Project Deliverable 4 Report

Completed by: Alessandro Gaggioli

1 Introduction

Stock trading was the unique prerogative of a lucky few Wall Street employees and high-income individuals for as long as most Americans can remember. However, within the past 10 years, stock and security trading has become increasingly popular with everyday people. This is evidenced by the meteoric rise of apps like Robinhood, a simple, online stock trading app, that reached 10.6 million active monthly users in 2023 [1]. The intention of my project was to create a program that simulates stock trading portfolios and utilizes a database to store all of this information.

The ***VirtualStocks*** application allows a user to register a new account with a first name, last name and a password. Using their credentials, a user may access and manage their account's one or more portfolios, which revolves around adding transactions. A transaction may represent the buying or selling of a certain amount of stocks for a certain dollar amount. The stocks available to buy and sell, including their historical prices, are stored in text files which are loaded into the database by the program on boot. In addition to stock and stock price information, the database also stores user credentials and information securely (using a hash function), portfolio values, stocks held by portfolios and all transactions. The user may also delete a portfolio, but they will always be prompted to create one if they own none before accessing the rest of the program's functions. The user interacts with the program through a command-line interface.

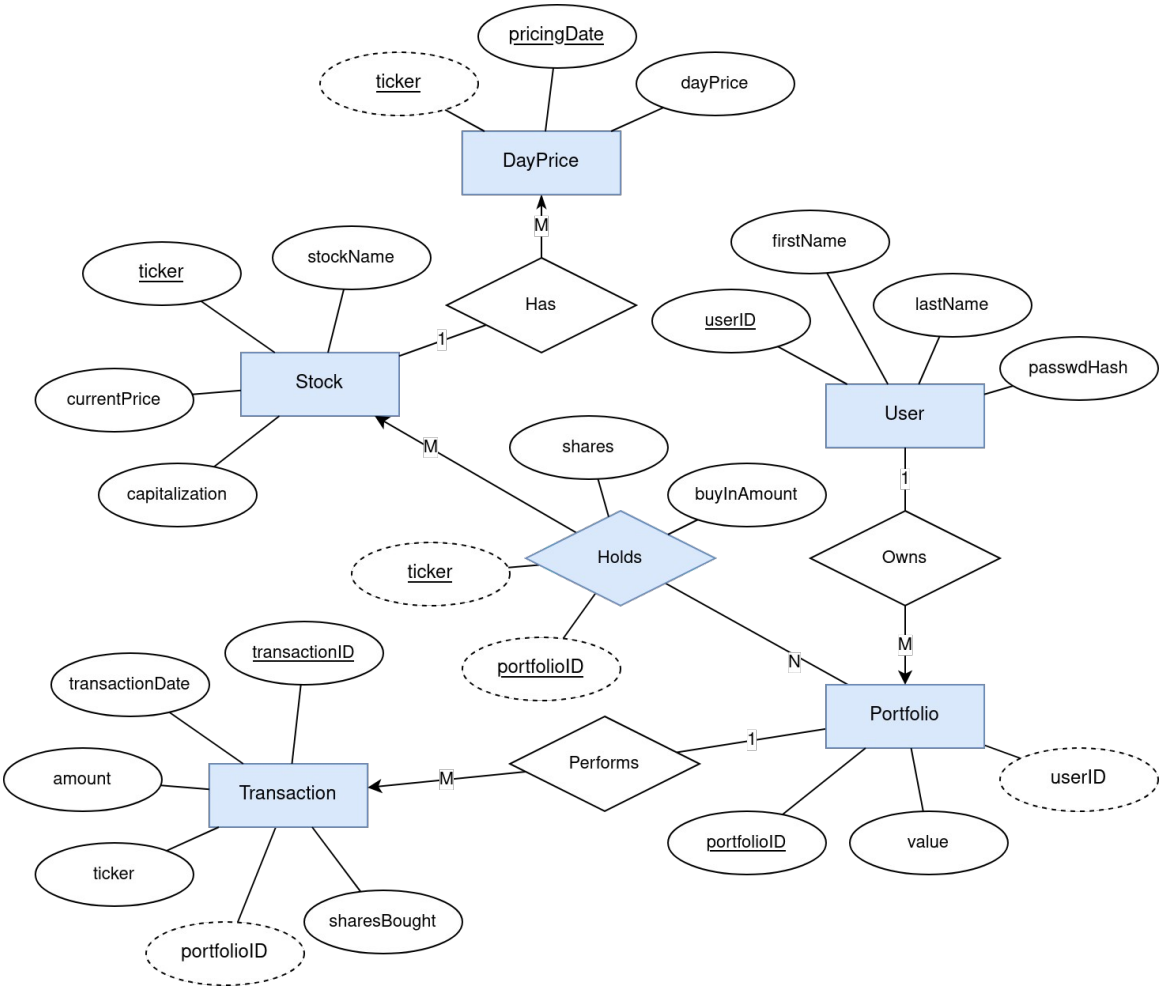
The program was built in stages: first I developed the relational database schema that would be the backbone of my program. This required employing my knowledge about relational databases I learned from CS 317: Files and Databases. I first listed all the basic features my program and determined what information needed to be stored to implement them. I then sorted the information into entities and relations (eg. A Portfolio → Owns Many → Stocks), producing an Entity/Relation or E/R diagram. I then implemented this diagram in SQL. This database schema would receive revisions as the program progressed, but did not undergo massive changes after its original development. After developing the database structure, I wrote the SQL queries required to implement my aforementioned feature list, and began implementing this in my code. In addition to dealing with the database, my code also had to include a terminal interface for the user, and appropriate data classes following Object Oriented Programming practices to store the information from the database in a cohesive way. Upon its completion, the application implemented all the features I wanted, however more could be done to make it more intuitive to use and offer more options to the user.

This report is organized into 4 main sections: first is this introduction, which covers the purpose of and methodology used to create this program. Second is the database design section, including its SQL implementation. Third is the design of the Java application itself, and fourth is the conclusion, including suggestions for future improvements. There are also two appendices, Appendix A and Appendix B, which include the full SQL schema and inserts, and the Java source code for the application respectively.

2 Database Design

This section presents the design milestones for the VirtualStock's database.

2.1 Final E/R Diagram



2.2 3NF Final Normalized Relational Schema

Here is the final database, in the form of normalized relational schemas derived from the above E/R diagram:

DayPrices(pricingDate:DATE, ticker:VARCHAR(5), dayPrice:FLOAT);
FDs: *pricingDate, ticker → dayPrice*
Key: *pricingDate, ticker*

Stocks(ticker:VARCHAR(5), stockName:VARCHAR(25), currentPrice:FLOAT, capitalization:FLOAT);
FDs: *ticker → stockname, currentPrice, capitalization*
Key: *ticker*

Holds(ticker:VARCHAR(5), portfolioID:INT, shares:FLOAT, buyInAmount:FLOAT);
FDs: *portfolioID, ticker → buyInAmount*
Key: *portfolioID, ticker*

Portfolios(portfolioID:INT, value:FLOAT, userID:VARCHAR(25));
FDs: *portfolioID → value, userID*
Key: *portfolioID*

Users(userID:VARCHAR(25), firstName:VARCHAR(25), lastName:VARCHAR(25), passwdHash:VARCHAR(65));
FDs: *userID → firstName, lastName, passwdHash*
Key: *userID*

Transactions(transactionID:INT, transactionDate:DATE, amount:FLOAT, sharesBought:FLOAT, ticker:VARCHAR(5), portfolioID:INT);
FDs: *transactionID → transactionDate, amount, sharesBought, ticker, portfolioID*
Key: *transactionID*

2.3 Constraints

In addition to PRIMARY keys, the following constraints are implemented:

Table	`DayPrices`
Constraint	`FOREIGN KEY (ticker) REFERENCES Stocks(ticker) ON DELETE CASCADE`
Justification	The dayPrices table stores information the prices of stocks. A DayPrice needs a stock to exist, hence a foreign key constraint is appropriate.

