Stock Mate

David Hedeen, Chris Nathan

SEIS630-01 (Saturday (b)), 5/18/2019

# Project 2 Sprint 2

GitHub: <https://github.com/hedeen/tp_2_stockmate>

SHA-1 ID: XXXXXXXXXXXXXXXXXXX

# How to Run

## Setup

1. Download the gitbub repository at the URI located on the title page.
2. Locate the runtime folder within the project directory (save this location for later)

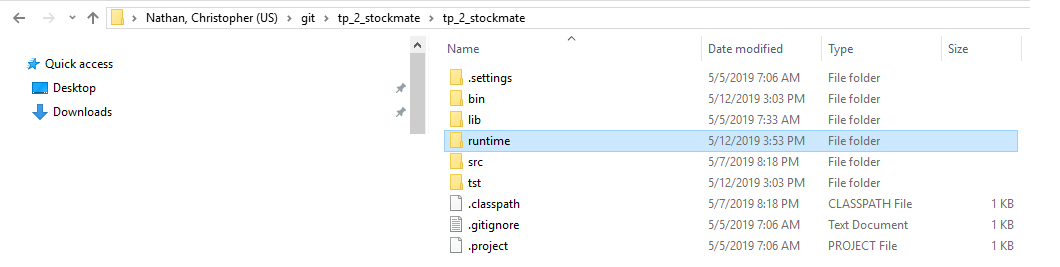


Figure 1: Runtime Folder

1. Create a temporary folder that will be used to save stock files on your local computer. The default setting will use “C:\Stock Mate”, but you may select any valid directly during program operation
   1. Record the temporary folder created
2. Confirm that you have a valid internet connection when running test cases.
   1. The software will poll data from <http://sec.gov> and will not operate correctly without internet connectitivity

## Starting the software

1. Open up a command line utility (cmd.exe) and navigate to the folder shown above (see below for example)

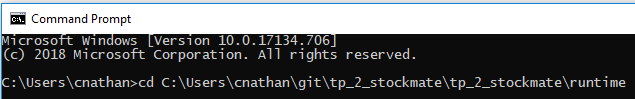


Figure 2: Change Directory to Runtime Folder

1. Enter in the following “java – jar “Stock Mate.jar” to start the program

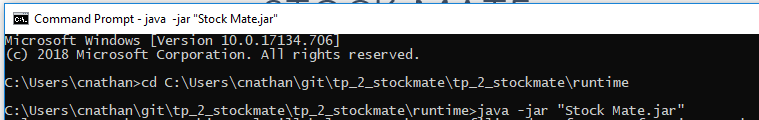


Figure 3: Run .jar File Using JVM

## Running the Software

1. Follow the prompts that appear on the command line utility and enter in your responses. Press the ‘Enter’ button for each response

### Sequence of Operation & Valid Entries

For some questions, a default response is shown. Press enter without typing any characters to select and use the default response.

1. Select how you want to save files
   1. Enter in a 1 or 2 for .txt or .csv files
2. Enter in a directory where you wish to save stock filing reports.

*NOTE: Enter in the directory that you record in the “Setup” section*.

1. Select the stock ticker you wish to retrieve information about
   1. Enter in any valid stock ticker (e.g. “AAPL” or “GOOG”)
2. Wait for the system to return the number of filings detected. If no filings are detected the software will ask for a new stock ticker
3. Select a filing data tag from the list of support options (earnings per share, earnings per share diluted, income)
4. Select whether you would like to return all filings or just the most recent
5. Wait for the software to retrieve the information
6. Read the message prompt to see the file that was created. Locate the file and review its contents.
7. Continue to use the program by selecting a new stock ticker, repeat as desired.

## Running Unit Test & Evaluating Test COverage

Unit testing can be verified using Eclipse IDE. The software was developed using Version: 2018-12 (4.10.0) although other versions are likely compatible. Confirm that you have a valid internet connection when running test cases.

