FIT3077 Assignment 2 – Stage 2

David San (27311589) and Wilbert Ongosari (26378078)

Note: Please refer to the readme.md file for in depth explanation of our UML diagram design.

The Design Patterns that we use were:

- Observer pattern
    - The design we made for our system is that of an observer pattern. It is used for click events such as monitoredStockManager that implements buttonPressObserver and monitored StockUpdater that implements the observer. The observers are made to separate the function of when the button is clicked and its display.
- Model view controller (MVC) pattern
    - We used the MVC pattern to solve tight coupling to have more control and because it does not use a view state, it reduces requests to the web service. Having MVC means that we have control over the model, view and controller separately. Such as monitoredStockManager which controls the response towards the user input whether to return the normal window or the graphical window. The other two graphUpdater and monitoredStockUpdater are controllers to update the graphical_Monitor view and normal monitor view in response to stock changes. While the view components serve as view type classes. Thus, model manages the stock data of the application, while view displays the information and controller response to the user input.

The Design Principles:

- The Open/Closed Principle (OCP)
    - The classes we made follows this principle where it can be modified by adding new functionality and is open for future extension. As both the monitor and stock have its own interfaces. Therefore, we can modify its behaviour without touching its source code.

- The Liskov Substitution Principle (LSP)
    - This principle says that a code should be able to perform itself without knowing its actual class object. As we made the new monitor graph, that does not change the properties of its inheritance. So that in the future if we wanted to make another type of monitor, we won't have to modify the existing code.
- The Dependency Inversion Principle (DIP)
    - This principal was used because with the addition of an abstact classes, it will reduce dependencies between classes. To avoid highly coupled distribution and increases re-usability in classes.
- The Don't Repeat Yourself Principle (DRY)
    - Code repetition were avoided by making more classes to maintain new different functionalities. This reduces redundancy and increases maintainability as modifications are made in a different place. For instance, the normal monitor and graph monitor originally have the same code with the same functionality but was then reconfigured into the class monitoredStockManager where it is handled and both its functionality is separated resulting in reducing code repetition.
- The Interface Segregation Principle (ISP)
    - We made this application based on this principal because we know that our module will be used by several clients. Thus, each different client accessing this application will have its own individual interface by implementing multiple inheritance. This is made so that any changes made by a client won't affect the other clients, in other words it removes dependencies between clients.

References:

https://www.c-sharpcorner.com/blogs/why-mvc-is-better-than-the-web-form1

https://www.tomdalling.com/blog/software-design/solid-class-design-the-liskov-substitution-principle/

https://en.wikipedia.org/wiki/SOLID