Table	`Portfolios`
Constraint	`FOREIGN KEY (userID) REFERENCES Users(userID) ON DELETE CASCADE`
Justification	The Portfolios table stores information on user portfolios. A portfolio needs a user to exist, and if a user were to be deleted, their portfolios should also be deleted. Hence a foreign key constraint is appropriate.

Table	`Holds`
Constraint	`FOREIGN KEY (portfolioID) REFERENCES Portfolios(portfolioID) ON DELETE CASCADE`
Justification	The Holds table stores the stocks owned and in what amount for each portfolio. For stocks to be held, there must be a portfolio in which they are held. Hence a foreign key here makes it impossible to add holdings for portfolioIDs which do not exist, and deletes holdings of portfolios that are deleted.

2.4 Database Schema in SQL

For the full SQL database schema and test insert statements for VirtualStocks, **please see Appendix A.**

3 Application and Database Integration

The VirtualStocks application was written in the Java programming language and makes use of a terminal interface to allow the user to interact with the program. I used the MariaDB Java Client library version 3.3.2, which was provided by Dr. Stansbury in his Java SQL example. This library implements various classes in Java, such as Connection for an SQL database connection, and the Statement type, which allows the program to craft SQL statements with less risk of injection.

<table>
<tr><td>Feature</td><td>User login screen</td></tr>
<tr><td>Description</td><td>When the program starts and connects to the database, it prompts the user to log in as follows:
<pre>Please enter a username:
> alex</pre>If the name the user enters is present in the database, the program prompts the user to enter a password:
<pre>Obtaining info for user alex...
Please enter the password for alex
> 1234</pre>If the user enters a password who's SHA256 hash corresponds to the one stored in the database, the user is logged in, and is prompted to select a portfolio.

If the user instead entered a username which doesn't exist, the program will prompt them to create a new user:
<pre>Please enter a username:
> adam
UserID adam not found.
Do you want to create this user?
> (Y/N) Y
Please enter a first name:
> Adam
Please enter a last name:
> Smith
Please enter a password:
> 1234
Please confirm password:
> 1234
Confirm user info:
UserID: adam
Name: Adam Smith
> (Y/N) Y</pre>
The program then continues as if a user had just logged in.</td></tr>
<tr><td>Where?</td><td>• LoginUser() [From VirtualStocks.java]
• findUser() [From SQL.java]
• addUser()
• deleteUser()
• replaceUser()</td></tr>
<tr><td>SQL Queries</td><td><pre>SELECT * FROM Users WHERE userID = ?;
-- If a user is created:
INSERT INTO Users VALUES (?, ?, ?, ?);
DELETE FROM Users WHERE userID = ?;</pre></td></tr>
</table>

Feature	**Portfolio Selection**
Description	Upon logging in, a user is prompted to select from one of their existing portfolios:

	Obtaining info for user alex... Index ID Value 1 1 $696649.00 Please pick a portfolio > **1** The portfolio is then loaded into the program, and the user is prompted with the main menu. If the user doesn't have a portfolio, they are prompted to create a new one: Create new portfolio for adam with ID 3? (Y/N) **Y** Creating new portfolio with ID: **3**
Where?	• promptPortfolios() [VirtualStocks.java] • createNewPortfolio() • getPortfolios() [SQL.java] • addPortfolio()
SQL Queries	SELECT * FROM Portfolios WHERE userID = ?; -- Get a unique portfolioID by adding 1 to this: SELECT MAX(portfolioID) FROM Portfolios; INSERT INTO Portfolios VALUES (?, ?, ?);

Feature	**Main Menu**
Description	The main() method has a loop which continually prompts the user with the following: Select from the following options: 1) View portfolio overview 2) View available stocks 3) View all transactions 4) Create new portfolio 5) Delete portfolio 6) New transaction 7) Change user info 8) Quit
Where?	main(), menu()
SQL Queries	N/A

Feature	**Portfolio overview**
Description	If the user selects option 1, they see an overview of all their portfolio information, including the total amount of transactions, total value of their portfolio and a breakdown of every stock held by it.
Where?	• toString() [Portfolio.java]
SQL Queries	SELECT * FROM Transactions WHERE portfolioID = ?; SELECT * FROM Holds WHERE portfolioID = ?;

Feature	**View all stocks**
Description	If the user selects option 2, they are presented with a table containing all the information for current stocks. This information is also loaded into the SQL database from files containing the latest stock information and price data.
Where?	• PrintStocks() [VirtualStocks.java] • parseDayPrices() [SQL.java] • parseStockInfo() • getStocks()

	• getPrices()

SQL Queries	`INSERT INTO Stocks VALUES (?, ?, ?, ?) ON DUPLICATE KEY UPDATE ticker = VALUES(ticker);`
	`INSERT INTO DayPrices VALUES (?, ?, ?) ON DUPLICATE KEY UPDATE ticker = VALUES(ticker);`
	`SELECT ticker, stockName, capitalization FROM Stocks;`
	`SELECT pricingDate, dayPrice FROM DayPrices WHERE ticker = ?;`

Feature	**View all transactions**

Description	If the user selects option 3, they see a full list of all transactions performed on their current portfolio.
	
	`TransactionID Date Ticker Shares Amount`
	`1 2025-04-10 AAPL 10.00 $1910.00`

Where?	• getTransactions()
SQL Queries	`SELECT * FROM Transactions WHERE portfolioID = ?;`

Feature	**Creating a new portfolio**
Description	If the user selects option 4, they are prompted to create a new portfolio. The same process is followed as with the Portfolio overview feature, where the program determines a new portfolio ID for them and simply asks the user for confirmation.
Where?	• CreateNewPortfolio() [VirtualStocks.java] • addPortfolio() [SQL.java]
SQL Queries	INSERT INTO Portfolios VALUES (?, ?, ?);

Feature	**Deleting a portfolio**
Description	If the user selects option 5, the current portfolio is deleted and the user is logged out of the program.
Where?	• DeletePortfolio() [SQL.java]
SQL Queries	`DELETE FROM Portfolios WHERE portfolioID = ?;`

Feature	**Adding a transaction**
Description	If the user selects option 6, they are prompted for all of the details of a transaction. So long as this transaction doesn't sell stocks they don't own, it is added to their transaction list, and the value of their portfolio changes.
Where?	• DeletePortfolio() [SQL.java]

SQL Queries	`INSERT INTO Transactions VALUES (?, ?, ?, ?, ?, ?);`
	`INSERT INTO Holds VALUES (?, ?, ?, ?);`
	`UPDATE Portfolios SET value = ? WHERE portfolioID = ?;`

Feature	**Changing user info**
Description	If the user selects option 7, they are prompted for a new first name, last name and password. These are then confirmed and the user's information is replaced.
Where?	• replaceUser() [SQL.java] • createNewUser() [VirtualStocks.java]
SQL Queries	`UPDATE Users SET firstName = ?, lastName = ?, passwdHash = ?` `WHERE userID = ?;`

4 Conclusion

In conclusion, most of the features I wanted to include ended up making their way into this final version of VirtualStocks. The program is able to insert into and retrieve information about stocks, transactions and portfolios from a MariaDB database and display this in an intuitive way to the user.

