# CS 583 Assignment 5

Authors: Christina Yu, Austin Short

November 7, 2017

# Introduction

The following document describes the test cases that were built for the StockWatcher project. We have concentrated on testing specific classes like StockWatcher Class and StockPrice Class using unit testing techniques.

One of our main focus is testing private methods in StockWatcher Class. Since the private methods are only helper methods, and not meant to be used from outside the class, we found it helpful to **remove the private modifier and make the method package private**. By dropping the private access modifier, the method is visible in the package of the class, and nowhere else. We also used the technique of **dividing the tasks up into small methods, and unit test each method individually**. This technique is very practical. For example, StockWatcher.refreshWatchList() method has two main functions: 1. To add the prices to the stock array 2. To update the GUI table. And we rewrote this method into two parts, so as to focused on testing the logics.

# Setting up the test class

testStockPriceCtor() and testSimple() passes.

```
58      public String getModuleName() {                                    // <span style="color:black;">**(2)**</
59          return "com.google.gwt.sample.stockwatcher.StockWatcher";
60      }
61
62      /**  * Verify that the instance fields in the StockPrice class are set correctly.  */
63      public void testStockPriceCtor() {
64          String symbol = "XYZ";
65          double price = 70.0;
66          double change = 2.0;
67          double changePercent = 100.0 * change / price;
68          StockPrice sp = new StockPrice(symbol, price, change);
69          assertNotNull(sp);
70          assertEquals(symbol, sp.getSymbol());
71          assertEquals(price, sp.getPrice(), 0.001);
72          assertEquals(change, sp.getChange(), 0.001);
73          assertEquals(changePercent, sp.getChangePercent(), 0.001);
74      }
75
76      public void testSimple() {                                         // <span style="color:black;">**(3)**</
77        assertTrue(true);
78      }
79
```

JUnit

Finished after 13.153 seconds

Runs: 2/2 Errors: 0 Failures: 0

com.google.gwt.sample.stockwatcher.client.StockWatcherTest [Runner: JUnit 4] (13.

testStockPriceCtor (13.060 s)

testSimple (0.063 s)

Failure Trace

# Test Case: testAddStock()

# Pass

The "testAddStock()" is a split method originally from StockWatcher.addStock()

```
// Add the stock to the table.
int row = stocksFlexTable.getRowCount();
addStock(symbol);
stocksFlexTable.setText(row, 0, symbol);
stocksFlexTable.setWidget(row, 2, new Label());
stocksFlexTable.getCellFormatter().addStyleName(row, 1, "watchListNumericColumn");
stocksFlexTable.getCellFormatter().addStyleName(row, 2, "watchListNumericColumn");
stocksFlexTable.getCellFormatter().addStyleName(row, 3, "watchListRemoveColumn");
```

addStock function is split out to be an individule method

In the original method addStock(), we split the functions, so we can focused on testing if the stock can be added correctly.

```
static void addStock(final String symbol){
    stocks.add(symbol);
}
```

We tried to add the stock ”XYZ” to the stocks array by calling addStock(symbol) in StockWatcher class. The test passes as “XYZ” is successfully added.

```
41      /**
42       * To test if the stock can be added correctly
43       */
44      public void testAddStock(){
45          String symbol = "XYZ";
46          double price = 70.0;
47          double change = 2.0;
48          double changePercent = 100.0 * change / price;
49          StockPrice sp = new StockPrice(symbol, price, change);
50          StockWatcher.addStock(sp.getSymbol());
51          assertEquals(1, StockWatcher.stocks.size());
52      }
```

We called addStock(symbol) in StockWatcher class
Then the stock was successfully added.

JUnit

Finished after 13.865 seconds

Runs: 4/4 Errors: 0 Failures: 0

com.google.gwt.sample.stockwatcher.client.StockWatcherTest [Runner: JUnit 4] (13.8

- testRemoveStock (13.696 s)
- testAddStock (0.072 s)
- testStockPriceCtor (0.045 s)
- testSimple (0.028 s)

Failure Trace

# Test Case: testRemoveStock()

# Pass

The "testRemoveStock()" is a split method originally from StockWatcher.addStock()

