2009

Report 2

Stock Market Fantasy Game

Presented by:

Alex Sood, John Grun, Kevin Folinus, Chris Zalewski & David Meng.

3/13/2009

	John Grun	Kevin Folinus	Chris Zalewski	Alex Sood	David Meng
Project Management	25%	25%	25%	25%	
Interaction Diagrams	50%			50%	
Class Diagram and Interface Specification				100%	
System Architecture and System Design	40%	10%	10%	40%	
Algorithms and Data Structures		50%	50%		
User Interface Design and Implementation		60%	40%		
Progress Report and Plan of Work		50%	50%		

Table of Contents

1. Interaction Diagrams……4

2. Class Diagram and Interface Specification……12
    a. Class Diagram
    b. Data Types and Operation Signatures

3. System Architecture and System Design……16
    a. Architectural Styles
    b. Identifying Subsystems
    c. Mapping Subsystems to Hardware
    d. Persistent Data Storage
    e. Network Protocol
    f. Global Control Flow
    g. Hardware Requirements

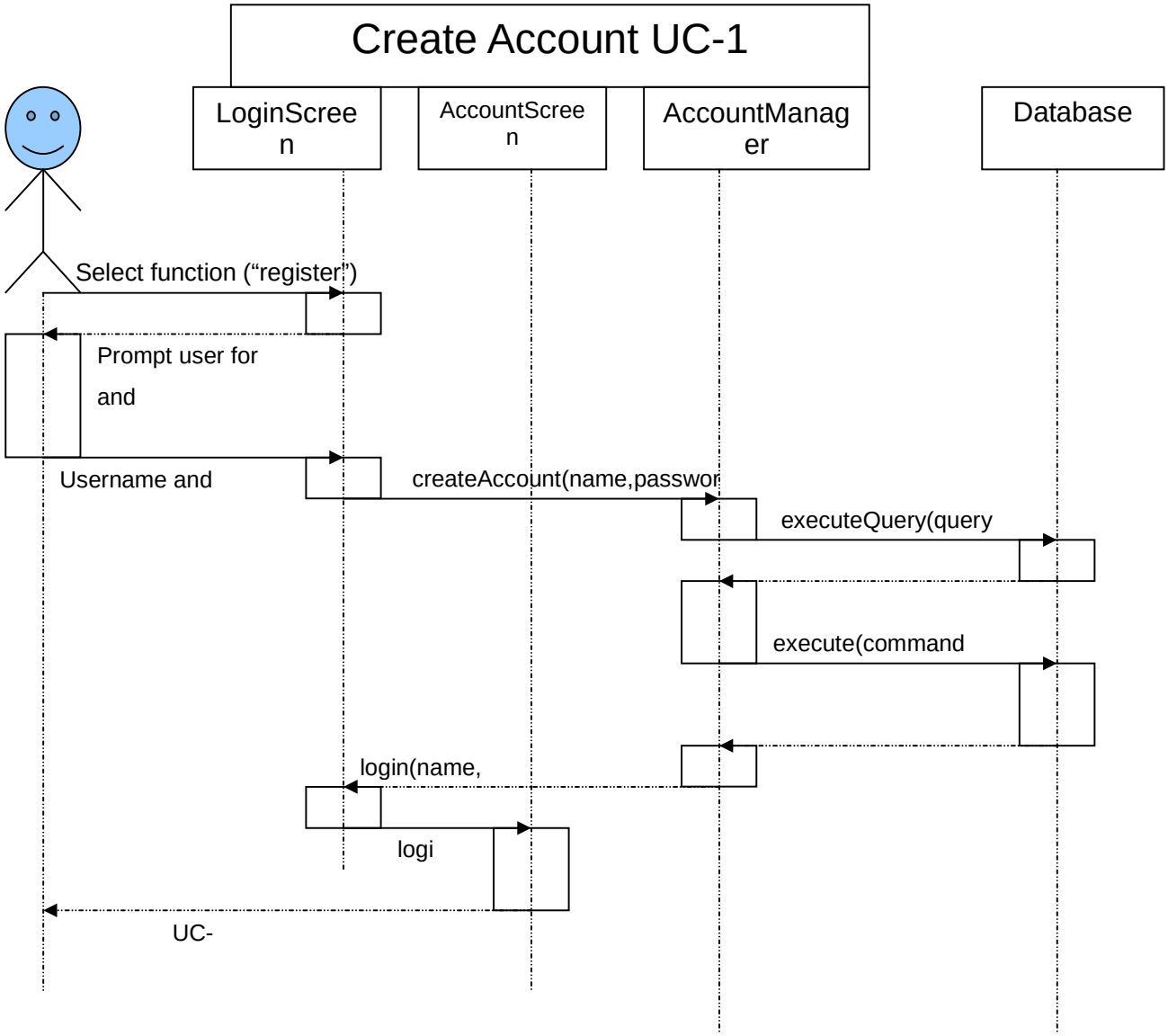
4. Algorithms and Data Structures……19
    a. Algorithms
    b. Data Structures

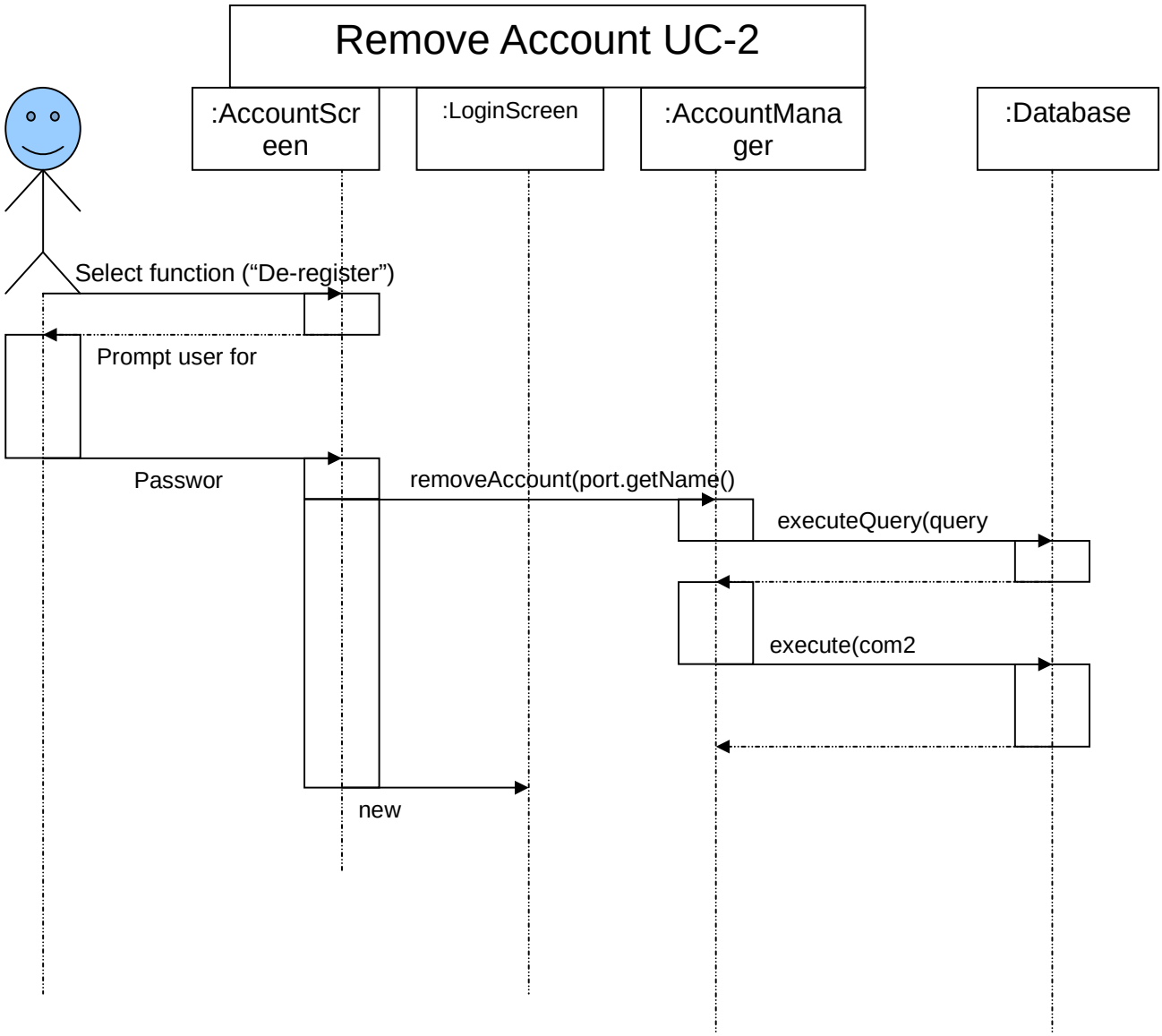
5. User Interface Design and Implementation……20

