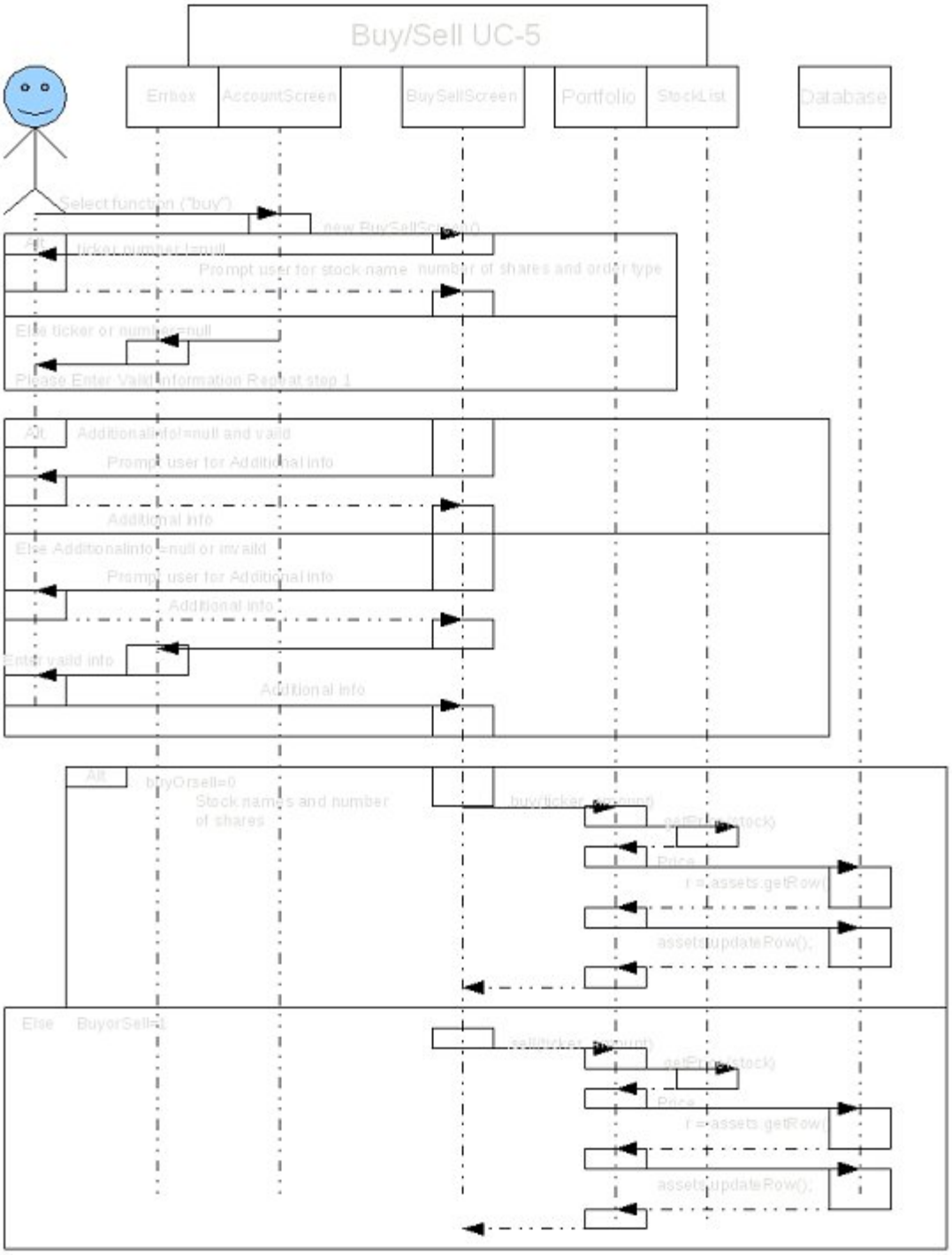
Operation	**View Stock Details**
Preconditions	Check if the entered name matches anything in TICKER
Postconditions	Display StockDetailScreen with the additional stock info

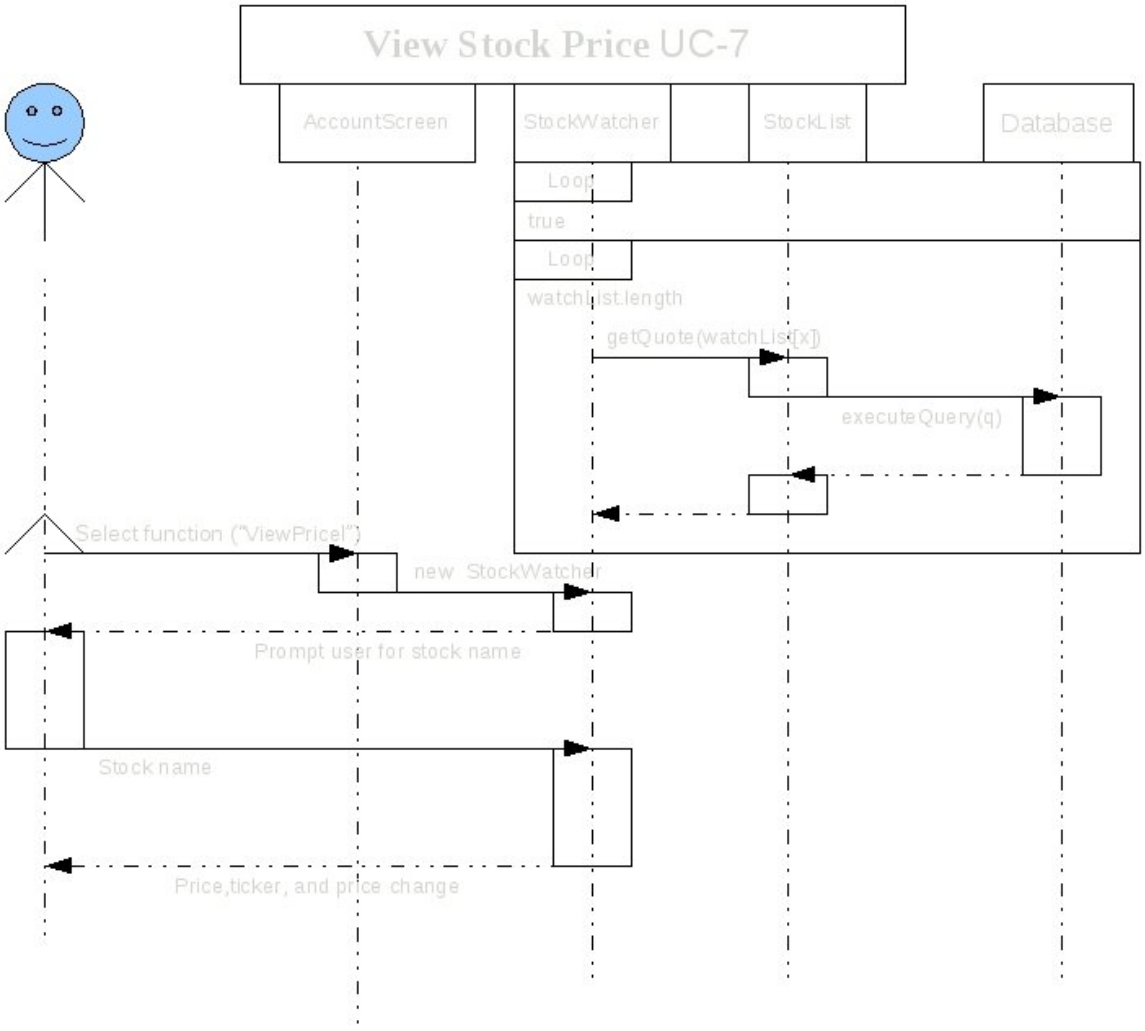
Operation	**View Leaderboard**
Preconditions	None
Postconditions	Display LoginScreen with the ranking information

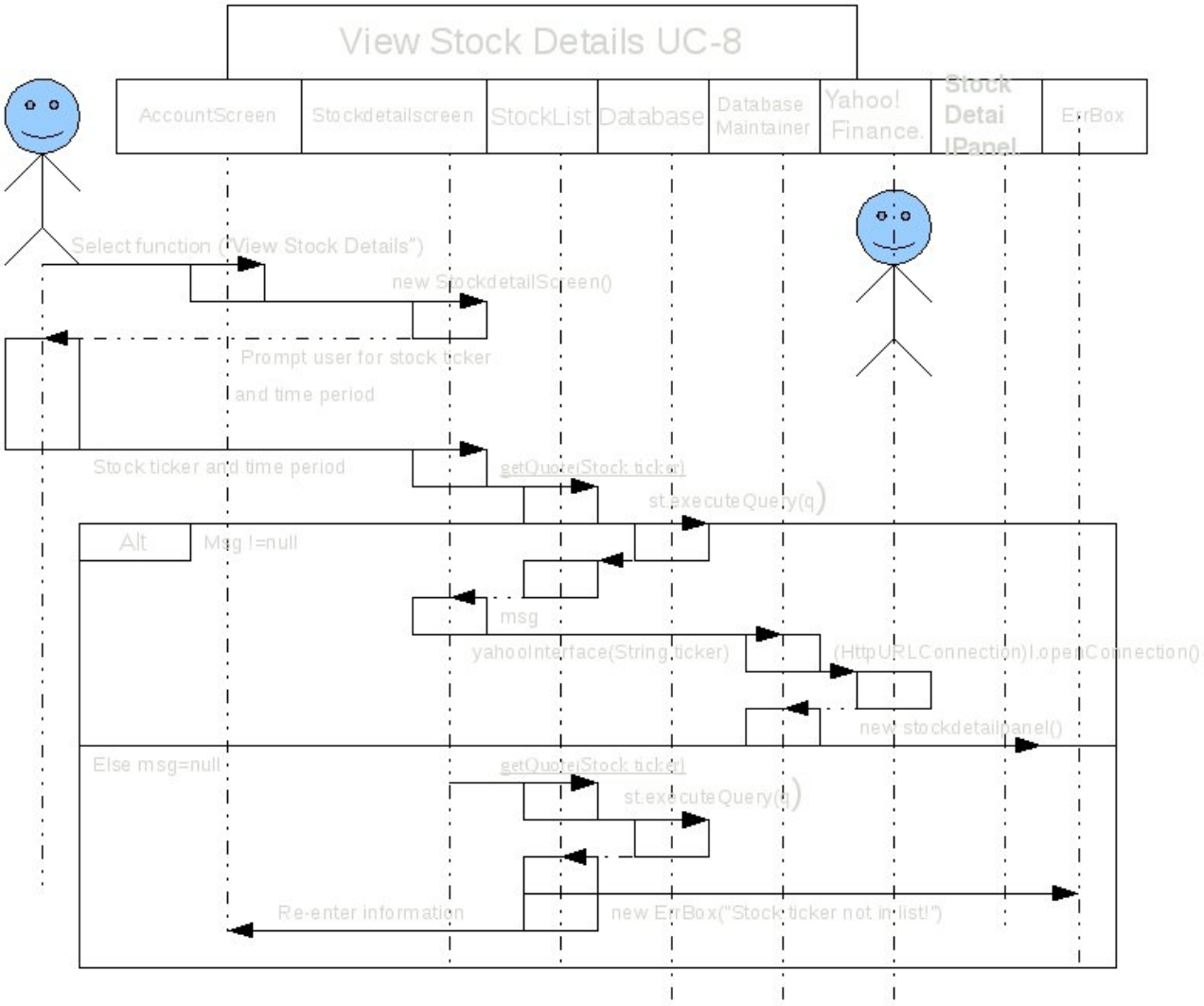
Operation	**Logout**
Preconditions	None
Postconditions	Displays Login screen User can exit from this point

