SE2: Software Design and Architecture
Winter 2017

Design Patterns Presentation – Observer Pattern

Team: JET

Team Members:
Eugene Lee, 20484586
Tim Silva, 20312737
Jake Stowe, 20382246
Ivan Teo, 20700986

Jake:

The Observer pattern is a one-to-many design pattern where an object called the subject is followed by many objects which we call observers. It is usually very easy for an observer to follow or unfollow which often creates an ever changing list of observers that the subject has to keep track of.

The Observer pattern can either be run by a push protocol or a pull protocol. The push protocol requires the subject to send a message to all the observers about any changes made. An example of this is when your operating system on your computer, phone or tablet gets updated. In this case you are the observer and when there is a new patch they let you know so that you can get the update. The pull protocol requires the observers to check to see if the subject has updated. An example of this is most types of clocks. The time is always changing but you will only know the correct time when you check it. Could you imagine if you had a clock that informed you every time it changed? It would never stop!

Tim:

This pattern is very easy to visualize. You have the subject, which maintains a list of observers. The subject will typically have a "register" and "unregister" method. If another object wants to become an observer, it calls the subject's register method, passing itself as a parameter, and the subject adds it to the list of observers. Likewise, it can call the unregister method and the subject will remove it from the list.

In the push protocol, the observers typically have a method called "notify." When the subscribed event occurs, the subject can simply loop through the list of observers and call their notify method. In the pull protocol, the subject might have a method like "checkForUpdate" or "getValue", which the observers can routinely call.

The great thing is that the subject and its observers only communicate with each other through these methods. Each of the observers could be a different class, and they could each have a notify method that does something completely different; the subject doesn't need to know how they work, and the observers don't need to know how the subject works.

Eugene:

Example: Piazza post!

In this activity, I will make a new thread on piazza called “Observer example”, and during our presentation we will ask our classmates to go on piazza, find this thread, and click “Follow” on the top right corner. Then, I will make a comment saying “If you get a notification, you are a observer!”. This will generate notifications to whoever is following the thread, making this an example of the Observer design pattern.

Ivan:

NFPs

- Flexibility - Adding observers is possible and expected by design, even after it is deployed. Subject would not be affected by the addition of observers.
- Resilient – The subject does not have to worry about the unavailability of the observers as observers are not generally expected to do any work on part of the subject.

Strengths and weaknesses:

Strengths:

- Supports the principle to strive for loosely coupled designs between objects that interact – Subject do not need to know about the Observer classes
- Allows you to send data to many other objects in a very efficient manner.
- No modification is needed to be done to the subject when adding new observers.
- Subject can add and remove observers at any point of time.

Weaknesses:

- Sending a message has a cost implied by number of observers, i.e. n observers take n time and messages (at least in a simplistic/naive implementation)
- The order of Observer notifications is undependable as events and interactions among objects become more complex, the assumption that observer equality becomes increasingly difficult to uphold. Among the observers for a subject, the observers should be treated equally. The moment a subject provides preferential treatment for a particular observer, it becomes highly-coupled with the observer. And this defeats the purpose of the observer pattern