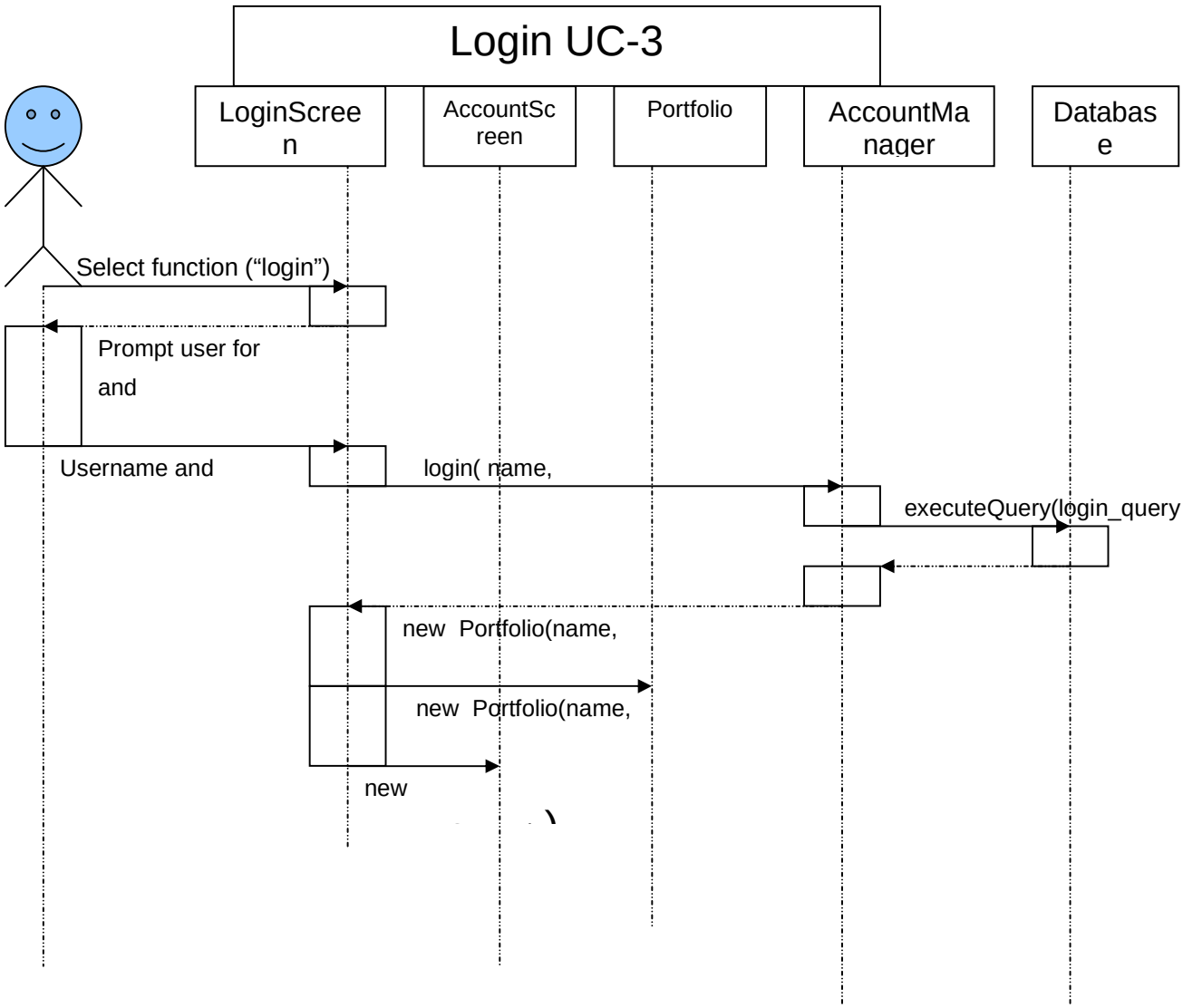
6. Progress Report and Plan of Work……25
    a. Progress Report
    b. Plan of Work
    c. Breakdown of Responsibilities

