What is Design Patterns and why anyone should use them?

Design patterns are a well-described solution to the most commonly encountered problems which occur during software development.Design pattern represents the best practices evolved over a period of time by experienced software developers. They promote reusability which leads to a more robust and maintainable code.

The Design patterns can be classified into three main categories:

Creational Patterns

Creational design patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.

1)**Singleton**

Singleton pattern comes under creational patterns category and introduces a single class which is responsible to create an object while making sure that only single object gets created

2)Factory or Factory method

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

Pro's:

- Allows you to hide implementation of an application seam (the core interfaces that make up your application)
- Allows you to easily test the seam of an application (that is to mock/ stub) certain parts of your application so you can build and test the other parts
- Allows you to change the design of your application more readily, this is known as loose coupling

Con's :

- Makes code more difficult to read as all of your code is behind an abstraction that may in turn hide abstractions.

- Can be classed as an anti-pattern when it is incorrectly used, for example some people use it to wire up a whole application when using an IOC container, instead use Dependency Injection.

3)Abstract Factory

Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.

4)Prototype

Prototype pattern refers to creating duplicate object while keeping performance in mind. This pattern involves implementing a prototype interface which tells to create a clone of the current object.

5)Builder

Builder pattern builds a complex object using simple objects and using a step by step approach. This builder is independent of other objects.

Behavioral Patterns

Behavioral design patterns are concerned with algorithms and the assignment of responsibilities between objects.

1)Chain of Responsibility

Chain of Responsibility lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

2)Command

Encapsulates a request as an object, thereby letting us parameterize other objects with different requests, queue or log requests, and support undoable operations.

3)Iterator

Iterator pattern is used to get a way to access the elements of a collection object in sequential manner without any need to know its underlying representation.

4)Observer or Publish-Subscribe

The observer pattern in which an object (a subject) keeps track of all of its dependents (observers) and notifies them of any state changes.

5)State

The State pattern allows an object to change its behavior when its internal state changes. This pattern can be observed in a vending machine.

6)Visitor

Visitor pattern is used when we have to perform an operation on a group of similar kind of Objects. With the help of visitor pattern, we can move the operational logic from the objects to another class.

7)Strategy

Strategy pattern allows you to change the behavior of an object at run time without any change in the class of that object.

<u>Structural patterns(Functional Patterns)</u>

These patterns explain how to assemble objects and classes into larger structures while keeping these structures flexible and efficient.

1) Adapter

Adapter pattern works as a bridge between two incompatible interfaces. This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces (adaptees).

2)Bridge

Bridge pattern allows the Abstraction and the Implementation to be developed independently and the client code can access only the Abstraction part without being concerned about the Implementation part.

4)Decorator

Decorator pattern allows a user to add new functionality to an existing object without altering its structure.

5)Proxy

In proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern.

In proxy pattern, we create object having original object to interface its functionality to outer world.