I haven't encountered any bugs so far. Error handling takes care of most erroneous inputs. The program is a little over-reliant on Java objects as opposed to SQL queries though, as a lot of its functionality is accomplished by manipulating these and later committing these changes to the database rather than changing the database in real-time.

There are some future improvements to consider, however:

- The date system has pretty much no use. The program is stuck using old data from a set date in the past. Maybe in the future, users would be able to set transaction dates manually.
- Online API queries could be used to obtain all the latest stock information.
- The user login screen could be more intuitive, such as informing the user after they've gotten a certain number of password attempts wrong instead of letting them continually guess.
- Passwords themselves could have limitations added to them, such as a minimum number of characters, special symbols, etc…
- Users could have the ability to view other users' portfolios if they so choose.
- Database authentication with MySQL could've been done with environment files instead of credentials being directly in the source code.

Writing this program taught me a lot about database design and SQL. The gradual improvements to my database, such as introducing foreign keys and double-checking to make sure it was 3NF compliant really helped me understand these concepts from class. If I had more time, I would've used triggers instead of implementing a lot of my functionality in Java, but overall I am happy with how much I learned about writing relational database schemas in SQL.

This project also taught me to manage my time wisely when taking on large programming tasks. Breaking the process down into smaller steps, building up functionality in self-contained bits helped a lot to keep the process motivating and not dragging it out.

5 References

[1] T. Nasr, “Robinhood Statistics (2023): AUM, Users, Revenues, and More!,” Feb. 18, 2023. https://investingintheweb.com/brokers/robinhood-statistics/

6 Appendix A: SQL Schema and INSERT Values

```
CREATE TABLE IF NOT EXISTS Stocks (
  ticker VARCHAR(5),
  stockName VARCHAR(25),
  currentPrice FLOAT,
  capitalization FLOAT,
  PRIMARY KEY (ticker)
);

CREATE TABLE IF NOT EXISTS DayPrices (
  pricingDate DATE,
  ticker VARCHAR(5),
  dayPrice FLOAT,
  PRIMARY KEY (ticker, pricingDate),
  FOREIGN KEY (ticker) REFERENCES Stocks(ticker) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS Users (
  userID VARCHAR(25),
  firstName VARCHAR(25),
  lastName VARCHAR(25),
  passwdHash VARCHAR(65),
  PRIMARY KEY (userID)
);

CREATE TABLE IF NOT EXISTS Portfolios (
  portfolioID INT,
  value FLOAT,
  userID VARCHAR(25),
  PRIMARY KEY (portfolioID),
  FOREIGN KEY (userID) REFERENCES Users(userID) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS Holds (
  ticker VARCHAR(5),
  portfolioID INT,
  shares FLOAT,
  buyInAmount FLOAT,
  PRIMARY KEY (ticker, portfolioID),
  FOREIGN KEY (ticker) REFERENCES Stocks(ticker),
  FOREIGN KEY (portfolioID) REFERENCES Portfolios(portfolioID) ON DELETE
CASCADE
);

CREATE TABLE IF NOT EXISTS Transactions (
  transactionID INT,
  transactionDate DATE,
  amount FLOAT,
  sharesBought FLOAT,
  ticker VARCHAR(5),
  portfolioID INT,
  PRIMARY KEY (transactionID),
  FOREIGN KEY (portfolioID) REFERENCES Portfolios(portfolioID) ON DELETE
CASCADE
);
```

```
-- Inserting data into the tables
-- Insert Stocks
INSERT INTO Stocks VALUES
('AAPL', 'Apple Inc.', 175.50, 2500000000000),
('GOOGL', 'Alphabet Inc.', 2800.75, 1800000000000),
('TSLA', 'Tesla Inc.', 900.25, 950000000000),
('MSFT', 'Microsoft Corp.', 320.80, 2400000000000);

-- Insert dayPrices (data for the stocks)
INSERT INTO DayPrices VALUES
('2024-03-06', 'AAPL', 175.50),
('2024-03-06', 'GOOGL', 2800.75),
('2024-03-06', 'TSLA', 900.25),
('2024-03-06', 'MSFT', 320.80);

-- Insert Users
INSERT INTO Users VALUES
('jdoe', 'John', 'Doe',
'5e884898da28047151d0e56f8dc6292773603d0d6aabbdd7eb629f99a647e4b4'),
('jsmith', 'Jane', 'Smith',
'2c9341ca4cf3d87b9a9619c48b24df558f8a0616604f760207809f21b251cb35'),
('ajohnson', 'Alice', 'Johnson',
'f5d1278e8109edd94e1e4197e04873b28dd25144d6996da7dc20bd722a65d116');

-- Insert Portfolios
INSERT INTO Portfolios VALUES
(1, 50000.00, 'jdoe'),
(2, 75000.50, 'jsmith'),
(3, 30000.75, 'ajohnson');

-- Insert Holds (portfolios holding stocks)
INSERT INTO Holds VALUES
('AAPL', 1, 50, 160.00),
('GOOGL', 1, 10, 2700.00),
('TSLA', 2, 20, 850.00),
('MSFT', 3, 15, 300.00);

-- Insert Transactions
INSERT INTO Transactions VALUES
(101, '2024-02-28', 8000.00, 50, 'AAPL', 1),
(102, '2024-02-25', 27000.00, 10, 'GOOGL', 1),
(103, '2024-02-26', 17000.00, 20, 'TSLA', 2),
(104, '2024-02-27', 4500.00, 15, 'MSFT', 3),
(105, '2024-02-27', 4500.00, 15, 'MSFT', 3);
```

7 Appendix B: Application Source Code

7.1 VirtualStocks.java:

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Date;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Scanner;

public class VirtualStocks {

	public static void main(String[] args) {
		// Connection arguments, TODO: PLACE IN ENVIRONMENT FILE!
		// TODO: make a user for this rather than root!!
		String DBurl = "jdbc:mariadb://localhost:3306/";
		String DBuser = "root";
		String DBpswd = "OrionessRed";

		// Create app object instance
		VirtualStocks app = new VirtualStocks();

		// Create the database interface
		SQL db = new SQL(DBurl, DBuser, DBpswd);

		// TODO actually get today's date,
		// Stop using this placeholder for testing
		Date today = Date.valueOf("2025-04-10");
		SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");
		String todayString = formatter.format(today);

		// Parse data from files
		db.parseStockInfo("stockInfo.csv");
		db.parseDayPrices("dayPrices.csv");

		// Obtain all stocks
		ArrayList<Stock> stocks = db.getStocks(today);

		// Create scanner
		Scanner scan = new Scanner(System.in);

		// Ask the user to log in
		User user = app.loginUser(db, scan);

		// Get the user's existing portfolios
		ArrayList<Portfolio> portfolios = db.getPortfolios(user.getUserID());
		Portfolio portfolio;

		// Create new portfolio if the user has 0
		if (portfolios.size() == 0) {
			portfolio = app.createNewPortfolio(scan, db.getUniquePortfolioID(),
user.getUserID());
			db.addPortfolio(portfolio);
		} else {
			// Prompt the user to pick one of their portfolio
			portfolio = app.promptPortfolios(portfolios, scan);
		}