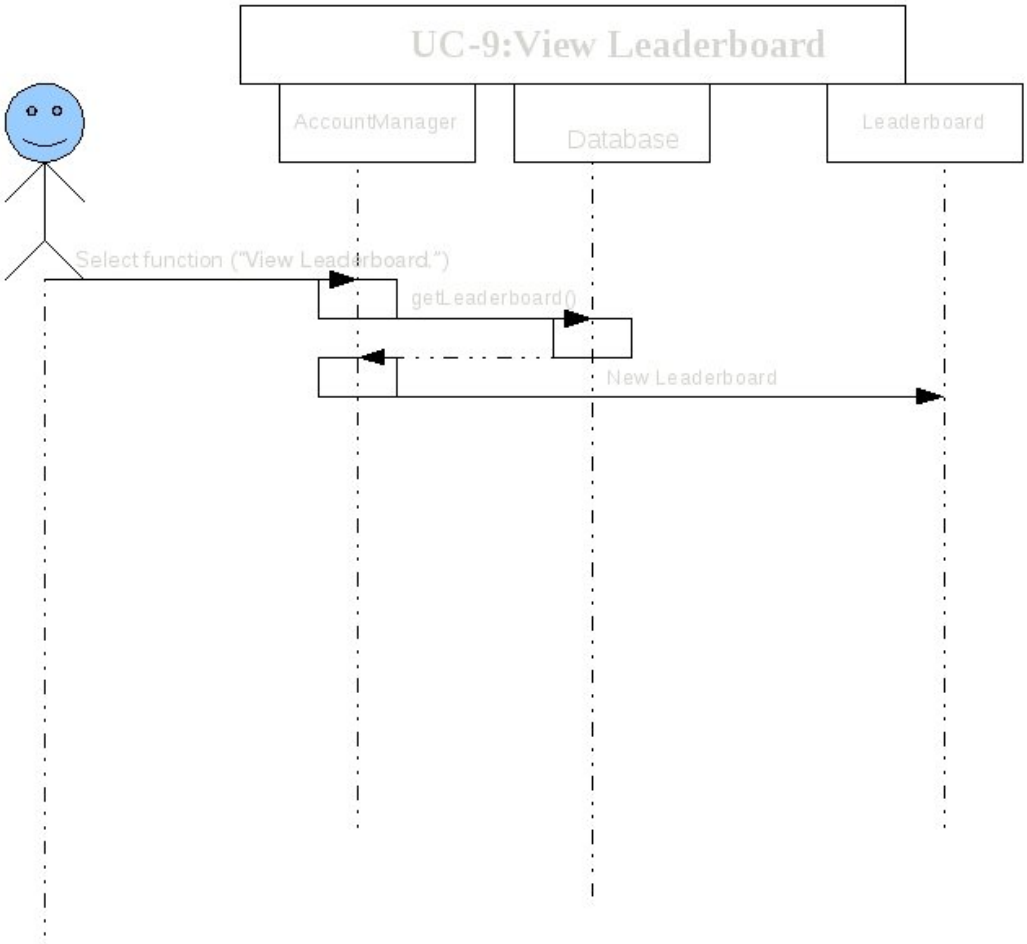
Operation	**Update Price**
Preconditions	None
Postconditions	Update stock prices in the database