1. Open the project within eclipse and confirm you can view the project directory on the left side (see below)

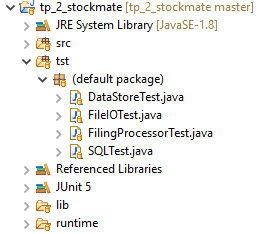
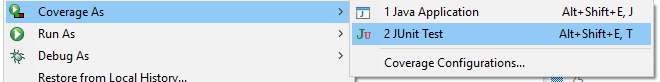


Figure 4: StockMate Project Directory

1. Locate the ‘tst’ folder and right click it
2. Select “Coverage As > JUnit Test” as shown below



1. Wait for the tests to complete and review their success using the test coverage panel and the test detail panel on the left pane.

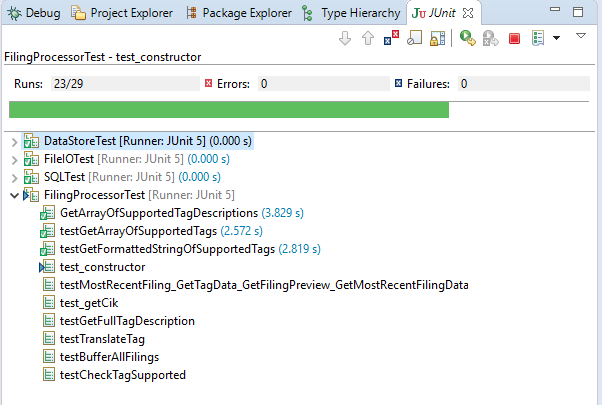


Figure 5: Test Detail Panel

# Domain Class Diagram

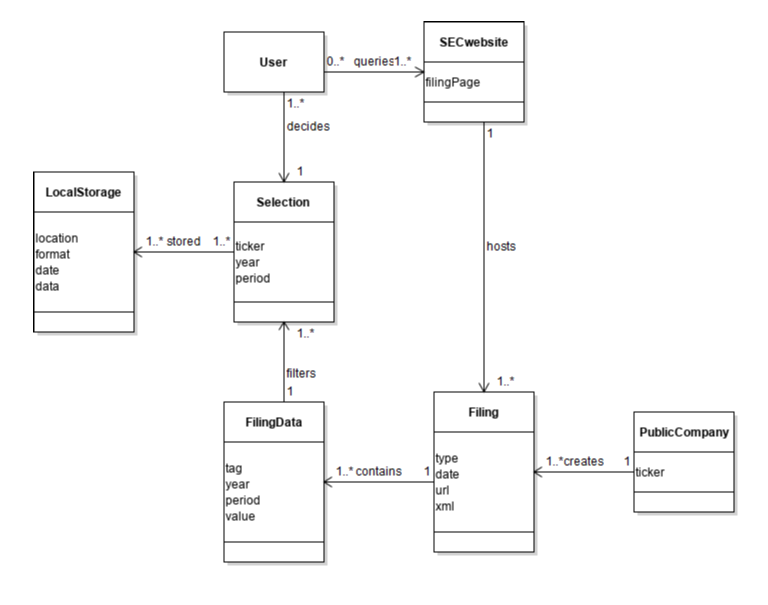


Figure 6: Domain Class Diagram

# Main Success Scenario (UC1 –Retrieve Stock Information and Save)

**Scope:** Stock trader retrieving and saving stock ticker information

**Level:** User goal

**Stakeholders and Interests:**

Stock trader: Desires easily accessible stock information. Desires accurate stock data in readable format.

Stock trader colleagues: Desires easily readable file format that can be exchanged between PCs.

SEC information technology: Wants stock information to be more easily digested by users.

**Primary Actor:** Stock trader

**Preconditions:** Stock trader has a reliable internet connection and can navigate to <https://www.sec.gov/> homepage. Stock trader is working on a PC with sufficient privileges to create and save files.

**Success Guarantee:** Stock trader has produced a readable .csv file that includes stock data related to the requested ticker.

**Main Success Scenario:**

1. SuD asks user how they would like data to be stored
2. User selects “TXT” file exports and the SuD asks the user to enter in the local folder to save stock files.
3. User enters in a filepath
4. SuD confirms the folder is valid, saves the information and asks user which stock ticker they would like to retrieve

[ -- Repeat until user closes application

1. SuD asks the user which stock ticker there are interested in
2. User enters in a stock ticker
3. SuD returns number of filings present, outputs available (supported) stock tags (earnings per share, income, etc.), and asks which information they would like to retrieve.
4. User enters in a stock tag selection
5. SuD retrieves information from sec.gov website, displays a preview of the data and saves the stock information to the selected local folder.

