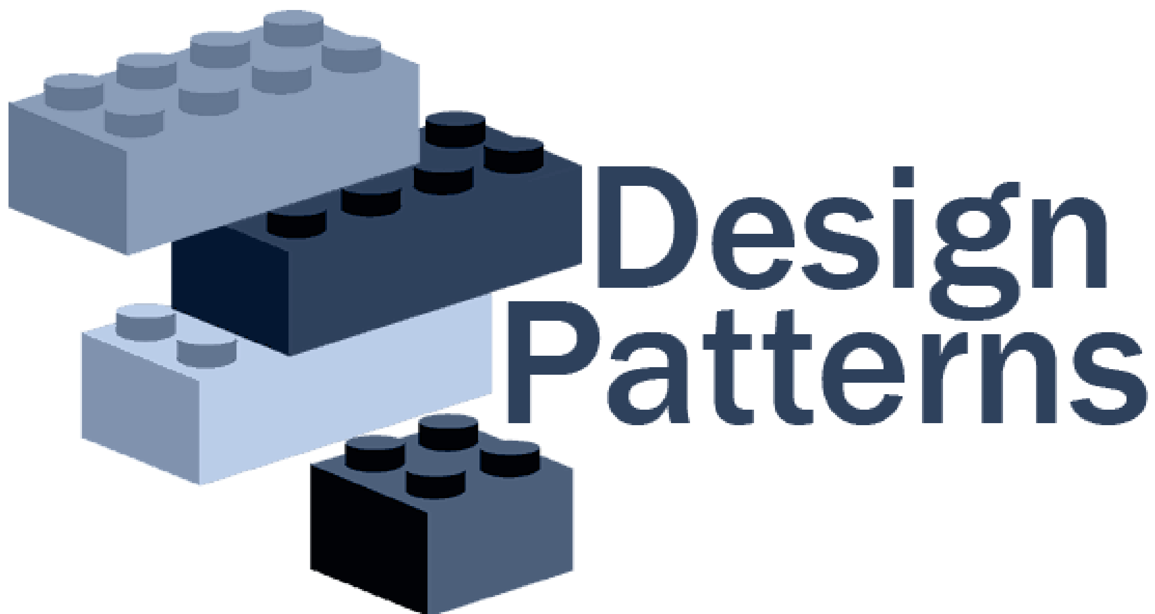
Design Patterns

▼ Introduction to Design Patterns



▼ What are Design Patterns?

Design patterns are reusable solutions to common problems in software design. They are best practices that provide developers with proven ways to structure their code, helping solve issues related to object creation, communication, and interaction. Design patterns come from decades of experience in object-oriented programming and are designed to make your code more **reusable**, **maintainable**, and **flexible**.

- Patterns help in writing reusable software components.
    - For example, if you're building a module to handle connections to a database, you don't want to write it from scratch every time.
    - A pattern like the **Singleton** ensures that only one instance of that module exists across your app, promoting code reuse.
- Design patterns embody the principles of **object-oriented programming (OOP)** such as encapsulation, abstraction, and inheritance. They make it easier to design solutions that can be extended or modified with minimal impact on the existing code.

Categories of Design Patterns

By Purpose				
		Creational	Structural	Behavioral
By Scope	Class	• Factory Method	• Adapter (class)	• Interpreter • Template Method
	Object	• Abstract Factory • Builder • Prototype • Singleton	• Adapter (object) • Bridge • Composite • Decorator • Façade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

Design patterns are generally categorized into three groups based on the types of problems they solve:

1. **Creational Patterns** — These patterns provide various mechanisms to create objects, allowing for greater flexibility in object creation. Examples:
    - **Singleton**: Ensures only one instance of a class.
    - **Factory**: Provides an interface for creating objects in a super class, but allows subclasses to alter the type of objects that will be created.
2. **Structural Patterns** — These deal with the composition of classes or objects. They help ensure that if one part of the system changes, the

whole system doesn’t break. Examples:

  - **Adapter**: Allows incompatible interfaces to work together.
  - **Decorator**: Adds responsibilities to objects dynamically.

3. **Behavioural Patterns** — These patterns are concerned with the interactions and responsibilities between objects. They help in defining communication between classes and objects. Examples:
    - **Observer**: Defines a dependency between objects so that when one changes, its dependents are notified.
    - **Strategy**: Enables selecting an algorithm's behaviour at runtime.

