In Software engineering, a **design pattern** is a *general repeatable solution to a commonly occurring problem in software design*. It is a **description or template** for how to solve a problem that can be used in many different situations. It is a **guideline** for **flexible** and **resilient** code design.

Created by Gang of Four. Original 23 patterns.

**Advantages:**

1. DS are already defined and provide **industry standard approach to solve a recurring problem,** so it leads to faster development and makes our code easy to understand and debug.
2. Using design pattern promotes **re-usability** that leads to more **robust** and **highly maintainable** code.

Divided into three categories:

1. Creational
2. Structural
3. Behavioural

**Creational design patterns**

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

1. **Abstract Factory**  
   Creates an instance of several families of classes
2. **Builder**  
   Separates object construction from its representation
3. **Factory Method**  
   Creates an instance of several derived classes
4. **Object Pool** Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
5. **Prototype**  
   A fully initialized instance to be copied or cloned
6. **Singleton**  
   A class of which only a single instance can exist

**Structural design patterns**

These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

1. **Adapter**Match interfaces of different classes
2. **Bridge**Separates an object’s interface from its implementation
3. **Composite**A tree structure of simple and composite objects
4. **Decorator**Add responsibilities to objects dynamically
5. **Facade**A single class that represents an entire subsystem
6. **Flyweight**A fine-grained instance used for efficient sharing
7. **Private Class Data**Restricts accessor/mutator access
8. **Proxy**An object representing another object

**Behavioural design patterns**

These design patterns are all about Class's objects communication. Behavioural patterns are those patterns that are most specifically concerned with communication between objects.

1. **Chain of responsibility**A way of passing a request between a chain of objects
2. **Command**Encapsulate a command request as an object
3. **Interpreter**A way to include language elements in a program
4. **Iterator**Sequentially access the elements of a collection
5. **Mediator**Defines simplified communication between classes
6. **Memento**Capture and restore an object's internal state
7. **Null Object**Designed to act as a default value of an object
8. **Observer**A way of notifying change to a number of classes
9. **State**Alter an object's behaviour when its state changes
10. **Strategy**Encapsulates an algorithm inside a class
11. **Template method**Defer the exact steps of an algorithm to a subclass
12. **Visitor**Defines a new operation to a class without change