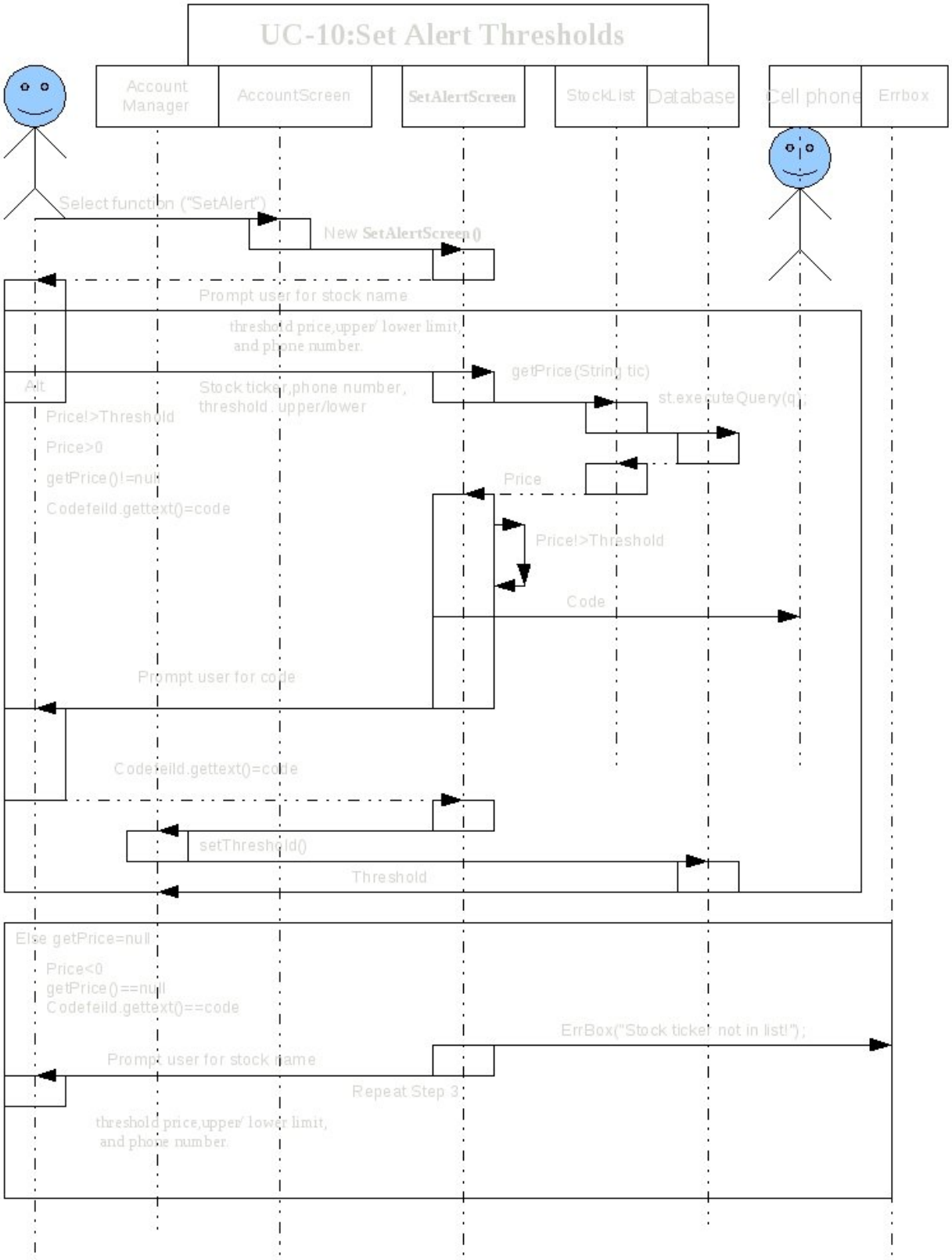
10. Interaction Diagrams

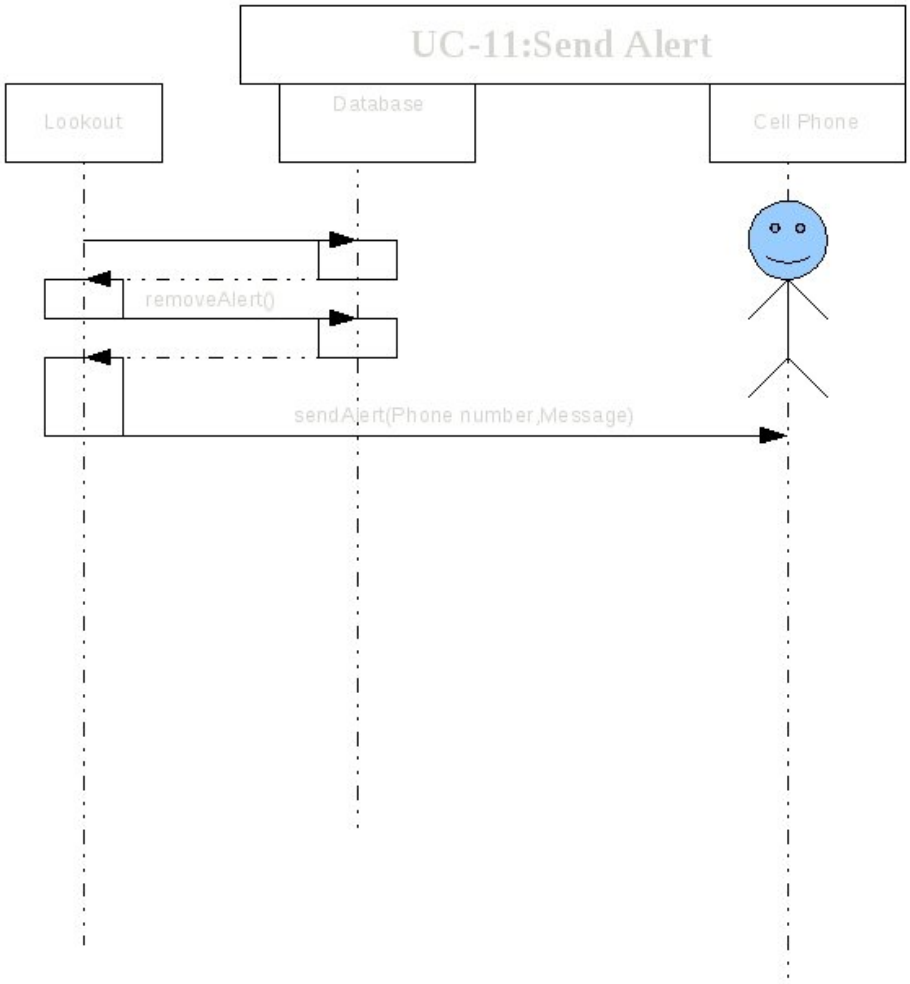


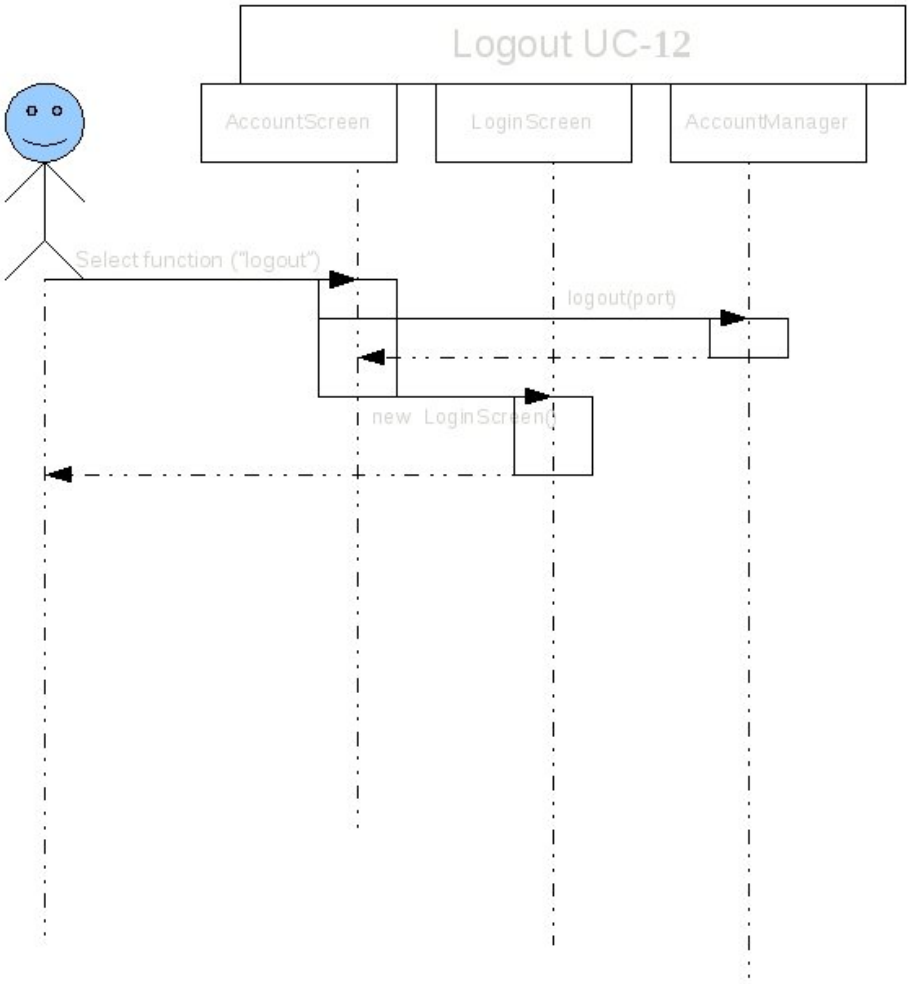

View Stock Price UC-7
AccountScreen
StockWatcher
StockList
Database
Loop
true
Loop
watchlist.length
getQuote(watchList[x])
executeQuery(q)
Select function ("ViewPrice!")
new StockWatcher()
Prompt user for stock name
Stock name
Price,ticker, and price change

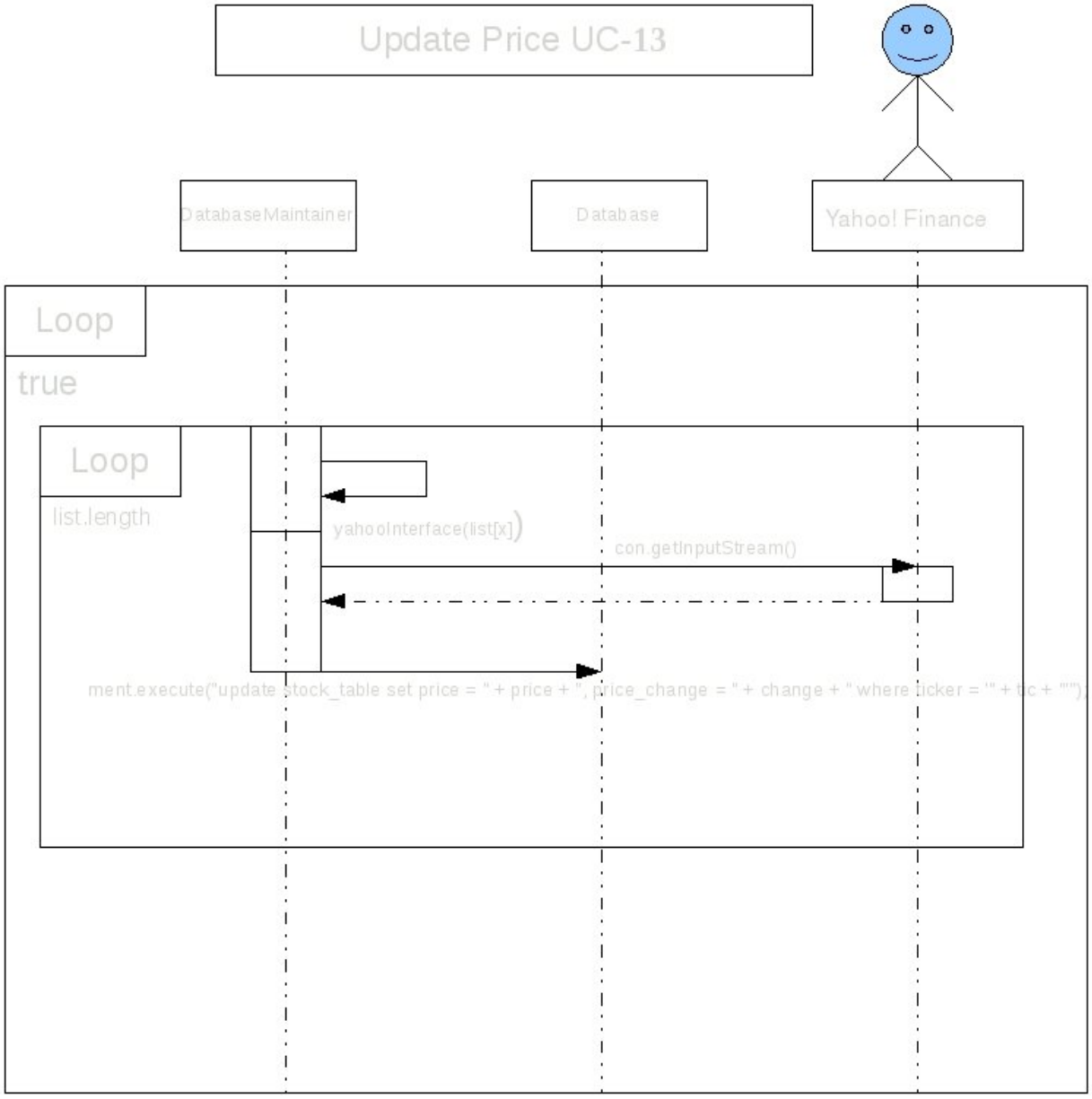

View Stock Details UC-8
AccountScreen
Stockdetailscreen
StockList
Database
Database Maintainer
Yahoo! Finance.
Stock Detai lPanel
ErrBox
Select function ("View Stock Details")
new StockdetailScreen()
Prompt user for stock ticker
and time period
Stock ticker and time period
getQuote(Stock ticker)
st.executeQuery(q)
Alt
Msg !=null
msg
yahooInterface(String ticker)
(HttpURLConnection)l.openConnection()
new stockdetailpanel()
Else msg=null
getQuote(Stock ticker)
st.executeQuery(q)
Re-enter information
new ErrBox("Stock ticker not in list!")

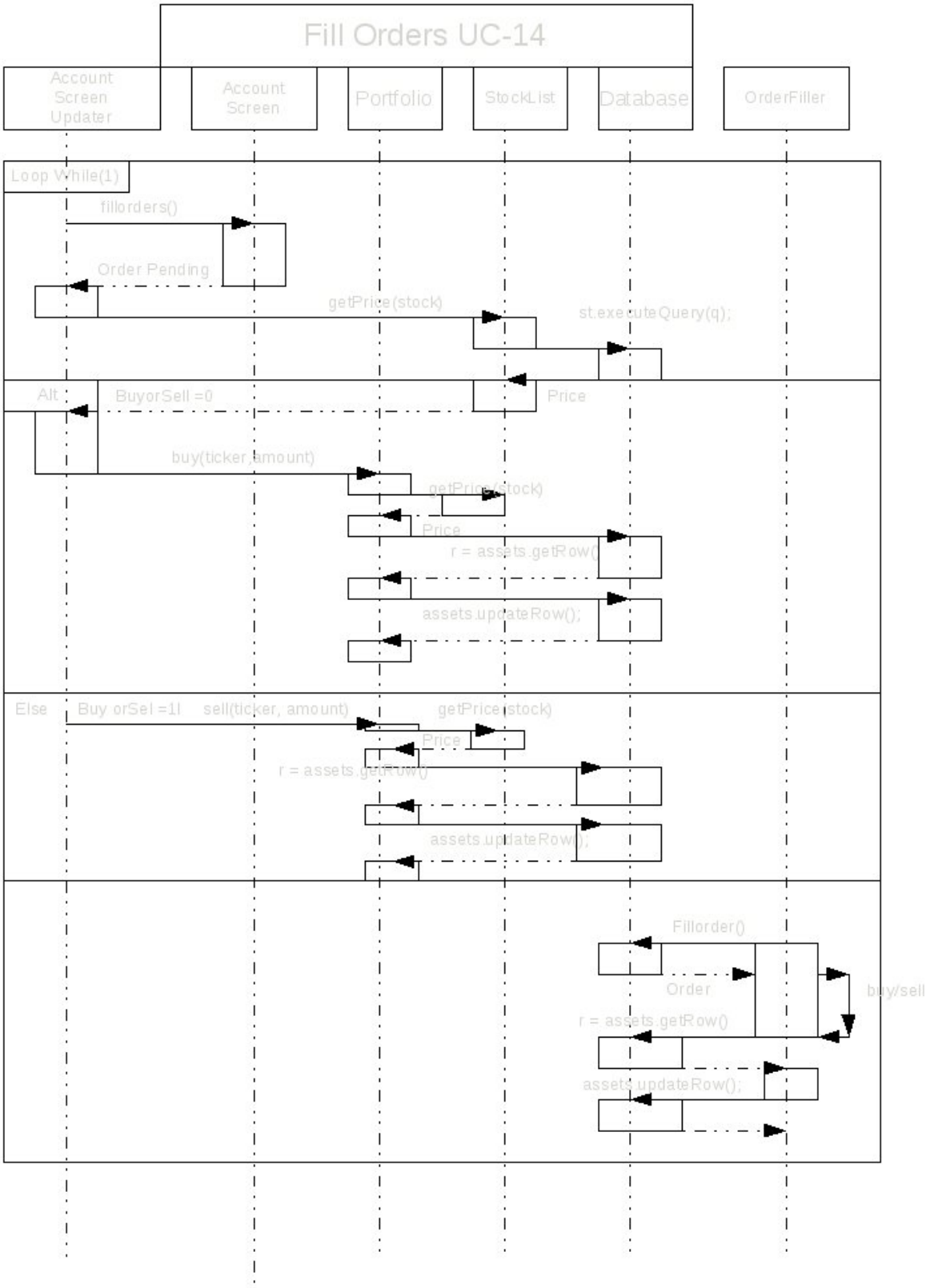

UC-9:View Leaderboard
AccountManager
Database
Leaderboard
Select function ("View Leaderboard")
getLeaderboard()
New Leaderboard


UC-10:Set Alert Thresholds
Account Manager
AccountScreen
SetAlertScreen
StockList
Database
Cell phone
Errbox
Select function ("SetAlert")
New SetAlertScreen()
Prompt user for stock name
threshold price,upper/ lower limit, and phone number.
Alt
Stock ticker,phone number, threshold ,upper/lower
getPrice(String tic)
st.executeQuery(q);
Price!>Threshold
Price>0
getPrice()!=null
Codefeild.gettext()=code
Price
Price!>Threshold
Code
Prompt user for code
Codefeild.gettext()=code
setThreshold()
Threshold
Else getPrice=null
Price<0
getPrice()==null
Codefeild.gettext()==code
ErrBox("Stock ticker not in list!");
Prompt user for stock name
Repeat Step 3
threshold price,upper/ lower limit, and phone number.


