- A Design Pattern is a proven solution to a common problem in software design. Instead of reinventing the wheel every time you encounter a design challenge, a design pattern offers a general reusable solution that has been tested and proven to work well for that specific problem.

Why Use Design Patterns?

1. Reuse solutions that have already been proven to work.

2. Improve communication among developers (for instance, when you say "use Singleton," other developers know exactly what you mean).

3. Organize code better and make it more understandable.

4. Facilitate maintenance and scalability in larger system

Main Categories of Design Patterns:

They are grouped into three main types:

1. Creational Patterns:

These focus on how objects are created in a flexible and appropriate manner.

Singleton: Ensures only one instance of a class exists in the system (e.g., settings or configurations).

Factory Method: Creates objects of a class without specifying the exact class of the object to be created.

Builder: Constructs a complex object step-by-step.

Abstract Factory: Creates families of related or dependent objects without specifying their concrete classes.

Prototype: Uses an existing object to create new objects, rather than creating a new one from scratch.

2. Structural Patterns:

These deal with how to compose classes or objects into larger structures.

Adapter: Converts one interface to another that a client expects.

Decorator: Adds new behavior or responsibilities to an object dynamically.

Facade: Provides a simplified interface to a complex subsystem of classes or operations.

Composite: Allows individual objects and compositions of objects to be treated uniformly.

3. Behavioral Patterns:

These focus on the interaction and responsibilities between objects.

Observer: Allows a subject to notify its observers automatically of state changes.

Strategy: Defines a family of algorithms and allows them to be interchangeable at runtime.

Command: Encapsulates a request as an object, allowing parameterization of clients with different requests.

State: Changes the behavior of an object when its internal state changes.

Mediator: Reduces dependencies between objects by centralizing communication through a mediator object.