▼ Why are Design Patterns Important?

- **Improves Code Maintainability**: By applying patterns, developers can ensure that the software is easier to modify and extend.
- **Provides Common Vocabulary**: Design patterns create a shared language, making communication between developers easier.
- **Prevents Over-Engineering**: Using design patterns promotes simplicity and avoids building complex, redundant code.
- **Increases Productivity**: Having a ready-made, proven solution accelerates development and debugging time.

▼ Example: Singleton Pattern

As we mentioned before, one of the simplest and most commonly used design patterns is **Singleton**. Here's a practical demonstration:

```
public class Singleton {
    private static Singleton instance;

    // Private constructor to restrict instantiation.
    private Singleton() {}

    // Public method to provide access to the single i
nstance.
    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
```

```
        return instance;
    }

    public void showMessage() {
        System.out.println("Hello from Singleton!");
    }

    public static void main(String[] args) {
        Singleton instance = Singleton.getInstance();
        instance.showMessage(); // Output: Hello from
Singleton!
    }
}
```

- This Singleton example prevents more than one instance of a class from being created.
- This pattern can be useful for classes that manage shared resources such as database connections or logging mechanisms.
- A Singleton can prevent unnecessary resource usage by ensuring only one instance exists, reducing the overhead of resource management.

Design Patterns

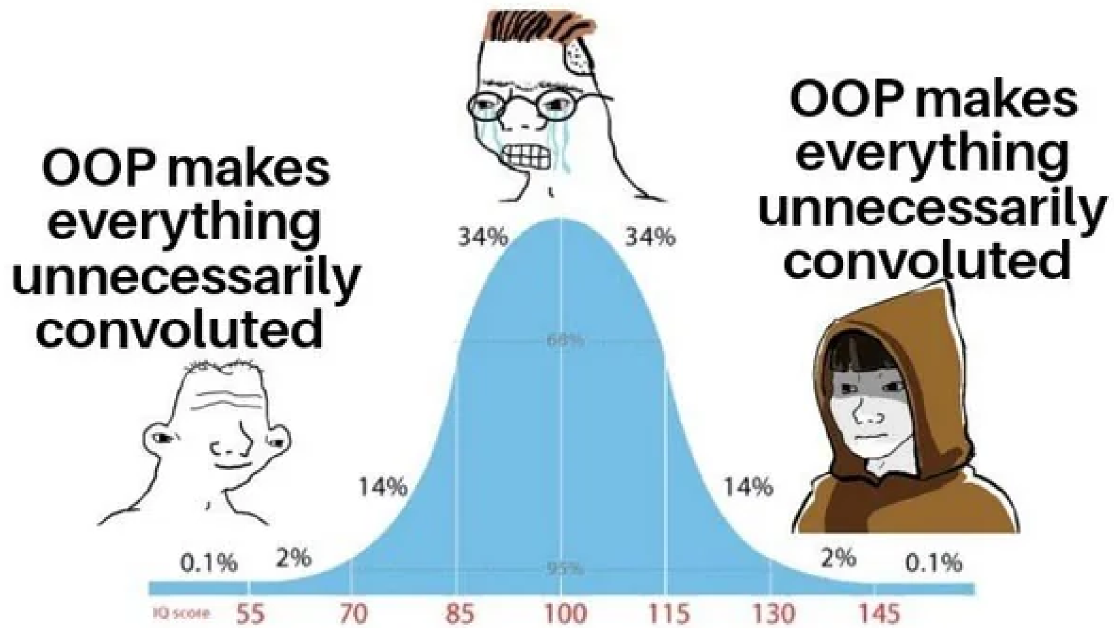
Design Patterns: Elements of Reusable Object-Oriented Software (1994) is a software engineering book describing software design patterns. The book was

https://en.wikipedia.org/wiki/Design_Patterns

What's a design pattern?

Design patterns are typical solutions to commonly occurring problems in software design. They are like pre-made blueprints that you can customize to solve a

https://refactoring.guru/design-patterns/what-is-pattern


Noooo, you need to use design patterns to make the code better. OOP helps us. Just listen to Uncle Bob
OOP makes everything unnecessarily convoluted
OOP makes everything unnecessarily convoluted
34%
34%
14%
14%
0.1%
2%
2%
0.1%
IQ score
55
70
85
100
115
130
145