Financial Portfolio Tracking System - Release 2

Design Documentation
Prepared by <s262-01p1e>:

- Daniel Dang <dxd2791@rit.edu>
- Jeffrey Kotowicz <jtk7412@rit.edu>
- Aaron Sloan <ars2201@rit.edu>
- Brett Kosciolek <bpk9701@rit.edu>
- Sadaf Chowdhury <sxc7870@rit.edu>
- Zachary Stauffer <zps6796@rit.edu>

Table of Contents

Summary

- The portfolio class has a collection of holdings which consists of multiple cash accounts and equities. This ties back to requirements of having multiple holdings as well as the attributes of those cash accounts and the equities.
- Without a database, we decided to store our data via object serialization so that the user could logout and re-login to and still have access to their portfolio information.
- For Importing and Exporting, we collaborated with Team 6 to come up with a standard format based on one put forward in the team discussions. It differs slightly in that the type of object is put as the first statement to make importing entire portfolios an easier task
- In our overall design we have high cohesion by making sure that the methods associated with a class is of that class and not does not depend on calls from other classes
- Password encryption is done via a Java library which handles encryption. It encrypts the username and password into a byte variable which then encodes it into a file.
- User has the option to undo or redo any recent portfolio modification by the click of a button.
- Due to time constraints, the updating of equity and market index prices had to be modified slightly. Instead of being update on a time interval the values are update when the portfolio is opened, a portfolio is imported, and there is also a update button located at the top of the GUI to manually update the price values at will.
- One of the first things we did going into R2 is break up the god classes of portfolio and FPTS to create better cohesion in our system. We implemented a visitor pattern with our implementation of watchlist and a combination of command and iterator for our UndoAction.
    - Changing the GUI to where it doesn't heavily rely on buttons would make it look a lot cleaner but is something we decided was too risky for the R2 time frame. We determined that an estimate for the LOC required to change this is not possible as the refactoring required would be so heavy as to make any estimates have a high level of inaccuracy.
    - By not implementing the observer pattern correctly in our initial design changing things into interfaces that work with the observers and subjects would have been too time consuming after receiving the feedback from R1. We would have to change how all of our classes receive their updates. I don't know estimated LOC that it would take to fully implement the observer pattern properly.
    - Also now having come to a full understanding of equities, the composite pattern could have been used for equities and market averages which would have been useful in different areas, but making the change now for R2 would be both too risky and too much work as our

system relies on our current implementation. Thus Portfolio, Equity, EquitiesView, and all classes related to history would require heavy refactoring in order to fully implement composite which would require way more work than we have time for R2.

