SWE2721 Lab 11: A Stock Market Ticker

1. Introduction

In last week's lecture, a discussion was held on using Mock objects to test the functionality of programs using TestNG. In this lab, you will 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. ☺) There is a video available showing how the tool works as well.

2. Lab Objectives

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

3. Deliverables / Submission

Tag your final version (which should be on the trunk) with the label LabSubmission1.0. Your code test cases should be thoroughly commented and build cleanly without warnings. You can view your test case execution in the CI system of GitHub.

In addition, submit one report in **pdf format**. The report should contain the following contents:

1. Title Page
   a. Names of all partners
   b. Date
   c. Assignment Title
   d. Repository Link (Please make link human readable – not just hyperlinked)
2. Introduction
   a. What are you trying to accomplish with this lab?
   b. This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment.
3. Testing Strategy
   a. What strategy did you use to create your test cases? Did the tester use equivalence classes, BVA, Input Domain Analysis, Control Flow Graphs, or another technique to determine the test cases?
   b. Based on the strategy used, provide any working documents you may have had (control flow graphs, equivalence partition diagrams, BVA diagrams, etc.) These do not need to be computer generated but should be neat and legible. Taking a photo and pasting into your report is fine so long as the items are legible.
4. Defects found
   a. How many defects did you find and fix?
5. Code coverage
   a. What level of code coverage did you achieve when you first wrote your test cases (i.e. the first time they ran)? You can express this as statement coverage, branch coverage, or any other measurable metric, but you need to explain why you selected the metric you did relative to the problem and the type of code you are testing. You can simply paste this into the report as a screen clip with the relevant explanation.
   b. What level of code coverage did you achieve over the StockQuoteAnalyzer class when the final tests were written? How many test cases did you need to add to achieve this?
6. Things gone right / Things gone wrong
   a. This section shall discuss the things which went correctly with this experiment as well as the things which posed problems during this lab.
7. Conclusions
   a. What have you learned with this experience?
   b. What improvements can be made in this experience in the future?

This material should be submitted as a single **pdf file** in Canvas by the developer who fixed the bugs.

If you have any questions, consult your instructor.

4. 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 four things. The first is an access code for the stock data provider. If you would like to run the full blown system, you will need to register for a free account with fcsapi (https://fcsapi.com/login). The second is a refresh rate. This determines how often the market analyzer queries the server to determine trading values. The third item is the number of times the application is to check the stock values or -1 if it is to continue perpetually. The fourth 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 to the program.

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 Figure 1 below.