--- ]

**Extensions:**

1a. User closes the console application (they have no more stocks that they are interested in)

4a. Local folder is invalid

7a. HTTP request to sec.gov fails

9a. HTTP request to sec.gov fails

# External System Events

Two external system events are defined/modeled below. Additional system events can be seen in the SSD

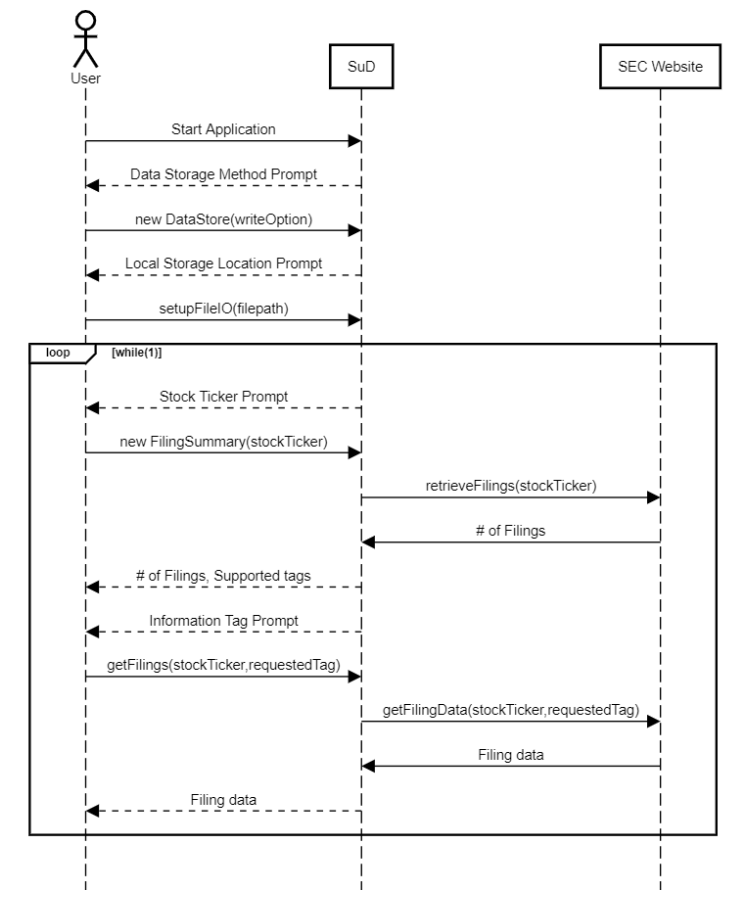
User Requests a Directory for Saving Files

checkDirectory(String dir) returns true/false for valid directory

User Requests a Stock Ticker:

getFilings(String ticker, Int requestedTag) returns integer for # of filings located