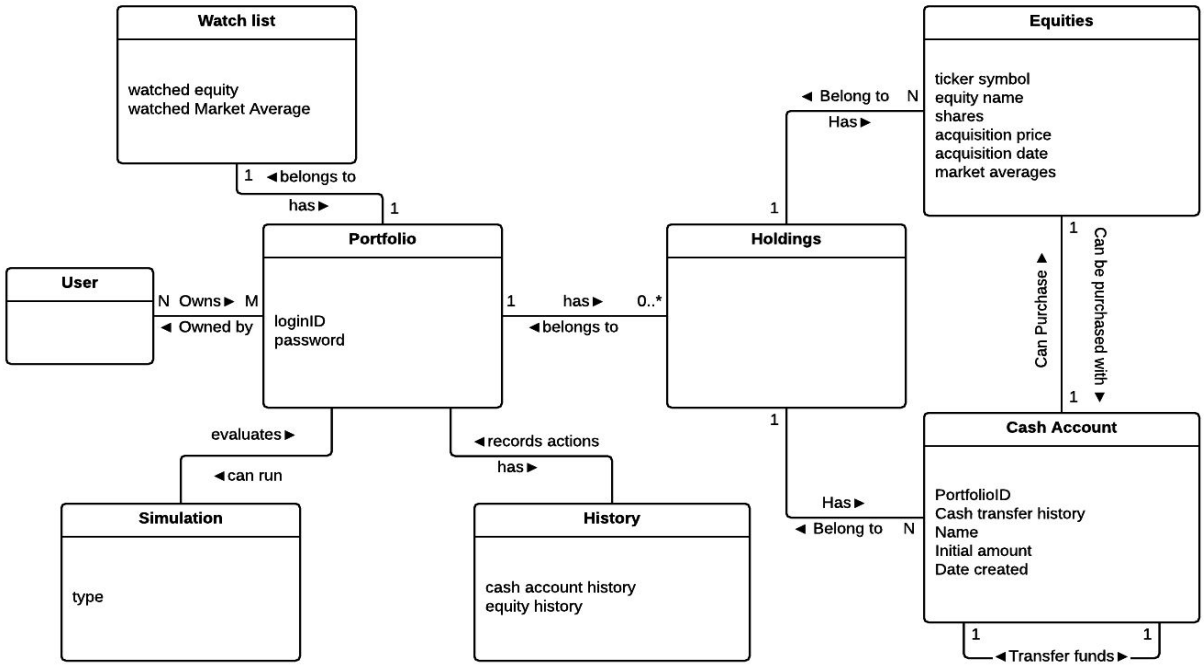
Requirements

R1	Release 1 Requirements
1	The system shall allow users to create and track multiple financial portfolios across multiple systems.
2	The user is allowed to register a new portfolio which will be tied to a login ID and a password.
3	The system shall maintain holdings of Equities and Cash Accounts in the user's portfolio. The user is allowed add equities either as from outside the system or using a cash account inside the system, remove equities and sell a specific number of shares of an equity, and import and export equities. The user is also allowed to add cash accounts, transfer funds between accounts, remove cash accounts, and import and export cash accounts.
4	The system shall maintain the following information about equities that a user can add to their portfolio: ticker symbol, equity name, and ~~index or sector~~ with which it is associated.
5	The user shall be able to search for equities based on their information.
6	The user shall be able to view holdings of cash accounts and equities. The system shall maintain a history of all portfolio transactions and also allow the user to view them.
7	The system shall allow users to simulate portfolio performance based on Bull Market, Bear Market, and No Growth algorithms.
8	The system shall allow users to save, export, and import portfolios.
9	Upon the user choosing to logout from the system, the system shall give the user a choice between saving the changes to their portfolio, not saving, or canceling the logout.
10	The user shall be able to delete a portfolio from the command line by typing in the command line in the format "-delete user-login-id", where "user-login-id" indicates the user to delete from the system.

R2	Release 2 Requirements
1	The system shall allow users to create and track multiple financial portfolios across multiple systems.
2	The user is allowed to register a new portfolio which will be tied to a login ID and a password.
3	The system shall maintain holdings of Equities and Cash Accounts in the user's portfolio. The user is allowed add equities either as from outside the system or using a cash account inside the system, remove equities and sell a specific number of shares of an equity, and import and export equities. The user is also allowed to add cash accounts, transfer funds between accounts, remove cash accounts, and import and export cash accounts. The user shall also

	be able to undo and redo previously done actions.
4	The system shall maintain the following information about equities that a user can add to their portfolio: ticker symbol, equity name, and market average with which it is associated.
5	The user shall be able to search for equities based on their information.
6	The user shall be able to view holdings of cash accounts and equities. The system shall maintain a history of all portfolio transactions and also allow the user to view them. When the system is first started and at an interval specified by the user, the system shall update the prices of equities per share given by the Yahoo financial web service. The user shall be able to set up a watchlist of equities and market averages that may or may not be owned by the user which will be saved and stored by the system and, based on a trigger system of low or high prices per share, will notify the user that a certain trigger condition has been met or at one point was met. The status of if a trigger condition has at one point been met is able to be reset by the user.
7	The system shall allow users to simulate portfolio performance based on Bull Market, Bear Market, and No Growth algorithms.
8	The system shall allow users to save, export, and import portfolios.
9	Upon the user choosing to logout from the system, the system shall give the user a choice between saving the changes to their portfolio, not saving, or canceling the logout.
10	The user shall be able to delete a portfolio from the command line by typing in the command line in the format “-delete user-login-id”, where “user-login-id” indicates the user to delete from the system.

Domain Model



In this financial portfolio tracking system, a portfolio will have a loginID and a password which will allow a user to access their portfolio(s).

There are zero to many cash account and holdings within a user's portfolio. A cash account has attributes of the portfolio that it belongs to, the name of the cash account, the amount of money in the account and the date that the account was created. Also money in cash accounts are also able to be transferred to and from other cash accounts or used to purchase equities in the system.

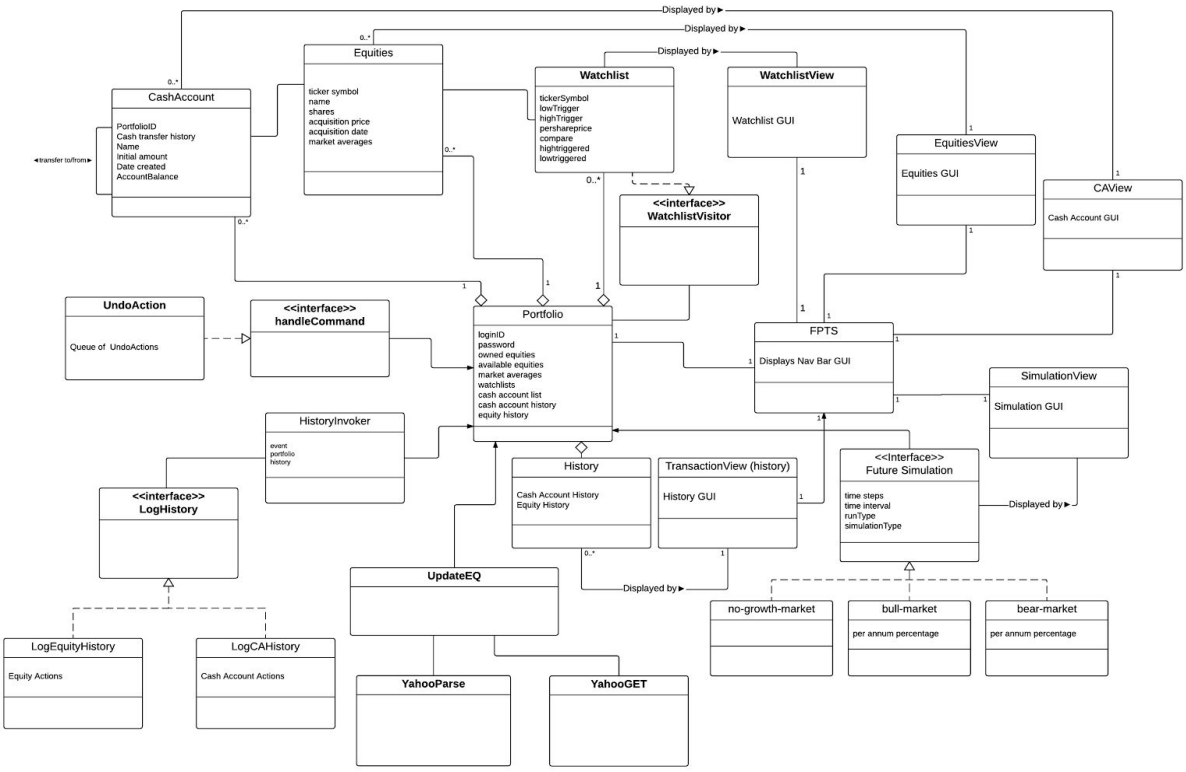
Equities is a collection of equities owned in the portfolio. Each equity as a ticker symbol, name, share, acquisition price and date. Equities reference to the Market Average that they belong. Equities and shares of equities can be purchased with or without a cash account.