		// Begin prompting the user with options
		int request;
		do {
			System.out.printf("Welcome, %s %s.\n\n", user.getFirstName(), user.getLastName());
			request = app.menu(scan);
			app.parseRequest(request, db, user, scan, todayString, stocks, portfolio);
		} while (request != 8);

		// Small message when the user quits
		System.out.println("Thanks for using VirtualStocks!");
	}

	// Prompt the user for login info
	public User loginUser(SQL db, Scanner scan) {
		System.out.print("Please enter a username:\n> ");
		String userID = scan.nextLine();
		User user = db.findUser(userID);
		// Look for the user
		if (user == null) {
			// Prompt to create a new user
			boolean answer = promptConfirm(
					"UserID " + userID + " not found.\n" + "Do you want to create this
user?\n> ", scan);
			if (!answer) {
				System.out.println("Not creating any new users. Quitting...");
				System.exit(0);
			}
			// Create a new user and add it to the database
```

```
                        user = createNewUser(scan, userID);
                        if (user == null) {
                                System.out.println("Cancelling attempt, quitting!");
                                return null;
                        } else {
                                db.addUser(user);
                                return user;
                        }

                } else {
                        // Prompt for the password if the user exists
                        passwdPrompt(scan, user);
                        return user;
                }
        }

        // Method to prompt the user through a new DB user creation process
        public User createNewUser(Scanner scan, String userID) {

                System.out.print("Please enter a first name:\n> ");
                String firstName = scan.nextLine();

                System.out.print("Please enter a last name:\n> ");
                String lastName = scan.nextLine();

                // Keep asking for a new password until they're the same
                String confirmPasswd, passwd;
                do {
                        System.out.print("Please enter a password:\n> ");
                        passwd = scan.nextLine();
                        System.out.print("Please confirm password:\n> ");
                        confirmPasswd = scan.nextLine();
                } while (!passwd.equals(confirmPasswd));

                // Hash the password
                String passwdHash = sha256sum(passwd);
                User user = new User(userID, firstName, lastName, passwdHash);

                // Confirmation from the user
                boolean result = promptConfirm("Confirm user info:\n" + user + "\n> ", scan);

                if (result) {
                        return user;
                } else {
                        return null;
                }
        }

        public void passwdPrompt(Scanner scan, User user) {
                String passwdHash;
                do {
                        System.out.print("Please enter the password for " + user.getUserID() + "\n> ");
                        String passwd = scan.nextLine();
                        passwdHash = sha256sum(passwd);
                } while (!passwdHash.equals(user.getPasswdHash()));
        }

        public void printStocks(ArrayList<Stock> stocks) {
                System.out.printf("%-4s %-8s %-15s %-15s %-15s\n", "No.", "Ticker", "Name", "Price",
"Capitalization");
                for (int i = 0; i < stocks.size(); i++) {
                        Stock stock = stocks.get(i);
                        System.out.printf("%-4s %-8s %-15s $%-15.2f $%-15.2f\n", i + 1, stock.getTicker(),
stock.getStockName(),
                                        stock.getCurrentPrice(), stock.getCapitalization());
                }
        }

        // Code to prompt user to pick a portfolio
        public Portfolio promptPortfolios(ArrayList<Portfolio> portfolios, Scanner scan) {
                System.out.printf("%-8s %-15s %-15s\n", "Index", "ID", "Value");
                for (int i = 0; i < portfolios.size(); i++) {
                        Portfolio portfolio = portfolios.get(i);
                        System.out.printf("%-8s %-15d $%-15.2f\n", i + 1, portfolio.getPortfolioID(),
portfolio.getValue());
                }
                int index = promptInt("Please pick a portfolio:\n> ", scan, 1, portfolios.size());
                return portfolios.get(index - 1);
        }

        // Code to prompt user to create new portfolio
        public Portfolio createNewPortfolio(Scanner scan, int portfolioID, String userID) {
                // Ask user to confirm the creation of this portfolio
                boolean result = promptConfirm("Create new portfolio for " + userID + " with ID " +
portfolioID + "?", scan);
                if (result) {
                        System.out.println("Creating new portfolio with ID: " + portfolioID);
                        Portfolio portfolio = new Portfolio(portfolioID, userID);
                        return portfolio;
                } else {
                        System.out.println("Not creating any new portfolios. Quitting...");
                        return null;
```

```
                }
        }

        // Prompt user to create a new transaction
        public Transaction createNewTransaction(Scanner scan, ArrayList<Stock> stocks, Portfolio portfolio,
                        String transactionDate, int transactionID) {
                printStocks(stocks);
                int index = promptInt("Please pick a stock:\n> ", scan, 1, stocks.size());
                Stock stock = stocks.get(index - 1);
                String ticker = stock.getTicker();
                double stocksOwned = portfolio.getStocksOwnedByTicker(ticker);
                System.out.println("You currently own " + stocksOwned + " of " + ticker);
                double stocksToBuy = promptDouble("Please enter an amount to buy/sell:\n>", scan, -1 *
stocksOwned, 9001);
                System.out.printf("Value of %.2f %s: $%.2f.\n", stocksToBuy, ticker,
stocksToBuy*stock.getCurrentPrice());
                double amount = promptDouble("Please enter the amount you bought/sold for:\n>", scan, -900001,
900001);
                Transaction transaction = new Transaction(transactionID, transactionDate, amount, stocksToBuy,
stock);
                return transaction;
        }

        // Method to hash a string and return the hash as a string
        // Especially useful for passwords in this app.
        public String sha256sum(String string) {
                String hashString = "";
                try {
                        MessageDigest digest;
                        digest = MessageDigest.getInstance("SHA-256");
                        byte[] hash = digest.digest(string.getBytes(StandardCharsets.UTF_8));
                        // Loop through, mask the bytes to get a string
                        for (int i = 0; i < hash.length; i++) {
                                String hex = Integer.toHexString(0xff & hash[i]);
                                if (hex.length() == 1)
                                        hashString += "0";
                                hashString += hex;
                        }
                } catch (NoSuchAlgorithmException e) {
                        System.out.println("Couldn't find hashing algorithm, quitting!");
                        System.out.println(e.getMessage());
                        System.exit(0);
                }
                return hashString;
        }

        public int menu(Scanner scan) {
                System.out.println("""
                                Select from the following options:
                                1) View portfolio overview
                                2) View available stocks
                                3) View all transactions
                                4) Create new portfolio
                                5) Delete portfolio
                                6) New transaction
                                7) Change user info
                                8) Quit
                                """);
                System.out.print("> ");
                String input = scan.nextLine();
                try {
                        int request = Integer.parseInt(input);
                        return request;
                } catch (Exception e) {
                        System.out.println("Invalid value!");
                        return -1;
                }
        }

        // Prompt the user for an integer
        public int promptInt(String message, Scanner scan, int min, int max) {
                int num = min - 1;
                while (num < min || num > max) {
                        try {
                                System.out.print(message);
                                num = Integer.parseInt(scan.nextLine());
                        } catch (Exception e) {
                                System.out.println("Invalid selection.");
                        }
                }
                return num;
        }

        // Prompt the user for a double
        public double promptDouble(String message, Scanner scan, double min, double max) {
                double num = min - 1;
                while (num < min || num > max) {
                        try {
                                System.out.print(message);
                                num = Double.parseDouble(scan.nextLine());
                        } catch (Exception e) {
                                System.out.println("Invalid selection.");
```

