SE2832 Lab 6: A Stock Market Ticker

Due: April 23, 2014 23:59

1. Introduction

In this week's lecture, a discussion was held on using Mock objects to test the functionality of programs using JUnit. In this lab, you will actually use mock objects to test the behavior of a stock ticker system. (And, if the source for stock market quotes stays down, that may be the only way you see the program run. ☺)

2. Lab Objectives

- To use Mock objects to verify the behavior of a class under test.
- Obtain familiarity with JMock, a tool for developing Mock Objects
- To apply JUnit/TestNG to the testing of an existing Java application
- To apply boundary conditions to determine proper test results for the system
- To debug existing Java code and remove faults which have been injected by an external entity

3. Problem Overview

The MSOE stock ticker is designed to monitor the stock market and determine the fate of vast numbers of investments. When the system is started, the user enters two things. The first is a refresh rate. This determines how often the market analyzer queries the server to determine trading values. The second piece of information provided is a listing of stocks that are to be monitored. This may be one stock, or it may be multiple stocks. This information is passed as command line parameters.

When operating, the system will query the remote server every n seconds (where n is the refresh rate) and generate a ticker report, which is written to the console. A sample output is shown in the Figure below.

The first segment lists the name of the company and the stock symbol. That is followed by the current value of the stock. The change since the last check is then shown, with a series of + or – signs designating if the stock has gone up or down and a percentage of the change. The number of signs is 5 if the change is greater than 10% since the last check, 4 if greater than 5%, 3 if greater than 2%, 2 if greater than 1%, 1 if greater than .5%, and zero otherwise.

This is then followed by the same data printed relative to the open of the stock today.

If the change since the last time is greater than ½% or the change since yesterday is greater than 2%, then either happy ("We're in the Money") or sad music ("GRRR") will be played, depending on whether the market is up or down. If an error occurs, and the system is unable to connect with the web site to download a quote, then an error message will be printed and an error message ("Failure is not an option") will be played.

Your job is to test the application to make certain it is working correctly. (After all, a lot of investments are riding on this high quality tool.:-)

```
################################################################################
Ford Motor Co.(F):              $1.99       (0%)       ---  (-4.33%)
Cedar Fair(FUN):                $13.89      (0%)       --   (-1.91%)
General Motors Corp(GM):        $4.89       (0%)       +++  (2.73%)
Verizon Communications(VZ):     $26.77      (0%)       +++  (3.24%)
AT&T Corp(T):                   $22.42      (0%)        ---  (-2.52%)
Genl Electric(GE):              $21.50      (0%)        +++++(13.1%)
Six Flags(SIX):                 $0.40       (0%)       -----(-11.11%)
Walgreen Company(WAG):          $23.22      (0%)        ---- (-8.15%)
Agilent Technologies Inc.(A): $22.48        (0%)        --   (-1.23%)
Microsoft Corp(MSFT):           $21.50      (0%)        ---  (-3.59%)
Intl Bus. Machines(IBM):        $87.75      (0%)        --   (-1.4%)
Morgan J.P. & Co. Inc.(JPM):    $41.64      (0%)        +++++(13.52%)
Wells Fargo & Company(WFC):     $28.31      (0%)        +++  (3.89%)
Bank of America(BAC):           $20.87      (0%)        ++++ (6.32%)
Citigroup Inc.(C):              $14.11      (0%)        ++++ (9.13%)
################################################################################
```

Figure 1 Sample program output.

4. Obtaining JMock

Easy Mock is available at http://www.jmock.org. You will need to download the tool from the site and follow the installation instructions given on the website. You will need to use version 2.5.1 of the tool.

Once you have installed JMock, take a look at the tutorial on developing mock objects with JMock. Essentially, you will need to record the expected behavior as part of a scenario inside of a Mock Object. Once the behavior is recorded, you can switch the mock object into a playback mode and make calls into the source code to invoke the sequences. When all is done, you then verify if the scenario that transpired matches the expected scenario.

The main area you should test is the StockQuoteAnalyzer class. Testing this class using JMock requires you to mock the behavior for the StockQuoteGeneratorInterface and the StockTickerAudioInterface class.