Equities and market averages can be set on a watchlist whether owned or not, and the user will receive a notification based on whether a high or low trigger point of price has been met or has been met in the past.

A history of all transactions done in the system involving cash accounts and equities is recorded and able to be viewed by the user.

The system will be able to do a simulation of the portfolio's projected value based on the portfolio's current value and the bull-market, bear-market, or no growth algorithm.

System Architecture



For the overall class design of the financial portfolio tracking system it is split up into a model, view, control system. Each of the separate model components of the system have a respective view that displays the current information associated with them within the portfolio.

FPTS is the main GUI class that contains the main border pane that will be used by the other classes in the view part. FPTS also handles the login and registration functions.

Portfolio is the main model class containing a list of all of the user's equities, cash accounts, market averages, watchlists, and the transaction histories of these objects. It is linked to a loginID and password. The Portfolio class also handles the importing and exporting the portfolio, and saving the portfolio.

CashAccount represents an account with a balance of money, a creation date, and a name. It also handles giving the user the option to add or delete cash accounts, deposit funds, withdraw funds, transfer funds between accounts, and export and import cash accounts. Its GUI display is handled by CAView.

Equity represents equities having a ticker symbol, name, shares, acquisition price, acquisition date, and market averages. It also handles allowing the user to add or delete equities, export and import equities, add and sell shares to/from equities, and add shares to market averages. EquitiesView handles displaying the GUI for equities.

Watchlist is a list of equities not necessarily owned by the user that the user wishes to track. A watchlist item can have a high and low trigger point at which if the price of the equity if above or below the point or has been in the past, the user is notified. WatchlistView handles displaying the GUI for watchlist, adding high and low trigger points, adding and removing items to and from the watchlist, and resetting if the trigger points have been met in the past.

History represents the transaction history of cash accounts and equities. TransactionView handles displaying the GUI for the transaction history. It also handles allowing the user to import/export the transaction history of equities and cash accounts.

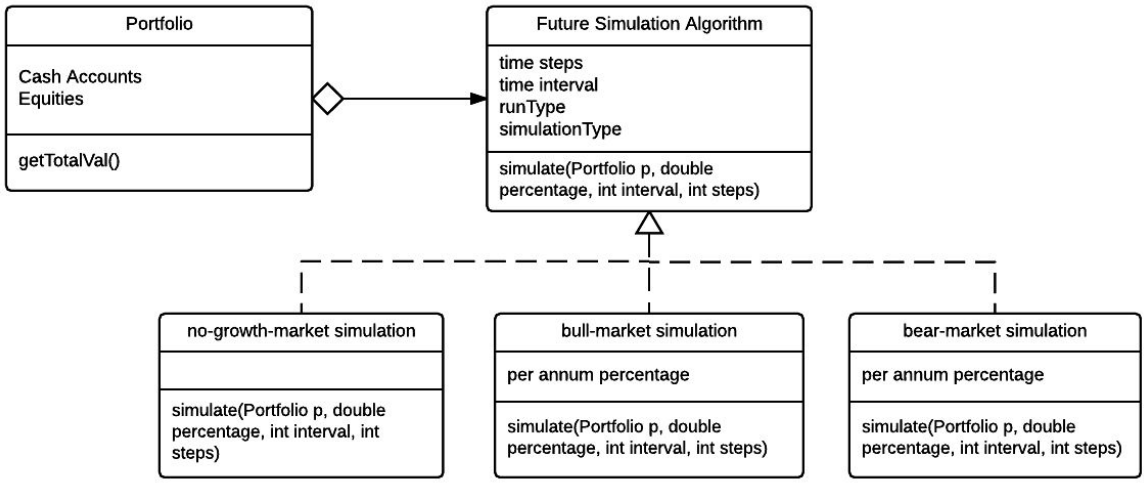
SimulationView handles displaying a simulation of the projected future value of the portfolio based on a chosen algorithm or strategy. FutureSimulation sets up an interface for the strategies which is implemented by NoGrowth, BullMarket, and BearMarket which represent the individual algorithms given by the requirements.

UndoAction represents the class that handles any last five modifications made to the Portfolio. The Undo/Redo buttons found on the GUI page utilizes the UndoAction class which undoes or redoes the action. This changes the CashAccount or Equity class, depending on what the last modification was (e.g. Undoing adding a cash account removes it from the CashAccount Arraylist). Updates the GUI as well to represent undoing the action and updates the history log accordingly.