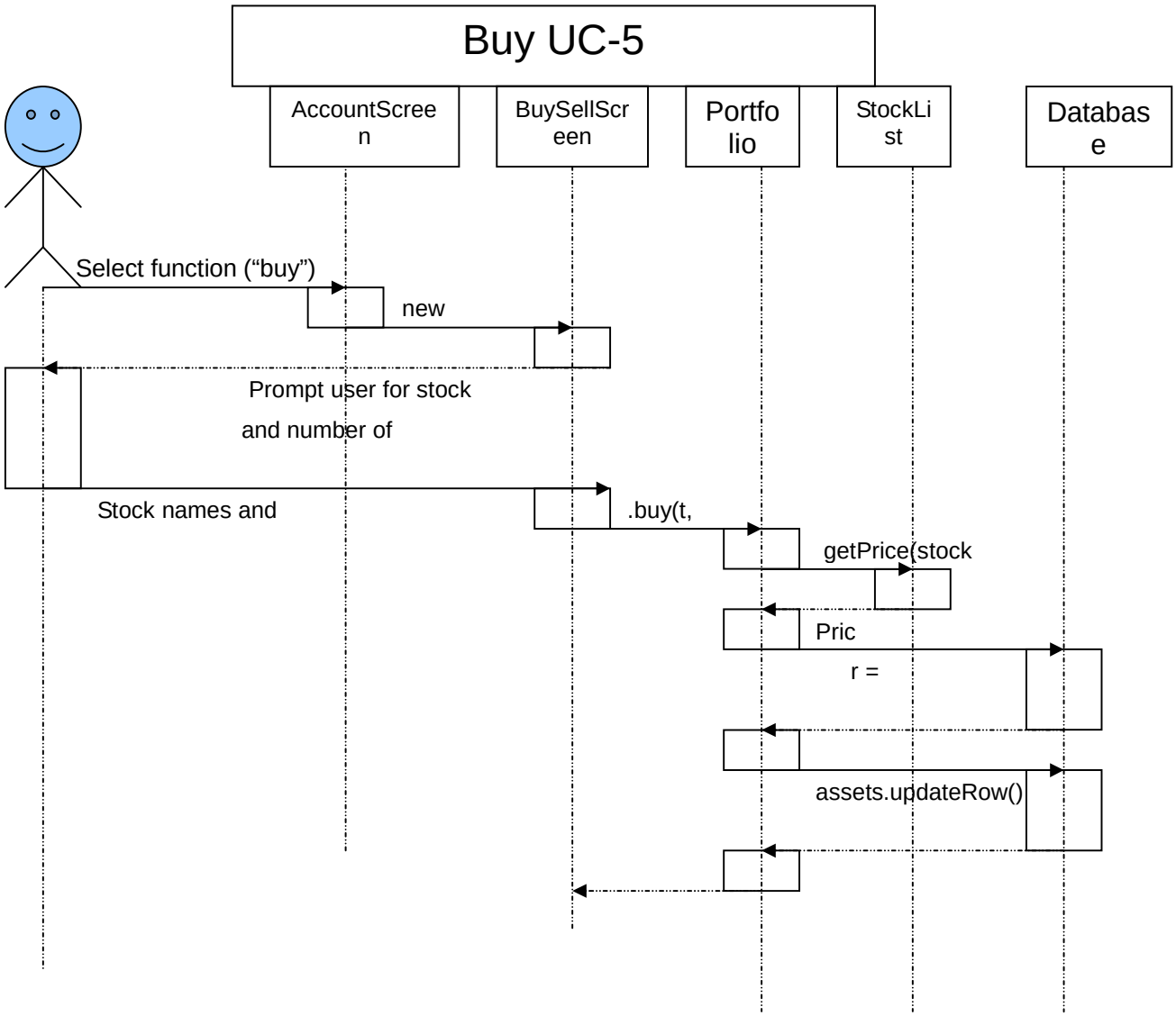
7. References……28

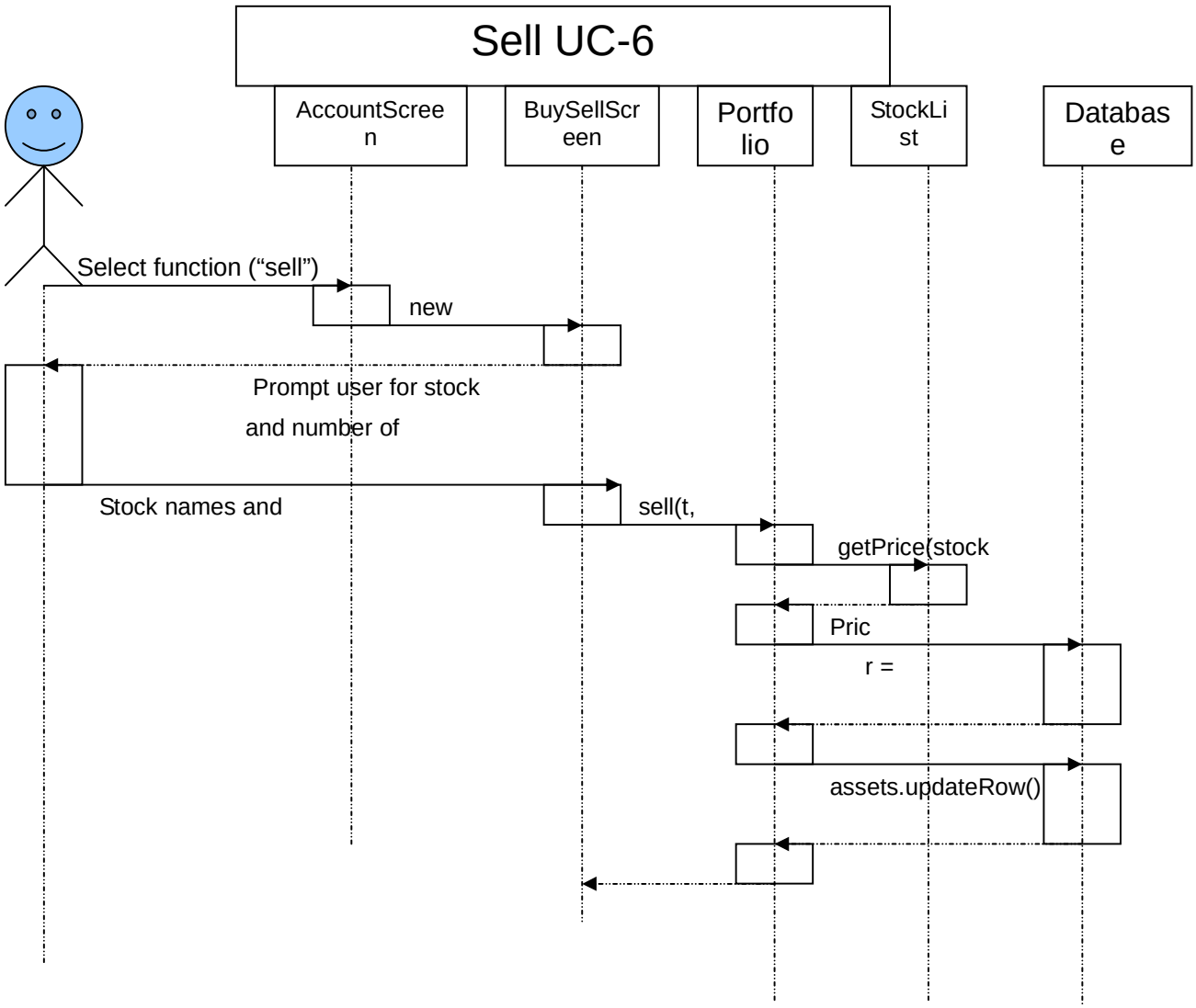
1. Interaction Diagrams

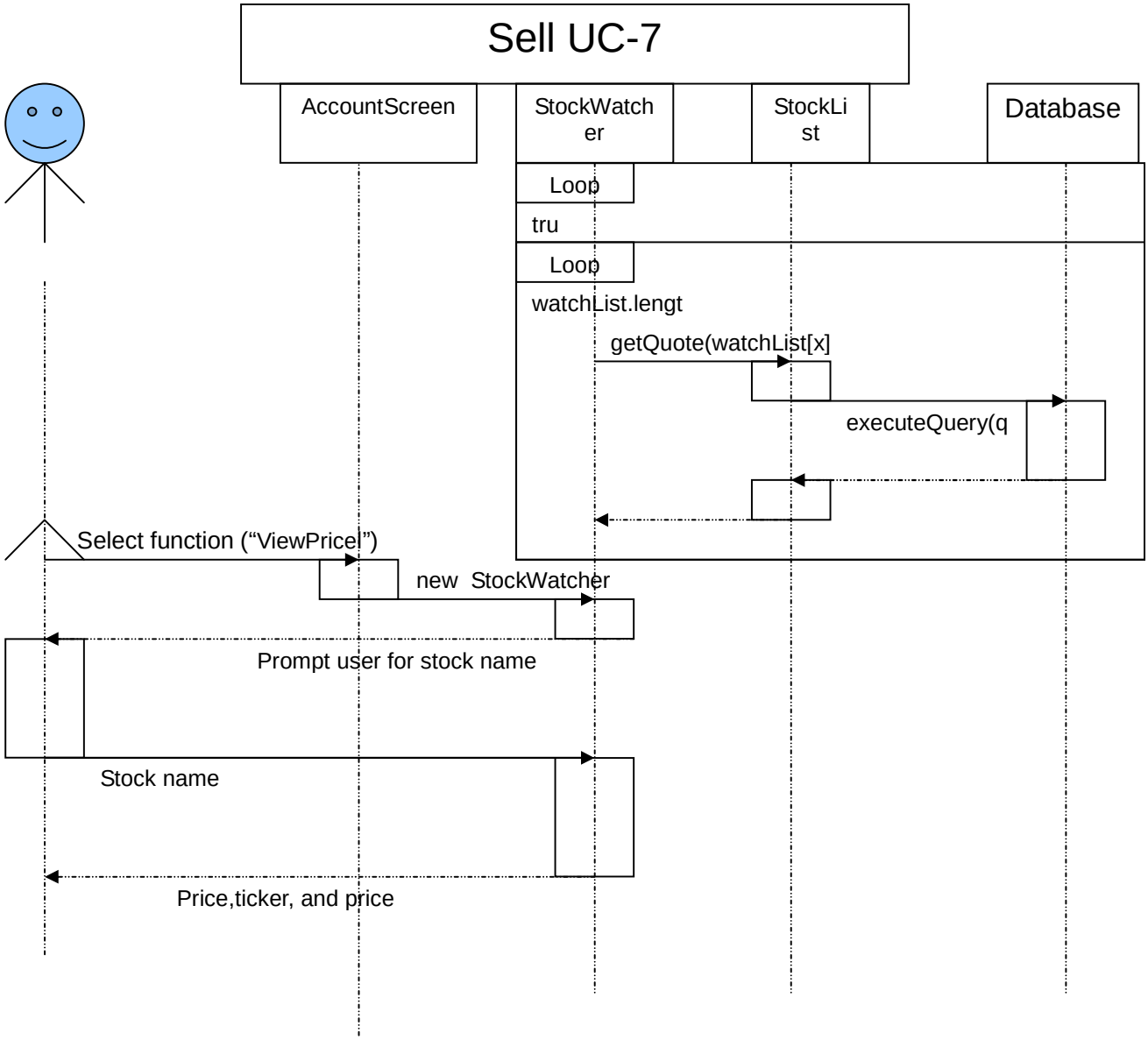


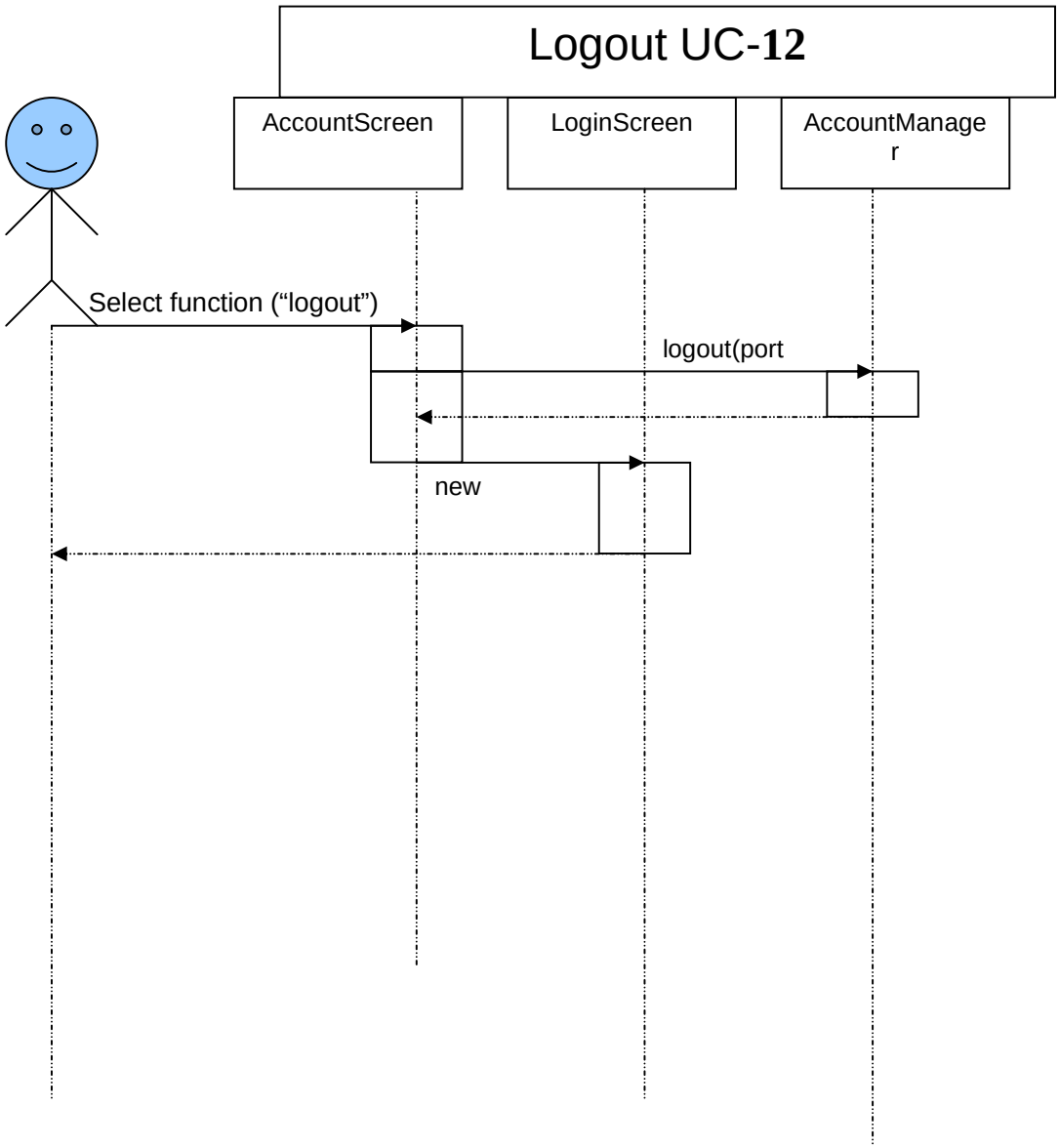

Remove Account UC-2
:AccountScr
een
:LoginScreen
:AccountMana
ger
:Database
Select function (“De-register”)
Prompt user for
Passwor
removeAccount(port.getName()
executeQuery(query
execute(com2
new