```
                        }
                }
                return num;
        }

        public void parseRequest(int request, SQL db, User user, Scanner scan, String date, ArrayList<Stock>
stocks,
                        Portfolio portfolio) {
                switch (request) {
                case 1:
                        System.out.println(portfolio);
                        break;
                case 2:
                        printStocks(stocks);
                        break;
                case 3:
                        System.out.println(portfolio.getTransactionsToString());
                        break;
                case 4:
                        Portfolio newPortfolio = createNewPortfolio(scan, db.getUniquePortfolioID(),
user.getUserID());
                        db.addPortfolio(newPortfolio);
                        break;
                case 5:
                        boolean confirm = promptConfirm(
                                        "Are you sure you want to delete portfolio " +
portfolio.getPortfolioID() + "?", scan);
                        if (confirm)
                                db.deletePortfolio(portfolio);
                        System.out.println("Log back in to create a new portfolio. Thank you for using
VirtualStocks!");
                                System.exit(0);
                        break;
                case 6:
                        int transactionID = db.getUniqueTransactionID();
                        Transaction newTransaction = createNewTransaction(scan, stocks, portfolio, date,
transactionID);
                        promptConfirm("Perform: " + newTransaction + "?", scan);
                        portfolio.addTransaction(newTransaction);
                        db.replacePortfolio(portfolio);
                        break;
                case 7:
                        user = createNewUser(scan, user.getUserID());
                        db.replaceUser(user);
                        break;
                case 8:
                        break;
                }
        }

        // Method to get a simple yes/no confirmation from the user
        public boolean promptConfirm(String message, Scanner scan) {
                // Loop forever until an answer is given
                String line = "";
                while (!line.toUpperCase().equals("N")) {
                        System.out.print(message + " (Y/N) ");
                        line = scan.nextLine();
                        if (line.toUpperCase().equals("Y"))
                                return true;
                        if (line.toUpperCase().equals("N"))
                                return false;
                        System.out.println("Unknown answer.");
                }
                return false;
        }
}
```

7.2 SQL.java:

```
import java.io.File;
import java.sql.*;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Map;
import java.util.Scanner;

// SQL Class
// Deals with all the interfacing with the database
// All of this is in useful methods, like findUser, getStocks etc...

public class SQL {
        Connection sql;
        ArrayList<Stock> stocks;

        public SQL(String url, String user, String pswd) {
                String databaseName = "VirtualStocks";
                stocks = new ArrayList<Stock>();

                // Try connecting to the database
                try {
                        this.sql = DriverManager.getConnection(url, user, pswd);
                } catch (SQLException e) {
                        System.out.println("Failed to getConnection!");
                        System.out.println(e.getMessage());
                        System.exit(0);
                }

                // Check if the database exists
                createDB(databaseName);
                // Go through and run the schema code
                createSchemas();
        }

        // Method to create the database
        private void createDB(String databaseName) {
                try {
                        Statement stmt;
                        stmt = sql.createStatement();
                        stmt.executeUpdate("CREATE DATABASE IF NOT EXISTS " + databaseName);
                        stmt.executeUpdate("USE " + databaseName);
                } catch (SQLException e) {
                        System.out.println("Failed to connect or create " + databaseName + ", quitting!");
                        System.out.println(e.getMessage());
                        System.exit(0);
                }
        }

        // Method to create all the initial schemas we need
        private void createSchemas() {
                try {
                        String schema;
                        Statement stmt;
                        stmt = sql.createStatement();

                        // Stocks Table
                        schema = """
                                        CREATE TABLE IF NOT EXISTS Stocks (
                                        ticker VARCHAR(5),
                                        stockName VARCHAR(25),
                                        currentPrice FLOAT,
                                        capitalization FLOAT,
                                        PRIMARY KEY (ticker)
                                        )""";
                        stmt.executeUpdate(schema);

                        // dayPrice table
                        schema = """
                                        CREATE TABLE IF NOT EXISTS DayPrices (
                                        pricingDate DATE,
                                        ticker VARCHAR(5),
                                        dayPrice FLOAT,
                                        PRIMARY KEY (ticker, pricingDate),
                                        FOREIGN KEY (ticker) REFERENCES Stocks(ticker) ON DELETE CASCADE
                                        )""";
                        stmt.executeUpdate(schema);

                        // User info table
                        schema = """
                                        CREATE TABLE IF NOT EXISTS Users (
                                        userID VARCHAR(25),
                                        firstName VARCHAR(25),
                                        lastName VARCHAR(25),
                                        passwdHash VARCHAR(65),
                                        PRIMARY KEY (userID)
                                        )""";
                        stmt.executeUpdate(schema);
```

```
                        // Portfolio table
                        schema = """
                                        CREATE TABLE IF NOT EXISTS Portfolios (
                                        portfolioID INT,
                                        value FLOAT,
                                        userID VARCHAR(25),
                                        PRIMARY KEY (portfolioID),
                                        FOREIGN KEY (userID) REFERENCES Users(userID) ON DELETE CASCADE
                                        )""";
                        stmt.executeUpdate(schema);

                        // Portfolio Holdings table
                        schema = """
                                        CREATE TABLE IF NOT EXISTS Holds (
                                        ticker VARCHAR(5),
                                        portfolioID INT,
                                        shares FLOAT,
                                        buyInAmount FLOAT,
                                        PRIMARY KEY (ticker, portfolioID),
                                        FOREIGN KEY (ticker) REFERENCES Stocks(ticker),
                                        FOREIGN KEY (portfolioID) REFERENCES Portfolios(portfolioID) ON
DELETE CASCADE
                                        )""";
                        stmt.executeUpdate(schema);

                        // Transaction table
                        schema = """
                                        CREATE TABLE IF NOT EXISTS Transactions (
                                        transactionID INT,
                                        transactionDate DATE,
                                        amount FLOAT,
                                        sharesBought FLOAT,
                                        ticker VARCHAR(5),
                                        portfolioID INT,
                                        PRIMARY KEY (transactionID),
                                        FOREIGN KEY (portfolioID) REFERENCES Portfolios(portfolioID) ON
DELETE CASCADE
                                        )""";
                        stmt.executeUpdate(schema);

                } catch (SQLException e) {
                        System.out.println("Failed to create tables, quitting!");
                        System.out.println(e.getMessage());
                        System.exit(0);
                }
        }

        // Method to parse CSV stockInfo file into the Stocks table
        public void parseStockInfo(String filename) {
                try {
                        File f = new File(filename);
                        Scanner fscanner = new Scanner(f);

                        // Skip first line, as it's only a label
                        fscanner.nextLine();

                        while (fscanner.hasNext()) {
                                String line = fscanner.nextLine();
                                String[] values = line.split(",");
                                // Move onto next line if something is wrong.
                                if (values.length != 4)
                                        continue;
                                // Parse the actual data from the csv file
                                String ticker = values[0];
                                String stockName = values[1];
                                double capitalization = Double.parseDouble(values[2]);
                                double currentPrice = Double.parseDouble(values[3]);

                                try {
                                        PreparedStatement insertStatement;
                                        insertStatement = sql.prepareStatement("INSERT INTO Stocks VALUES
(?, ?, ?, ?) ON DUPLICATE KEY UPDATE ticker = VALUES(ticker)");
                                        insertStatement.setString(1, ticker);
                                        insertStatement.setString(2, stockName);
                                        insertStatement.setFloat(3, (float) currentPrice);
                                        insertStatement.setFloat(4, (float) capitalization);
                                        insertStatement.execute();
                                } catch (Exception e) {
                                        System.out.println("FAILED to add " + line + " to SQL.");
                                }
                        }
                        fscanner.close();