YahooGET handles getting raw information from Yahoo and putting the information into XML format. YahooParse takes in the XML and grabs the prices of each equity which is then used by UpdateEQ to set the correct sale price of equities in the system.

Subsystems

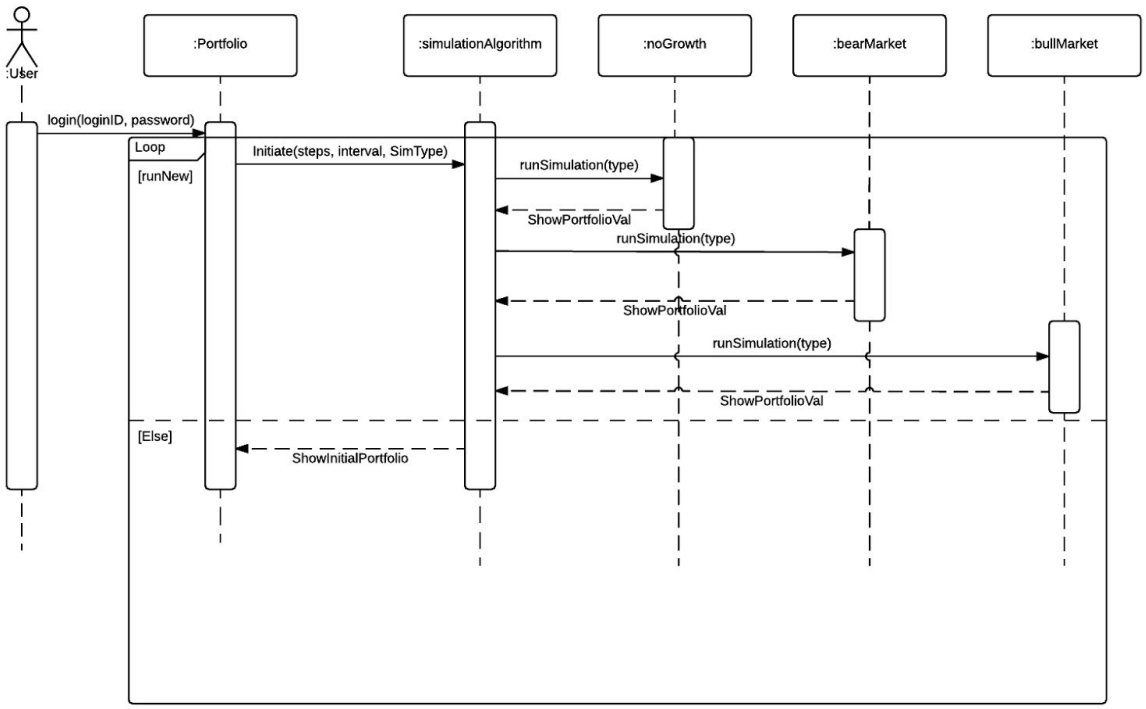
Subsystem #1 – Simulation Algorithms



Name: Future Simulations	GoF Design Pattern Name: Strategy	
Pattern Name in terms of system context: Market Simulation Prediction		
Participants: Portfolio, FutureSimulation, NoGrowth, BullMarket, BearMarket		
Class	**Role in pattern**	**Participant's contribution in the context of the application**
Portfolio	Context	The future status of the portfolio is configured by the concrete strategies
FutureSimulation	Strategy	Defines the interface that all of the concrete strategies inherit from
NoGrowth	ConcreteStrategy	A choice of simulation in which equity prices do not change
BullMarket	ConcreteStrategy	A choice of simulation in which a per annum percentage at which equities increase is specified
BearMarket	ConcreteStrategy	A choice of simulation in which a per annum percentage at which equities decrease is specified
Deviations from the standard pattern: None		

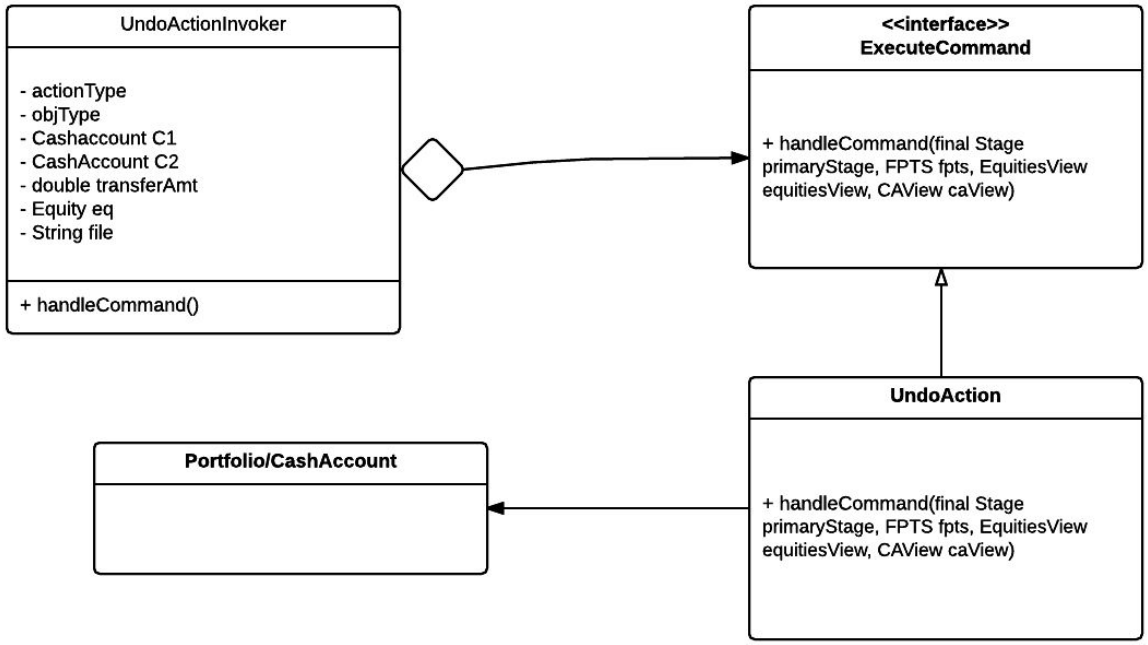
Requirements being covered:

- Able to show future predictions of a given portfolio with the simulations outlined
- All equity prices are greater than or equal to $0.00 in each simulation.
- The user will choose an algorithm for simulation. Based on what algorithm is chosen, a specific concrete implementation of the market simulation algorithm will be chosen and run.



A user that is logged into their portfolio may run one of three algorithms at a time that will show the future predictions of the market depending on which algorithm has been run. The user has an option to run a new simulation after the initial simulation is finished executing. After the desired number of simulations are ran the user reverts back to the initial portfolio.

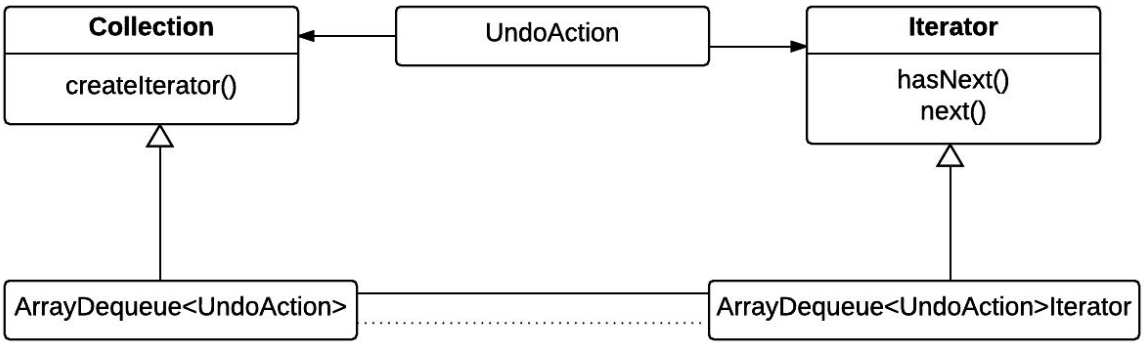
Subsystem #2 – UndoAction (command)



Name: UndoAction	GoF Design Pattern Name: Command	
Pattern Name in terms of system context: UndoAction		
Participants: FPTS,UndoAction, ExecuteCommand, Portfolio, CashAccount		
Class	**Role in pattern**	**Participant's contribution in the context of the application**
ExecutiveCommand	Command	Defines the interface that all the concrete commands inherit from.
handleCommand()	ConcCommand	Given instruction, the method undoes/redoes any portfolio modification and goes to the last five modifications.
UndoAction	Invoker	Gets a request from the GUI and encapsulates that into UndoAction object.
Portfolio CashAccount	Receiver	Updates the CashAccounts and Equities accordingly as well as the GUI.
Deviations from the standard pattern: This Command Pattern has Redo in the UndoAction class instead of its own Redo class.		
Requirements being covered: ● The user shall be able to undo and redo any of the last 5 portfolio modifications.The command pattern ensures that no matter what modification, the command is encapsulated into an		

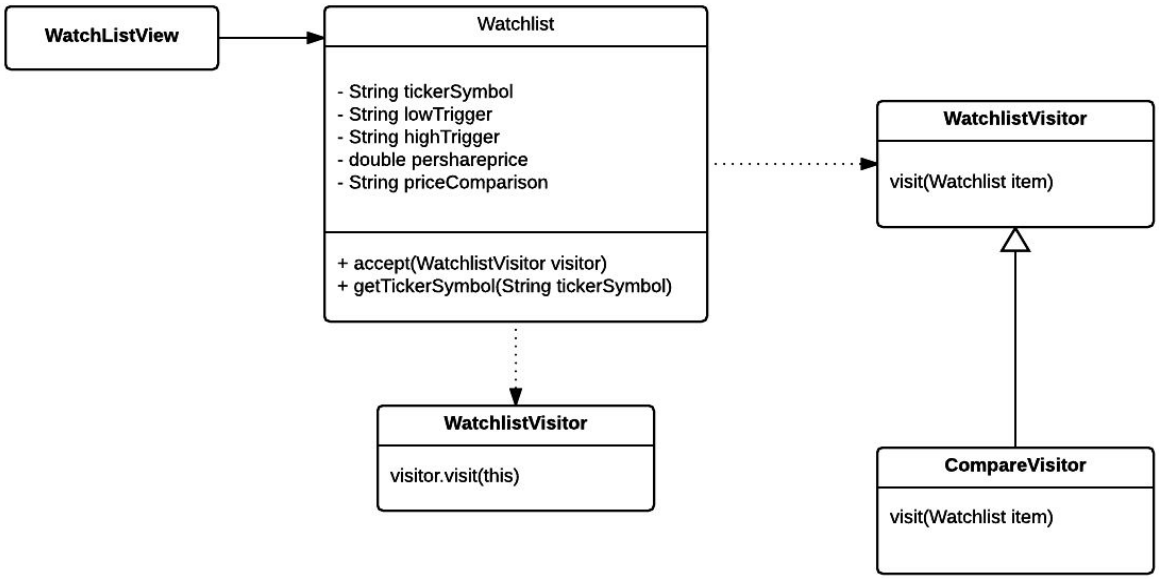
UndoAction object, added into a queue, and executed by handleCommand(). Then it returns that to the client which is the GUI (FTPS) and the Portfolio / CashAccount classes.