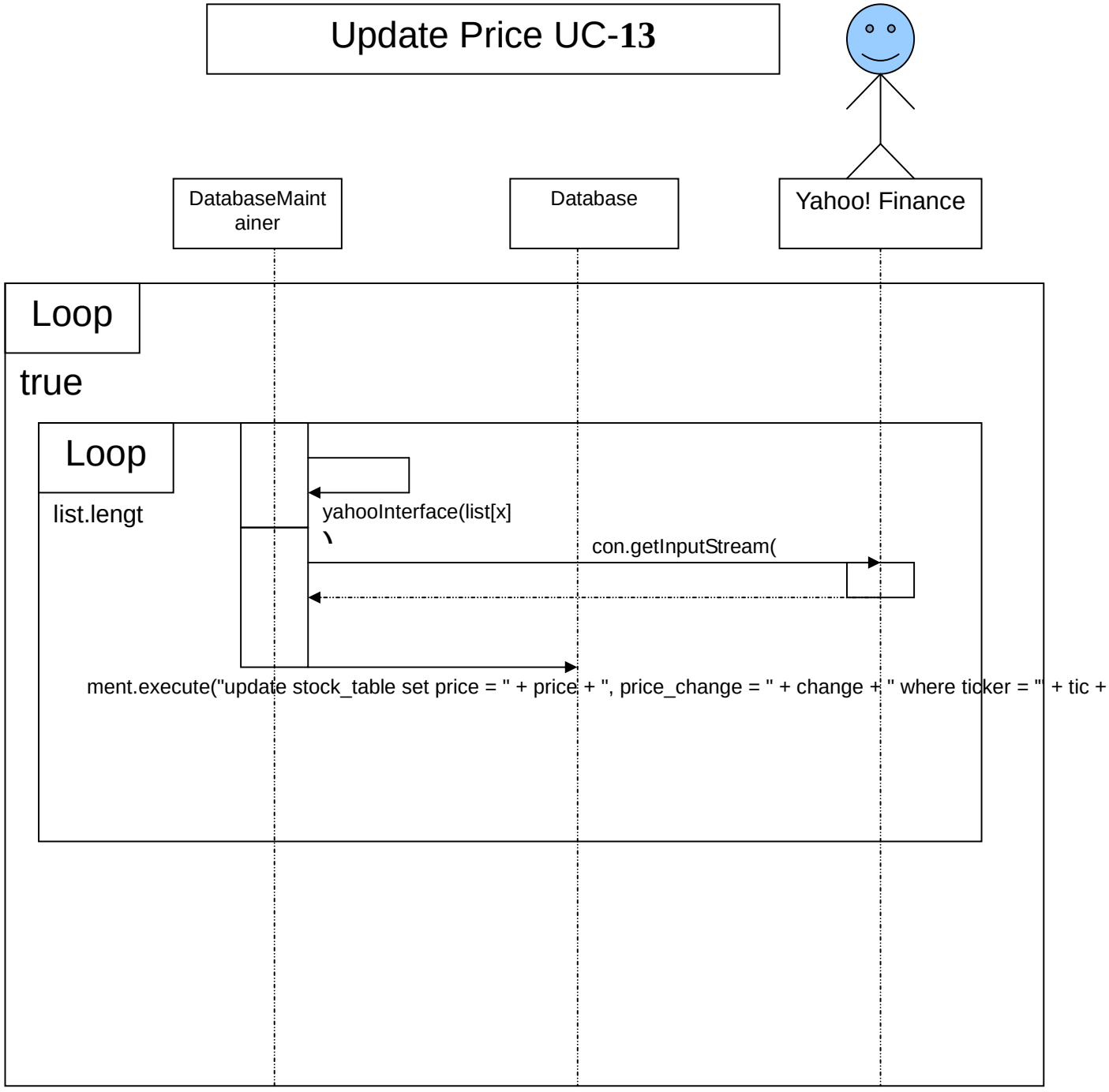

Login UC-3
LoginScree n
AccountSc reen
Portfolio
AccountMa nager
Databas e
Select function (“login”)
Prompt user for
and
Username and
login(name,
executeQuery(login_query
new Portfolio(name,
new Portfolio(name,
new


Buy UC-5
AccountScreen
BuySellScreen
Portfolio
StockList
Database
Select function (“buy”)
new
Prompt user for stock
and number of
Stock names and
.buy(t,
getPrice(stock
Pric
r =
assets.updateRow()


Sell UC-6
AccountScreen
BuySellScreen
Portfolio
StockList
Database
Select function (“sell”)
new
Prompt user for stock
and number of
Stock names and
sell(t,
getPrice(stock
Pric
r =
assets.updateRow()

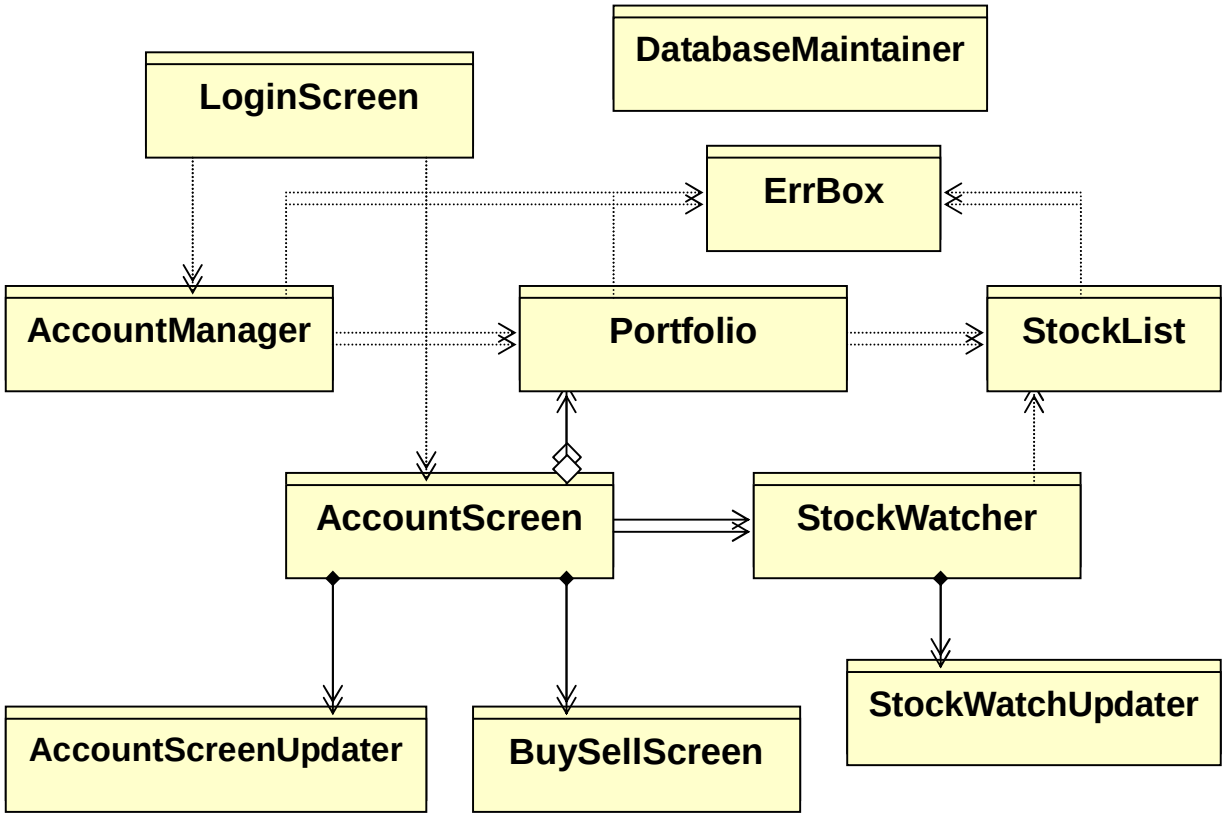

Sell UC-7
AccountScreen
StockWatcher
StockList
Database
Loop
tru
Loop
watchList.lengt
getQuote(watchList[x]
executeQuery(q
Select function (“ViewPrice!”)
new StockWatcher
Prompt user for stock name
Stock name
Price,ticker, and price


Logout UC-12
AccountScreen
LoginScreen
AccountManage
r
Select function (“logout”)
logout(port
new


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

2. Class Diagram and Interface Specification

a. Class Diagram



b. 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.}

AccountScreen:

-dispPanel: JPanel
-butPanel: JPanel
-dispArea: JTextArea
-buySell: JButton
-viewPrice: JButton
-logout: JButton
-deregister: JButton
-port: Portfolio
-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 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().}

AccountScreen.BuySellScreen:

-inPanel: JPanel
-buttonPanel: JPanel
-stockField: JTextField
-numField: JTextField
-buy: JButton
-sell: JButton
-cancel: JButton
-stockLabel: JLabel
-numLabel: JLabel

+actionPerformed(in ev: ActionEvent): void {precondition: stockField and numField are non-empty, bodycondition: if buy is the source of ev, Portfolio.buy(...) is called with entered stock ticker and number fo shares, bodycondition: if sell is the source of ev, Portfolio.sell(...) is called with entered stock ticker and number of shares.}

DatabaseMaintainer:

-list: String[*] {readOnly}
-url: String = “jdbc:mysql://localhost/group902?user=user&password=password” {readOnly}
-ment: Statement
+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

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.

