**Paper Trading Simulator**

*Group 9: Nate, Raymond, and Shawn*

# Checklist for Submission:

1. Group 9: Nate, Raymond, and Shawn (Aaron dropped the class).
2. Java Files:
   1. Iterator.java (Dr. Zhang)
   2. PaperTrader.java (test driver for Nate’s class files, written by Nate)
   3. StockList.java (Nate, using Dr. Zhang’s example of Iterator design pattern)
   4. StockQuote.java (Nate)
   5. yahooConnectForStockInfo.java (The method for making a RESTful call from Java was referenced and modified from Koscen Chung’s public example. The rest was written by Nate.)
   6. Order.java (Raymond)
   7. User.java (Raymond)
   8. UserSystem.java (Raymond)
   9. DisplayPanel (Shawn)
   10. MainView(Shawn)
   11. TradeCenterPanel(Shawn)
   12. PortfolioPanel(Shawn)
   13. Portfolio (Shawn)
   14. OwnedStock (Shawn)
   15. All .form files (GUI)
3. Sequence Diagrams were created by the creators of those components.

# Design Patterns:

1. **Polymorphism** (*DisplayPanel*)
   * Display Panel is a jPanel with an additional User System attribute and methods to access and mutate that attribute. TradeCenter, and PortfolioPanel are both subclasses of DisplayPanel. When the Main renders, uses, and changes the currently displayed View, it does so by referencing a variable defined as a DisplayPanel. In reality that variable may be one of several subclasses. This is to enable uniform functionality and simple transitions across all components of the View. It also greatly encourages extensibility through allowing designers to add any new subclasses, all of which may be used without any redesign of the existing class structure.
2. **Interface** (*Iterator*)
   * The interface design pattern was used as a means of modularizing the StockList class. Its use shortened the overall length of the StockList source code and increased readability for StepThrough, the class nested in StockList.
3. **Decorator** (*DisplayPanel*)
   * MainView uses DisplayPanel as a decorator polymorphically by being decorated with either the Portfolio or the Trade Center
4. **Encapsulation/Data-Hiding** (*StockQuote*)
   * This design pattern was used to protect any StockQuote object from being used incorrectly by other programmers.
5. **Iterator** (*StockList.StepThrough*)
   * This design pattern was used due to its efficiency in moving through an ArrayList object type.
6. **Events/Listeners** (*GUI*)
   * Many listeners are used for various components in the GUI. Each button has a corresponding callback that listens for buttons presses. Additionally, there is a ComboBox in PortfolioPanel that listens for changes in its index in order to for commands from the Controller to pass to the View to change its display of the data in the Model.
7. **Strategy Pattern** (*User, UserSystem*)
   * This design pattern was used to store and manipulate data easily through the user interface, allowing variables to be assigned and used as they were needed.
8. **Composite Pattern** *(OwnedStock)*
   * OwnedStock composites StockList and a HashMap in order to easily associate quantities with each StockQuote within the StockList.
9. **MVC Passive** *(entire package)*
   * The Stock Trader uses the MVC Passive design pattern. The Model processes stock quotes and performs operations upon their data according to input from the user through the Controller. The View displays the data in the Model, though it does not actively interact with it in order to update itself. This distinguishes it from MVC Active. The buttons and objects on the Display Panels allow the user to control both the data in the model as well as how it is used, with several filtering capabilities. In addition, there is an Update button that queries for new input in the model from Yahoo, performs operations on that input with the business logic, and then displays the data to the user in the view. File input and output is also available through the Save button.

# Use Cases:

Major Use Cases:

1. Search Stock
   1. Stock Found
   2. Stock not Found
2. Place Buy Order
3. Place Sell Order
4. Change Buy/Sell Order
   1. Buy/Sell More Order
   2. Cancel Buy/Sell Order
5. View Portfolio
   1. Currently Owned
   2. Pending Buys
   3. Pending Sells
6. Update All Stock Data
   1. Updates all Stock Lists and Executes Orders if conditions are met
   2. Execute Orders
7. Save all Data

**Search Stock**

*Variation 1:* Stock Found

* User enters the symbol of the stock they wish to view.
* The system searches for the stock by the given symbol with a call to a YQL open datatable.
* The stock is found, and its data is placed into a StockQuote object.
* The system displays the stock on screen.