Subsystem #3 - UndoAction (Iterator)



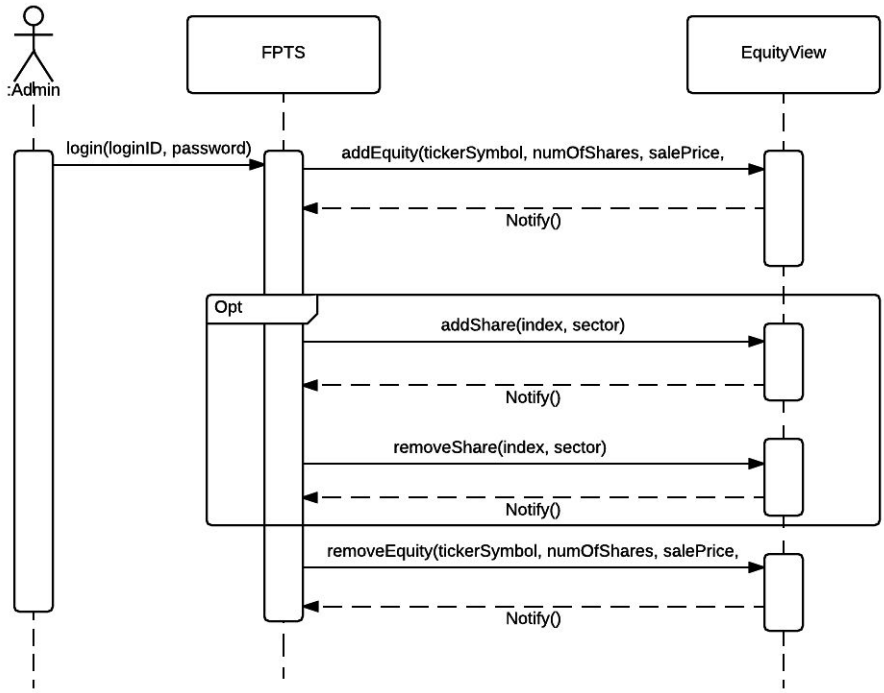
Name: UndoAction	GoF Design Pattern Name: Iterator	
Pattern Name in terms of system context: UndoAction Iterator		
Participants: UndoAction		
Class	**Role in pattern**	**Participant's contribution in the context of the application**
UndoAction	ConcreteIterator, Aggregate, ConcreteAggregate	The handleCommand() gets the ArrayDeque iterator from the createIterator() function and uses that to traverse and execute each object in the Undo/Redo ArrayDequeue
Deviations from the standard pattern: ● Iterator only contains two functions needed for the actual iteration, which is hasNext() and next()		
Requirements being covered: ● Undoing and Redoing saves each recent portfolio modification as an UndoAction object which is stored into an ArrayDeqeue. Upon clicking the undo/redo buttons, we use the iterator pattern to traverse through the respective ArrayDeqeue and perform the action to be undone. Depending on how many times Undo/Redo is clicked, we undo/redo up to the last five modifications.		

Subsystem #4 - Watchlist (Visitor)



Name: Watchlist	GoF Design Pattern Name: Visitor	
Pattern Name in terms of system context: Watchlist		
Participants: Watchlist, WatchlistVisitor		
Class	**Role in pattern**	**Participant's contribution in the context of the application**
WatchlistVisitor	Visitor	Defines the Interface that lets the Watchlist class implement it and accept the visitor.
Watchlist	ConcreteElement	Actual watchlist class accepts the visitor.
CompareVisitor	ConcreteVisitor	Actual visitor that Visits each and individual watchlist to compare prices to their respective triggers
WatchListView	Client	Updates Watchlist objects (GUI and Backend)
Deviations from the standard pattern: ● Went straight to concreteElement instead of the element itself, meaning there is no element but just a concrete element.		
Requirements being covered: ● Pattern does not modify anything from each and individual watchlist that it visits, but it does warn the user of high and low triggers.		