5. Project specifics

For this lab, you are to work with a lab partner. You will be developing a JUnit/TestNG test suite for the program. In doing so, you will be required to use Mock objects to simulate the behavior of the network connection with the data source for the stock quotes. To do this, you will need to create a set of classes which implement the `StockTickerAudioInterface` and `StockQuoteGeneratorInterface` Interfaces. These mock objects will allow the system to obtain simulated stock quotes without the need for a connection to the real server. Furthermore, this will also allow the user to guarantee that the appropriate behavior has been obtained regardless of how stocks perform in this volatile market.

In this program, bugs have been injected into the code. Your job, as in previous weeks, is to find them. They are lurking again in common places. The common errors you might make are probably there.

A UML class diagram for the system is shown in Figure 2. Notice that the classes which needed to be mocked are defined as interfaces. Figure 3 provides sequence information showing the execution sequences for the program.

The design for the system is shown in the class diagram, and Java source code is available from the course website.

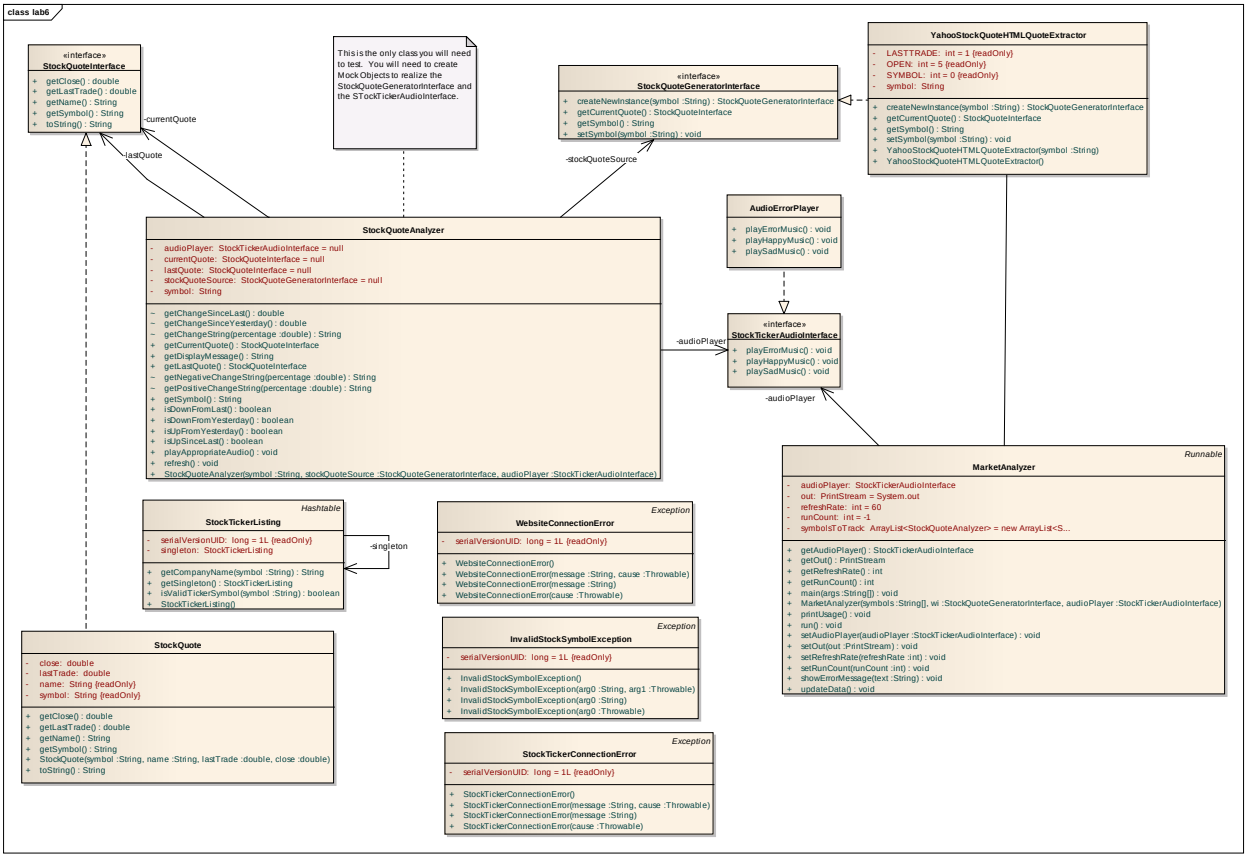

Figure 2 UML Class diagram for Stock Quote Analyzer

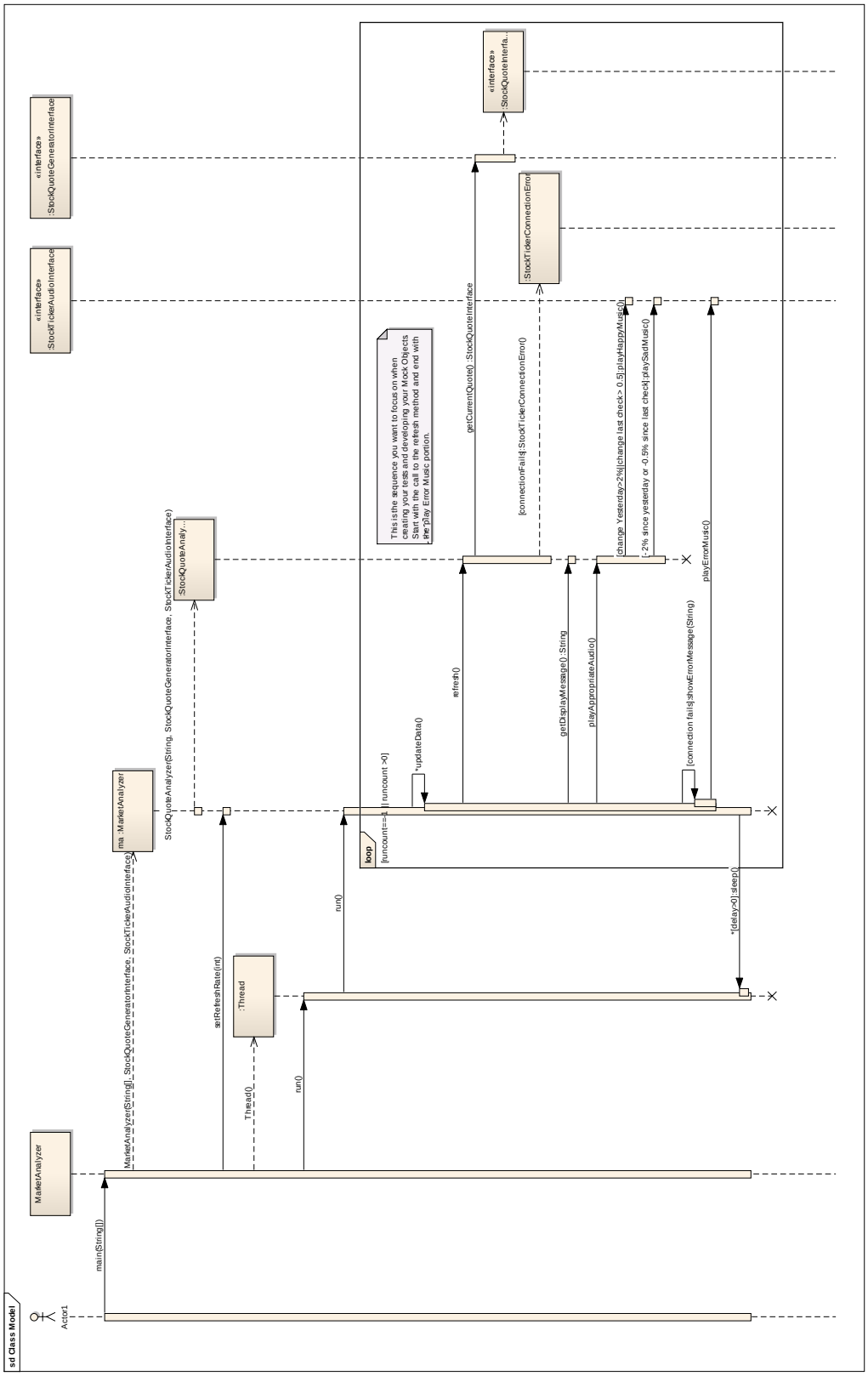
sd Class Model
Actor1
MarketAnalyzer
main(String[])
ma :MarketAnalyzer
MarketAnalyzer(String[], StockQuoteGeneratorInterface, StockTickerAudioInterface)
StockQuoteAnalyzer(String, StockQuoteGeneratorInterface, StockTickerAudioInterface)
:StockQuoteAnaly...
setRefreshRate(int)
:Thread
Thread()
run()
run()
loop
[runcount==-1 || runcount > 0]
updateData()
refresh()
getDisplayMessage():String
playAppropriateAudio()
[connection fail]showErrorMessage(String)
playErrorMusic()
change Yesterday>2%||change last check> 0.5]playHappyMusic()
-2% since yesterday or -0.5% since last check]playSadMusic()
[connectionFail]:StockTickerConnectionError()
getCurrentQuote():StockQuoteInterface
This is the sequence you want to focus on when creating your tests and developing your Mock Objects. Start with the call to the refresh method and end with the play Error Music portion.
«interface» :StockQuoteGeneratorInterface
«interface» :StockTickerAudioInterface
«interface» :StockQuoteInterfa...
:StockTickerConnectionError
*[delay>0]sleep()

