**Software Design Pattern**

# Software Design Pattern

Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

Patterns are commonly found in objected-oriented programming languages like C++ or Java. They can be seen as a template for how to solve a problem that occurs in many different situations or applications. **It is not code reuse, as it usually does not specify code**, but code can be easily created from a design pattern.

Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

Concept was described by Christopher Alexander in its architectural theories

It only gathered some traction in programming due to the publication of **Design Patterns: Elements of Reusable Object-Oriented Software** book in October 1994 by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, known as the Gang of Four (GoF) that identifies and describes 23 classic software design patterns.

**A design pattern is neither a static solution, nor is it an algorithm**. A pattern is a way to describe and address by name (mostly a simplistic description of its goal), a repeatable solution or approach to a common design problem, that is, a common way to solve a generic problem (how generic or complex, depends on how restricted the target goal is). Patterns can emerge on their own or by design. This is why design patterns are useful as an abstraction over the implementation and a help at design stage. With this concept, an easier way to facilitate communication over a design choice as normalization technique is given so that every person can share the design concept.

Each design pattern consists of the following parts:

1. **Problem/requirement**

To use a design pattern, we need to go through a mini analysis design that may be coded to test out the solution. This section states the requirements of the problem we want to solve. This is usually a common problem that will occur in more than one application.

1. **Forces**

This section states the technological boundaries that helps and guides the creation of the solution.

1. **Solution**

This section describes how to write the code to solve the above problem. This is the design part of the design pattern. It may contain class diagrams, sequence diagrams, and or whatever is needed to describe how to code the solution.

Design patterns can be considered as a **standardization of commonly agreed best practices to solve specific design problems**. One should understand them as a way to implement good design patterns within applications. Doing so will reduce the use of inefficient and obscure solutions. Using design patterns speeds up your design and helps to communicate it to other programmers.

# Classification

Depending on the design problem they address, design patterns can be classified in different categories, of which the main categories are:

1. **Creational patterns** provide the capability to create objects based on a required criterion and in a controlled way.
2. **Structural patterns** are about organizing different classes and objects to form larger structures and provide new functionality.
3. **Behavioral patterns** are about identifying common communication patterns between objects and realizing these patterns.

|  |  |  |  |
| --- | --- | --- | --- |
| Sr.No. | Creational | Structural | Behavioral |
| 1. | Abstract factory | Adapter | Chain of responsibility |
| 2. | Builder | Bridge | Command |
| 3. | Factory method | Composite | Interpreter |
| 4. | Prototype | Decorator | Iterator |
| 5. | Singleton | Facade | Mediator |
| 6. |  | Flyweight | Memento |
| 7. |  | Proxy | Observer |
| 8. |  |  | State |
| 9. |  |  | Strategy |
| 10. |  |  | Template method |
| 11. |  |  | Visitor |

# References

<https://en.wikipedia.org/wiki/Software_design_pattern>

# END