                } catch (Exception e) {
                        System.out.println("Error reading file:" + e.getMessage());
                }
        }

        // Method to parse CSV dayPrices file into the DayPrices table
        public void parseDayPrices(String filename) {
                try {
                        File f = new File(filename);
```

```
                        Scanner fscanner = new Scanner(f);

                        // Skip first line, as it's only a label
                        fscanner.nextLine();

                        while (fscanner.hasNext()) {
                                String line = fscanner.nextLine();
                                String[] values = line.split(",");
                                // Move onto next line if something is wrong.
                                if (values.length != 3)
                                        continue;
                                // Parse the actual data from the csv file
                                String ticker = values[0];
                                Date pricingDate = Date.valueOf(values[1]);
                                double dayPrice = Double.parseDouble(values[2]);
                                // Insert it with an SQL statement
                                try {
                                        PreparedStatement insertStatement;
                                        insertStatement = sql.prepareStatement("INSERT INTO DayPrices VALUES
(?, ?, ?) ON DUPLICATE KEY UPDATE ticker = VALUES(ticker)");
                                        insertStatement.setDate(1, pricingDate);
                                        insertStatement.setString(2, ticker);
                                        insertStatement.setFloat(3, (float) dayPrice);
                                        insertStatement.execute();
                                } catch (Exception e) {
                                        System.out.println("FAILED to add " + line + " to SQL.");
                                }
                        }
                        // Close the Scanner.
                        fscanner.close();

                } catch (Exception e) {
                        System.out.println("Error reading file:" + e.getMessage());
                }
        }

        // Method to add a user object to the database
        public void addUser(User user) {
                User findResult = findUser(user.getUserID());
                if (findResult == null) {
                        try {
                                PreparedStatement insertStatement;
                                insertStatement = sql.prepareStatement("INSERT INTO Users VALUES
(?, ?, ?, ?)");
                                insertStatement.setString(1, user.getUserID());
                                insertStatement.setString(2, user.getFirstName());
                                insertStatement.setString(3, user.getLastName());
                                insertStatement.setString(4, user.getPasswdHash());
                                insertStatement.execute();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                } else {
                        System.out.println("Username " + user.getUserID() + " found, cannot add/replace!");
                }
        }

        public void deleteUser(User user) {
                try {
                        PreparedStatement insertStatement;
                        // Delete old entry
                        insertStatement = sql.prepareStatement("DELETE FROM Users WHERE userID = ?");
                        insertStatement.setString(1, user.getUserID());
                        insertStatement.execute();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        // Method to find a user in the database
        public User findUser(String userID) {
                User user;
                try {
                        PreparedStatement queryStatement;
                        queryStatement = sql.prepareStatement("SELECT * FROM Users WHERE userID = ?");
                        queryStatement.setString(1, userID);
                        ResultSet results = queryStatement.executeQuery();

                        if (!results.next()) {
                                return null;
                        }

                        String firstName = results.getString(2);
                        String lastName = results.getString(3);
                        String passwdHash = results.getString(4);
                        user = new User(userID, firstName, lastName, passwdHash);
                        return user;

                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return null;
```

```
        }

        // Replaces a user's old information with the new information passed in
        public void replaceUser(User user) {
                User findResult = findUser(user.getUserID());
                if (findResult == null) {
                        addUser(user);
                } else {
                        try {
                                PreparedStatement updateStatement = sql.prepareStatement("UPDATE Users SET
firstName = ?, lastName = ?, passwdHash = ? WHERE userID = ?");
                                updateStatement.setString(1, user.getFirstName());
                                updateStatement.setString(2, user.getLastName());
                                updateStatement.setString(3, user.getPasswdHash());
                                updateStatement.setString(4, user.getUserID());
                                updateStatement.execute();
                        } catch (Exception e) {
                                System.out.println("Failed to update user entry!");
                        }
                }
        }

        // Get list of stock tickers
        public ArrayList<Stock> getStocks(Date date) {
                try {
                        PreparedStatement queryStatement = sql
                                        .prepareStatement("SELECT ticker, stockName, capitalization FROM
Stocks");
                        ResultSet results = queryStatement.executeQuery();

                        while (results.next()) {
                                Stock stock = new Stock(results.getString(1), results.getString(2));
                                stock.setCapitalization((double) results.getFloat(3));
                                stock.setPrices(getPrices(stock.getTicker()), date);
                                stocks.add(stock);
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return stocks;
        }

        // Takes in a ticker and returns all the prices in a hashmap (date -> price)
        public Map<Date, Double> getPrices(String ticker) {
                Map<Date, Double> prices = new Hashtable<>();
                try {
                        PreparedStatement queryStatement = sql.prepareStatement("SELECT pricingDate, dayPrice
FROM DayPrices WHERE ticker = ?");
                        queryStatement.setString(1, ticker);
                        ResultSet results = queryStatement.executeQuery();

                        while (results.next()) {
                                prices.put(results.getDate(1), (double) results.getFloat(2));
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return prices;
        }

        public boolean addPortfolio(Portfolio portfolio) {
                Portfolio findResult = findPortfolio(portfolio.getPortfolioID());
                if (findResult == null) {
                        try {
                                PreparedStatement insertStatement;
                                insertStatement = sql.prepareStatement("INSERT INTO Portfolios VALUES
(?, ?, ?)");
                                insertStatement.setInt(1, portfolio.getPortfolioID());
                                insertStatement.setFloat(2, (float) portfolio.getValue());
                                insertStatement.setString(3, portfolio.getUserID());
                                insertStatement.execute();
                                setPortfolioTransactions(portfolio);
                                setPortfolioHolds(portfolio);
                                return true;
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                } else {
                        System.out.println("Portfolio " + portfolio.getPortfolioID() + " found, cannot
add/replace!");
                        return false;
                }
                return false;
        }

        public void deletePortfolio(Portfolio portfolio) {
                try {
                        PreparedStatement deleteStatement;
                        deleteStatement = sql.prepareStatement("DELETE FROM Portfolios WHERE portfolioID
= ?");
                        deleteStatement.setInt(1, portfolio.getPortfolioID());
                        deleteStatement.execute();
```

```
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        // Delete all existing transactions and replace them with the current ones
        public void setPortfolioTransactions(Portfolio portfolio) {
                try {
                        PreparedStatement deleteStatement;
                        deleteStatement = sql.prepareStatement("DELETE FROM Transactions WHERE portfolioID
= ?");
                        deleteStatement.setInt(1, portfolio.getPortfolioID());
                        deleteStatement.execute();
                        // Now go through and all all entries
                        for (int i = 0; i < portfolio.getNumTransactions(); i++) {
                                Transaction transaction = portfolio.getTransactionByIndex(i);
                                PreparedStatement insertStatement;
                                insertStatement = sql.prepareStatement("INSERT INTO Transactions VALUES
(?, ?, ?, ?, ?, ?)");
                                insertStatement.setInt(1, transaction.getTransactionID());
                                insertStatement.setDate(2, Date.valueOf(transaction.getTransactionDate()));
                                insertStatement.setFloat(3, (float) transaction.getAmount());
                                insertStatement.setFloat(4, (float) transaction.getSharesBought());
                                insertStatement.setString(5, transaction.getTicker());
                                insertStatement.setInt(6, portfolio.getPortfolioID());
                                insertStatement.execute();
                        }