*Variation 2:* Stock not Found

* User performs Search Stock.
* The provided stock is not found in the YQL open datatable.
* The system informs the User that the given stock cannot be found.

**Place Buy Order**

* The user carries out Search Stock for the stock they wish to buy.
* The user selects what stock they wish to buy.
* The user enters the minimum price they would pay for the stock.
* The user enters the maximum price they would pay for the stock.
* The user presses the place order button.
* The system places the order in the “Pending Orders” StockList.

**Place Sell Order**

* The user indicates what stock they wish to sell.
* The user enters the price and quantity they would like to sell.
* The system places the sell order.

**Change Buy/Sell Order**

* The user carries out Place Buy Order or Place Sell Order.
* The user carries out View Portfolio, variation 2 or 3.

*Variation* 1: Buy/Sell More Order

* The user selects the Order in the list that they would like to change.
* The user enters the quantity amount that they would like to change the order quantity by.
* The user presses the “Buy More” or “Sell More” button.
* The system updates the quantity field in the pending order.

*Variation* 2: Cancel Buy/Sell Order

* The user selects the Order in the list that they would like to cancel.
* The user presses the Cancel Order button.
* The system cancels the order and removes it from the list of pending orders.

**View Portfolio**

* The user presses the view portfolio button.
* The system reads the user’s portfolio.
* The system displays the user’s portfolio.

*Variation* 1: Currently Owned

* The user carries out View Portfolio.
* The user selects the “currently owned” option from the drop down menu.
* The system displays the user’s currently owned stocks.

*Variation* 2: Pending Buy

* The user carries out View Portfolio.
* The user selects the “pending buy” option from the drop down menu.
* The system displays the user’s stocks that are pending purchase.

*Variation* 3: Pending Sell

* The user carries out View Portfolio.
* The user selects the “pending sale” option from the drop down menu.
* The system displays the user’s stocks that are pending sale.

**Update All Stock Data**

* The user presses the “update” button in the window.
* The system retrieves the most recent stock data it can, or the last update if the market is closed..
* The system updates all stock lists with the newly retrieved values.
* The system executes all pending orders if their conditions are met.
* The system displays the updated stock list.

**Execute Pending Orders**

* The user carries out Place Buy Order or Place Sell Order.
* The user carries out the Update All Stock Data.
* The system iterates through the updated lists (Buy Orders and Sell Orders).

*Variation #1: Conditions of an order are not met*

* The system checks the conditions of the pending order.
* The system finds that at least one order’s conditions have not been met.
* The system does nothing to the order.
* //The system executes only the orders whose conditions have been met.
* //The system updates the “Currently Owned” list in the Portfolio.

*Variation #2: Conditions of an order are met*

* The system checks the conditions of the pending order.
  + For “Buy Order,” the system adds the stock to the user’s portfolio and adjusts the user’s virtual wallet according to the cost of executing the Buy Order.
  + For “Sell Order,” the system removes the stock from the user’s portfolio and adjusts the user’s virtual wallet according to the cost of executing the Sell Order.
* The system updates the “Currently Owned” list in the Portfolio.

**Save All Data**

* The user carries out any of the other use cases.
* The user presses the Save button.
* The system serializes all user data.
* The system saves serialized data into “StockTrader.ser”
* Upon opening the program the next time, the new data will be input into the system.

# Self-Evaluations:

## Nate McCain:

**Self-Evaluation**

1. *Specifications: 5*
   1. All of the functions I was responsible for meet the specifications we laid out.
2. *Readability: 5*
   1. I organized the code, and made sure that all methods are organized by the number of other methods they need to access (the top methods don’t call other methods, the bottom methods do call other methods). Also, method names were chosen to reflect what they do.
3. *Documentation: 5*
   1. Every class and method has a detailed description of what they do, and comments inside of the functions help to explain what happens in more detail.
4. *Software Architecture: 5*
   1. Information hiding is practiced in all methods, and all classes are reduced as much as possible.

