DATE:18/07/2024

DAY 4:

Software Design Pattern

Software design patterns are established solutions to common design problems that software developers face. They are templates or best practices for designing software architectures and solving issues in a way that is both effective and reusable.Some of the most important design patterns, categorized into three main types: **Creational, Structural, and Behavioral.**

1. Creational Design Patterns

Creational patterns focus on how objects are created. They abstract the instantiation process, making it more flexible and efficient.

Singleton

- ➢ Purpose: Ensures that a class has only one instance and provides a global point of access to it.
- ➢ Example: A configuration manager that reads configuration settings from a file.

Factory Method

- ➢ Purpose: Defines an interface for creating objects, but allows subclasses to alter the type of objects that will be created.
- ➢ Example: A document creation application where the type of document (Word, PDF, etc.) is decided at runtime.

Abstract Factory

- ➢ Purpose: Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
- ➢ -Example: Creating user interfaces with different themes (light mode, dark mode).

Builder

- ➢ Purpose: Separates the construction of a complex object from its representation so that the same construction process can create different representations.
- ➢ Example: Building a complex meal with different combinations of dishes.

Prototype

- Purpose: Creates new objects by copying an existing object, known as the prototype.
- Example: Copying objects with default settings for a new configuration.

2. Structural Design Patterns

Structural patterns focus on how objects and classes are composed to form larger structures.

Adapter (or Wrapper)

- Purpose: Allows incompatible interfaces to work together.
- Example: Adapting a legacy system interface to a new system.

Decorator

- Purpose: Adds new functionality to an object without altering its structure.
- Example: Adding scroll bars to a window.

Composite

- Purpose: Allows clients to treat individual objects and compositions of objects uniformly.
- Example: A file system where files and directories are treated similarly.

Facade

- Purpose: Provides a simplified interface to a complex subsystem.
- Example: A simplified API for a complex library.

Bridge

- Purpose: Decouples an abstraction from its implementation so that the two can vary independently.
- Example: Drawing different shapes (circle, square) in different colors.

Proxy

- Purpose: Provides a surrogate or placeholder for another object.
- Example: A proxy that manages access to a resource-heavy object.

3. Behavioral Design Patterns

Behavioral patterns focus on communication between objects and how responsibilities are distributed.

Chain of Responsibility

- Purpose: Passes a request along a chain of potential handlers until one of them handles it.
- Example: A help desk where requests are escalated through different levels.