W03 Assertions and JUnit Coverage Rationale

for PortfolioTracker

Assertions

Assertions have been used throughout the program to assure the state of the program at that current point is correct.

This assertion inside `addStock()` method (line 34) of `Portfolio` class checks whether or not the stock price was updated after it was found in stock set. The only two conditions under which it may not be found in the set are either the reference passed is a null reference or the stock object being already in the set. The former is excluded because the stock object is constructed programmatically and cannot be null. Hence, `updateStock()` should not return false under normal circumstances.

```
assert (stockUpdated) : "The stock must exist in the set, but its quantity
is not updated. The stock is " + stock;
```

The assertion inside `mouseClicked()` method in `TableRowMouseListener` (line 48) checks if there is right object mapping from to the view to the model. Because a click can only be registered on a table row of portfolio view, there **must** be corresponding portfolio associated with it. Even though `getStock()` method in `Portfolio` class may potentially return null if the match is not found, it should never happen.

```
assert stock != null : "Stock is not supposed to be null, but it is null";
```

The number of rows in the table that represents portfolio view must be exactly the number of stocks in the portfolio. The assertion is in the `refresh()` method of `PortfolioTableModel` (line 69).

```
assert compareStockSizeAndDataTable(stocks, data) : "The table should
contain as many rows as there are stocks in the portfolio. Stock size is :
" + stocks.size() + " Row number is : " + data.size();
```

The assertion in `getTotalValue()` method of *Portfolio* class (line 102) checks the number of shares in a stock and share price.

```
assert(checkNumberOfShares(stock) && checkSharePrice(stock)) : "The number
of shares should only be positive or 0" + " and price of the stock should
always be positive. The numbe of shares is " + stock.getNumberOfShares()
+ "and the share price is " + stock.getPricePerShare();
```

Tests

We were able to achieve high line coverage across all our primary classes.

Class	**Line Coverage %**	**Comment**
Stock	94	Exquisite
PortfolioManager	94	Quite astounding if I do say so myself
Portfolio	92	Surprising to say the least
StrathQuoteServer	72	Came at quite a shock

Why is that so?
Stock contains methods that are very accessible. We were able to fully test all methods apart from `updateStockPrice()`, a method that fetches and updates stock. The stock value state is unknown before running the method. Stock objects were tested for equality based on five defining characteristics.

Portfolio also contains methods that are quite lightweight functionality wise. Portfolio objects were tested for equality based on five defining characteristics. Several tests had to be written for the addStock method. `addStock()` method that adds a new stock to the portfolio or updates the price if that stock is present. updateStock() is private and `return false` statement wouldn't be reachable, hence assertion was used instead. Portfolio objects can be serialised, therefore saving and deleting methods were tested.

`PortfolioManager` was close to 100% coverage apart from method that was inherited due to the `IPortfolioManager` interface extending the Iterable class.

QuoteServer contained several methods we were unable to exercise fully. As it relies on dynamic content from an external source.