BEHAVIOURAL DESIGN PATTERNS

1. ITERATOR DESIGN PATTERNS

DEFINITION OF ITERATOR PATTERN

The Iterator Pattern provides a way to access elements of a collection sequentially without exposing the underlying structure of the collection.

It allows traversal of data without depending on the internal implementation.

Follows the Behavioral category of design patterns.

COMPONENTS -

- Concrete Aggregate: Implements the collection and returns an iterator instance
- Iterator Interface: Declares methods like Next(), HasNext()
- Concrete Iterator: Implements the iterator and keeps track of the current position

WHEN SHOULD WE USE THIS :

1. You have a collection (list, tree, array, etc.) and want to loop through its elements.
2. You want to hide the internal structure of the collection.
3. You need multiple types of iteration (e.g., forward, backward, custom).
4. You want to implement your own custom collection class.

REAL WORLD APPLICATION:-

Custom lists in C#
File readers
Menu systems in games or apps
Paginated results

ADVANTAGES:-

1. Simplifies traversal of collections.
2. Hides internal structure of collections.
3. Supports multiple traversals (e.g., forward, reverse).
4. Makes code cleaner and more maintainable.
5. Can be generic to support multiple data types.

DISADVANTAGES:-

6. More classes to write (extra boilerplate).
7. Custom iterators can be harder to maintain for complex collections.
8. Not as efficient for some special data structures (like graphs, trees without built-in iterators).

2. OBSERVER DESIGN PATTERNS

DEFINITION OF OBSERVER PATTERN

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

In the Observer Design Pattern, an object (called a Subject) maintains a list of its dependents (called Observers). It notifies them automatically whenever any state changes.

COMPONENTS -

1. Subject: They are also called Publishers. When a change occurs to a subject, it should notify all its Subscribers/Observers.

2. Observers: They are also called subscribers. They listen to the changes in the subjects.

WHEN SHOULD WE USE THIS :

-When a change to one object requires changing others, you don’t know how many objects need to be changed.
-When an object should be able to notify other objects without making assumptions about who these objects are.

REAL WORLD APPLICATION:-

Weather Station and Mobile Apps.
YouTube Channel.

ADVANTAGES:-

1. Loose Coupling:- The Subject and Observers are independent. The subject doesn’t need to know how observers work.
2. Easy to Add/Remove Observers:- You can add or remove observers at runtime without changing the subject’s code.
3. Real-Time Updates:- All observers get immediate notifications when something changes.

DISADVANTAGES:-

1. Memory Leaks (if not handled well):- If you forget to unsubscribe an observer, it might stay in memory, causing performance issues.
2. Order of Notification is Not Guaranteed:- If the update order is important, you might need extra logic to control it.

THANK YOU