                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        // Delete all existing held stocks and replace them with the current ones
        public void setPortfolioHolds(Portfolio portfolio) {
                try {
                        PreparedStatement deleteStatement;
                        deleteStatement = sql.prepareStatement("DELETE FROM Holds WHERE portfolioID = ?");
                        deleteStatement.setInt(1, portfolio.getPortfolioID());
                        deleteStatement.execute();
                        // Now go through and all all entries
                        for (int i = 0; i < portfolio.getNumStocks(); i++) {
                                int portfolioID = portfolio.getPortfolioID();
                                String ticker = portfolio.getTickerByIndex(i);
                                double sharesOwned = portfolio.getStocksOwnedByTicker(ticker);
                                double buyInAmount = portfolio.getInvestmentByTicker(ticker);
                                PreparedStatement insertStatement;
                                insertStatement = sql.prepareStatement("INSERT INTO Holds VALUES
(?, ?, ?, ?)");
                                insertStatement.setString(1, ticker);
                                insertStatement.setInt(2, portfolioID);
                                insertStatement.setFloat(3, (float) sharesOwned);
                                insertStatement.setFloat(4, (float) buyInAmount);
                                insertStatement.execute();
                        }

                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        public Portfolio findPortfolio(int portfolioID) {
                Portfolio portfolio;
                try {
                        PreparedStatement queryStatement = sql.prepareStatement("SELECT * FROM Portfolios
WHERE portfolioID = ?");
                        queryStatement.setInt(1, portfolioID);
                        ResultSet results = queryStatement.executeQuery();

                        if (!results.next()) {
                                return null;
                        }

                        portfolio = new Portfolio(results.getInt(1), results.getString(3));
                        return portfolio;
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return null;
        }

        // Replaces a portfolio's old information with the new information passed in
        public void replacePortfolio(Portfolio portfolio) {
                Portfolio findResult = findPortfolio(portfolio.getPortfolioID());
                if (findResult == null) {
                        addPortfolio(portfolio);
                } else {
                        try {
                                PreparedStatement updateStatement = sql.prepareStatement("UPDATE Portfolios
SET value = ? WHERE portfolioID = ?");
                                updateStatement.setFloat(1, (float) portfolio.getValue());
                                updateStatement.setFloat(2, portfolio.getPortfolioID());
```

```
                                updateStatement.execute();
                        } catch (Exception e) {
                                System.out.println("Failed to update user entry!");
                        }
                }
                setPortfolioHolds(portfolio);
                setPortfolioTransactions(portfolio);
        }

        // Get portfolios owned by a user
        public ArrayList<Portfolio> getPortfolios(String userID) {
                if (findUser(userID) == null)
                        return null;
                ArrayList<Portfolio> portfolios = new ArrayList<Portfolio>();
                try {
                        PreparedStatement queryStatement = sql.prepareStatement("SELECT * FROM Portfolios
WHERE userID = ?");
                        queryStatement.setString(1, userID);
                        ResultSet results = queryStatement.executeQuery();

                        while (results.next()) {
                                int portfolioID = results.getInt(1);
                                Portfolio portfolio = new Portfolio(portfolioID, userID);
                                // Get all transactions and add them
                                ArrayList<Transaction> transactions = getTransactions(portfolioID);
                                for (int i = 0; i < transactions.size(); i++) {
                                        portfolio.addTransaction(transactions.get(i));
                                }
                                // Add the portfolio to the array
                                portfolios.add(portfolio);
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return portfolios;
        }

        // Get a unique portfolioID
        public int getUniquePortfolioID() {
                int portfolioID = 1000;
                try {
                        PreparedStatement queryStatement = sql.prepareStatement("SELECT MAX(portfolioID) FROM
Portfolios");
                        ResultSet results = queryStatement.executeQuery();

                        if (!results.next())
                                return portfolioID;
                        portfolioID = results.getInt(1) + 1;

                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return portfolioID;
        }

        // Get a unique transactionID
        public int getUniqueTransactionID() {
                int transactionID = 2000;
                try {
                        PreparedStatement queryStatement = sql.prepareStatement("SELECT MAX(transactionID)
FROM Transactions");
                        ResultSet results = queryStatement.executeQuery();

                        while (results.next()) {
                                transactionID = results.getInt(1) + 1;
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return transactionID;
        }

        public Stock getStockFromTicker(String ticker) {
                for (int i = 0; i < stocks.size(); i++) {
                        if (stocks.get(i).getTicker().equals(ticker))
                                return stocks.get(i);
                }
                return null;
        }

        // Get all the transactions for a particular portfolio
        public ArrayList<Transaction> getTransactions(int portfolioID) {
                ArrayList<Transaction> transactions = new ArrayList<Transaction>();
                try {
                        PreparedStatement queryStatement = sql.prepareStatement("SELECT * FROM Transactions
WHERE portfolioID = ?");
                        queryStatement.setInt(1, portfolioID);
                        ResultSet results = queryStatement.executeQuery();

                        while (results.next()) {
                                String ticker = results.getString(5);
                                Stock stock = getStockFromTicker(ticker);
```

```
                                Transaction transaction = new Transaction(results.getInt(1),
results.getString(2),
                                                (double) results.getFloat(3), (double) results.getFloat(4),
stock);
                                transactions.add(transaction);
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return transactions;
        }
}
```

7.3 Portfolio.java:

```
import java.util.ArrayList;

public class Portfolio {
        private int portfolioID;
        private double value;
        private String userID;
        // Values derived from relational DB
        private ArrayList<Transaction> transactions;
        private ArrayList<Stock> holds;


        public Portfolio(int portfolioID, String userID) {
                this.portfolioID = portfolioID;
                this.userID = userID;
                this.value = 0.0;
                this.transactions = new ArrayList<Transaction>();
                this.holds = new ArrayList<Stock>();
        }

        public int getPortfolioID() {
                return portfolioID;
        }

        public double getValue() {
                calcValue();
                return value;
        }

        public void setValue(double value) {
                this.value = value;
        }

        public String getUserID() {
                return userID;
        }

        public void calcValue() {
                value = 0.0;
                for (int i = 0; i < getNumTransactions(); i++) {
                        Transaction transaction = transactions.get(i);
                        value += transaction.getSharesBought() * transaction.getStock().getCurrentPrice();
                }
        }

        public int getNumStocks() {
                return holds.size();
        }

        public String getTickerByIndex(int index) {
                return holds.get(index).getTicker();
        }

        // Return the index of a stock in the stocks array by its ticker
        public int getIndexByTicker(String ticker) {
                for (int i = 0; i < holds.size(); i++) {
                        Stock stock = holds.get(i);
                        if (stock.getTicker().equalsIgnoreCase(ticker)) return i;
                }
                return -1;
        }

        public void addTransaction(Transaction transaction) {
                // Do not add transaction if it already exists
                if (getTransactionByID(transaction.getTransactionID()) != null) return;
                transactions.add(transaction);

                // Add the stock if it is new
                Stock stock = transaction.getStock();
                String ticker = transaction.getTicker();
                if (getIndexByTicker(ticker) == -1) {
                        holds.add(stock);
                }
                // Remove the stock if its share count is now 0
                if (getStocksOwnedByTicker(ticker) == 0.0) {
                        holds.remove(stock);
                }
```

