REFLECTION

For our third programming assignment, I learned what delegates were, how to implement events and event handlers, in addition to how threads can be used for the first time and how to apply these in C#. In C#, delegates are a data type which can be used as references to methods which have the same signature -- that is, any method whose parameters are of the same data type and whose return value is also of the same type. For this reason, we can use delegates to change which methods will be called, combine methods or connect two classes in this way. I also learned that there are specific kinds of delegates, like the built-in Event Handler delegate supported by .NET which, like delegates, can be set to multiple different methods for handling events. These Event Handler delegates usually take in two parameters, the object of interest and an EventArgs type object. This EventArgs type object is passed to the Event Handler in addition to the object involved in the event which occurred as a way to pass on any information to those classes which are registered to be notified when this event occurs. If no information needs to be passed on to any outside classes, we can choose to pass this Event Handler delegate “EventArgs.Empty.” Otherwise, as in the case of our lab assignment, we would have to create a class dedicated for this event to extend or derive the EventArgs class in order to be able to support or send more than one data value to its subscribers -- in our case, this was our StockNotification class.

In order to send this data or notify its subscribers of an event, the subscriber class must subscribe, or “register” to this event somehow. This is done by setting the event handler delegate member of the object of interest to the subscriber’s event-handling method -- we can do this because the Stock’s Event Handler delegate method and the StockBroker’s event-handling method both take in the same types of parameters and return the same type. Here, for our programming assignment, the Stock Broker represents the subscriber and the stock’s event, StockValueChanged is the event the subscriber is interested in, so under the StockBroker’s AddStock() method, we decided it would make sense that whenever a StockBroker adds a stock to a list of stocks he or she owns, they will want to be notified when the Stock’s value has changed and so it is here that the stock’s Event Handler delegate is set to the subscriber’s event-handling method and the StockBroker is successfully “registered” to be notified when this event occurs through this way.

We also learned how to “raise” and “consume” an event, or a class of the EventArgs type, our StockNotification class. To raise an event, after an event occurs or happens to an object (once the value of a stock changes in our case) we call the object’s protected and virtual method (usually with a prefix of “On”) after the process is finished. Here is where we “raise” the event as the method is what invokes the Stock’s Event Handler delegate, “StockValueChanged.” Once the delegate is invoked, any class who has “registered” to this Event-Handler delegate will be “notified” and their respective methods will be called as well.

% of CONTRIBUTIONS:

- Jerry Belmonte - 100%