The first segment lists the name of the company and the stock symbol. That is followed by the previous closing value for the stock and the current price for the stock. The change since the close and percent change are then displayed in two subsequent columns. The status lists whether the stock is STABLE, RISING, or FALLING. The last column then shows the timestamp of when the quote was published by the stock exchange.

```
> Task :edu.msoe.swe2721.lab11.Main.main()
Start the Market Analyzer.
Stocks will be refreshed every 60 seconds.
The program will run 3 times.
The program will monitor the following stocks:
F          FUN        DIS        NVDA       INTC
Standby for the first quotes....
Tue Apr 01 16:16:59 CDT 2025
##############################################################################################################################################################
                          Stock Name and Symbol        Last Close            Cur. Price              Change   % Change     Status  Timestamp
                           FORD MOTOR CO (    F)       $  10.03   $   9.92   $  -0.11     -1.10%      STABLE 2025-04-01 19:59:59
                           CEDAR FAIR LP (  FUN)       $  35.67   $  36.11   $   0.44      1.23%      STABLE 2025-04-01 19:59:59
                      WALT DISNEY CO/THE (  DIS)       $  98.70   $  97.68   $  -1.02     -1.03%     FALLING 2025-04-01 19:59:59
                             NVIDIA CORP ( NVDA)       $ 108.38   $ 110.15   $   1.77      1.63%      RISING 2025-04-01 19:59:59
                              INTEL CORP ( INTC)       $  22.71   $  22.06   $  -0.65     -2.86%     FALLING 2025-04-01 19:59:59
##############################################################################################################################################################
Tue Apr 01 16:17:59 CDT 2025
##############################################################################################################################################################
                          Stock Name and Symbol        Last Close            Cur. Price              Change   % Change     Status  Timestamp
                           FORD MOTOR CO (    F)       $  10.03   $   9.92   $  -0.11     -1.10%      STABLE 2025-04-01 19:59:59
                           CEDAR FAIR LP (  FUN)       $  35.67   $  36.11   $   0.44      1.23%      STABLE 2025-04-01 19:59:59
                      WALT DISNEY CO/THE (  DIS)       $  98.70   $  97.68   $  -1.02     -1.03%     FALLING 2025-04-01 19:59:59
                             NVIDIA CORP ( NVDA)       $ 108.38   $ 110.15   $   1.77      1.63%      RISING 2025-04-01 19:59:59
                              INTEL CORP ( INTC)       $  22.71   $  22.06   $  -0.65     -2.86%     FALLING 2025-04-01 19:59:59
##############################################################################################################################################################
Tue Apr 01 16:18:59 CDT 2025
##############################################################################################################################################################
                          Stock Name and Symbol        Last Close            Cur. Price              Change   % Change     Status  Timestamp
                           FORD MOTOR CO (    F)       $  10.03   $   9.92   $  -0.11     -1.10%      STABLE 2025-04-01 19:59:59
                           CEDAR FAIR LP (  FUN)       $  35.67   $  36.11   $   0.44      1.23%      STABLE 2025-04-01 19:59:59
                      WALT DISNEY CO/THE (  DIS)       $  98.70   $  97.68   $  -1.02     -1.03%     FALLING 2025-04-01 19:59:59
                             NVIDIA CORP ( NVDA)       $ 108.38   $ 110.15   $   1.77      1.63%      RISING 2025-04-01 19:59:59
                              INTEL CORP ( INTC)       $  22.71   $  22.06   $  -0.65     -2.86%     FALLING 2025-04-01 19:59:59
##############################################################################################################################################################

BUILD SUCCESSFUL in 3m 24s
3 actionable tasks: 1 executed, 2 up-to-date
4:19:22 PM: Execution finished ':edu.msoe.swe2721.lab11.Main.main()'.
```
Figure 1 Sample program output.

While this program is executing it will play either happy music, sad music, or no music depending on the status of the stock that is being followed. If the stock is stable, no music will be played. If the stock is rising, “We’re in the money” will be played, and if it is falling the sound of a bear growl will be heard. 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.:-)

Working with a partner, you are tasked with developing the unit tests and mock objects to fully test the system, as well as uncovering any defects in the supplied code.

5. Obtaining Mockito

Mockito is readily available to you through the gradle configuration of this project. Documentation on using Mockito is available here https://site.mockito.org/ and there is a tutorial available here that is helpful to work with the tool: https://www.tutorialspoint.com/mockito/mockito_junit_integration.htm

The main area you should test is the StockQuoteAnalyzer class. Testing this class using Mockito requires you to mock the behavior for the StockQuoteGeneratorInterface and the StockTickerAudioInterface class. However, you may use the existing StockQuote class, as it is mainly a container class.

6. Project specifics

For this lab, you are to work with a lab partner. You will be developing a TestNG unit test suite for the StockQuoteAnalyzer class. 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 mocks 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. When you find a bug, clearly denote with a comment the correction you made.

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 in the repository.