UC-11:Send Alert
Lookout
Database
Cell Phone
removeAlert()
sendAlert(Phone number,Message)

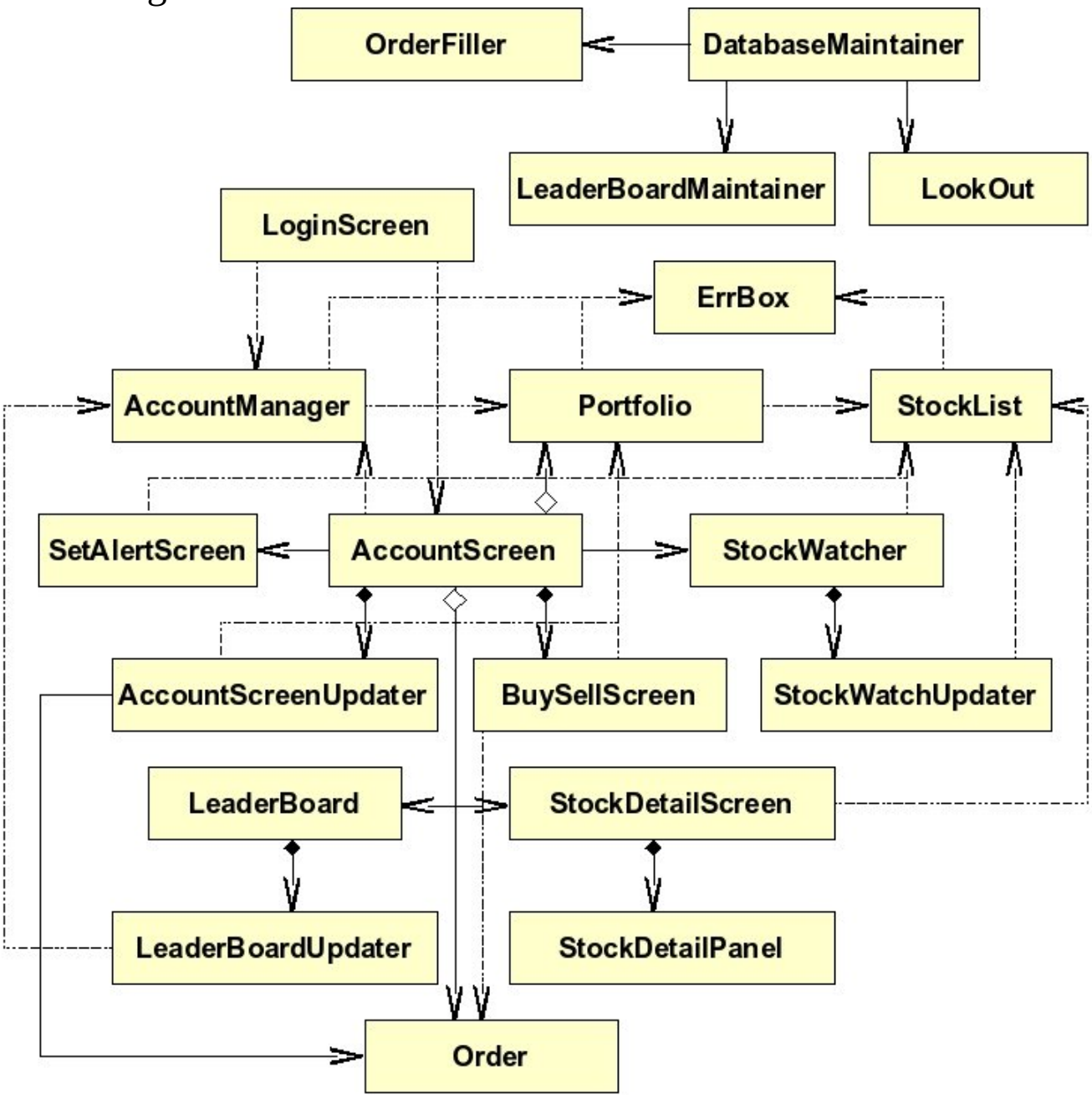

Logout UC-12
AccountScreen
LoginScreen
AccountManager
Select function ("logout")
logout(port)
new LoginScreen()


Update Price UC-13
DatabaseMaintainer
Database
Yahoo! Finance
Loop
true
Loop
list.length
yahooInterface(list[x])
con.getInputStream()
ment.execute("update stock_table set price = " + price + ", price_change = " + change + " where ticker = '" + tic + "'");


Fill Orders UC-14
Account Screen Updater
Account Screen
Portfolio
StockList
Database
OrderFiller
Loop While(1)
fillorders()
Order Pending
getPrice(stock)
st.executeQuery(q);
Alt
BuyorSell =0
Price
buy(ticker,amount)
getPrice(stock)
Price
r = assets.getRow()
assets.updateRow();
Else
Buy orSel =1
sell(ticker, amount)
getPrice(stock)
Price
r = assets.getRow()
assets.updateRow();
Fillorder()
Order
buy/sell
r = assets.getRow()
assets.updateRow();

11. Class Diagram and Interface Specification

Class Diagram:



11c Design Patterns

Notice from this diagram that the whole system is made of two completely isolated subsystems. In the upper right-hand corner of the diagram is a small subsystem that runs on the server. The rest of the system runs on the user's machine. The whole system has a database-centric architecture, in which each of these subsystems communicates with the Account Database rather than communicating with each other. The design of the user subsystem is greatly simplified by use of the proxy design pattern. The StockList class serves as a proxy for accessing stock price information from the Account Database, and the AccountManager class serves as a proxy for accessing account information from the Account Database. This encapsulates responsibilities that would otherwise have to be given to the LoginScreen, AccountScreen, Portfolio, SetAlertScreen, StockWatcher, StockWatchUpdater, LeaderBoardUpdater, and StockDetailScreen classes, all of which need to access the Account Database at some point in their operation.

11b Data Types and Operation Signatures

AccountManager:

-url : String = “jdbc:mysql://software-ece.rutgers.edu/group902?user=user&password=password” {readOnly}

-con : Connection

+getCon() : Connection {query}

-connect() : void {postcondition: con is connected to Account Database.}

+createAccount(in name : String, in pass : String) : Portfolio {postcondition: if name does not match a username in Account Database, an entry with name and pass is added to Account Database.}

+login(in name : String, in pass : String) : Portfolio {postcondition: if name and pass match an entry in Account Database, that user's assets are retrieved and put into a Portfolio object.}

+logout(inout p : Portfolio) : void {postcondition: p is set to null.}

+removeAccount(in name : String) : void {postcondition: all entries containing name are removed from Account Database.}

+getStandings() : ResultsSet

+getOrders(in name : String) : Order {postcondition: all orders placed by this user are removed from Account Database.}

+storeOrders(in ord : Order) : void {postcondition: all orders in the linked list headed by ord are added to Account Database.}

+setThreshold(in address : String, in ticker : String, in price : double, in upLow : int) : void {postcondition: an entry with address, ticker, price, and upLow is added to Account Database.}

AccountScreen:

-dispPanel : JPanel

-butPanel : JPanel

-dispArea : JTextArea

-buySell : JButton

-viewPrice : JButton

-viewDetails : JButton

-viewLeaders : JButton

-setAlert : JButton

-logout : JButton

-deregister : JButton

-port : Portfolio

-pending : Order

-asu : AccountScreenUpdater

+actionPerformed(in ev : ActionEvent) : void {bodycondition: if buySell is the source of ev, a BuySellScreen is launched, bodycondition: if viewPrice is the source of ev, a StockWatcher is launched, bodycondition: if viewDetails is the source of ev, a StockDetailScreen is launched, bodycondition: if viewLeaders is the source of ev, a LeaderBoard is launched, bodycondition: if setAlert is the source of ev, a SetAlertScreen is launched, bodycondition: if logout is the source of ev, AccountManager.logout(port) is called, bodycondition: if deregister is the source of ev, AccountManager.removeAccount(port.getName()) is called.}

AccountScreen.AccountScreenUpdater:

+run() : void {bodycondition: dispArea's text is set to port.toString().}

-fillOrders() : void {postcondition: all orders in the linked list headed by AccountScreen.pending that are ready to be executed are executed and removed from the linked list.}

AccountScreen.BuySellScreen:

-inPanel : JPanel

-radPanel : JPanel

-buttonPanel : JPanel

-stockField : JTextField

-numField : JTextField

-priceField : JTextField

-percentField : JTextField

-market : JRadioButton

-limit : JRadioButton

-stopLoss : JRadioButton

-trailStop : JRadioButton

-buy : JButton

-sell : JButton

-cancel : JButton

-stockLabel : JLabel

-numLabel : JLabel

-priceLabel : JLabel

-percentLabel : JLabel

+actionPerformed(in ev : ActionEvent) : void

+placeOrder() : void {precondition: stockField and numField are non-empty, postcondition: AccountScreen.pending is replaced with a new order that is linked to the previous AccountScreen.pending.}

DatabaseMaintainer:

-list : String[*] {readOnly}

-url : String = “jdbc:mysql://localhost/group902?user=user&password=password” {readOnly}

-conn : Connection

+getConn() : Connection {query}

+run() : void

-update() : void {postcondition: each stock price in Account Database is updated with its current value.}

-yahooInterface(in ticker: String): String {postcondition: returns a String containing ticker, the current price, and today's change in price.}

+main(in args: String[]): void

ErrBox:

-msgLabel : JLabel

-ok : JButton

+actionPerformed(in ev : ActionEvent) : void{postcondition: ErrBox is closed.}

LeaderBoard:

-dispPanel : JPanel

-butPanel : JPanel

-leaderPanel : JPanel

-dispPane : JScrollPane

-leaders : JTextField[10]

-player : JTextField

-exit : JButton

-updater : LeaderBoardUpdater

+actionPerformed(in ev : ActionEvent) : void {postcondition: LeaderBoard is closed}

LeaderBoardMaintainer:

-ment : Statement

+run() : void

-update() : void {postcondition: the value of each player's portfolio stored in Account Database is updated with its current value.}

LeaderBoard.LeaderBoardUpdater:

+run() : void {bodycondition: LeaderBoard.player and each element of LeaderBoard.leaders is updated to reflect the current standings.}

LookOut:

-ment : Statement

+run() : void {bodycondition: if an alert threshold has been exceeded, sendAlert() is called, and that alert threshold is deleted from Account Database.}

-sendAlert(in address : String, in msg : String) : void {postcondition: msg is sent to cell phone corresponding to address.}

LoginScreen:

-inPanel : JPanel

-butPanel : JPanel

-nameField : JTextField

-passField : JTextField

-nameLabel : JLabel

-passLabel : JLabel

-login : JButton

-register : JButton

-exit : JButton

+actionPerformed(in ev : ActionEvent) : void {precondition: nameField and passField are non-empty, bodycondition: if login is the source of ev, AccountManager.login(...) is called, bodycondition: if register is the source of ev, AccountManager.createAccount(...) is called, postcondition: if login/registration information is valid, AccountScreen is launched.}

+main(in args : String[*]) : void {bodycondition: launches a LoginScreen.}

There are two important things to note here. You may have noticed that there are two main methods. That's because DatabaseMaintainer runs on the server machine and therefore needs its own main method. The second thing to note is that the main method in the LoginScreen class have been placed in any class without changing anything. It was placed in the LoginScreen class because the primary responsibility of the main method is to launch a LoginScreen.

