Design Patterns

Design patterns are typical solutions to common problems in software design.

Design Patterns

The design pattern is typical solutions to problems that commonly occur in software design. They are like pre-made plans that you can customize to solve a recurring design problem in your code.

You cannot immediately find a pattern and copy it into your program, such as ready-to-use functions or libraries. A design pattern is a general concept for solving a specific problem, not a specific piece of code. You follow the pattern details and apply a solution that fits the realities of your own program.

Design patterns are implemented in accordance **with Software Design Principles.** Therefore, we should learn these principles before the subject of design patterns.

Patterns are often confused with algorithms because both concepts describe typical solutions to some known problems. **An algorithm defines a clear set of actions that can always achieve a goal, while a pattern is a higher-level description of a solution.** The code of the same pattern applied to two different programs can be different.

An algorithm is like a recipe: Both have clear steps to achieve a goal. On the other hand, a pattern is more like an outline. You can see what the result and its properties are, but the exact order of application is up to you.

What Does a Pattern Consist of?

Most design patterns are defined very formally so that people reproduce them in many contexts. The sections commonly found in a pattern description are:

- **The purpose of the pattern** briefly describes both the problem and the solution.
- **The motivation** further explains the problem and the solution that the pattern makes possible.
- **The structure of classes** shows each part of the pattern and how they are related.
- **A code example** in one of the popular programming languages makes it easy to grasp the idea behind the pattern.

Some mold catalogs list other useful details such as mold applicability, application steps, and relationships with other molds.

Classification of Patterns

The most basic and low-level patterns are often called **idioms**. Often they only apply to a single programming language.

Design patterns differ in their complexity, level of detail, and degree of applicability to the entire system being designed. It can be likened to the construction of a road. You can make an intersection safer by installing some traffic lights or by building a multi-level intersection with underground passages for pedestrians.

The most universal and high-end patterns are architectural patterns. Developers implement these patterns in almost every language. Unlike other patterns, they can be used to design the architecture of an entire application.

Additionally, all patterns are categorized according to their intent or purpose. This series of articles will describe the three main groups of design patterns:

- **Creational patterns** provide object creation mechanisms that increase flexibility and reuse of existing code.
- **Structural patterns** describe how to combine objects and classes into larger structures while keeping structures flexible and efficient.
- **Behavioral patterns** provide effective communication and assignment of responsibilities between objects.

The design patterns under these three main categories are as follows;

Creational Patterns

- Factory Method
- Abstract Factory
- Builder
- Prototype
- Singleton

Structural Design Patterns

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Behavioral Design Patterns

- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

Who Invented Design Patterns?

That's a good, but not very accurate question. Design patterns are not vague, complex concepts. On the contrary, patterns are typical solutions to common problems in object-oriented design. When a solution is repeated over and over in various projects, someone eventually gives it a name and explains the solution in detail. This is basically how a design pattern is discovered.

The concept of pattern was first defined by **Christopher Alexander** in his book **A Pattern Language: Towns, Buildings, Construction.** The book defines a "language" for designing the urban environment. The units of this language are patterns. These patterns indicate how high the windows should be, how many floors a building should have, how big green areas should be in a neighborhood, etc. they can identify.

The idea was taken up by four authors: **Erich Gamma, John Vlissides, Ralph Johnson and Richard Helm.** In 1994, they released **Design Patterns: Elements of Reusable Object-Oriented Software**, in which they applied the concept of design patterns to programming. The book contains 23 models that solve various problems of object-oriented design. Moreover, it has become a best seller very quickly. Because of its long name, people started calling it "**the book by the gang of four**" and soon shortened it to just "**the GoF book**".

Since then, dozens of other object-oriented design patterns have emerged. The "**pattern approach**" has become very popular in other programming areas, so there are now many other patterns outside of object-oriented design as well.

Why Should I Learn Design Patterns?

The truth is that you can manage to work as a programmer for many years without knowing a single pattern. Lots of people do just that. Even then, you may be applying some patterns unknowingly. So why spend time learning about them?

• Design patterns are a toolkit of tried and tested solutions to common problems in software design. Even if you never encounter these problems, knowing the design pattern still benefits you. Because it teaches you how to solve any problem using object-oriented design principles.

• The design pattern defines a common language that you and your teammates can use to communicate more efficiently. You can say "Oh, use a Singleton for this" and everyone gets the idea behind your suggestion. There is no need to explain what a singleton is if you know the pattern and its name.