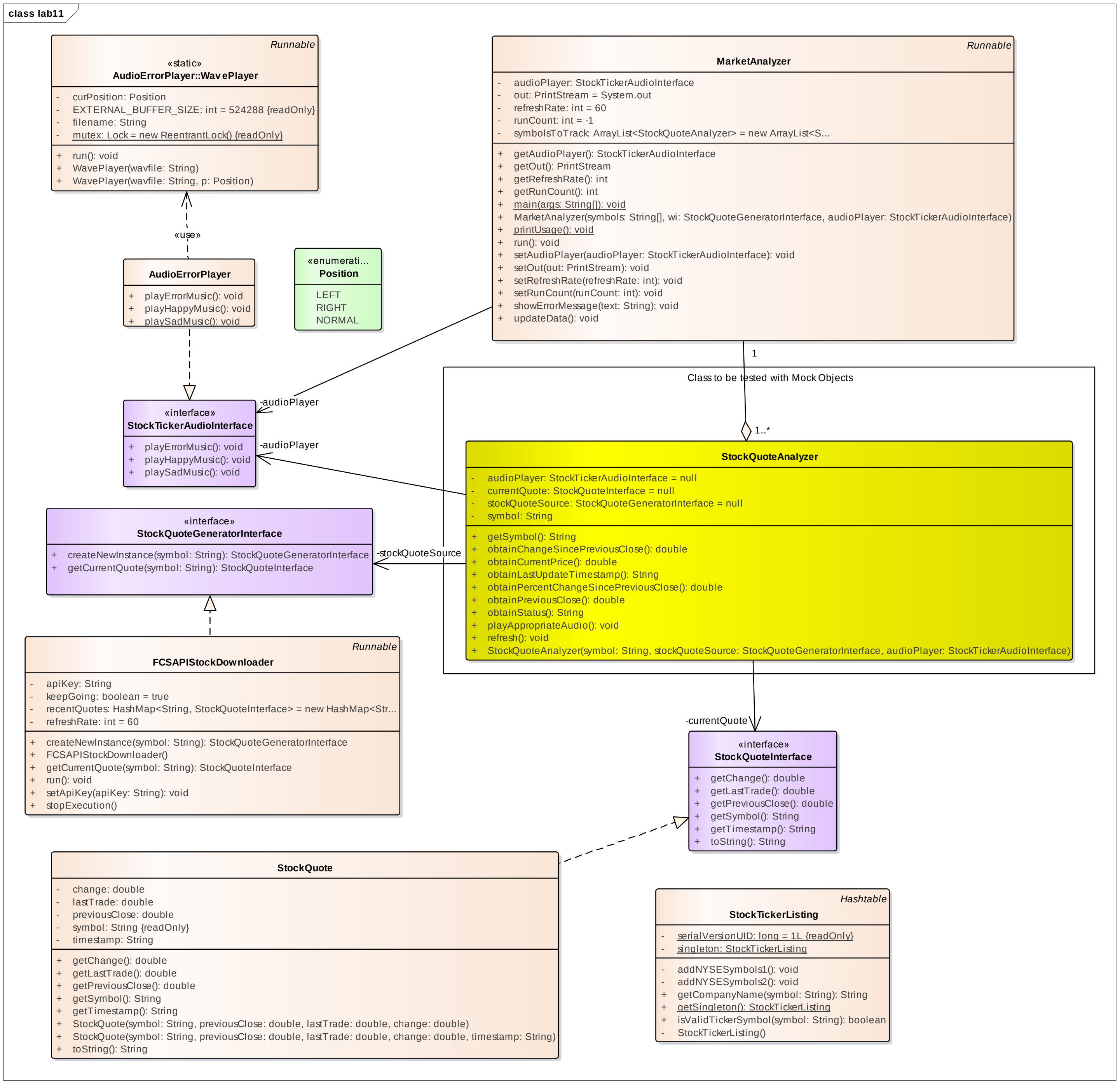
class lab11
Runnable
«static»
AudioErrorPlayer::WavePlayer
- curPosition: Position
- EXTERNAL_BUFFER_SIZE: int = 524288 {readOnly}
- filename: String
- mutex: Lock = new ReentrantLock() {readOnly}
+ run(): void
+ WavePlayer(wavfile: String)
+ WavePlayer(wavfile: String, p: Position)
«use»
AudioErrorPlayer
+ playErrorMusic(): void
+ playHappyMusic(): void
+ playSadMusic(): void
«enumerati...
Position
LEFT
RIGHT
NORMAL
Runnable
MarketAnalyzer
- audioPlayer: StockTickerAudioInterface
- out: PrintStream = System.out
- refreshRate: int = 60
- runCount: int = -1
- symbolsToTrack: ArrayList<StockQuoteAnalyzer> = new ArrayList<S...
+ getAudioPlayer(): StockTickerAudioInterface
+ getOut(): PrintStream
+ getRefreshRate(): int
+ getRunCount(): int
+ main(args: String[]): void
+ MarketAnalyzer(symbols: String[], wi: StockQuoteGeneratorInterface, audioPlayer: StockTickerAudioInterface)
+ printUsage(): void
+ run(): void
+ setAudioPlayer(audioPlayer: StockTickerAudioInterface): void
+ setOut(out: PrintStream): void
+ setRefreshRate(refreshRate: int): void
+ setRunCount(runCount: int): void
+ showErrorMessage(text: String): void
+ updateData(): void
1
Class to be tested with Mock Objects
1..*
-audioPlayer
-audioPlayer
«interface»
StockTickerAudioInterface
+ playErrorMusic(): void
+ playHappyMusic(): void
+ playSadMusic(): void
StockQuoteAnalyzer
- audioPlayer: StockTickerAudioInterface = null
- currentQuote: StockQuoteInterface = null
- stockQuoteSource: StockQuoteGeneratorInterface = null
- symbol: String
+ getSymbol(): String
+ obtainChangeSincePreviousClose(): double
+ obtainCurrentPrice(): double
+ obtainLastUpdateTimestamp(): String
+ obtainPercentChangeSincePreviousClose(): double