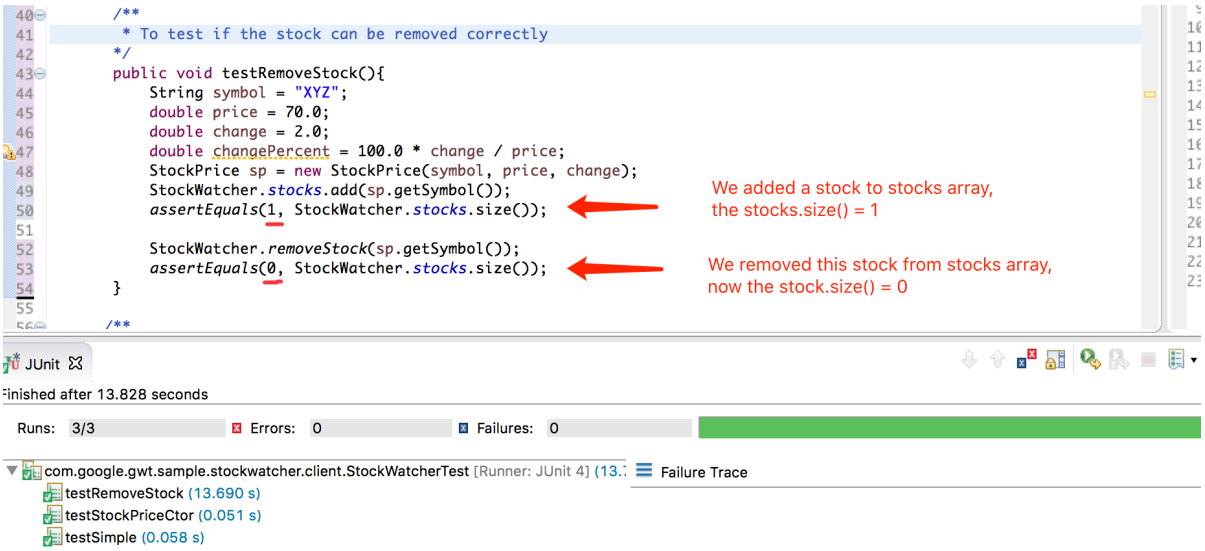
```
// Add a button to remove this stock from the table.
    Button removeStockButton = new Button("x");
    removeStockButton.addStyleDependentName("remove");
    removeStockButton.addClickHandler(new ClickHandler() {
        public void onClick(ClickEvent event) {
            int removedIndex = removeStock(symbol);
            stocksFlexTable.removeRow(removedIndex + 1);
        }
    });
  stocksFlexTable.setWidget(row, 3, removeStockButton);
```

removeStock function is split out to be an individule method

In the original method addStock(), we split the functions, so we can focused on testing if the stock can be removed correctly.

```
static int removeStock(final String symbol){
    int removedIndex = stocks.indexOf(symbol);
    stocks.remove(removedIndex);
    return removedIndex;
}
```

We first added a stock to the stocks array, and checked the size was 1. Then we deleted that stock, then the size become 0. Thus the test passed.




# Test Case: testAddPrices()

# Pass

---

The "testAddPrices()" is a split method originally from StockWatcher.refreshWatchList()

```
/**
 * Generate random stock prices.
 */
private static void refreshWatchList() {
    StockPrice[] prices = addPrices();
    updateTable(prices);
    }
```

We split the functions in the original method refreshWatchList(), so we can focused on testing if the stockPrices can be added to the stocks correctly.

```
static StockPrice[] addPrices() {
    final double MAX_PRICE = 100.0; // $100.00
    final double MAX_PRICE_CHANGE = 0.02; // +/- 2%
    StockPrice[] prices = new StockPrice[stocks.size()];
    for (int i = 0; i < stocks.size(); i++) {
    double price = Random.nextDouble() * MAX_PRICE;
    double change = price * MAX_PRICE_CHANGE
    * (Random.nextDouble() * 2.0 - 1.0);
    prices[i] = new StockPrice(stocks.get(i), price, change);
    }
    return prices;
    }
```

In the test, we first added 5 stocks, and called addPrices() in the StockWatcher class, then we checked the price.length to see if they were added correctly to the StockPrice arrayList. And the test passed.

```
71      public void testAddPrices() {
72              StockWatcher.stocks.add("Alibaba");
73              StockWatcher.stocks.add("Apple");
74              StockWatcher.stocks.add("Amazon");
75              StockWatcher.stocks.add("Android");
76              StockPrice[] prices = StockWatcher.addPrices();
77              String symbol = "XYZ";
78              double price = 70.0;
79              double change = 2.0;
80              double changePercent = 100.0 * change / price;
81              StockPrice sp = new StockPrice(symbol, price, change);
82              prices[4] = sp;
83
84              assertEquals(5, prices.length);
85              assertTrue(prices[0].getSymbol().equals("Alibaba"));
86              }
```

We added 5 sotcks and called the addPrices() method in StockWatcher class. We then checked the prices.length, it becomes 5.