Figure 3 Sequence Diagram for Stock Analyzer System.

6. Deliverables / Submission

Each lab group will be responsible for submitting one report with the following contents:

1. Introduction -> What are you trying to accomplish with this lab? This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment.
2. Testing Strategy -> What strategy did you use to create your test cases? How did you go about coming up with stock values to sue for testing?
3. Fault Locations -> Did you find any locations in your testing, and if so, how did you fix them? (NOTE: You do not need to file bugs in Mantis. However, you do need to keep a log of the problems you find.)
4. Things gone right / Things gone wrong -> This section shall discuss the things which went correctly with this experiment as well as the things which posed problems during this lab.
5. Conclusions -> What have you learned with this experience? What improvements can be made in this experience in the future?
6. Appendix -> JUnit test code. Submit your JUnit test code.

This material should be submitted as a single pdf file.

Additionally, your complete development package (corrected code, JUnit test suite, etc.) should be uploaded to the course website.

Because you are working as a group, only one report and code submission is necessary.

If you have any questions, consult your instructor.

Preliminary Grading Rubric

	Weight Factor	Rubric Score	
Test Cases	2	«Test_ Cases»	Test cases - Test cases are thorough and complete. - Test cases properly simulate all possible behaviors based upon the potential sequences. - Test cases fully verify appropriate aspects of the component. - Test cases contain comments and other documentation explaining the reason for their existence.
	2	«Bug_ Fixes»	Bug fixes - Only significant changes in bug locations. - Bug fixes commented / noted - Bugs fixed efficiently
Submission	1	«Introd uction»	Introduction - Introduction fully describes what was intended for the lab - Introduction written in words separate from the lab assignment - Introduction written in multiple sentences
	1	«Testin g_Strat egy»	Testing Strategy - Strategy for approaching test design described - Explanation of stock values used for testing provided - Multiple, complete sentences provided.
	2	«Bug_ Log»	Bug Log - Bug log details the bugs that were uncovered - Bug log details location of bugs in source code - Bug log details the fixes applied to the source code
	2	«Reflec tion»	Reflection: Things gone right and things gone wrong - Things which went right with the lab fully described. - Things which went wrong with the lab fully described.
	1	«Concl usions»	Conclusions - What was learned from the lab described - Written in multiple sentences
	2	«NonL ab_Mat erial_S ubmissi ons»	Non-Lab Material Submission - JUnit test cases submitted both in report and electronically - Corrected source code submitted as well