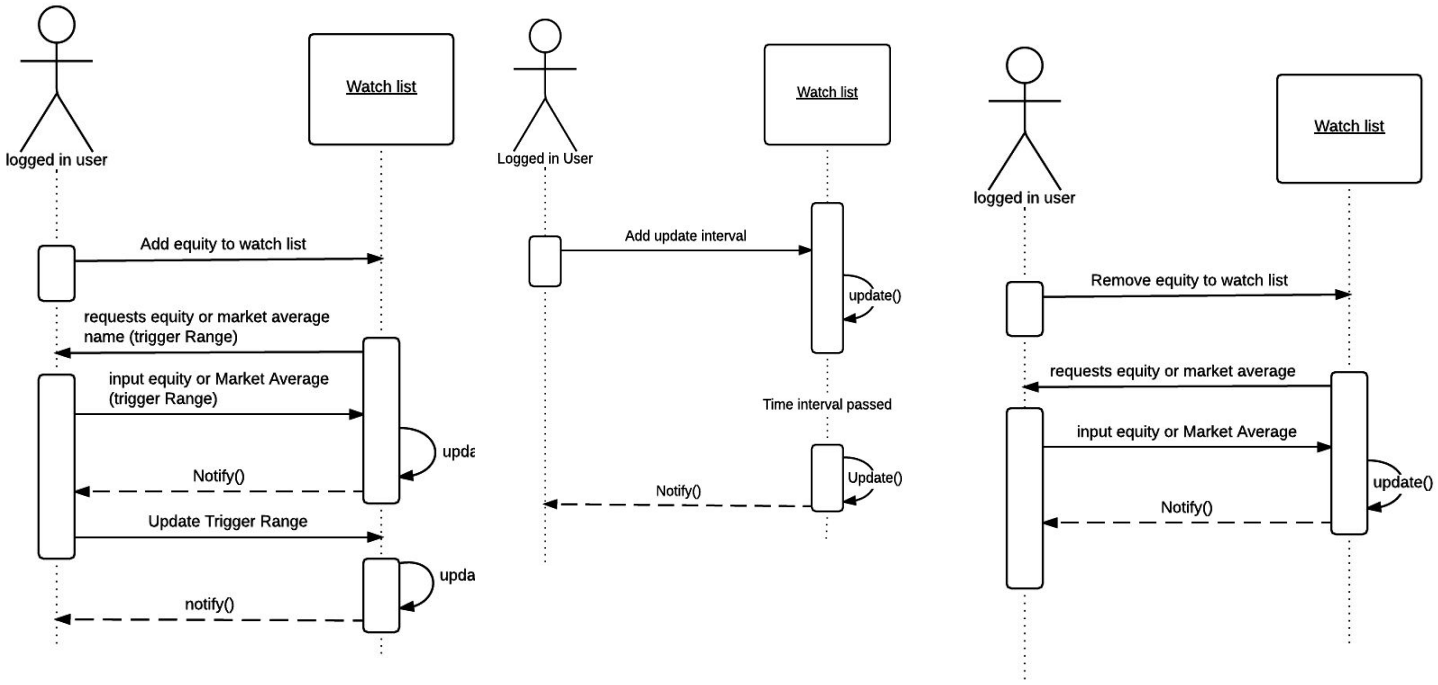
Add/Remove Equities Sequence Diagram



A user is able to add an Equity to their respective portfolio by first logging into their portfolio. After the user is successfully logged in, they will have an add Equity option which they must then input the associated fields of the equity to properly add it to the portfolio. To remove an equity, it is essentially the same action except remove Equity is chosen instead. Again to remove an equity the user must specify the equities fields that the user wishes to remove.

Update & Add/Remove from user's watchlist Sequence Diagrams

Add to watchlist **Update watchlist** **Remove from watchlist**



A user can add an equity or a market average to their watchlist. When adding to the watchlist the user has the option to add a low trigger and or a high trigger to a watchlist item. If one of the item's value goes lower than the low trigger or higher than the upper trigger, then the user is notified. The user is also able to specify the duration of time between individual updates done to the watchlist. The user is also able to remove watchlist items as they please. Also a user is able to add or remove trigger(s) from watchlist items.

Release 2 Design Updates

Release 1:

Our old command pattern was used incorrectly for Release 1, thus we changed it and applied it to our Undo/Redo design which better represents the intent of the command pattern. Also in Release 1, we had Portfolio and FPTS as god classes. Based on the feedback given, we broke these up and separated the concerns in order to have higher cohesion in our system.

Release 2:

Along with release 2, we implemented two more patterns, the iterator & visitor pattern. The iterator pattern was implemented with our UndoAction class as well. By instantiating an iterator, we traversed through the ArrayDeque that contained UndoAction classes to be executed upon instruction. The visitor pattern was associated with our Watchlist class and it visited the array of Watchlist objects individually to see if it had reached the low or high trigger, which would then alert the user.

Status of the Implementation

Missing Features:

- Filtering transaction history between two dates
- Options for continuing simulations after finishing a simulation
- Updating on an interval; to remedy this, there is a button that allows the user to update

Known Issues/Bugs:

-

Appendix

Class: Portfolio	
Responsibilities: A Portfolio is mapped to a user login and password which contains a list of equities as well as cash accounts. This class is also responsible for exporting and importing the whole portfolio.	
Collaborators	
Uses: Cash Account, Equity	**Used by:** FTPS
Author: Everyone	

Class: CashAccount	
Responsibilities: The overall responsibility of this class is to interact with equities where you buy and sell equities. Has the ability to transfer funds from one CashAccount object to another and is mapped to a single Portfolio object. Handles imports and exports of cash accounts. Can add and remove cash accounts from it's list and updates the balance in its respective cash accounts. Can transfer from one cash account to another.	
Collaborators	
Uses: N/A	**Used by:** Portfolio, CAView
Author: Everyone	

Class: Equity	
Responsibilities: Contains shares and represents a certain number of stock. Also represented a ticker symbol / acquisition date / sector index. Handles imports and exports of equities. Can add and remove equities from it's list and updates the balance in its respective cash accounts.	
Collaborators	
Uses: N/A	**Used by:** Portfolio, EquitiesView
Author: Everyone	