Order:

-MARKET : double = 0.02 {readOnly}

-LIMIT : double = 0.04 {readOnly}

-STOPLOSS : double = 0.05 {readOnly}

-TRAILSTOP : double = 0.07 {readOnly}

-type : String

-ticker : String

-buyOrSell : int

-numShares : double

-priceOrPercent : double

-commission : double

-next : Order

-prev : Order

+getType() : String {query}

+getTicker() : String {query}

+getBuyOrSell() : int {query}

+getNumShares() : double {query}

+getPriceOrPercent() : double {query}

+getCommission() : double {query}

+getNext() : Order {query}

+get Prev() : Order {query}

+setNext(Order ord) : void {postcondition: next = ord.}

+setPrev(Order ord) : void {postcondition: prev = ord.}

OrderFiller:

-ment : Statement

+run() : void

-fillOrders() : void {postcondition: all orders that were ready to execute have been executed and removed from Account Database, postcondition: all changes to players' portfolios from orders that were excuted have been entered in Account Database.}

Portfolio:

-name : String

-assets : ResultSet

+getName() : String {query}

+getValue() : double {query}

+toString() : String {query}

+buy(in stock : String, in amnt : double, in com : double) : boolean {postcondition: if the user has enough cash for the purchase, the stock shares are added to the portfolio and the cash is subtracted from the portfolio.}

+sell(in stock : String, in amnt : double, in com : double) : boolean {postcondition: if the user has enough shares to sell, the shares are subtracted from the portfolio and the income is added to the portfolio.}

SetAlertScreen:

-carrier : JComboBox

-phoneField : JTextField

-tickField : JTextField

-priceField : JTextField

-codeField : JTextField

-upper : JRadioButton

-lower : JRadioButton

-submit : JButton

-verify : JButton

-phoneLabel : JLabel

-tickLabel : JLabel

-priceLabel : JLabel

-codeLabel : JLabel

-upLowLabel : JLabel

-subLabel : JLabel

-numTrials : int

-maxTrials : int = 5 {readOnly}

-code : String

-hosts : String[6] {“message.alltel.com”, “txt.att.net”, “messaging.nextel.com”, “messaging.sprintpcs.com”, “tmomail.net”, “vtext.com”} {readOnly}

+actionPerformed(in ev : ActionEvent) : void {precondition: numTrails < maxTrials, precondition: AccountManager.con is not null, bodycondition: if submit is the source of ev, sendCode(...) is called, bodycondition: if verify is the source of ev and the correct code has been entered in codeField, AccountManger.setThreshold(...) is called and code is set to null, postcondition: if verify is the source of ev and the correct code has been entered in codeField, numTrials is set to 0.}

-sendCode(in address : String) : void {postcondition: code is set to randomly generated string.}

StockDetailScreen.StockDetailPanel:

-chart : Image

#paintComponent(in g : Graphics) : void {precondition: StockDetailScreen.ticker is not null, precondition: StockDetailScreen.period is not null, postcondition: stock chart image is loaded from Yahoo! Finance to chart and displayed on the panel.}

StockDetailScreen:

-dispPanel : StockDetailPanel

-inPanel : JPanel

-oneDay : JRadioButton

-fiveDay : JRadioButton

-threeMonth : JRadioButton

-sixMonth : JRadioButton

-oneYear : JRadioButton

-twoYear : JRadioButton

-fiveYear : JRadioButton

-tickField : JTextField

-go : JButton

-ticker : String

-period : String

+actionPerformed(in ev : ActionEvent) : void {precondition: AccountManager.con is not null, postcondition: ticker is set to the text of tickField, postcondition: period is set to either “1d”, “5d”, “3m”, “6m”, “1y”, “2y”, or “5y”.}

StockList:

+getPrice(in tic : String) : double {precondition: AccountManager.con is not null, postcondition: returns the price of the stock associated with tic.}

+getQuote(in tic : String) : String {precondition: AccountManager.con is not null, postcondition: returns a String containing the stock ticker, its current price, and today's change in price.}

StockWatcher:

-dispPanel : JPanel

-inPanel : JPanel

-fieldPanel : JPanel

-dispPane : JScrollPane

-watchList : String[*]

-quotes : JTextField[*]

-header : JTextField

-input : JTextField

-go : JButton

-exit : JButton

-nextOpenField : int

-updater : StockWatchUpdater

+actionPerformed(in ev : ActionEvent) : void

+add(in tick : String) : void {postcondition: tick is placed in watchList, and the corresponding stock quote is placed in one the elements of quotes.}

+remove(in pos : int) : void {postcondition: the ticker at watchList[pos] is removed, and the stock quote at quotes[pos] is removed.}

StockWatcher.StockWatchUpdater:

+run() : void {bodycondition: each non-empty element of StockWatcher.quotes is updated with the current stock quote.}

11d Object Constraint Language Contracts (OCL)

context AccountManager::connect() : void

post: self.con = DriverManager.getConnection(self.url)

context AccountManager::createAccount(name : String, pass : String) : Portfolio

post: let userNames : Set = self.con.createStatement().executeQuery(“select name from accounts”)

let passwords : Set = self.con.createStatement().executeQuery(“select pass from accounts”)

if !userNames.exists(n | n = name) @ pre then

userNames.exists(n | n = name)

passwords.exists(p | p = pass)

context AccountManager::logout(p : Portfolio) : void

post: p = null

context AccountManager::removeAccount(name : String) : void

post: let account : Set = self.con.createStatement().executeQuery(“select * from accounts where name = '” + name “’”)

let assets : Set = self.con.createStatement().executeQuery(“select * from assets where name = '” + name “’”)

let orders : Set = self.con.createStatement().executeQuery(“select * from orders where name = '” + name “’”)

account.isEmpty()

assets.isEmpty()

orders.isEmpty()

context AccountManager::getOrders(name : String) : void

post: let orders : Set = self.con.createStatement().executeQuery(“select * from orders where name = '” + name “'”)

orders.isEmpty()

context AccountManager::storeOrders(ord : Order) : void

post: let new_orders : Set = {ord, ord.next(), ord.next().next() ... }

let all_orders : Set = self.con.createStatement().executeQuery(“select * from orders where name = '” + name “'”)

all_orders.includesAll(new_orders)

context AccountManger::setThreshold(address : String, ticker : String, price : Real, upLow : Int) : void

post: let alerts : Set = self.con.createStatement().excuteQuery(“select * from alerts”);

alerts.exists(a | a.getString(“address”) = address and a.getString(“ticker”) = ticker and a.getDouble(“price”) = price and a.getInt(“upLow”) = upLow)

context AccountScreen.BuySellScreen inv:

self.market.isSelected() or self.limit.isSelected() or self.stopLoss.isSelected() or self.trailStop.isSelected()

context AccountScreen.BuySellScreen::placeOrder() : void

pre: self.stockField.getText() <> null

self.numField.getText() <> null

post: let ord : Order = new Order()

ord.next() = AccountScreen.pending @ pre

AccountScreen.pending = ord

```
context DatabaseMaintainer inv:
let entries : Set = conn.createStatement().executeQuery(“select * from stock_table”)
entries.isUnique(getString(“ticker”))

context DatabaseMaintainer::update() : void
let entries : Set = conn.createStatement().executeQuery(“select * from stock_table”)
post: forAll(tick | self.yahooInterface(tick) <> self.yahooInterface(tick) @ pre)
entries.select(getString(“ticker”)=tick) <> entries.select(getString(“ticker”)=tick) @ pre

context LeaderBoard::actionPerformed(ev : ActionEvent) : void
post: self = null

context Portfolio inv:
self.assets.isUnique(a | a.getString(“type”));

context Portfolio::buy(stock : String, amnt: Real, com : Real) : boolean
post: let cash : Real = self.assets.select(getString(“type”) = cash).getDouble(“amount”)
let amount : Real = self.assets.select(getString(“type”) = stock).getDouble(“amount”)
if cash > (1 + com) * amnt * StockList.getPrice(ticker) then
self.assets.exists(a | a.getString(“type”) = stock)
amount = amnt + amount @ pre
cash = cash @ pre – (1 + com) * amnt * StockList.getPrice(ticker)

context Portfolio::sell(stock : String, amnt : Real, com : Real) : boolean
```

post: let cash : Real = self.assets.select(getString(“type”) = cash).getDouble(“amount”)

let amount : Real = self.assets.select(getString(“type”) = stock).getDouble(“amount”)

if amount > amnt then

amount = amount @ pre – amnt

cash = cash @ pre + (1 – com) * amnt * StockList.getPrice(ticker)

else if amount = amnt then

not self.assets.exists(a | a.getString(“type”) = stock)

context SetAlertScreen inv:

self.numTrials >= 0

context SetAlertScreen::actionPerformed(ev : ActionEvent) : void

pre: self.numTrials < self.maxTrials

post: if ev.getSource() = self.verify then

if self.codeField.getText() = self.code @ pre then

self.numTrials = 0

else

self.numTrials = self.numTrials @ pre + 1

context SetAlertScreen::sendCode(phoneNumber : String) : void

post: self.code <> null

context StockDetailScreen inv:

self.oneDay.isSelected() or self.fiveDay.isSelected() or self.threeMonth.isSelected() or self.sixMonth.isSelected() or self.oneYear.isSelected() or self.twoYear.isSelected() or self.fiveYear.isSelected()

context StockDetailScreen::actionPerformed(ev : ActionEvent) : void

pre: AccountManager.con <> null

post: if StockList.getPrice(self.tickField.getText()) <> -1 then

self.ticker = tickField.getText()

post: if self.oneDay.isSelected() then

self.period = “1d”

else if self.fiveDay.isSelected() then

self.per iod = “5d”

else if self.threeMonth.isSelected() then

self.period = “3m”

else if self.sixMonth.isSelected() then

self.period = “6m”

else if self.oneYear.isSelected() then

self.period = “1y”

else if self.twoYear.isSelected() then

self.period = “2y”

else if self.fiveYear.isSelected() then

self.period = “5y”

context StockList::getPrice(tic : String) : Real

pre: AccountManager.con <> null

context StockList::getQuote(tic : String) : String

pre: AccountManager.con <> null

context StockWatcher inv:

self.nextOpenField >= 0

context StockWatcher::add(tick : String) : void

post: watchList[nextOpenField @ pre] = tic

quotes[nextOpenField @ pre].getText() = StockList.getQuote(tick)

nextOpenField = nextOpenField @ pre + 1

context StockWatcher::remove(pos : int) : void

pre: watchList.size() > pos

post: forAll(x | pos <= x < nextOpenField)

