**Singleton** is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.

The Singleton pattern solves two problems at the same time, violating the Single Responsibility Principle:

1. **Ensure that a class has just a single instance**. Why would anyone want to control how many instances a class has? The most common reason for this is to control access to some shared resource—for example, a database or a file.
2. **Provide a global access point to that instance**. Remember those global variables that you (all right, me) used to store some essential objects? While they’re very handy, they’re also very unsafe since any code can potentially overwrite the contents of those variables and crash the app.
3. Just like a global variable, the Singleton pattern lets you access some object from anywhere in the program. However, it also protects that instance from being overwritten by other code.

**Iterator** is a behavioral design pattern that allows sequential traversal through a complex data structure without exposing its internal details.

Thanks to the Iterator, clients can go over elements of different collections in a similar fashion using a single iterator interface.

**Observer** is a behavioral design pattern that allows some objects to notify other objects about changes in their state.

The Observer pattern provides a way to subscribe and unsubscribe to and from these events for any object that implements a subscriber interface.

**Strategy Pattern**

Factory Pattern: It encapsulate object creation.

It will be used when we are not sure what type of objects we need in our system in future.

Abstract Factory : It is useful when its user expects to receive a family of related objects at a given time but doesn’t have to know which family it is until run time