Design Patterns

01 // Adapter – Structural Pattern

- Convert the interface of a class into another interface clients expect.
- Adapter lets classes work together, that could not otherwise because of incompatible interfaces.

02 // Bridge – Structural Pattern

Decouple abstraction from implementation so that the two can vary independently.

03 // Builder – Creational Pattern

- Defines an instance for creating an object but letting subclasses decide which class to instantiate.
- Separate the construction of a complex object from its representation so that the same construction process can create different representations.

04 // Chain of Responsibility – Behavioural Pattern

Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

05 // Command – Behavioural Pattern

Encapsulate a request as an object to parametrize clients with different requests, queue or log requests, and support undoable operations.

06 // Composite – Structural Pattern

- Compose objects into tree structures to represent part-whole hierarchies.
- Composite lets clients treat individual objects and compositions of objects uniformly.

07 // Decorator – Structural Pattern

Add additional responsibilities dynamically to an object.

08 // Facade – Structural Pattern

Provide a unified interface to a set of interfaces in a subsystem.

09 // Factory – Creational Pattern

- Creates objects without exposing the instantiation logic to the client.
- Refers to the newly created object through a common interface.

10 // Filter – Structural Pattern

Filter a set of objects using different criteria and chaining them in a decoupled way.

11 // Flyweight – Structural Pattern

Use sharing to support a large number of objects that have part of their internal state in common where the other part of state can vary.

12 // Iterator – Behavioural Pattern

Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

13 // Mediator – Behavioural Pattern

- Define an object that encapsulates how a set of objects interact.
- Promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.
- Design an intermediary to decouple many peers.

14 // Memento – Behavioural Pattern

- Without violating encapsulation, capture and externalize an object's internal state so that the object can be returned to this state later.
- Promote undo or rollback to full object status.

15 // Observer – Behavioural Pattern

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

16 // Proxy – Structural Pattern

Provide a placeholder for an object to control references to it.

17 // State – Behavioural Pattern

- Allow an object to alter its behaviour when its internal state changes.
- Class behaviour changes based on its state.

18 // Strategy – Behavioural Pattern

- Define a family of algorithms, encapsulate each one, and make them interchangeable.
- Strategy lets the algorithm vary independently from the clients that use it.

19 // Template – Behavioural Pattern

Define the skeleton of an algorithm.

20 // Visitor – Behavioural Pattern

- Represent an operation to be performed on the elements of an object structure.
- Visitor class is used to change the executing algorithm of an element class.