Portfolio:

-commission: double = 0.02 {readOnly}
-name: String
-assets: ResultSet

+getName(): String {query}
+getValue(): double {query}
+toString(): String {query}
+buy(in stock: String, in amnt: 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): 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.}

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:

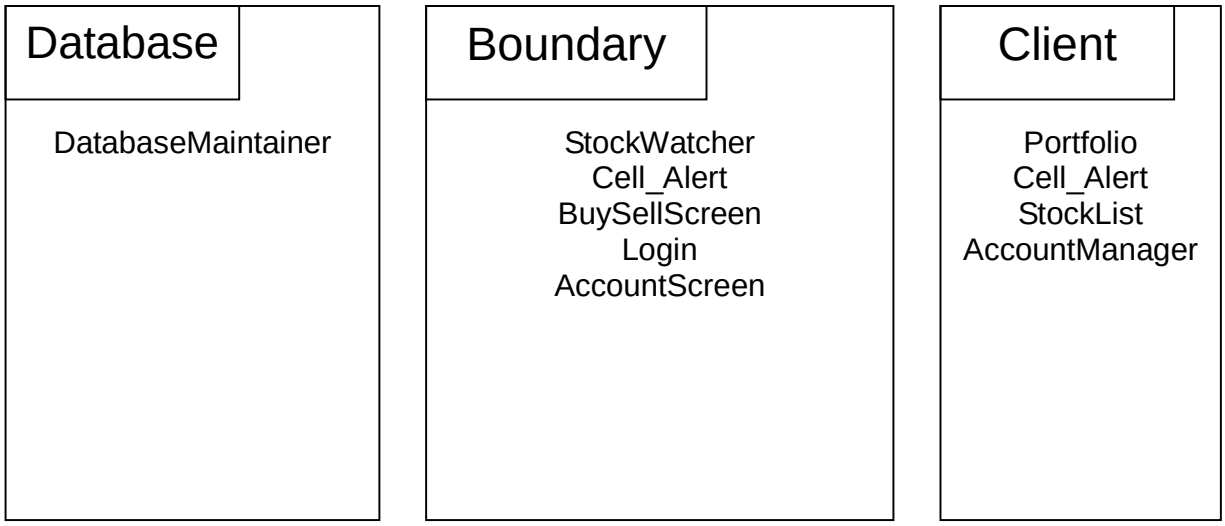
+run(): void {bodycondition: each non-empty element of quotes is updated with the current stock quote.}

3. System Architecture and System Design

a. Architecture Styles

We are creating a component-based database-centric application. This is a combination of the component-based software engineering and the database-centric architectures. The program as a 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.

b. Identifying Subsystems



c. Mapping Subsystems to Hardware

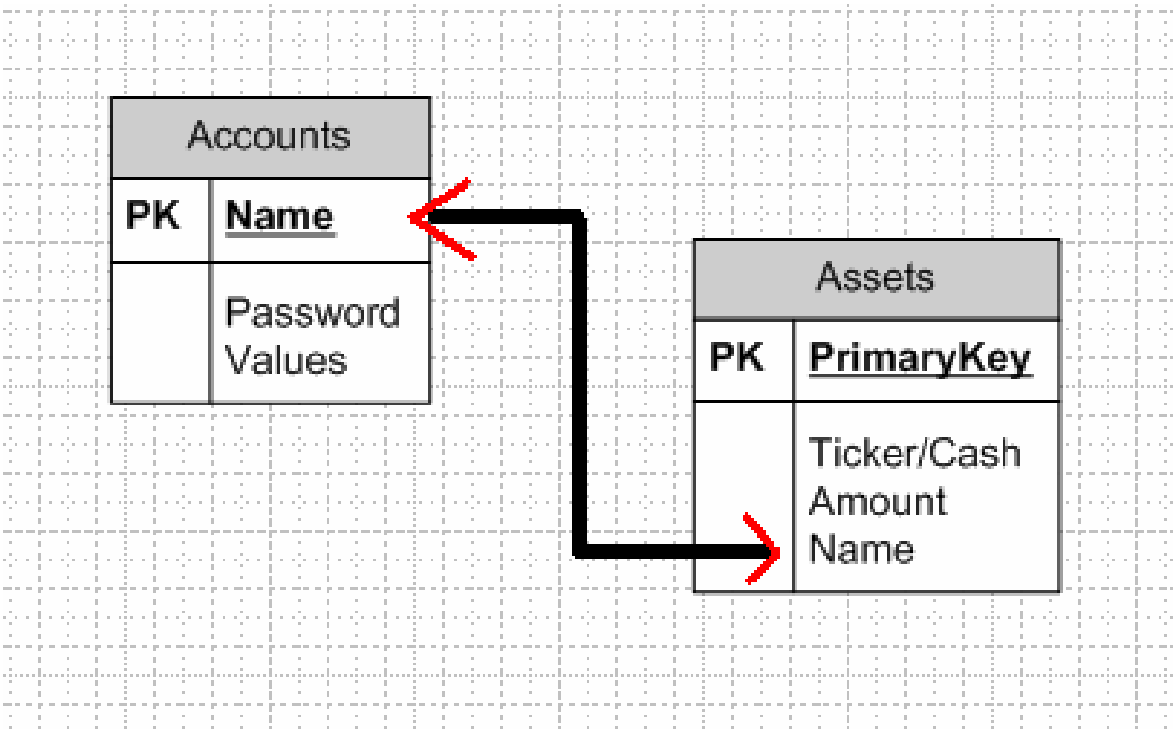
Does your system need to run on multiple computers? For example, you may have client (web browser) and server (web server) processes, running on different machines.

Yes, our system requires several computers to operate. The main database is on a server, while the rest of the program resides on the user's computer. Currently, if the program loses connection with the main database, it will fail to complete its current task. The problem limitation should be removed before the release of the final version

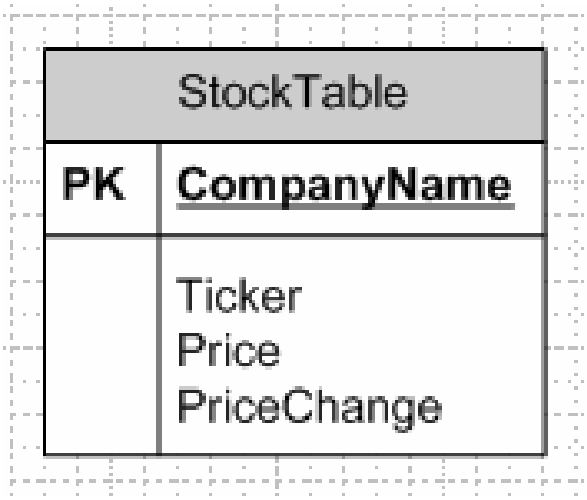
d. Persistent Data Storage

The application needs to save the data containing user's login information, their stock portfolio and a table of stock tickers and their current values.