```
	}

	public int getNumTransactions() {
		return transactions.size();
	}

	public Transaction getTransactionByIndex(int index) {
		if (index >= transactions.size())
			return null;
		return transactions.get(index);
	}

	public Transaction getTransactionByID(int transactionID) {
		for (int i = 0; i < transactions.size(); i++) {
			Transaction transaction = transactions.get(i);
			if (transaction.getTransactionID() == transactionID)
				return transaction;
		}
		return null;
	}

	public ArrayList<Transaction> getTransactionsByTicker(String ticker) {
		ArrayList<Transaction> transactionsByTicker = new ArrayList<Transaction>();
		for (int i = 0; i < transactions.size(); i++) {
			Transaction transaction = transactions.get(i);
			if (transaction.getTicker().equals(ticker)) {
				transactionsByTicker.add(transaction);
			}
		}
		return transactionsByTicker;
	}

	// Turn the transaction list into a string
	public String getTransactionsToString() {
		String transactionsString = String.format("%-18s %-15s %-15s %-15s %-15s\n", "TransactionID",
"Date", "Ticker",
				"Shares", "Amount");
		for (int i = 0; i < transactions.size(); i++) {
			Transaction transaction = transactions.get(i);
			transactionsString += String.format("%-18d %-15s %-15s %-15.2f $%-15.2f\n",
transaction.getTransactionID(),
					transaction.getTransactionDate(), transaction.getTicker(),
transaction.getSharesBought(),
					transaction.getAmount());
		}
		return transactionsString;
	}

	public double getStocksOwnedByTicker(String ticker) {
		double shares = 0.0;
		for (int i = 0; i < getNumTransactions(); i++) {
			Transaction transaction = getTransactionByIndex(i);
			String ticker2 = transaction.getTicker();
			if (ticker.equals(ticker2)) {
				double theseShares = transaction.getSharesBought();
				shares += theseShares;
			}
		}
		return shares;
	}

	public double getValueByTicker(String ticker) {
		Stock stock = holds.get(getIndexByTicker(ticker));
		double sharesBought = getStocksOwnedByTicker(ticker);
		return stock.getCurrentPrice() * sharesBought;
	}

	public double getInvestmentByTicker(String ticker) {
		double investment = 0.0;
		for (int i = 0; i < getNumTransactions(); i++) {
			Transaction transaction = getTransactionByIndex(i);
			String ticker2 = transaction.getTicker();
			if (ticker.equals(ticker2)) {
				double thisAmount = transaction.getAmount();
				investment += thisAmount;
			}
		}
		return investment;
	}

	public double getProfitByTicker(String ticker) {
		return getValueByTicker(ticker) - getInvestmentByTicker(ticker);
	}

	public double getProfitPercentageByTicker(String ticker) {
		return 100 * (getProfitByTicker(ticker) / getInvestmentByTicker(ticker));
	}

	// Get a summary of the portfolio
	public String toString() {
		String portfolio = "Portfolio " + portfolioID + ", owned by " +  userID +":\n";
		portfolio += String.format("Value: $%.2f\n", value);
```

```
                portfolio += "Transactions: " + getNumTransactions() + "\n";
                portfolio += String.format("%-8s %-15s %-15s %-15s %-15s\n", "Ticker", "Shares", "Value",
"Profit/Loss", "%Change");
                for (int i = 0; i < getNumStocks(); i++) {
                        String ticker = holds.get(i).getTicker();
                        double stocksOwned = getStocksOwnedByTicker(ticker);
                        double value = getValueByTicker(ticker);
                        double profit = getProfitByTicker(ticker);
                        double profitPercentage = getProfitPercentageByTicker(ticker);
                        portfolio += String.format("%-8s %-15.2f $%-15.2f $%-15.2f %.2f%%\n", ticker,
stocksOwned, value, profit, profitPercentage);
                }
                return portfolio;
        }

}
```

7.4 Stock.java:

```
import java.sql.Date;
import java.util.Hashtable;
import java.util.Map;

public class Stock {

        private String ticker;
        private String stockName;
        private double currentPrice;
        private double capitalization;
        private Map<Date, Double> prices;

        // Once created, a stock's ticker and name cannot be changed
        public Stock(String ticker, String stockName) {
                prices = new Hashtable<>();
                this.ticker = ticker;
                this.stockName = stockName;
                this.currentPrice = 0.0;
                this.capitalization = 0.0;
        }

        public double getDatedPrice(Date date) {
                return prices.get(date);
        }

        // Sets the historical price map and the current price
        public void setPrices(Map<Date, Double> prices, Date today) {
                this.prices = prices;
                this.currentPrice = prices.get(today);
        }

        public String getTicker() {
                return ticker;
        }

        public String getStockName() {
                return stockName;
        }

        public double getCurrentPrice() {
                return currentPrice;
        }

        public void setCurrentPrice(double currentPrice) {
                this.currentPrice = currentPrice;
        }

        public double getCapitalization() {
                return capitalization;
        }

        public void setCapitalization(double capitalization) {
                this.capitalization = capitalization;
        }

        public String toString() {
                return stockName + "(" + ticker + ")\t"+ currentPrice + ",\t Cap: " + capitalization;
        }

}
```

7.5 User.java:

```
import java.util.ArrayList;

public class User {
	private String userID;
	private String firstName;
	private String lastName;
	private String passwdHash;
	private ArrayList<Portfolio> portfolios;

	public User(String userID, String firstName, String lastName, String passwdHash) {
		this.userID = userID;
		this.firstName = firstName;
		this.lastName = lastName;
		this.passwdHash = passwdHash;
	}

	public String getFirstName() {
		return firstName;
	}
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}
	public String getPasswdHash() {
		return passwdHash;
	}
	public void setPasswdHash(String passwdhash) {
		this.passwdHash = passwdhash;
	}

	public String getUserID() {
		return userID;
	}

	public String toString() {
		String userInfo;
		userInfo = "UserID:\t" + userID;
		userInfo += "\nName:\t" + firstName + " " + lastName;
		return userInfo;
	}

	public Portfolio getPortfolioByID(int portfolioID) {
		for (int i = 0; i < portfolios.size(); i++) {
			if (portfolios.get(i).getPortfolioID() == portfolioID) return portfolios.get(i);
		}
		return null;
	}

	public Portfolio getPortfolioByIndex(int index) {
		return portfolios.get(index);
	}

}
```

7.6 Transaction.java:

```
public class Transaction {
        private int transactionID;
        private String transactionDate;
        private double amount;
        private double sharesBought;
        private String ticker;
        private Stock stock;

        public Transaction(int transactionID, String transactionDate, double amount, double sharesBought,
Stock stock) {
                this.transactionID = transactionID;
                this.transactionDate = transactionDate;
                this.amount = Math.abs(amount);
                if (sharesBought < 0) this.amount *= -1.0;

                this.sharesBought = sharesBought;
                this.stock = stock;
                this.ticker = stock.getTicker();
        }
        public int getTransactionID() {
                return transactionID;
        }

        public String getTransactionDate() {
                return transactionDate;
        }

        public double getAmount() {
                return amount;
        }

        public double getSharesBought() {
                return sharesBought;
        }

        public String getTicker() {
                return ticker;
        }

        public Stock getStock() {
                return stock;
        }

        public String toString() {
                String transaction = transactionID + " (" + transactionDate + "): ";
                if (sharesBought > 0) {
                        transaction += " BUY ";
                } else {
                        transaction += " SELL ";
                }
                transaction += ticker + " FOR $" + amount;
                return transaction;
        }
}
```