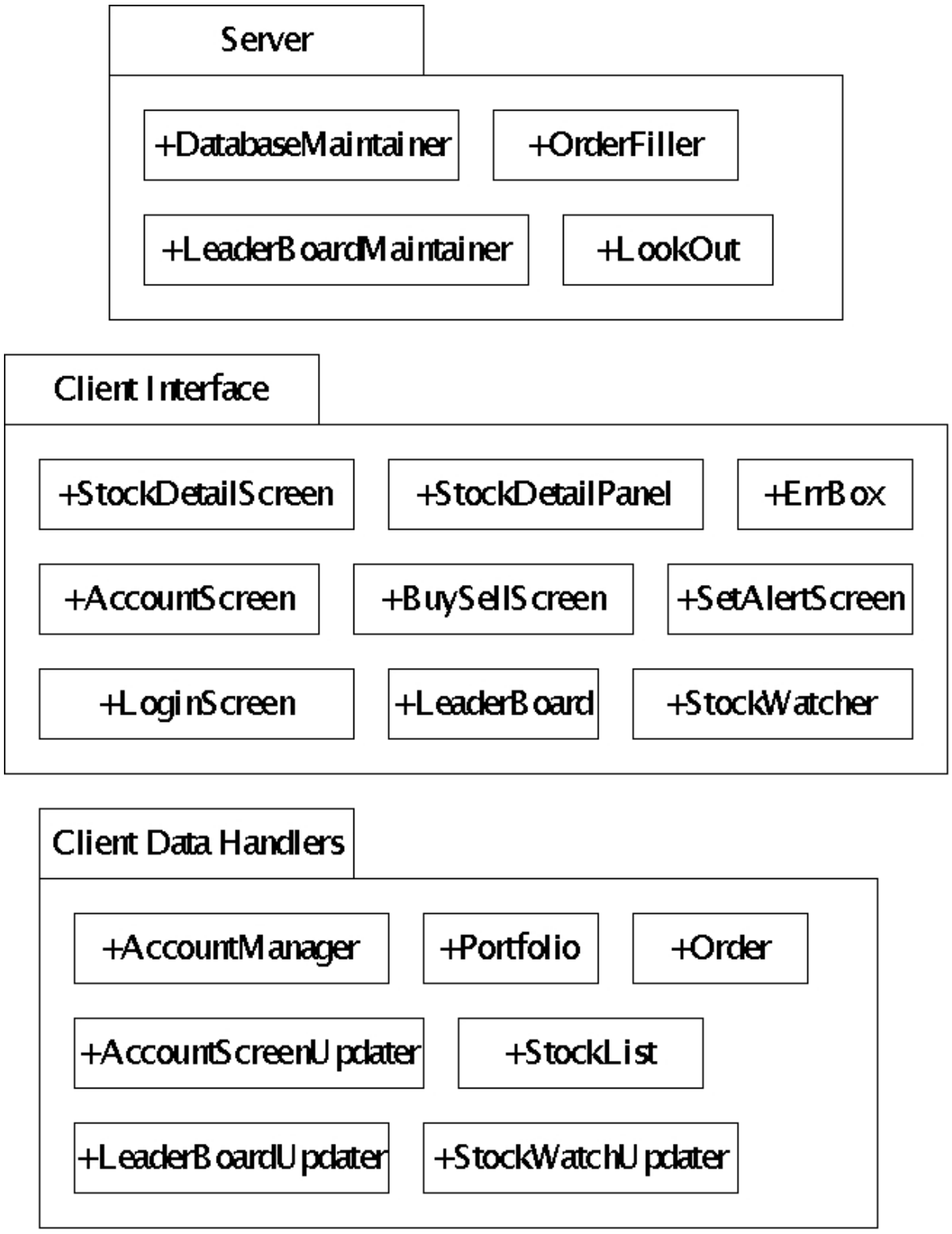
watchList[x] = watchList[x+1] @ pre

quotes[x] = quotes[x+1] @ pre

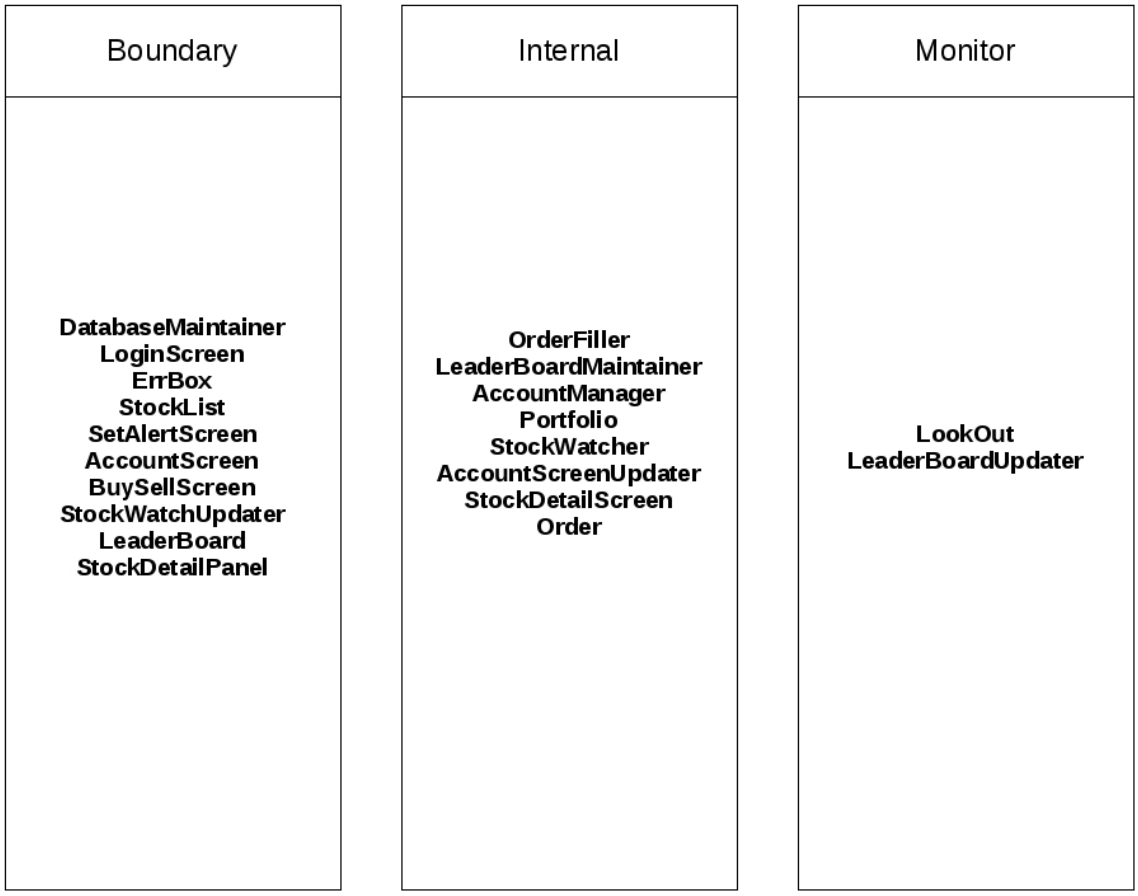
nextOpenField = nextOpenField @ pre – 1

12. System Architecture and System Design

12b Identifying Subsystems



As shown above, this system can broken down into 3 subsystems. The server subsystem consists of classes that run on the server and are responsible for keeping Account Database up-to-date. The client interface subsystem consists of GUIs that allow the user to enter input and view output. The process of obtaining output from user inputs is handled behind-the-scenes by the client data handlers subsystem. The StockDetailPanel class, hoowever, is an exception. It generates the output it displays on its own. The reason for this is that the output for the StockDetailPanel is an image, and it is easier to generate the image inside the GUI than to have another class generate the image and pass it to the GUI.



12a Architectural Styles

We are creating a component-based database-centric application. This is a combination of the component-based software engineering, client-server architectural style, and database-centric architectures. The server and client subsystems each rely heavily on data stored in Account Database. The client interface subsystem also uses an event-driven architecture.

The program as a whole is organized into "components" in order to ease debugging, scalability, reduce complexity, and improve organization of the code. In addition to the component layout of the code, the application is heavily relent on a central database. All use cases involve the main database at some point in their execution.

12c Mapping Subsystems to Hardware

This system is split between a server machine and client machines. The server subsystem runs on the server machine. The other subsystems run on a client machine.

12d Persistent Data Storage

This system uses a lot of data that must persist between executions of the system. We use two different methods of storing data. Most data is stored in a MYSQL database in the following tables:

accounts
name : varchar(50)
pass : varchar(50)
value : double

assets
name : varchar(50)
type : varchar(50)
amount : double
<<primary key>> row_key : varchar(60)

stock_table
name : varchar(50)
ticker : varchar(50)
price : double
price_change : double

alerts
address : varchar(50)
ticker : varchar(50)
price : double
up_low : int

orders
name : varchar(50)
ticker : varchar(50)
type : varchar(50)
amount : double
buy_sell : int
price : double
percent : double
commission : double

The database is used to store every piece of persistent data except for SetAlertScreen.numTrials. This piece of data is saved on the user's machine as a simple text file. The reason for not putting it on the database with everything else is that this data is only changed during the execution of the user subsystem. All of the other data may need to be changed by the server subsystem while the user subsystem is turned off.

12e Network Protocol

This system uses two different communications protocols. The system connects to the Account Database using Java JDBC and connects to Yahoo! Finance using HTTP. JDBC provides several classes and interfaces for connecting to an SQL database within a Java program. Since the system is implemented entirely in Java, that makes JDBC the logical choice for connecting to the Account Database. The one drawback of JDBC is that the implementation of the classes and interfaces it provides are dependent on the JDBC driver that is used. This requires that a JDBC driver be distributed with the program. The choice of HTTP is driven simply by the fact that HTTP urls for stock quotes and charts on Yahoo! Finance are readily available.

12f Global Control Flow

This system uses several threads, some of which are process-driven and some of which are event-driven. The login screen has a single, event-driven thread that simply waits for the user to enter his or her login information. The account screen has two threads that run concurrently. One is event-driven; it waits for the user to select an operation. The other is process-driven; it periodically processes pending orders and updates the display of the portfolio contents and takes no input from the user. Since the event-driven thread does not alter the contents of the portfolio at all, there is no synchronization required between threads. The StockWatcher and LeaderBoard windows follow the same pattern. Each has one thread that waits for user input and one thread that periodically updates the displayed information. Similarly, the StockDetialsScreen has one thread that waits for user input and one that periodically repaints the display, but in this case the second thread is already implemented by Java. In addition to these threads, which run on the user's machine, there are several process-driven threads that are constantly running on the server machine. A DatabaseMaintainer thread periodically updates the stock prices in Account Database, a LeaderBoardMaintainer thread updates the portfolio values in Account Database, a OrderFiller thread processes pending orders stored in Account Database, and a LookOut thread periodically checks to see if any of the alert thresholds in Account Database have been exceeded.

12g Hardware Requirements

The system's server is going to require an internet connection to connect to Yahoo! Finance and so the users can log in and play the game. In addition, the server should be able to run Java and MySQL smoothly. There should be enough hard drive space in order to hold user data in databases.

Users will need an internet connection, monitor, and enough hard drive space to install the game. The specific minimum requirements of each hardware device have not yet been determined since our program is still in the development stage.

13. Algorithms and Data Structures

13a Algorithms

There are currently no complex algorithms in use in the program. If algorithms were to exist in the project, they would deal with certain predictions of the stock market. One instance of such an algorithm would be predicting the expected price of a stock of interest, and providing the user with recommendations. If time permits, an implementation of a database can be used with these algorithms to track the trend of each stock. Another algorithm could exist where it can predict whether or not a user's choice is too risky considering the condition of their portfolio.

13b Data Structures

The program uses two types of data structures. The StockWatch application uses an array of strings to store the ticker names the user is currently watching. The other data structures used are data tables loaded from the databases.

