Design Pattern:

Give the solution to solve the problems in the software development

1.Creational pattern: creating the object

1.Prototype Design Pattern:

Instead of creating new instance for an object just clone the object and make use of the existing object

- ✓ Shallow copy - create same instance
- ✓ Deep copy - create 2 diff instance

2.Singelton Pattern:

one instance for one class

- ✓ Early Instantiation - At load time itself
- ✓ Lazy Instantiation - Whenever it required

3.Builder Pattern:

- ✓ Used in software design to construct a complex object step by step.
- ✓ It allows the construction of a product in a step-by-step fashion, where the construction process can vary based on the type of product being built

4.Factory Pattern:

- ✓ Define an interface for creating object but let subclass decide which class to instantiate.
- ✓ Single class handling all object

5.Abstract Factory Pattern:

- ✓ Single type of object creating – factory pattern
- ✓ Different type of object is creating – abstract factory pattern

- ✓ AFP is a factory of factory

2.Structural design pattern: Structural Design Patterns are concerned with how classes and objects are composed to form larger structures. Structural class patterns use inheritance to compose interfaces or implementations. Adv: reusability and loosely coupling

1.Adapter Pattern:

Make in compactable interface as compactable one

Components

- ✓ Client code
- ✓ Target
- ✓ Adapter
- ✓ Adaptee

2.Bridge Pattern:

- ✓ The 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.
- ✓ It can use the existing methods instead of creating new class or make changes in code

3.Composite Pattern:

- ✓ Composite means group
- ✓ Compose objects into treelike structure

Components

- ✓ Component
- ✓ leaf

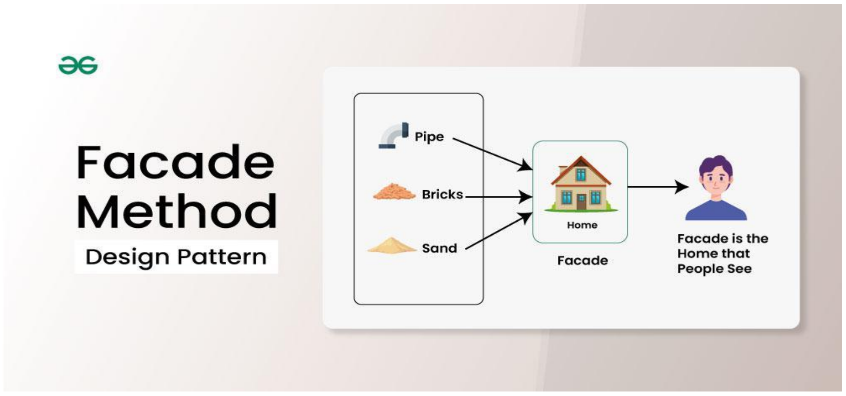
- ✓ Composite
- ✓ Client

4.Decorator Pattern:

Allows user to add new functionalities to an existing object without alternating its structure

5.Facade Pattern:

It hides the complexity of the underlying system and provides a simple interface that clients can use to interact with the system.



6.Flyweight Pattern:

- ✓ This pattern provides ways to decrease object count thus improving application
- ✓ The flyweight pattern is used when we need to create a large number of similar objects (say 10^5).
- ✓ In Flyweight pattern we use a HashMap that stores reference to the object which have already been created, every object is associated with a key
  - Intrinsic – Constant (e.g. all balls are in round shape)
  - Extrinsic – It can change (flyweight) (e.g. balls have different color)

7.Proxy Pattern:

- ✓ When you want to add an extra layer of control over access to an object
  And for redirecting to which page, it would be decided by proxy

- ✓ The proxy acts as an intermediary, controlling access to the real object.

3.Behavioral Design Pattern:

- ✓ Behavioral Patterns are concerned with algorithms and the assignment of responsibilities between objects.
- ✓ Behavioral class patterns use inheritance to distribute behavior between classes.

1.Chain of Responsibility:

- ✓ That allows an object to pass a request along a chain of handlers
- ✓ Each handler in the chain decides either to process the request or to pass it along the chain to the next handler.

