ECE 421 Project 1: Stock Market Monitor

Winter 2023

Qi Zhou
Tianyuan Fang
Jakob Lau

Design Rationale:

This application is designed to retrieve the latest stock shares data for the current trading day by using Alpha Vantage API then return the stock with the highest price, with a value less than 500. We will need to modify the code for APIFinance to account for the limitation of 5 API requests per minute and 500 requests per day. The original imperative code used for selecting high shares will be refactored to a functional style.

In the design of the program, we will be rewriting the PickShareImperative class to create a new class called pickShareFunctional. This class will feature a static method called findHighPriced, which takes a stream of strings as input. By utilizing the JDK's specialized functional-style method, we can more easily compare and filter the data.

To streamline the process, we first use the map() method to convert the input strings into ShareInfo objects, which are easier to work with for further comparison and filtering purposes. Next, we use the filter() method to eliminate shares with a price above 500 dollars. Finally, we utilize the max() method to identify the share with the highest price among the filtered results.

APIFinance class was modified to handle the API request restriction. As the free API key permits only 5 requests per minute and 500 requests per day, the program will pause for one minute when the API returns a limit warning. After the pause, the program will make a re-request. If the second request receives a limit warning again, the 500 requests per day limit has been reached. The restriction is also handled when parallelstream is used. Semaphore is used to ensure that there can only be 5 threads waiting to send the re-request at any given time. Also, no other thread can get into the function when there is at least one thread waiting to send the re-request. This makes the program thread safe.

Testing

Out of the three steps in part a, "creating a list of ShareInfo" should cost most of the execution time, since it contains the API calls, which is slower than other local operations.

	5 Requests	**10 Requests**
stream	1087 milliseconds	61798 milliseconds
parallelstream	674 milliseconds	61006 milliseconds

The difference in execution time between stream and parallelstream is obvious under 5 requests. The difference is less noticeable under 10 requests due to the 60 seconds delay caused by the API limit.

The parallelstream frees up the cpu when one thread is waiting for the response from API, which gives other threads the chance to do work. This leads better cpu utilization and shorter execution time. Whereas, in stream, the requests are performed sequentially. Therefore, the cpu time is wasted when the program is wait for a response from API.

We also tested the program with more than 10 requests including invalid requests. Overall, the program behaved as expected in both stream and parallel stream. It appears to be thread safe based on our test results.

Test ID	Description	expected	Pass
1	Test with share symbol list that include invalid symbol using parallel stream	program should display warning message for invalid share symbol	Yes

2	Test with share symbol list that include 501 shares using parallel stream	program should display daily limit exceeding warning at the end	Yes
3	Test with share symbol list that include invalid symbol using stream()	program should display warning message for invalid share symbol	Yes
4	Test with share symbol list that include 501 shares using stream()	program should display daily limit exceeding warning at the end	Yes

Defects

Due to the API limitation, the programs run rather slowly when there are more than 5 requests. This problem cannot be resolved unless we upgrade our API access.