In both cases these data structures were the only sensible options available.

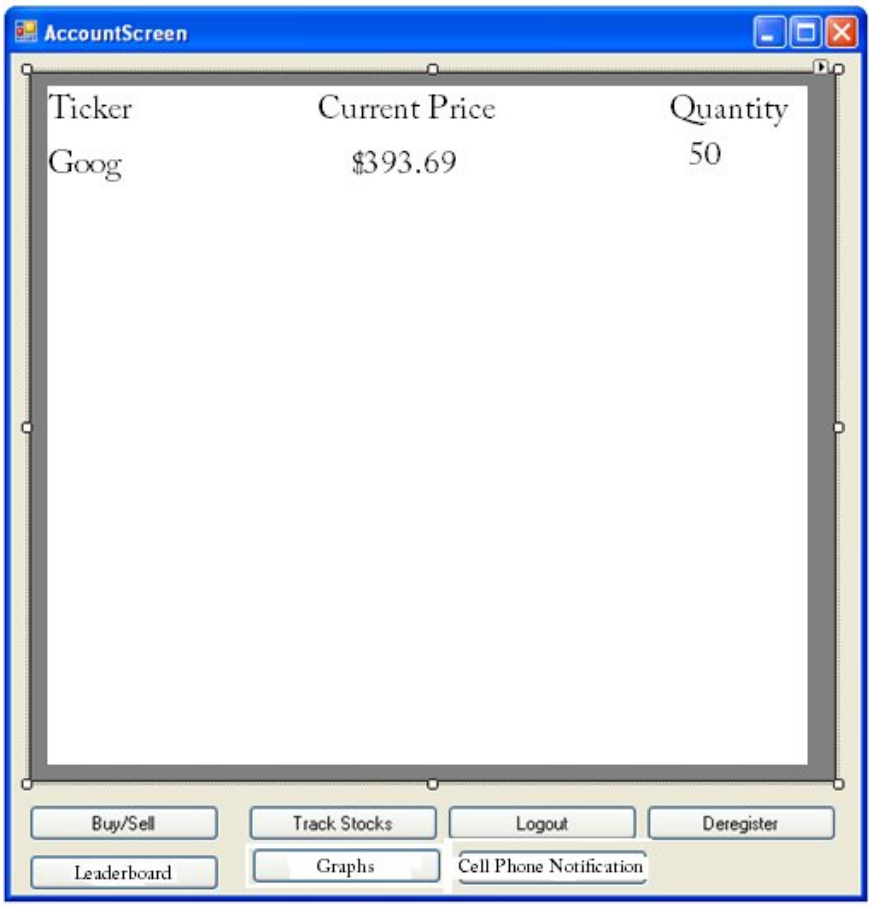
14. User Interface Design Implementation

The primary focus of our user interface design is to make the user's interaction as simple and efficient as possible

LoginScreen:

The login screen allows a user to login using an existing account, create a new one, or exit the program. Login and Register require two text entries and a button click, the exit button only requires one click.

AccountScreen:

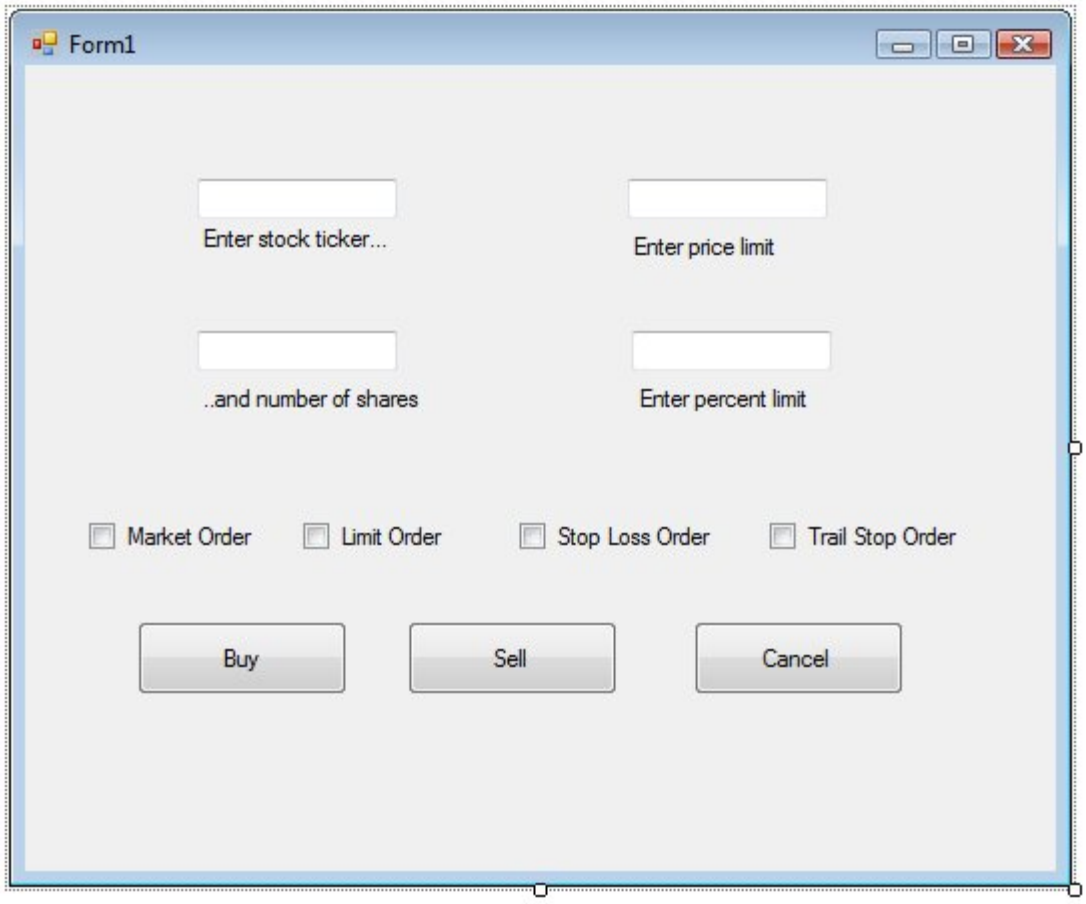


When a user successfully logs on they will first encounter the AccountScreen. The AccountScreen has four click boxes and a display pane. The boxes enable the user to Buy/Sell stocks, Track Specific Stocks, Logout or Deregister.

The display pane shows the current user their holdings and the value of these holdings. It does this by calling the Portfolio class's toString() overloaded function. This function accesses the database of the current user and displays their holdings in the display pane.

In order to keep the display updated an AccountScreenUpdater is initiated as a background process. This daemon application continuously updates the AccountScreen window pane by calling the toString() function at a specified interval of time.

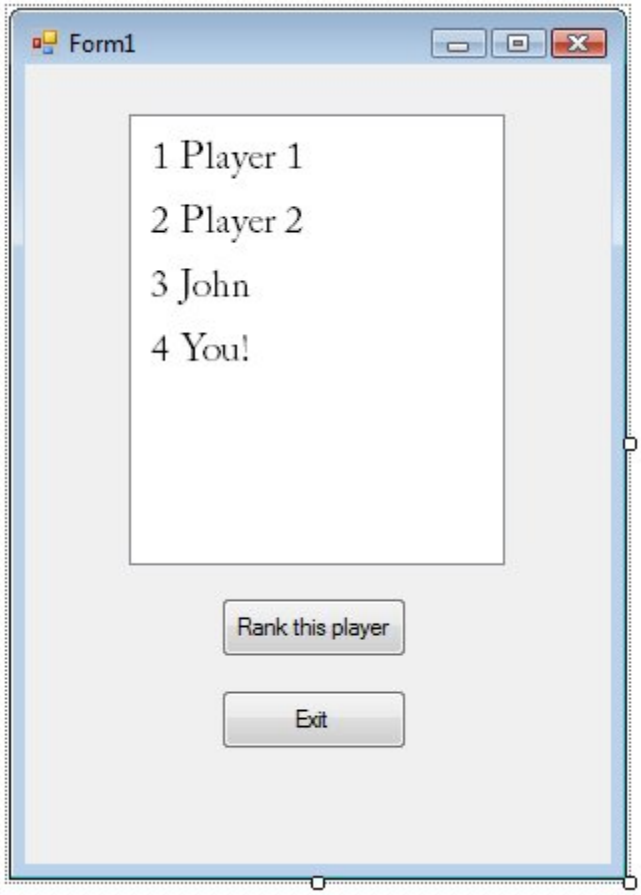
Buy/Sell Screen



The Buy/Sell screen gives the functionality of Market Orders and Limit Orders (w/ percent limit). The user is given text boxes to enter in the required information, options of what type of purchase via a check box and then buttons to perform the buy or sell. The cancel button allows them to exit this screen if they do not wish to buy or sell a security. A market order requires one ticker entry, one number of share entry, and two clicks. A limit order requires three text entries and two button clicks.

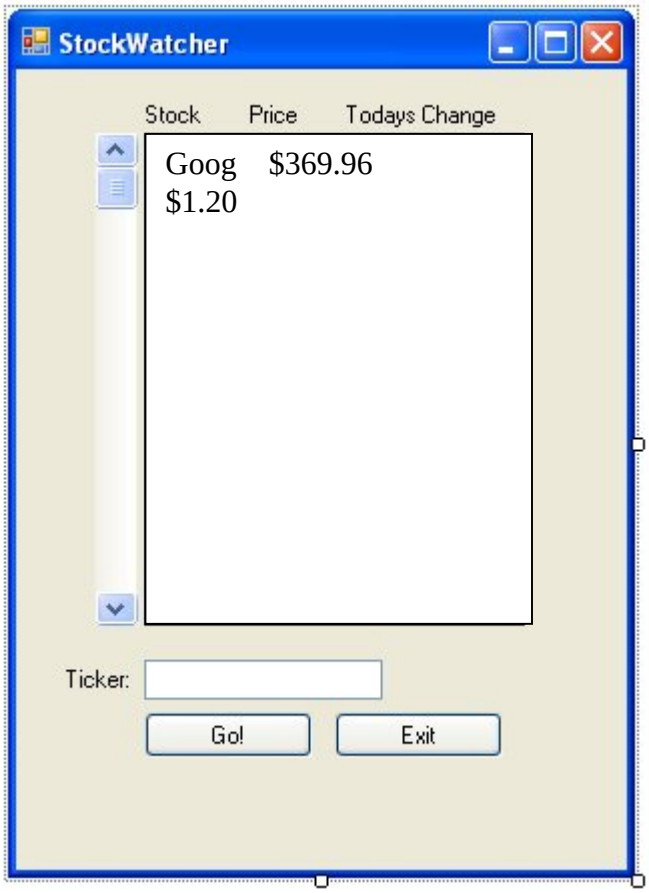
The previous user interface incorporated the buy and sell features into the account screen. This was done to minimize the number of mouse clicks needed to use our program. It, however, made the program feel less intuitive and has now been changed to create a more fluid environment.