2.Command Pattern:

- ✓ The **Command Pattern** encapsulates a request as an object, allowing for the separation of sender and receiver.
- ✓ Allowing parameterization of clients with different requests, queuing of requests, Use: undo, redo, Logging System, User Authentication

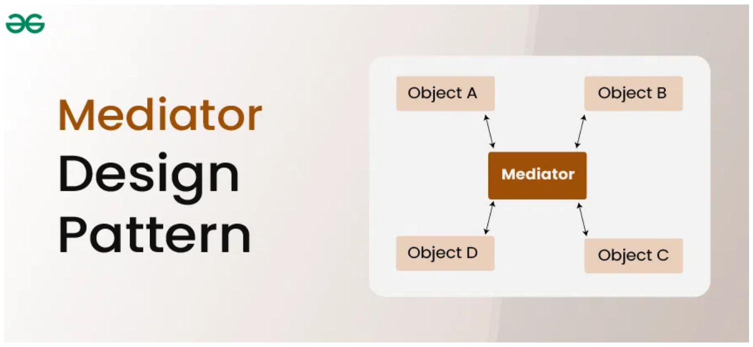
3.Interpreter Pattern:

- ✓ Interpreter pattern is used to define a grammatical representation for a language and provides an interpreter to deal with this grammar.

- ✓ This involves defining the behavior of interpreting expressions, parsing input strings, building expression trees, and recursively evaluating expression nodes based on predefined grammar rules.

4.Mediator Pattern:

- ✓ There is a mediator, to centralize communication between various components or objects in a system.
- ✓ This promotes loose coupling by preventing direct interactions between components, instead of having them communicate through the mediator, facilitating better maintainability and flexibility in the system architecture.



5.Iterator Pattern:

- ✓ The Iterator pattern is a widely used design pattern in software development that provides a way to access the elements of an aggregate object (such as a list or collection) sequentially
- ✓ It defines a separate object, called an iterator, which encapsulates the details of traversing the elements of the aggregate

6.Memento Pattern:

- ✓ It is used to restore the state of an object to a previous state.

- ✓ As your application is progressing, you may want to save checkpoints in your application and restore them back to those checkpoints later.

7.Observer Pattern:

- ✓ It defines a one-to-many dependency between objects,
- ✓ so that when one object (the subject) changes its state, all its dependents (observers) are notified and updated automatically.
- ✓ e.g.: Push Notification

8.State Pattern:

- ✓ That allows an object to alter its behavior when its internal state changes.
- ✓ This pattern is particularly useful when an object's behavior depends on its state, and the state can change during the object's lifecycle.
- ✓ e.g.: when mobile is in ringing and silent mode

9.Strategy Pattern:

- ✓ That allows the behavior of an object to be selected at runtime
- ✓ The Strategy Pattern allows you to define a family of algorithms, encapsulate each one of them, and make them interchangeable
- ✓ e.g.: Payment Processing, Google map (shortest route, fastest route)

10.Template Pattern:

- ✓ That defines the skeleton of an algorithm in a superclass but allows subclasses to override specific steps of the algorithm without changing its structure.
- ✓ It promotes code reuse by encapsulating the common algorithmic structure in the superclass while allowing subclasses to provide concrete implementations for certain steps, thus enabling customization and flexibility.
- ✓ e.g.: Bank ATM

11.Visitor Pattern:

- ✓ It is used when we must perform an operation on a group of similar kind of Objects.
- ✓ Define new operation without affecting the existing one
- ✓ e.g.: Online Shopping System

When to Use Which Design Pattern

Creational

If the Problem is related to Object Creation.

Singleton	Makes sure there is just one instance.
Factory Method	Assigns subclasses the task of instantiating objects.
Abstract Factory	Constructs related object families without defining their concrete classes.
Prototype	Clones objects to provide a template example.
Builder	Helps in building the complex objects step by step.

Structural

If the Problem is related to Object Assembly.

Adapter	Acts as a bridge between two incompatible interfaces
Bridge	Separates the abstraction from the implementation.
Composite	Handles single and composite objects equally.
Decorator	Adds behaviors to objects dynamically.
Facade	Helps in Simplifying the complex system interfaces.
Flyweight	Shares common parts of state between multiple objects to reduce memory.
Proxy	Controls the access to an object.

Behavioral

If the Problem is related to Object Interactions.

Observer	Observes and notifies changes in multiple objects.
Strategy	Encapsulates the interchangeable algorithms.
Command	Encapsulates requests as objects for decoupled execution.
State	It Changes the behavior of object with internal state.
Visitor	It separates algorithms from objects.
Memento	Pattern to manage object state and actions.
Iterator	It Sequentially accesses the elements of a collection.
Mediator	Central controller managing communication between objects.
Chain of Responsibility	Pass request through handlers until one handles it.
Template Method	Defines the skeleton of an algorithm.