The database 'accounts' has a primary key index used to access the 'assets' database column 'name'.



The second database the program uses is a database to keep track of the current stock market prices. This database has a thread running on the server to continually update it.



e. 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. Multiple databases need to be accessed, and 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 JDBC driver attempts to convert the underlying data to the Java type specified in the getter method and returns a suitable Java value. The JDBC specification has a table showing the allowable mappings from SQL types to Java types that can be used by the ResultSet getter methods. 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 on Yahoo! Finance are readily available.

f. 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 updates the display of the portfolio contents and takes no input from the user. Since the process-driven thread does not alter the contents of the portfolio at all, there is no synchronization required between threads. The StockWatcher window follows the same pattern. It has one thread that waits for the user to input stock tickers and one thread that periodically updates the displayed stock quotes. In addition to these threads, which run on the user's machine, there is one thread that is constantly running on the server machine which periodically updates the stock prices in the Account Database.

g. 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.

4. Algorithms and Data Structures

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.

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.

5. User Interface Design and Implementation

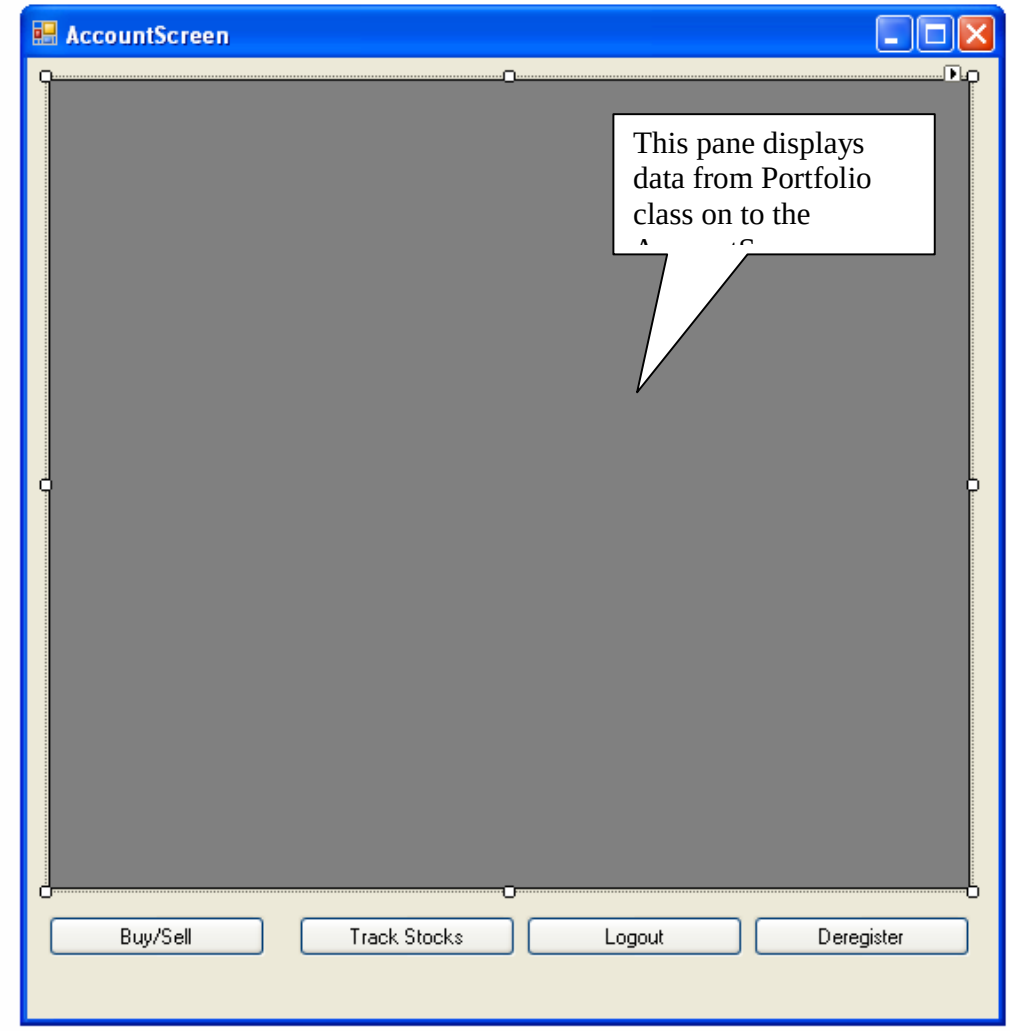
The primary focus of our user interface design is to make the user's interaction as simple and efficient as possible. In some circumstances efficiency is slightly sacrificed in order to create a more intuitive application, these cases will be described in detail later.

The interface can be broken down into five main components.
LoginScreen
AccountScreen
BuySellScreen
StockWatch
ErrBox

LoginScreen:

The login screen allows a user to login using an existing account, create a new one, or exit the program. The previous implementation did not have an exit button. This is a small but convenient modification.

AccountScreen:

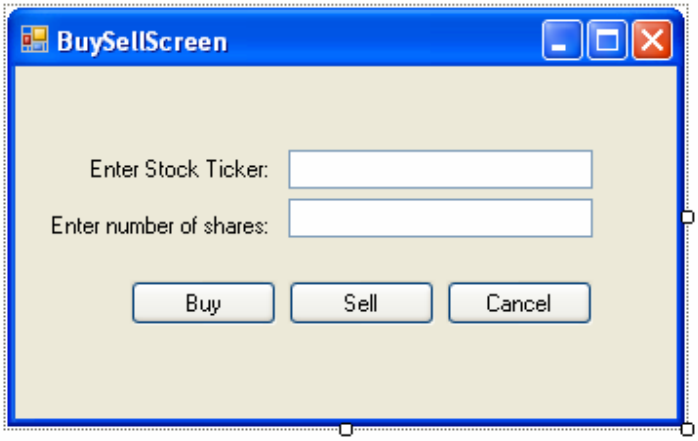


When a user successfully logs on they will first encounter the AccountScreen. The AccountScreen has four click boxes and a display pane. The boxes enable the user to Buy/Sell stocks, Track Specific Stocks, Logout or Deregister.

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.