+ obtainPreviousClose(): double
+ obtainStatus(): String
+ playAppropriateAudio(): void
+ refresh(): void
+ StockQuoteAnalyzer(symbol: String, stockQuoteSource: StockQuoteGeneratorInterface, audioPlayer: StockTickerAudioInterface)
«interface»
StockQuoteGeneratorInterface
+ createNewInstance(symbol: String): StockQuoteGeneratorInterface
+ getCurrentQuote(symbol: String): StockQuoteInterface
-stockQuoteSource
Runnable
FCSAPIStockDownloader
- apiKey: String
- keepGoing: boolean = true
- recentQuotes: HashMap<String, StockQuoteInterface> = new HashMap<Str...
- refreshRate: int = 60
+ createNewInstance(symbol: String): StockQuoteGeneratorInterface
+ FCSAPIStockDownloader()
+ getCurrentQuote(symbol: String): StockQuoteInterface
+ run(): void
+ setApiKey(apiKey: String): void
+ stopExecution()
-currentQuote
«interface»
StockQuoteInterface
+ getChange(): double
+ getLastTrade(): double
+ getPreviousClose(): double
+ getSymbol(): String
+ getTimestamp(): String
+ toString(): String
StockQuote
- change: double
- lastTrade: double
- previousClose: double
- symbol: String {readOnly}
- timestamp: String
+ getChange(): double
+ getLastTrade(): double
+ getPreviousClose(): double
+ getSymbol(): String
+ getTimestamp(): String
+ StockQuote(symbol: String, previousClose: double, lastTrade: double, change: double)
+ StockQuote(symbol: String, previousClose: double, lastTrade: double, change: double, timestamp: String)
+ toString(): String
Hashtable
StockTickerListing
- serialVersionUID: long = 1L {readOnly}
- singleton: StockTickerListing
- addNYSESymbols1(): void
- addNYSESymbols2(): void
+ getCompanyName(symbol: String): String
+ getSingleton(): StockTickerListing
+ isValidTickerSymbol(symbol: String): boolean
- StockTickerListing()