**Group-Evaluation**

1. *Division of Labor: 5*
   1. Everything was split more evenly when we had Aaron, but we were able to pivot and take on a few of the parts he was assigned. We could have done so much more if we had a 4th person in our group.
2. *Contributions: 5*
   1. Everyone made useful contributions to the project, especially since we were reduced to a 3-person group right before Phase 1 was due.
3. *Project Milestones: 5*
   1. We were almost always right on schedule (with the exception of my responsibility of working with YQL, which was due to Yahoo! Removing all documentation on how to interact with their system using OAuth 1.0). I think we accomplished a ton considering that we were short-staffed.
4. *Meetings and Communication: 5*
   1. We had 100% attendance at every meeting and we regularly communicated with each other outside of class and meetings.

**Notes:**

I really wish I could have implemented the RESTful call to YQL with Open Authorization 1.0 so that we wouldn’t be limited to 2,000 calls to the service per hour. YQL is the only Yahoo! Service that is still running on OAuth 1.0 (everything/everybody else upgraded to OAuth 2.0 about 8 years ago). On the few pages that do describe how to interact with OAuth 1.0 for YQL, the libraries referenced and outside reference materials are mostly deleted.

## Raymond Huffman:

CS 321 Group 9 project

Raymond Huffman program self-evaluation

* Specifications, 4; Meets most specifications, the program runs and performs operations correctly, though the group didn’t have enough time to fulfill all proposed ideas with only three members
* Readability, 5; The code is organized and well commented, with code separated and easy to read throughout the program.
* Documentation, 5; The documentation is clearly written, every function has a comment block introducing it as well as comments within the function itself. Each class has a comment block explaining its purpose.
* Software Architecture, 5; The classes each have their own data, all within themselves, and each class is constructed to work with and call on other classes for many functions.

Group self-evaluation

* Division of Labor, 5; Each member was responsible for one part of the program: GUI, user handling, and stock data. This allowed each part to be completed on time.
* Contributions, 5; Each member made useful contributions, everyone completed their assigned part and all were present throughout the work done on the project. Everyone made the meetings and communicated regularly.
* Project Milestones, 5; Project milestones were completed on time, and all were clearly defined before hand through group meetings.
* Meetings and Communication, 5; Meetings were held at least once a week throughout the time spent working on the project, everyone was present throughout the meeting times as well.

## Shawn Edwards:

Project Quality:

* Specifications: (5) Every specification and design element (after adjusting for removing the user management) was successfully met. Intended use cases were all satisfied. The program runs, and performs validation and error handling to input.
* Readability: (5) Code is readable and commented. Purpose of variables is provided, functions are described, and parameters are explained.
* Documentation (5): Comment blocks are widely available, Javadocs are provided, and explanatory diagrams and text are included in order to explain design choices and usability. Use cases well describe a users operation of the program.
* Software Architecture: (5) The system and class designs all incorporate information hiding, exhibit programming-by-contract usability. Class inheritance and interfaces are well structure in a way that ensures each class has a clear purpose. Overall, the program is very extensible, and could easily have additional panels, user management, or paper trading simulation added without restructuring any off the architecture.

Group Performance Quality Self-Evaluation:

* Division of Labor: (5) The task was neatly divided into the capabilities and preferences of all three team members, where Raymond wrote the User and Order components (the data structures for our project), Nate wrote and designed the interface with Yahoo and the stock market (which he has participated in), and I designed the GUI and integrated the various classes (which I have done before).
* Contributions: (5) Each team member made useful, necessary contributions. As the GUI designer and program integrator, I highly valued having ready access to the data structures and YQL interfaces.
* Project Milestone: (5) Project goals were all successfully met. Goals for the project were well-defined, though the User management system had to be dropped in order to account for the lower number of team members.
* Meetings and Communication: (5) Meetings were held regularly and often, with perfect attendance.

# Meeting Minutes:

**Group Meeting Minutes**

**02/03/2017, 100% Attendance**

* Brainstormed on ideas for the project.
* Discussed the overall design of the project.
* Began the Functional Specifications document.
* Set up future times of availability for team meetings.

Goal for next meeting: Complete the Functional Specifications document.

**02/06/2017, 100% Attendance**

* Completed the Functional Specifications document.
* Read and discussed the group project requirements.
* Discussed the project specifications.
* Created a Google Drive shared folder to hold project documents.
* Created a meeting minutes document.

