ECE 421 Project 1: Stock Market Monitor

Please note all projects are undertaken as GROUPS! Groups are normally of size 3. All members of the group will receive the same grade.

Consider the following Java code segment:

- Shares.java

```
public static final List<String> symbols=
Arrays.asList("IBM","AAPL","AMZN","CSCO","SNE","GOOG","MSFT","ORCL","FB","
VRSN");
```

- APIFinance.java

The class utilizes the Alpha Vantage API to provide stock data for a list of shares. The API returns the latest data points from the current trading day. To use the API, you need to obtain a free API key from https://www.alphavantage.co/support/#api-key

```
private static final String BASE_URL = "https://www.alphavantage.co/query?";
private final static String apiKey = "YOUR_API_KEY_HERE";

public static BigDecimal getPrice(final String symbol) {
        BigDecimal price = new BigDecimal(0);
        try {
                URL url = new URL(BASE_URL +
"function=GLOBAL_QUOTE&symbol=" + symbol + "&apikey=" + apiKey);
                URLConnection connection = url.openConnection();
                InputStreamReader inputStream = new
InputStreamReader(connection.getInputStream(), "UTF-8");
                BufferedReader bufferedReader = new
BufferedReader(inputStream);
                String line;
                while ((line = bufferedReader.readLine()) != null) {
                        if (line.contains("price")) {
                                price = new BigDecimal(line.split("\"")[3].trim());
                        }
                }
        bufferedReader.close();
        } catch (IOException e) {
                System.out.println("failure sending request");
```

```
        }
        return price;
}
```

- ShareInfo.java

```
public final String symbol;
public final BigDecimal price;

public ShareInfo (final String theSymbol, final BigDecimal thePrice) {
        symbol = theSymbol;
        price = thePrice;
}
public String toString() {
        return String.format("symbol: %s price: %g", symbol, price);
}
```

- ShareUtil.java

```
public static ShareInfo getPrice(final String symbol) {
    return new ShareInfo (symbol, APIFinance.getPrice(symbol));
}
public static Predicate<ShareInfo> isPriceLessThan(final int price) {
    return shareInfo -> shareInfo.price.compareTo(BigDecimal.valueOf(price)) < 0;
}
public static ShareInfo pickHigh(final ShareInfo share1, final ShareInfo share2) {
    return share1.price.compareTo(share2.price) > 0 ? share1: share2;
}
```

- PickShareImperative.java

```
final Predicate isPriceLessThan500 = ShareUtil.isPriceLessThan(500);
        for(String symbol : Shares.symbols) {
            ShareInfo shareInfo = ShareUtil.getPrice(symbol);
            if(isPriceLessThan500.test(shareInfo))
                    highPriced = ShareUtil.pickHigh(highPriced, shareInfo);
        }
        System.out.println("High priced under $500 is " + highPriced);
```

The previous code simply compares some shares' prices and returns the stock with the highest price whose value is less than 500$. However, the code uses an imperative style.

As you can see, the code uses mutating variables. Furthermore, if we want to change the logic a little (i.e., pick the share with the highest price under $1,000), we will have to modify this code. This makes the code not reusable.

a- Rewrite the previous code in a functional style. Functional programming was added to Java in version 8. It is now a mainstream practice. In this project, you will be required to demonstrate that you've mastered this aspect of Java programming. Remember, Java programming was covered in ECE 325 and CMPUT 301. For reference, the following article provides a good overview of functional style in Java: https://www.javaworld.com/article/3314640/functional-programming-for-java-developers-part-1.html

You should create a class named *PickShareFunctional* in which you will define one method called *findHighPrices*.

To help you, this is how you should call *findHighPrices* method:

```
findHighPriced(Shares.symbols.stream());
```

As you can see, converting symbols from **List** to a **Stream** of symbols will allow you to use the JDK specialized functional-style methods (i.e., *map*, *reduce (fold)*, and *filter*). ***Hint***: In this method, you should do the following steps:

1- Create a list of *ShareInfo* filled with the price for each of the symbols in *Shares*
2- Trim down this list to a list of shares whose prices under $500
3- Return the highest-priced share.

Think Functional!

b- Calculate the execution time for the *findHighPriced* method. i.e., try to use timing methods in java before your call to *findHighPriced* methods.
What is the execution time for the *findHighPriced* method?

Out of the three steps explained above in part (a), which step do you think is responsible for most of this time?

c- Change the way you are calling the *findHighPriced* method as:

```
findHighPriced(Shares.symbols.parallelStream());
```

Calculate the execution time as in part (b) again, but now with parallelStream() instead of stream(). Explain why the execution times are different?

The system must be completed by **Tuesday 4th February @ 5:00 p.m.** You are required to hand-in:

1) A detailed rationale for your augmented decisions with regard to the above design questions.
2) A copy of the code
3) A description of any additional testing beyond that described by your contracts.
4) A list of known errors, faults, defects, missing functionality, etc ….telling us about your system's limitations will score better than letting us find them!

Please hand in all components by submitting via eclass to the group account, and hence all sub-components, by definition, must be machine-readable. Also, file types should be only .pdf or zipped files for your Java project.