Figure 2 UML Class diagram for Stock Quote Analyzer

In developing your test cases, you may see that there are interdependencies between the methods that this class invokes, the data that it stores internally, and previous calls. You will need to take that into account when testing your system. Ideally, you should be able to still use a AAA format, but you may need to be careful about how you structure your tests to comply with the AAA format. You should be able to use data providers for some tests, but not necessarily all tests. Some methods may need independent testing that a data provider may not allow. You will also need to think about what approaches you are going to use to verify correct behavior and how to select stock values that meet the required behaviors. You may find boundary value analysis, equivalence classes, and other techniques to also be useful.

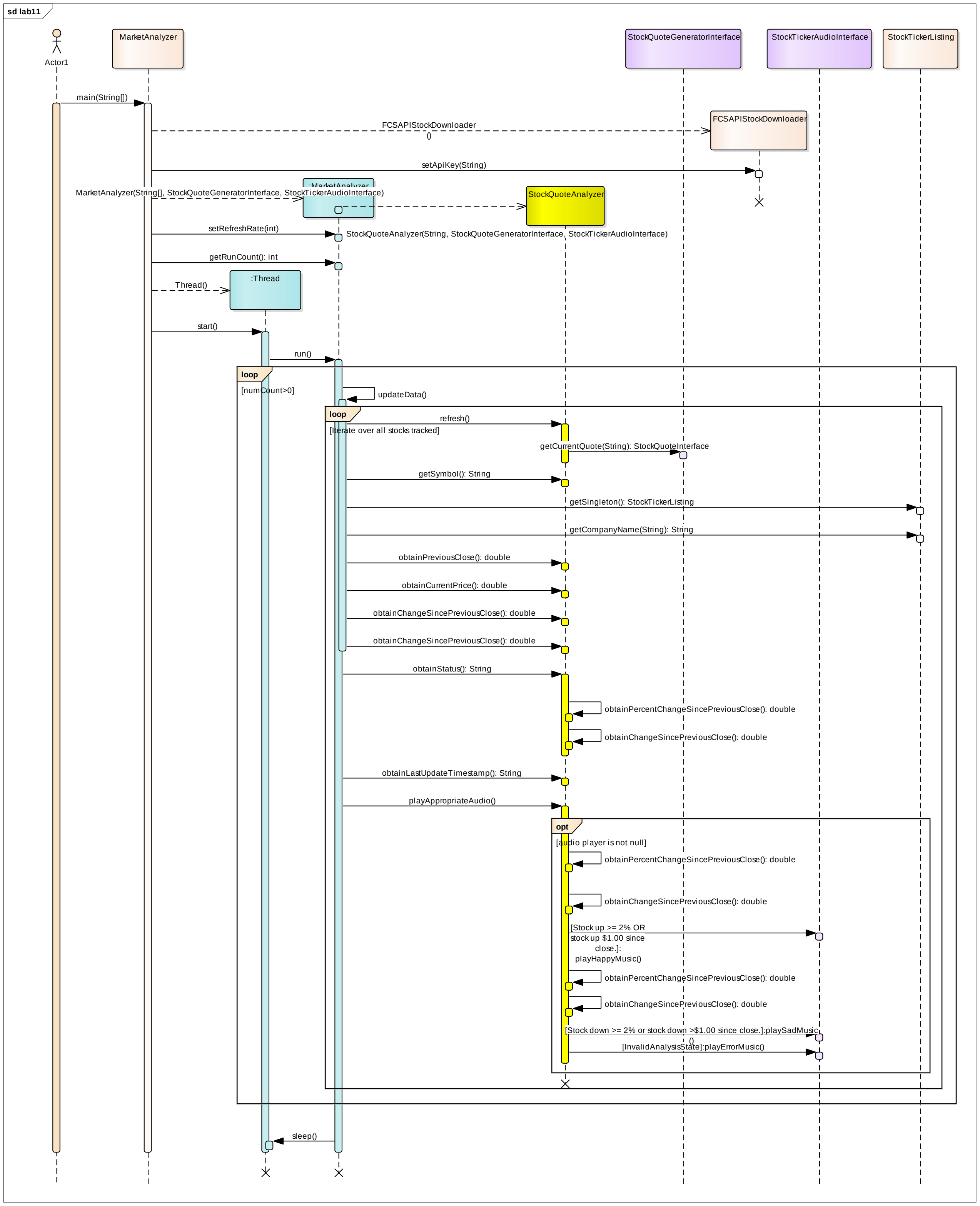
sd lab11
Actor1
MarketAnalyzer
StockQuoteGeneratorInterface
StockTickerAudioInterface
StockTickerListing
main(String[])
FCSAPIStockDownloader
FCSAPIStockDownloader()
setApiKey(String)
:MarketAnalyzer
MarketAnalyzer(String[], StockQuoteGeneratorInterface, StockTickerAudioInterface)
StockQuoteAnalyzer
setRefreshRate(int)
StockQuoteAnalyzer(String, StockQuoteGeneratorInterface, StockTickerAudioInterface)
getRunCount(): int
:Thread
Thread()
start()
run()
loop
[numCount>0]
updateData()
loop
[Iterate over all stocks tracked]
refresh()
getCurrentQuote(String): StockQuoteInterface
getSymbol(): String
getSingleton(): StockTickerListing
getCompanyName(String): String
obtainPreviousClose(): double
obtainCurrentPrice(): double
obtainChangeSincePreviousClose(): double
obtainChangeSincePreviousClose(): double
obtainStatus(): String
obtainPercentChangeSincePreviousClose(): double
obtainChangeSincePreviousClose(): double
obtainLastUpdateTimestamp(): String
playAppropriateAudio()
opt
[audio player is not null]
obtainPercentChangeSincePreviousClose(): double
obtainChangeSincePreviousClose(): double
[Stock up >= 2% OR stock up $1.00 since close.]: playHappyMusic()
obtainPercentChangeSincePreviousClose(): double
obtainChangeSincePreviousClose(): double
[Stock down >= 2% or stock down >$1.00 since close.]:playSadMusic()
[InvalidAnalysisState]:playErrorMusic()
sleep()

Figure 3 Sequence Diagram for Stock Analyzer System.