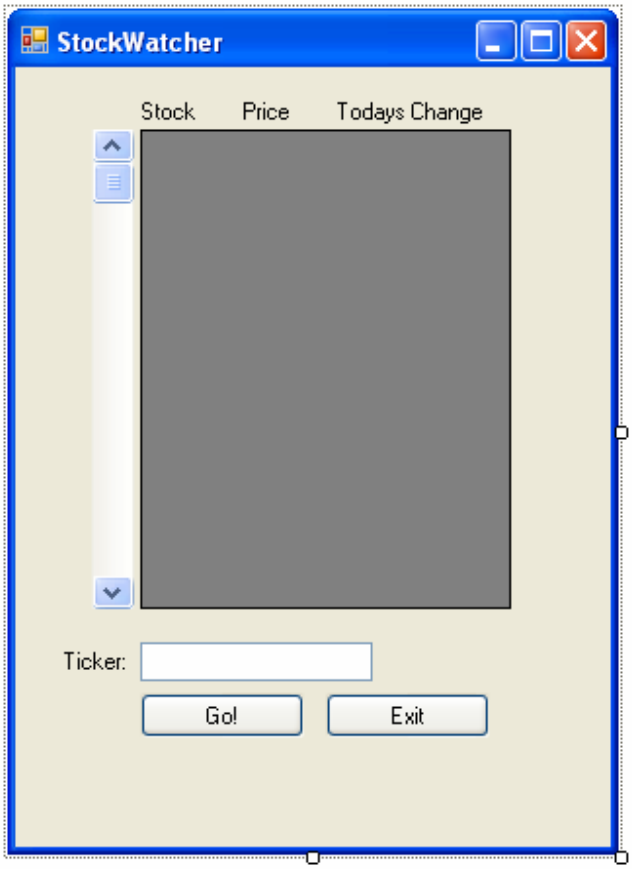
BuySellScreen:



The Buy/Sell Screen is simple and intuitive. The user is given two text boxes to enter a stock ticker and the number of share's they wish to buy. When the proper option is selected the program executed their order if they have the funds/shares to do so. The cancel button allows them to exit this screen if they do not wish to buy or sell a security.

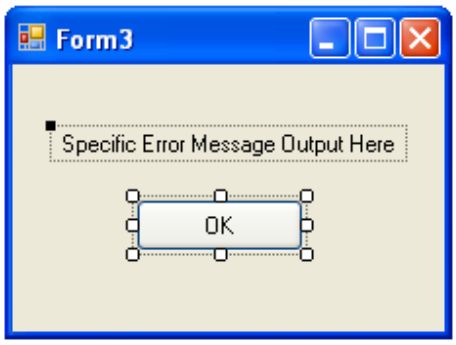
The previous user interface incorporated the buy and sell features into the AccountScreen. This was done to minimize the number of mouse clicks needed to use our program. It, however, made the program feel less intuitive and has now been changed to create a more fluid environment.

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.

The user interface is very simple and intuitive. All of the buttons needed for a desired task are on the main screen of the user's portfolio. Once a button a pressed, most options will lead to another window which is also very intuitive (Example: Pressing "Check Stock Price" will open a window which allows a user to type in the desired stock and the price will show up). If all options were on one screen, a button may be accidentally pressed due to confusion or not paying attention. For example, perhaps a user wanted to check a price of a stock, but had a field filled for buying a certain amount of shares. The user may accidentally click "Buy" instead of "Check Price," leading to unwanted results. This is why the ease-of-use is high for this interface; it is difficult for one to make a mistake.

There has been an added element to User Effort Estimation
Error
Navigation
- 1 Click "OK"

6. Progress Report and Plan of Work

a. Progress Report

Classes that have been created are:

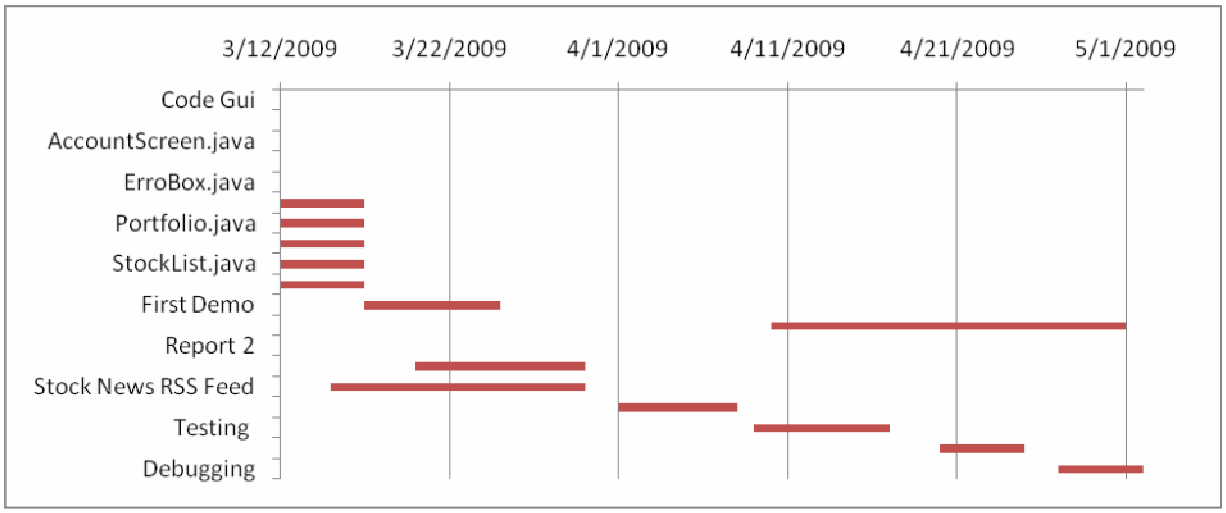
Portfolio
LoginScreen
DataBaseManager
StockLost
AccountScreen
StockWatcher
BuySellScreen
AccountManager
AccountScreenUpdater

In these classes, the following use cases are being implemented:

UC-1 Create Account
UC-2 Remove Account
UC-3 Login
UC-5 Buy Stock
UC-6 Sell Stock
UC-7 View Stock Price
UC-12 Logout

The Portfolio is functional with the Yahoo! Finance website and the DataBaseManager is also functional.

b. Plan of Work



c. Breakdown of Responsibilities

Project Management - Gantt Chart

Tasks	Start Date	Duration(days)	End Data	
Code Gui	3/2/2009	10	3/12/2009	Alex, John, Chris, and Kevin
StockWatcher.java	3/2/2009	10	3/12/2009	Chris, Kevin
AccountScreen.java	3/2/2009	10	3/12/2009	Chris, Kevin
LoginScreen.java	3/2/2009	10	3/12/2009	Alex, John
ErroBox.java	3/2/2009	10	3/12/2009	Alex, John
Code Back-End Operations	3/2/2009	15	3/17/2009	Alex, John, Chris, and Kevin
Portfolio.java	3/2/2009	15	3/17/2009	Alex, John, Chris, and Kevin
AccountManager.java	3/2/2009	15	3/17/2009	Alex
StockList.java	3/2/2009	15	3/17/2009	Alex
DatabaseMaintainer.java	3/2/2009	15	3/17/2009	Alex
First Demo	3/17/2009	8	3/25/2009	Alex, John, Chris, and Kevin
Second Demo	4/10/2009	21	5/1/2009	Alex, John, Chris, and Kevin
Report 2	3/2/2009	10	3/12/2009	Alex, John, Chris, and Kevin
Cell Phone Interface	3/20/2009	10	3/30/2009	John, Kevin
Stock News RSS Feed	3/15/2009	15	3/30/2009	Chris, Kevin
Cell Phone Testing	4/1/2009	7	4/8/2009	John
Testing	4/9/2009	8	4/17/2009	Chris, Kevin, John
Clean Up	4/20/2009	5	4/25/2009	Alex, John, Chris, and Kevin
Debugging	4/27/2009	5	5/2/2009	Alex, John, Chris, and Kevin

7. 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. ULM 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.

Wikipedi. Wikimeda Foundation. www.wikipedia.org

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