ASSIGNMENT 8: DESIGN PATTERNS

EMAIL: GREGOIRE.COUSIN@UCDCONNECT.IE;
GREGOIRE COUSIN STUDENT NUMBER: 18204188; COMP 47480 PRACTICAL

INTRODUCTION TO DESIGN PATTERNS

Before we dive into the characteristics of observer or strategy patterns we need to understand a lot more about the importance of design patterns, what they are and what they are used for. Design patterns is a template for how to solve a particular problem. A design pattern can be used in many different situations and there are usually set rules for each one of them. It uses common code practices such as Inheritance, Encapsulation, Instances a local and external variables to accomplish its particular goal. The most common design patterns include the abstract factory, singleton, strategy, observer and template methods.

OBSERVER

As an introduction to one of the most common design patterns, we will mostly talk about the observer and strategy patterns. The observer pattern is defined as one to many dependencies between objects that said that every time an object changes its state all dependencies are notified and automatically updated. The observer pattern uses encapsulation to produce this task. It encapsulated the core elements based on a subject abstraction and its variable components in an observer based system. This is closely related to the MVC idea. This pattern is mostly used when an application need to scale more than it already is. Let's take for instance a bank, a mailing system in which a banker is responsible for and a customer. The bank holds the customers money however the customer still needs to have some sort of control knowing how much money he has in his account since he buys mostly online. The bank account is linked to another few family members who also need to know this as fast as possible. This is where we have a mailing system that checks when a transaction is being processed from the bank (in this case our interface that sits between the subject and the customer) the mailing system gathers the data from the bank and updates all the family members. This would be the same things in design pattern. Where you would have to have a subject and interface which gathers the information form the client and informs the customers how much money they've taken out to be taken out and then also updates the other family members.

STRATEGY

The other commonly used pattern is the Strategy Pattern. This describes a family of algorithms, encapsulates each one and and makes them interchangeable. This makes it so the algorithm at use varies independently from its client. If we have an open closed principle issue with our program, this pattern would help fix a lot of issues since its main focus is isolation of different components. The strategy pattern helps figure out what behaviors are meant to be used from derived classes separates them. However, we sometimes want to come out with a goal that are shared between these different behaviors. Let's take for instance a dog and a bird. The dog does not have the ability to fly but is able to walk while the bird can do both. In this case it would be great to create a flies interface that are both connected to two other other classes a can't fly and a can fly class. Since our client is the animal in this case we can dynamically change the animals behavior by connecting it to the interface as an instance variable. In this case we can now create many different types of animals without affecting any sub classes. It becomes very interesting when you are able to mix both the strategy patten and the observer. In out case if a bird decides that it needs to start flying the animal is notified creating a mechanism that is based on how everything works around us.