Goal for next meeting: Begin to form ideas about what classes are needed.

**02/13/2017, 100% Attendance**

* Signed up for free Moqup account (for GUI design).
* Conceptualized the different classes needed for the project.
* Discussed how the business logic would work with the classes we came up with.

Goal for next meeting: Start fleshing out some of the diagrams needed to describe class behaviors.

**02/15/2017, 100% Attendance**

* Set-up CRC cards in a powerpoint drive.
* Established a Github location.
* Established a Lucid Chart account to share diagrams.
* Consolidated a few classes and expanded a few classes in order to simplify the business logic.

Goal for next meeting: Prepare most of the documents required for the Phase 1 presentation.

**02/24/2017, 100% Attendance**

* Goodbye Aaron.
* Expanded UML Class diagram.
* Reviewed basic methods and attributes for presentation.
* Decided upon what states should be used.
* Created a Use Case document, and began to populate it with major use cases.
* Gave class writing responsibilities out (Nate handles Stock Fetch/Organization, Raymond handles User System, and Shawn handles the GUI).
* Reviewed what diagrams are needed before this coming tuesday.

Goal for next meeting: Finalize all documents required for the Phase 1 presentation.

**03/03/2017, 100% Attendance**

* Finished writing all major use cases.
* Made the state diagram.
* Began to make the major sequence diagrams.

Goal for next meeting: Have presentation prepared for Phase 1.

**03/08/2017, 100% Attendance**

* Created PowerPoint for for Phase 1 presentation.
* Discussed data update process for stock trader (reviewed each person’s progress).

Goal for next meeting: Begin writing out methods for classes.

**03/20/2017, 100% Attendance**

* Discussed method by which to collect stock data from Yahoo Finance (using YQL).
* Discussed workings of Model in regards to stock data.
* Edited the class diagram to reflect changes in class structure.

Goal for next meeting: Come up with GUI designs and how to implement them.

**03/31/2017, 100% Attendance**

* Demoed GUI designs.
* Discussed plans for Model, View, Controller interaction.
* Discussed interaction between user class and Stock information.
* Joined the Yahoo Developer Network to get OAuth key and secret.

Goal for next meeting: Keep working on classes and learn more about OAuth.

**04/07/2017, 100% Attendance**

* Reviewed progress in writing classes.
* Discussed how Open Authorization works in a RESTful call.
* Found places in classes to implement design patterns.

Goal for next meeting: Begin to finalize the project.

**04/10/2017, 100% Attendance**

* Discussed finalization of project.
* Refined project scope (removed optional components of design).
* Determined Input/output method.
* Discussed data format and display in View.
* Discussed logout function (this will be implemented if we have time).

Goal for next meeting: Find a way to implement OAuth in the stock fetch method (YDN has deleted all of their pages describing how to implement OAuth 1.0 with their utilities).

**04/13/17, 100% Attendance**

* Discussed method of getting data from Yahoo (don’t have to use OAuth, but are limited to 2,000 calls/hour).
* Understood why Yahoo! has been overtaken by Google; Yahoo! has removed almost all of their documentation on how they implement Open Authorization 1.0 (which is still being used by YQL).
* Reworked class diagrams.
* Planned for finishing the project.
* Refined data format for stock quotes.
* Designed an Order system for pending Buys/Sales.
* Updated GUI with relevant changes.
* Most class methods (for everyone), while still messy, are working.

Goal for next meeting: Finalize classes and integrate code.

**04/14/2017, 100% Attendance**

* Reworked the User class designs.
* Shared code (fixed everyone’s code so that it works together).
* Refined and expanded upon class diagrams.
* Improved GUI design.
* Discussed input/output functionality.

Goal for next meeting: Clean up code style (and functionality), and try to implement as much as we can (Aaron made that a bit more difficult).

**04/19/2017, 100% Attendance**

* Combined all files into a project.
* Finished making all documentation (javadoc and diagrams).
* Practiced the demo.
* Cleaned the code and made it Javadoc ready.

**04/21/2017, 100% Attendance**

* Integrated and tested all code.
* Compiled report and materials to submit