# 6 Lab 6: The Observer Pattern

## 6.1 Reflection on my Learning on the Observer Pattern

This practical was on design patterns with respect to the observer pattern. This observer pattern tries to address the issue where classes can be too dependent on each other. These systems are what are known as tightly coupled systems. Consequently, by implementing the observer pattern it allows us to have a one to many relationships between classes by keeping a track of an observers list where the observers are notified if the state changes. In a nutshell design patterns just provide a way to solve a software design issue that occurs regularly. These patterns have been established time and again, so we know that they will work when we implement them and don't have to waste time trying to discover our own. The observer pattern when the Person is notified there is a change of state it broadcasts that change to all the observers in the list. It can be known as multicasting once the observers have received the state change they know about the updated state. This can be difficult to handle in large systems as the change cascades though the observers list and their dependant's bugs are easy to introduce so care must be taken. This is driven by events and has a somewhat publisher subscriber feel to it in the way the changes manifest in the system.

There are two basic models of the observer pattern. The first is the push model I implemented this as a first step in the practical. The idea is that the person pushes the changed information to the observers list. However, sending all the changed information may not be a good solution as I suppose some of the observers may not be affected by the change. Afterward I then change the code to implement the pull model, personally I found the pull model more intuitive and easier to implement even though the push model is theoretically the simplest method. In comparison the pull method tells the observers list a change has occurred. It is then up to the observers to then ask what has changed within the system and update the changes as needed.

There are a few matters to consider when implementing this pattern first we need a way to have a map of the interactions between subject -> observers we can do this by creating a list of observers in the Alarm Clock class. This holds a reference to all the observer objects that will need to be updated. We need to also ensure that the observers are correctly detached from the subject to ensure that the observer can be collected by the garbage collection when not needed anymore. It is possible to observer multiple subjects in this case the observer needs to know where the update is coming from, a way that we can achieve this is to pass a refence to itself with the update message. Message delivery is not ensured by this pattern. The observer pattern plays a key role in the model view controller pattern which is utilized in many GUI based designs. Subsequently, it is a key architectural pattern utilized in many systems and libraries.