**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.

**Design Rational**

Overall:

Our design is split up into the three main parts, model, view and controller. Our view is mainly passive with the views taking all user input. This user input is then collected in the controller and passed through to specific methods in our model. Once the model has done the action asked by the controller, if something has changed (like creation of a portfolio) then the model updates all of its observers with the appropriate ViewUpdateType enum. In our case it is the view which is the observer and once it receives the update type it can then request the new information and refresh accordingly.

Model:

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. We also handle the setting name of a specific stock as this means we can keep the stock object encapsulated within the IPortfolio.

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

Some of the feedback we received on our initial design was that we may want to think about the visibility of our Prices class. We didn’t feel the need change from our original design and we kept Prices hidden and only used by the backend. The idea behind this class was that it offers a centralised place where we can store the most recent prices for the stocks we know about. This was preferable to us over the alternative of keeping a copy of the share price with each stock object. This meant that when we go to refresh prices we just need to do it once per stock. This also meant that instead of having to set each stock’s share price we can just refer to this centralised price storage from stock whenever the price is wanted. We didn’t feel it was nessecary to move this class up to the api level as other implementations of the model may not want to use a centralised system like ours.

**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

modifies: this

effects: if the ticker is associated with a stock then the stock’s name is set to newname and return true, else return false

Boolean setNameOfStock(String ticker, String newName);

requires: ticker != null

effects: returns the IStock with the ticker value equal to 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 ticker 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 ticker 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 current price of a given stock

Double getPricePerShare();

effects: returns the total value of this stock

Double getValueOfHolding();

effects: returns the initial share price of the stock

Double getInitalPricePerShare();

effects: returns the total profit of a stock since buying at initial price

Double getProfitOfHolding();

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(File file);

effects: returns true if folios loaded from disk otherwise false

Boolean loadPortfolioFromFile(File file);

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 name, Observer observer)

requires: observer != null

effects: adds observer parameter as an observer to prices

void addObserverToPrices(Observer observer);