**CS308 Folio Tracker Final GUI and Back-end API Design**

**Overview**

This application is used to manage stock portfolios and view the value of both individual stocks and the total value of your portfolio

**Changes**

* Added JFileChooser options for saving portfolios
* Added profit/loss column as well as initial share price.
* Moved the buy button to other side so of the GUI.
* Added waiting screen for when a stock is being bought and added to the portfolio.
* Moved total folio price away from the edge of the screen.

**API Overview**

IStock is mainly used as a way to get data out of the stock object the only modification that happens directly is of the name. We felt that everything else like buying and selling of stock could be handled a level up in IPortfolio.

IPortfolio is used to maintain the stocks in a portfolio. Such as the buying and selling of stocks by ticker name, this is handled in IPortfolio as this allows it to be handled centrally and it can handle cases where a stock will have to be created or deleted.

IPortfolioTracker is used to handle and maintain individual portfolios. It allows basic operations such as creating/deleting and saving/loading.

**Specification API**

IPortfolio:

effects: returns the name associated with this portfolio

String getPortfolioName();

effects: returns all the tickers owned within this portfolio

Set<String> getStockTickers();

requires: ticker != null

effects: returns the IStock with the ticker value of the parameter ticker, if ticker is not associated to any stock then returns null

IStock getStockByTicker(String ticker);

requires: ticker != null, numOfShares >0

modifies: this

effects: if tickerSymbol is known to the portfolio then increases number of shares by numOfShares in the associated stock and returns true, else if the ticker is a real stock ticker then a stock is created with the tickerSymbol and numOfShares and returns true, otherwise false.

Boolean buyStock(String ticker, int numOfShares);

requires: ticker != null, numOfShares >0

modifies: this

effects: if ticker is not known to the portfolio returns null otherwise if associated stock has enough shares then decreases number of shares by numOfShares and returns true else false;

Boolean sellStock(String ticker, int numOfShares);

IStock:

effects: returns the ticker associated with this Stock

String getTickerSymbol();

effects: returns the name associated with this stock

String getStockName();

effects: returns the number of shares owned

int getNumShares();

effects: returns the value of a share of this stock

Double getPricePerShare();

effects: returns the total value of this stock

Double getValueOfHolding();

requires: stockName != null

effects: sets the name associated with this stock to stockName

void setStockName(String stockName);

IPortfolioTracker:

effects: returns all the portfolio names in the tracker

Set<String> getPortfolioNames();

requires: name != null

effects: returns the folio associated with name, if none found then return null

IPortfolio getPortfolioByName(String name);

effects: returns false if folio with name doesn’t exist otherwise removes folio with name returns true

Boolean deletePortfolioByName(String name);

effects: returns false if folio with name already exists otherwise creates new folio with name and returns true

Boolean createPortfolio(String name);

effects: returns true if folios saved to disk otherwise false

Boolean savePortfolios();

effects: returns true if folios loaded from disk otherwise false

Boolean loadPortfolioFromFile();

requires: folioName != null, observer != null

effects: folioName isn’t associated to a folio then return false, otherwise adds observer as an observer to the folio associated with folioName returns true;

Boolean addObserverToFolio(String folioName, Observer observer)

requires: observer != null

effects: adds observer as an observer to prices

void addObserverToPrices(Observer observer);

**Java Assertions**

StockListener

line 81, assertion used to check the if condition forms the correct operation if the sell action is required.

PortfolioListener

line 51 used to check for correct switch statement return.

PortfolioTracker

Line 62 – checking that the delete method has performed it expect operation

Line 75 – checking that create portfolio has indeed created the given folio

Line 82 - checking that the portfolioBefore arrayList is not equal to null

StockTable

Line 184 – checking that the updateField method is working by checking the field parameter is the same NumSharesField

Portfolio

Line 62 – checking that the number of shares before selling is not equal to null

Line 76 – checking the checkSell method returns the expected values

**Junit Test**

There were 4 test classes created for each class within the model. They were PortfolioTest, PortfolioTrackerTest, PricesTest and StockTest. The tests within each class were used to assure that each class performed as expected. Each test class tests the Reflective, Symmetric, Transitive, consistence and not equal too null. All the methods within each class are also tested to get the high line coverage possible and ensure high level of testing.

Junit Coverage

As can be seen in the screenshot below, this is the coverage of our model package.