Leaderboard



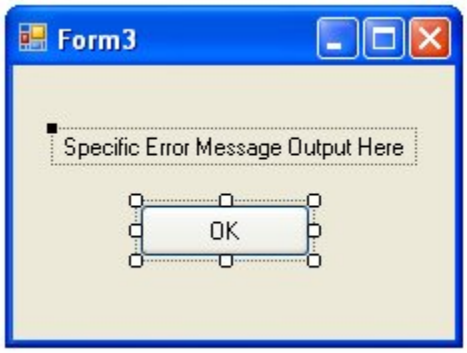
The leaderboard shows a lists of the leaders in the current game. You can click the 'Rank this player' button to see your current rank or exit to leave this window.

StockWatch:

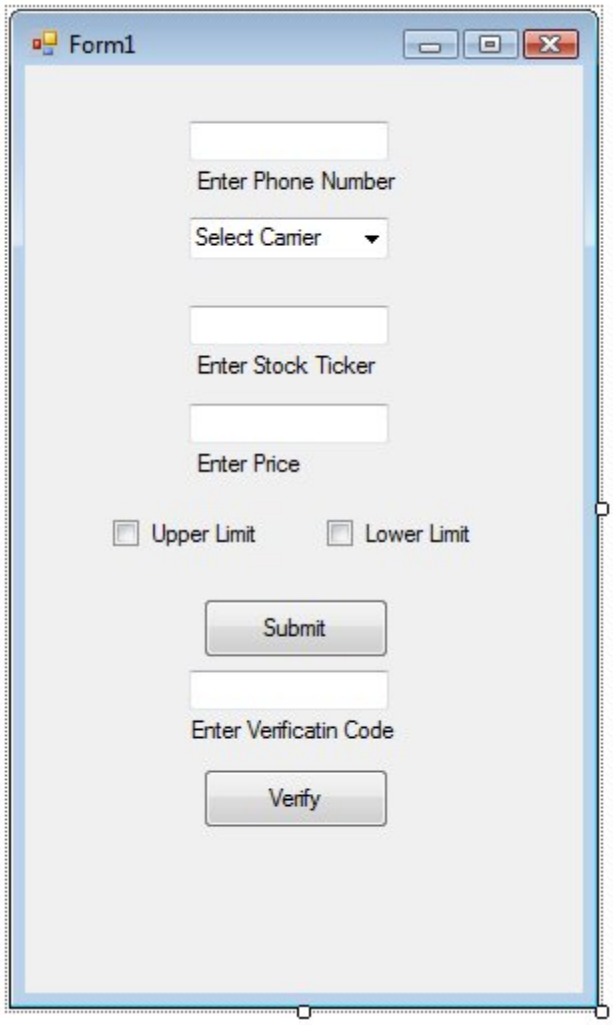


The StockWatch interface is a newly added piece of interface in our design. It enables the user to enter multiple different ticker prices and watch their values change throughout the day. It does this by storing a string array of tickers and calling a StockWatchUpdater, a daemon application, similarly to the AccountScreenUpdater. The StockWatchUpdater runs through the string of tickers and outputs their data stored in the database onto the screen.

ErrBox:



When an error occurs this simple error box will be displayed with the appropriate error message.
Cell Phone Notification UI



This user interface item allows a user to enter a phone number and receive a text notification when a stock price reaches a certain price. There is always 4 entries, and 5 other clicks.

Stock Graphs

The stock graph window allows users to see the display of an inputted stock ticker. The user can choose from 7 different Y-axis options from 1 day to 5 years.

Button Functionality
Price Find: Searches the stock ticker string and will return the price of that stock.
Buy/Sell Stock: Checks if your fund are sufficient and if so purchases or sells the specified number of shares.
Leader Board: Changes the display from your data to a leader board.
Cell Phone Notification: Enter phone number for text notifications
Graph: Views the trend of a stock's price over a specified amount of time
Log Out: Returns the user to the log in screen.
Deregisters: Erases a user from the database

User Effort Estimation:

Login
NAVIGATION
1 click

DATA ENTRY
a. Click to text window "Username"
b. Type Username
c. Click to text window "Password"
d. Type Password
e. Click "Login"

Buy Stock/Sell Stock
NAVIGATION
1 click
DATA ENTRY
a. Click to text window Enter Ticker
b. Type stock's ticker
c. Type quantity of shares
d. Enter price limit
e. Enter percent limit
f. Click either market order, limit order, stop loss order, trail stop order
g. Click buy or sell

Ticker Finder
NAVIGATION
1 click
DATA ENTRY
a. Click to text window
b. Type stock's ticker

Player Data/Leaderboard
NAVIGATION
1 click

Error

Navigation
1 Click “OK”

Signup

NAVIGATION
1 click
DATA ENTRY
a. Click to text window “Username”
b. Type new Username
c. Click to text window “Password”
d. Type new Password
e. Click “Confirm”

Stockwatcher

NAVIGATION
1 click
DATA ENTRY
a. Click to text window
b. Type stock’s ticker
c. Press enter

Portfolio

NAVIGATION
1 click to account screen

Remove Account

NAVIGATION
1 Click (De-Register)
DATA ENTRY
a. Click to text box
b. Enter password
c. Press enter

Create Account

NAVIGATION
1 Click (Register)
DATA ENTRY
a. Click to text box
b. Enter Username
c. Enter Password

View Graph

NAVIGATION

1 Click (Graphs)
DATA ENTRY
a. Enter Ticker
b. Select either 1 day, 5 days, 3 months, 6 months, 1 year, 2 years, 5 years

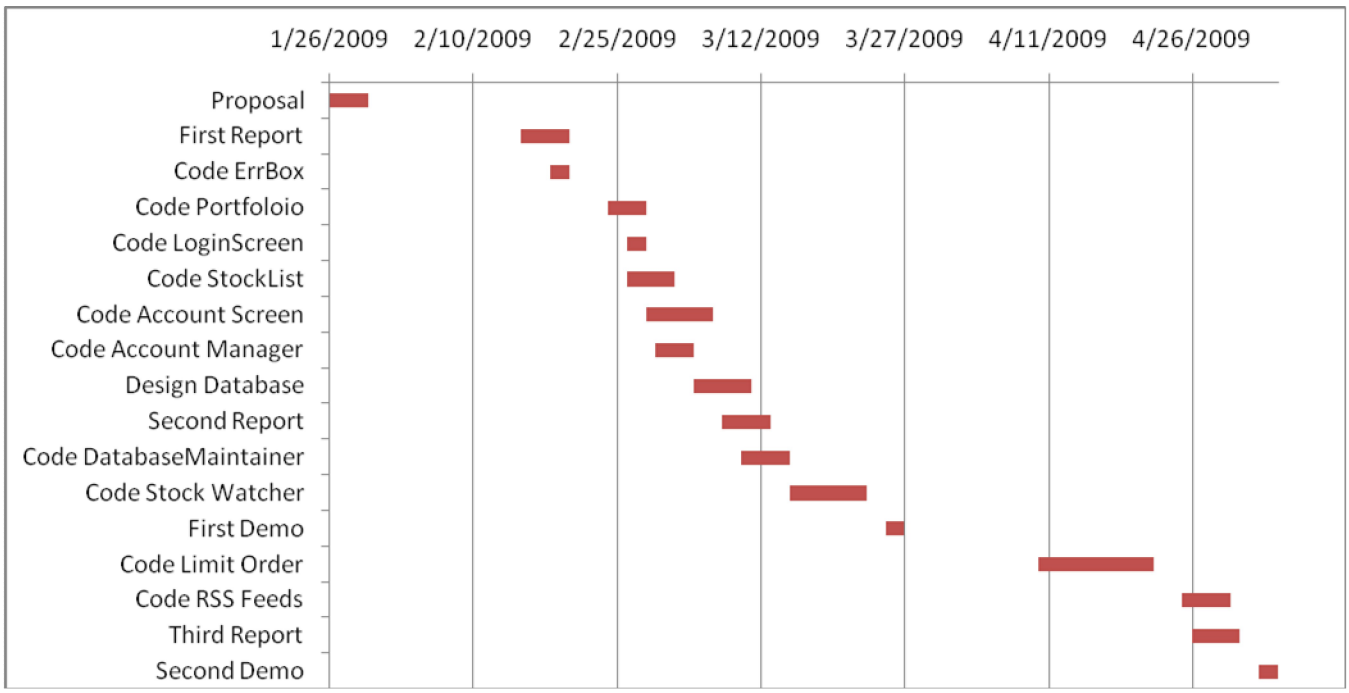
Update Price

No navigation or data entry, automatically done

Cell Phone Notification

NAVIGATION
1 Click (Cell Phone Notification)
DATA ENTRY
a. Enter phone number
b. Select carrier (drop down box)
c. Enter stock ticker
d. Enter Price
e. Select either upper limit or lower limit
f. Enter verification code
g. Submit/Verify

15. History of Work & Current Status of Implementation



- User Interface Design
- Database Maintainer
- UI-Database Interactions
- Graphs
- Cell Phone Notifications
- RSS Feed
- Limit/Money Orders

When we estimated the time it would take to code our program we overestimated the amount of time programming the interface and database queries would be but greatly underestimated how hard the databases would be to set up. We also believed the Limit Order and RSS Feeds would take less to code than they did.

Currently we only need to finish the cell phone notification system and integrating graphs from the internet.

16. Conclusions and Future Work

Conclusions

In developing the stock market fantasy league we had three challenges to overcome. When we first set out to develop this product we needed to increase our understanding of the stock market in order to write a program that emulates a real online brokerage service well, a lack of experience programming with databases, and having our program pull information off of the internet.

An important challenge all software engineers must face when developing software is a need for expertise in not only programming but also in the field that software will be implemented for. This was true in coding a stock market fantasy league. We overcame this hurdle by working together to learn and better understand the stock market. Once we had this knowledge we were able to model the stock market in UML and see a direct translation into object oriented code upon its implementation.

On the programming side we were challenged by needing to code with databases. Our group had a lack of prior knowledge on this subject and set out to learn how to immediately. With the abundance of examples on the internet we were capable of learning databases functionality and in implementing our queries. The thread that runs concurrently on the server was also beyond what we had learned previously, however, in the end we successfully coded this item.

The most common translation for RSS is really simple synchronization. Unfortunately these really simple synchronization feeds caused more headaches than a lot of the other programming. At first the RSS aggregator did not function as it should, but eventually these issues were resolved.

Furthermore, the telephone notification system is still in development and is giving our group some difficulties.

Future Work

A goal that we set out to accomplish in the beginning of our project was to implement the software on a cell phone. The time element of one implementation has been too intensive to be able to continue on and do a second for a mobile device. I, however, argue that it might be more practical to simply wait for cell phone technology to improve and be capable of running the same java virtual machine computers run. When this comes to fruition our software will already be mobile compatible.

Another item that could be implemented by our program is a means to test different algorithms on the stock market. If we could implement this it would create some very interesting methods of testing investing strategies.

17. References

Brend Bruegge & Allen H. Dutoit. Object-Oriented Software Engineering. Upper Saddle River: Pearson Education, Inc., 2001.

Paul T. Tymann & G. Michael Schneider. Modern Software Development Using Java. Pacific Grove, Thomson, 2004.

Dan Pilone & Neil Pitman. UML 2.0 In a Nutshell. Sebastopol, O'Reilly Media, Inc., 2005.

Cay S. Horstmann & Gary Cornell. Core Java: Volume 1 – Fundamentals. Santa Clara, Sun Microsystems Inc., 1959.

Wikipedia. Wikimeda Foundation. www.wikipedia.org

Microsoft Developer Network. Microsoft. Msdn.microsoft.com/en-us/default.aspx