Brendan Kay and Gavin Forsberg

Design Patterns Project: Written Report

CIS445: 12/11/2020

The Observer Pattern

**Observer Pattern Overview:**

The design pattern we chose for the final project is the observer pattern. The observer pattern is a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically. It provides an object design where subjects and observers are loosely coupled. The best way to think about this pattern is like a newspaper. Newspapers have publishers that push out data and subscribers who want to receive updates on new data. People can subscribe and unsubscribe to the newspaper whenever they’d like. In relation to the observer pattern, the subject is equivalent to the newspaper publisher and the observers are the newspaper subscribers.

**What We Learned:**

In this project we learned a new approach to object oriented programming, loose coupling of data structures, and group coding dynamics. Some real-life situations we learned that the observer pattern is used in include social media (this is why we chose do build a very simple, small scale social media representation of the pattern), email subscriptions, and app stores with their updates for application users. Even the event listener in Java is modeled after this pattern — the button is the subject and the onClickListener is the observer. Many modern programming languages implement built-in event constructs, i.e., implementing the observer pattern in background.

The subject and the observer are loosely coupled. This basically means that the observer implements a concrete class, but the subject just needs to be aware of that. It does not need to know everything about the class. Due to the loosely coupled nature of the subject and observers, it makes adding, removing, and replacing of observers fairly easy and the subject will not need any modification to add them. Furthermore, they can each be reused independently of each other.

Group coding is something that each of us will probably become a part of in the future, so it is good to find good dynamics and software that assist in this. GitHub is one of the platforms we used heavily throughout the project. Mostly, we used GitHub to store all of our project files, and took use of the issues, labels, and milestones to keep track of our progress, assigning components of the project, and keep deadlines. It was useful to comment on issues and tag them with labels. Tracking changes to the documents and code was also beneficial with each new commit. For the actual coding of the project, we were in Zoom talking while using VS Code’s extension for live coding sessions. This way we could talk and edit the code simultaneously on a platform that incorporates GitHub.

**Our Observer Application:**

Our concept is basically that we have created a very basic ‘social media’ platform, in order to replicate the observer design pattern. The platform consists of 2 friends, Alice and Bob, who receive updates about the main character’s status. Bob has been designated a ‘best friend,’ while Alice was left as a normal friend. This means that they should not receive the exact same updates. Bob may get more updates than Alice but never less. Due to the nature of the pattern, many more observers could be added and removing any of them could be done without trouble.

**Applicability and Faults:**

This design pattern is great for spreading information to different sources utilizing loosely linked objects, but we cannot justify our implementation for an actual social media site to use. This is not the best pattern to use for this specific project due to the fact that new classes are required for each person in the friend circle, it includes only one-way communication from the main character, friends cannot connect to each other, and finally, each observer would have to be “Persons.” The perspective is quite interesting but relatively useless without adding more functionality to it, which was not required to showcase the concept. Once we decided to use Java as the language for our project, we did not encounter very many problems. Originally, when we planned to use C++, one of the biggest initial issues and deterrent to the language was setting up an interface. Having experience with this in Java, the program came together much easier.

We did not necessarily encounter the following issues but were noted in the research stage of the project. The observer pattern can cause memory leaks. This is known as a lapsed listener problem and can occur when the observer fails to unsubscribe the subject who no longer wants to subscribe. There are strong references between the observers and the subject and therefore can be kept alive when attempting to unsubscribe. If the subject keeps these references to the observer, it can prevent garbage collection from cleaning them up. These memory leaks can mostly be prevented by using weaker references to observers.

**Contributions:**

We each contributed to the project equally. We did individual research and came together to write the code for the project. Brendan handled the creation of the presentation and Gavin finished the written report.

**Research Citations:**

Kumar, Sulabh. (14 February 2019) *Observer Pattern.* Geeks for Geeks. https://www.geeksfor geeks.org/observer-pattern-set-1-introduction/.

Nehra, Mahipal. (23 April 2020) *Design Patterns: A quick guide to Observer pattern in Java.* Decipher Zone. https://www.decipherzone.com/blog-detail/design-pattern-observer-java.