**Observer Pattern:**

Defines one to many dependency between objects so that when one object changes state all it's dependents are notified and updated automatically.

There are 2 methods of passing the data from the subject to the observer when the state is being changed in the subject side:

* Push model- The subjects send detailed information about the change to the observer whether it uses it or not. Because the subject needs to send the detailed information to the observer this might be inefficient when a large amount of data needs to be sent and it is not used. Another approach would be to send only the information required by the observer. In this case the subject should be able to distinguish between different types of observers and to know the required data of each of them, meaning that the subject layer is more coupled to observer layer.
* Pull model- The subject just notifies the observers when a change in his state appears and it's the responsibility of each observer to pull the required data from the subject. This can be inefficient because the communication is done in 2 steps and problems might appear in multi threading environments.

**Facade pattern**

Provides a unified interface for a set of interfaces in a subsystem. It defines a higher level of interface that makes the subsystem easier to use.

**Strategy Pattern**

Defines a family of algorithms, encapsulate each one, and make them interchangeable. It lets the algorithm vary independently from the clients that use it. Strategy pattern is used when we have multiple algorithm for a specific task and client/context decides the actual implementation to be used at runtime.

In Strategy pattern the determination of which algorithm to use can be done either by Client itself or we can delegate it to the Context based on the implementation.

**Composite Pattern**

Compose objects in Tree structures to represent part-whole hierarchies. It lets clients treat individual objects and composition of objects uniformly. Composite pattern is used where we need to treat a group of objects in similar way as a single object. This pattern creates a class that contains group of its own objects.

For example In Haskell we had,

Tree t = Null | Leaf(value) | Tree(value, Tree, Tree)

Here Tree is composed either of Null or Leaf or another subtrees.