# System Sequence Diagram (SSD)



# UML Class Diagram

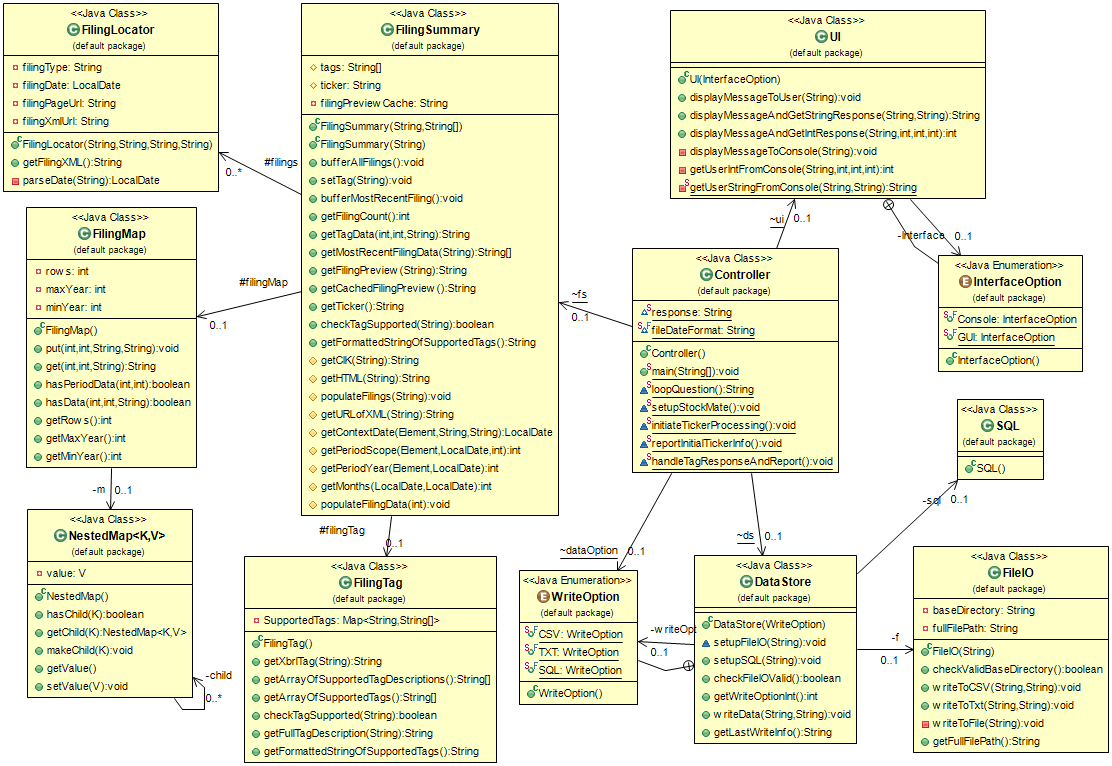


Figure 7: UML Class Diagram

# GRASP

One of the GRASP heuristics used in developing StockMATE was the principle of Low Coupling. Coupling is the concept of how strongly an element depends on other elements of the design. Coupling also infers the interconnectedness of objects and even awareness of them. A software package that has many highly coupled, intertwined objects would be very difficult to change; since small adjustments could ripple through the entire design and could also lead to unforeseen bugs and broken functionality. The ideal is design with Low Coupling Pattern, so elements can freely act independent of one another. This also affords allows better understanding of the intended purpose of the elements, as well as gives option for reusability across the same or perhaps entirely different projects.

StockMATE strives to adapt the Low Coupling pattern across its design. For instance, the NestedMap Class has zero concept of almost every other class in the design, save FilingMap. FilingMap implements the NestedMap for the ease of storing and retrieving filing data, but no other classes know what NestedMap is. NestedMap could be altered, perhaps at a future date it is decided that a different approach for storing the Filing data is preferred. The only other class that would have to change would be FilingMap. NestedMap could also be brought into another, completely different project and reused in its entirety.

Not all elements in the StockMATE design have this single relationship with another class. Larman recognizes as an element as having “too many” relationships context dependent. As an example, the class with the highest number of the relationships in StockMATE is the Controller class. However, this is intentional by design, actually following another GRASP concept: Controller. Where this class receives and coordinates the system operation of the major subsystems. By pure virtue of this pattern, this class must reach out, connect the major players, and pass messages to accomplish the desired use cases. The Controller class in StockMATE is the root object, the conductor of all the moving parts between the abstracted methods and passing data.

An overemphasis of low coupling can lead to small number of classes that do too much. This steps on the toes of another GRASP heuristic: high cohesion. High cohesion is the patter to focus object design into manageable and understandable roles. Recognizing this with StockMATE some late development adjustments were made to tone down the activities of relatively large class: FilingSummary. Early design concepts had this class owning the responsibility of assigning filing and decoding filing tags. This was done in efforts to apply the low coupling effort. However, a balancing act must be achieved between low coupling and high cohesion. By requiring FilingSummary the responsibility of managing filing tags in addition to summarizing the data of the filings was simply too much, so it was decided to separate the two responsibilities into two different classes. Thus, achieving the goal of low coupling while maintaining high cohesion.

Good software design needs to have a global conscience of all GRASP concepts in implementation. Some of these patterns were discussed here, however, the focus for StockMATE’s was low coupling, high cohesion and controller implementation.