JUnit

Finished after 13.988 seconds

Runs: 5/5 Errors: 0 Failures: 0

com.google.gwt.sample.stockwatcher.client.StockWatcherTest [Runner: JUnit 4] (1

- testRemoveStock (13.802 s)
- testAddStock (0.053 s)
- testAddPrices (0.049 s)
- testStockPriceCtor (0.031 s)
- testSimple (0.022 s)

Failure Trace

# Test Cases: testUpdateTable(StockPrice prices) & testUpdateTable(StockPrice[] prices) Pass

This method tests adding stocks, and then updating their prices and determining if the table updates. The new addStock() function is used to add stocks to the table, so that we could add stocks to the table through code, instead of through the GUI. We decided to test with an array of prices, because the update method loops through the array of prices and calls a method on each element of the array. So this test method effectively tests both the updateTable(StockPrice[] prices) method and the updateTable(StockPrice price) method. These methods were tested by removing the private modifier and making the methods package private.

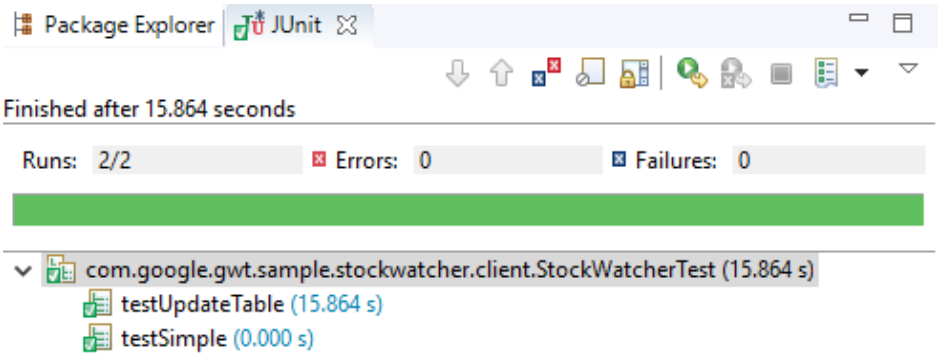
```
public void testUpdateTable() {
    StockWatcher watch = new StockWatcher();
    watch.onModuleLoad();
    FlexTable table = watch.getStocksFlexTable();
    //Table should just have 1 row to start
    assertEquals(table.getRowCount(), 1);
    //add some stocks
    watch.addStock("Alibaba");
    watch.addStock("Apple");
    watch.addStock("Amazon");

    //Create some prices for the stocks, and add them to an array
    StockPrice[] prices = watch.addPrices();
    prices[0] = new StockPrice("Alibaba", 5, 3);
    prices[1] = new StockPrice("Apple", 2, 1);
    prices[2] = new StockPrice("Amazon", 4, -1);
```

```
    //update the table with the prices
    watch.updateTable(prices);
    //Get the updated table
    table = watch.getStocksFlexTable();

    //verify that the changes came through
    //There should be 4 rows now, a header and 3 stocks
    assertEquals(table.getRowCount(), 4);
    //Verify that the name of the stocks is still the same
    assertEquals(table.getText(1, 0), "Alibaba");
    assertEquals(table.getText(2, 0), "Apple");
    assertEquals(table.getText(3, 0), "Amazon");
    //Verify that the prices are showing
    assertEquals(table.getText(1, 1), "5.00");
    assertEquals(table.getText(2, 1), "2.00");
    assertEquals(table.getText(3, 1), "4.00");
    //Verify that the change in price is reflected
    assertTrue(((Label)table.getWidget(1, 2)).getText().contains("+3"));
    assertTrue(((Label)table.getWidget(2, 2)).getText().contains("+1"));
    assertTrue(((Label)table.getWidget(3, 2)).getText().contains("-1"));
}
```

As you can see, our testUpdateTable(StockPrice[] prices) and testUpdateTable(StockPrice prices) tests passed. The table was updated with all new stock information.




# Summary

- Methods rewrite: 3
- Test case created:

  Christina Yu: 7;  Austin Short: 3