CIS 3515 Lab
Worksheet 9

An application that will allow a user to search for and store a list of stocks, called their portfolio, for which they'd like to keep track. The application should use the standard dual-pane display mode, with a list displayed in one pane (Portfolio pane), and details displayed in the other (Stock Details pane). Large screens and landscape layouts should display both panes simultaneously (dual-pane mode), while smaller screens in portrait mode should display one pane at a time (single-pane mode) starting with the navigation pane.

A user must be able to:

- Add a stock to their portfolio using the stock's ticker symbol (e.g. GOOG for Alphabet)
    - The Add function must be initiated from a **Floating Action Button** (FAB) (https://material.io/guidelines/components/buttons-floating-action-button.html - https://developer.android.com/reference/android/support/design/widget/FloatingActionButton.html)
    - The search interface should be presented to the user as an activity *ideally* formatted as a windowed dialog. The dialog will have an EditText to capture the stock symbol as well as 2 buttons to Add and Cancel respectively
    - If the user attempts to enter a stock ticker symbol that doesn't exist, they should be shown an error message using a Toast
    - The search FAB must always be present regardless of if the device is displaying single or dual-pane mode.
- Have their portfolio persist across application restarts (the data must be saved to storage)
- View details of a stock when the user clicks on the stock from the list (see below)

The main interface of the application must implement fragments. The portfolio pane will display a list of stocks that the user has previously saved to their portfolio. It must display the ticker symbol and current price for each stock. Additionally, each stock ticker's view must either have a green or red background depending on if the stock price is higher or lower for the day respectively (a stock that is the same as it's opening price should be green).
They should be displayed in a ListView (or similar adapter view that allows seeing multiple stocks at once) that displays the stock symbol for each stock saved in the portfolio.

Selecting a stock from the portfolio pane will display details of that stock in the stock details pane. Those details must (at a minimum) include:

- The Company Name (see appendix ii)
- The 1 day stock chart (see appendix ii)
- The current stock price (see appendix ii) (big, bold font size)
- The opening stock price (see appendix ii)

When your application is running, it should continuously update all the stocks stored in a portfolio, even if a user is not currently looking at that stock. This must be done with a Worker Thread. The thread should be started once the application begins and continuously update the stock price in the background (**update each stock every 1 minute or other reasonable interval**). When the thread receives an updated stock price, it should save the information to a file (portfolio_file). ***Nothing Else***.

NOTE A regular ol' worker thread exists past the lifecycle of the activity that created it, and holds an implicit reference to said activity. This means that whenever your activity restarts (for example, when the device is rotated from Portrait to Landscape, triggering a configuration change) the "old thread" is still running. Additionally, threads that never stop running are never garbage collected. These issues lead to the dreaded **Memory Leak!** You must ensure that any running thread is either killed and restarted whenever such events occur, or ensure that only one instance of your thread is ever running (Investigate private static threads). Regardless of your chosen approach, ensure that background threads are stopped entirely when your application exits (FYI, calling **stop()** on a thread has been deprecated since API Version 1 [it has always been a bad idea], so come up with another solution).

Whenever the worker thread updates portfolio_file, your application needs to react to this update by

a) displaying the updated stock information in the portfolio pane (updated colors and all), and
b) display updated stock information in the stock details pane if the stock being viewed has updated information

The be notified of changes to portfolio_file (remember, the worker thread only fetches updated portfolio information form the web and updates the file, it doesn't directly notify anyone), your activity must monitor it for changes using a FileObserver object (https://developer.android.com/reference/android/os/FileObserver).

Whenever the user selects a stock from the portfolio pane, the activity should retrieve information about that stock from portfolio_file, and thereafter provide that information to the stock details pane. This way the user will always see the stock's information instantly instead of having to fetch updated information from the web directly. Remember that as portfolio_file is updated by the worker thread, you must also have the stock details pane continuously show updated information for the displayed stock as well.
The only data the stock details fragment will not get directly from its parent activity directly is the stock chart. Instead, it should only receive the stock ticker (or the full stock image URL) which it will then use to fetch the stock chart and update its ImageView (see appendix ii).

Upon launch, if the application does not have any stocks saved, it should display a message in the navigation fragment instructing the user to add a stock. The message **must** be fixed (For instance, a textview inside the navigation fragment. You must **not** use a temporary message notification mechanism such as a Toast).

All string displayed to the user must be retrieved from a string resource. Additionally, there must be an additional string resource file in a non-English language of your choice.

Please see the appendix II for the APIs to obtain all required information.

Appendix I

IMPORTANT NOTE:

Using the right components and the correct design patterns to implement each requirement is paramount. It is not sufficient to simply get a feature working; proper use of the framework is required for full marks to be awarded. Refer to your notes and other course resources to determine the right component to do a particular job and how to use them (e.g. Custom adapters and adapter views. Fragment reuse, etc.)

Appendix II

Useful Third Party APIs:

Stock information (JSON):
http://dev.markitondemand.com/MODApis/Api/v2/Quote/json/?symbol=<*stock_symbol*>
replace <stock_symbol> with a the symbol of the stock for which you would like information
e.g. http://dev.markitondemand.com/MODApis/Api/v2/Quote/json/?symbol=*goog*

See more information on the JSON format here: http://www.json.org/ and here:
https://developer.android.com/reference/org/json/JSONObject.html

Stock Chart (HTML):
https://macc.io/lab/cis3515/?symbol=<stock_symbol>

This URL will return a chart inside an HTML document that should be used inside a WebView. In order for the chart to be displayed, the webview object must have JavaScript enabled. Example:

```
WebView webView = findViewById(R.id.webView);
webView.getSettings().setJavaScriptEnabled(true);
webView.loadUrl("https://macc.io/lab/cis3515/?symbol=goog");
```

You can also append the variables **width** and **height** to adjust the image size. Example:

String url = `https://macc.io/lab/cis3515/?symbol=goog&width=400&height=200`

Extra API:
If you would like to allow the user to add a stock to their portfolio by searching for the company by name rather than by stock symbol (which they may not know), you can use the API below. **Note - This is not a requirement of the application**:

Company Search (JSON):
http://dev.markitondemand.com/MODApis/Api/v2/Lookup/json?input=<query_string>