Class: Future Simulation Algorithm	
Responsibilities: The overall responsibility of this class is to facilitate the simulation of portfolio values over a certain period of time using different market growth algorithms including bull market, bear market, and no growth algorithms.	
Collaborators	
Uses: BearMarket, BullMarket, NoGrowth	**Used by:** Portfolio, FPTS
Author: Zachary Stauffer	

Class: FPTS	
Responsibilities: Handles views for login, registration, and the general tabular view of the FPTS	
Collaborators	
Uses: Portfolio, UndoAction	**Used by:** N/A
Author: Everyone	

Class: CAView	
Responsibilities: This class displays the information within the portfolio pertaining to cash accounts. For example all of the owned accounts with their balances and other info. Also provides buttons to execute various actions with cash accounts.	
Collaborators	
Uses: Cash Account	**Used by:** N/A
Author: Everyone	

Class: EquitiesView	
Responsibilities: This class displays the information within the portfolio pertaining to equities. For example all of the owned equities with their shares owned and other info. There is a table with the available equities as well. Also provides buttons to execute various actions with cash accounts.	
Collaborators	
Uses: Equity	**Used by:** N/A
Author: Everyone	

Class: simulationView	
Responsibilities: This class handles the viewing of simulations that you want to execute as well as a graph displaying the effects of the simulation over time.	
Collaborators	
Uses: Future Simulation Algorithm	**Used by:** N/A
Author: Everyone	

Class: transactionView	
Responsibilities: This class displays all of the actions that have been logged in the history of the portfolio. There are also buttons that allow the user to do various actions with the history.	
Collaborators	
Uses: History	**Used by:** N/A
Author: Everyone	

Class: History	
Responsibilities: This class has two different objects that will be used by the history loggers depending on if it is a cash account or equity. This class also has various getter functions to be used by other classes. Also handles imports and exports of transaction history.	
Collaborators	
Uses: CashAccount, Equity	**Used by:** Portfolio, FPTS, LogHistory, HistoryInvoker
Author: Brett Kosciolek	

Class: HistoryInvoker	
Responsibilities: After the Portfolio gets a request that LogHistory will evaluate and call a method depending on what action was made. If it is an equity action then the LogEquityHistory will log a new history event and this same thing will happen with cash accounts it will just use LogCAHistory instead.	
Collaborators	
Uses: LogHistory, Portfolio, History	**Used by:** Portfolio, FPTS, CashAccount
Author: Brett Kosciolek	

Class: LogCAHistory	
Responsibilities: Logs history relating to a Portfolio's transactions with Cash Accounts.	
Collaborators	
Uses: LogHistory	**Used by:** HistoryInvoker
Author: Brett Kosciolek	

Class: LogEquityHistory	
Responsibilities: Logs history relating to a Portfolio's transactions with Equities.	
Collaborators	
Uses: LogHistory	**Used by:** HistoryInvoker
Author: Brett Kosciolek	

Class: LogHistory	
Responsibilities: Act as an interface in the Command Pattern.	
Collaborators	
Uses: History, Portfolio	**Used by:** LogEquityHistory, LogCAHistory, HistoryInvoker
Author: Brett Kosciolek	

Class: UndoAction	
Responsibilities: Undoes any recent modification to the portfolio. Undoes the action and logs it into our log history. Also handles the redo action.	
Collaborators	
Uses: Equity, CashAccount, Portfolio, handleCommand	**Used by:** CAView, EquitiesView, FPTS, transactionView
Author: Sadaf Chowdhury, Daniel Dang, Aaron Sloan	

Class: YahooParse	
Responsibilities: Takes in the XML of YahooGET and returns the prices of each equity.	
Collaborators	
Uses: N/A	**Used by:** UpdateEQ
Author: Brett Kosciolek	

Class: YahooGET	
Responsibilities: Handles getting raw information from Yahoo and putting the information into XML format.	
Collaborators	
Uses: N/A	**Used by:** UpdateEQ
Author: Brett Kosciolek	

Class: UpdateEQ	
Responsibilities: Uses YahooGET and YahooParse to get and set the current asking price of equities in the system.	
Collaborators	
Uses: YahooGET, YahooParse	**Used by:** N/A
Author: Brett Kosciolek	

Class: WatchlistView	
Responsibilities: Handles displaying the GUI of watchlist. Displays watchlist objects as a list with its high and low triggers and whether or not those triggers have been met or are met currently. Also provides buttons which allow the user to add items to the watchlist, remove items from the watchlist, set the high and low triggers of an item in the watchlist, and reset if a trigger has been met for a watchlist item.	
Collaborators	

Uses: YahooGET, YahooParse, Equity, Watchlist	**Used by:** N/A
Author: Jeff Kotowicz, Aaron Sloan, Daniel Dang	

Class: Watchlist	
Responsibilities: Contains name of watchlist item, its low and high triggered, and information telling whether its low or high triggers have been tripped.	
Collaborators	
Uses: N/A	**Used by:** WatchlistView
Author: Daniel Dang	

Class: handleCommand	
Responsibilities: Interface for handleCommand in UndoAction.	
Collaborators	
Uses: N/A	**Used by:** UndoAction
Author: Sadaf Chowdhury	

Class: WatchlistVisitor	
Responsibilities: Interface for visitor pattern in Watchlist.	
Collaborators	
Uses: N/A	**Used by:** Watchlist
Author: Sadaf Chowdhury	