# Design Patterns in Software Development

Design patterns are reusable solutions to commonly occurring problems in software design. They provide a structured approach to solving design issues and improve code maintainability, scalability, and readability.

---

# 1. Creational Design Patterns

Creational patterns deal with object creation mechanisms, optimizing for flexibility and reuse.

### a. Singleton Pattern

- **Definition:** Ensures a class has only one instance and provides a global access point.
- **Example:** Database connection manager.

### b. Factory Method Pattern

- **Definition:** Defines an interface for creating an object but allows subclasses to alter the type of objects that will be created.
- **Example:** A vehicle factory that produces different types of vehicles (Car, Bike, Truck).

### c. Abstract Factory Pattern

- **Definition:** Provides an interface for creating families of related objects without specifying their concrete classes.
- **Example:** GUI toolkit supporting multiple OS themes (Windows, Mac, Linux).

### d. Builder Pattern

- **Definition:** It separates the construction of a complex object from its representation.
- **Example:** Creating a complex meal order with multiple components like burgers, drinks, and sides.

### e. Prototype Pattern

- **Definition:** Creates new objects by cloning an existing object.
- **Example:** A drawing application where users duplicate a shape with its properties.

---

# 2. Structural Design Patterns

Structural patterns deal with object composition, ensuring that components are organized efficiently.

### a. Adapter Pattern

- **Definition:** Allows incompatible interfaces to work together by providing a wrapper that translates requests.
- **Example:** A power adapter that allows a US plug to work in a European socket.

### b. Bridge Pattern

- **Definition:** It separates abstraction from implementation so both can evolve independently.
- **Example:** A remote control that works with multiple TV brands.

### c. Composite Pattern

- **Definition:** Composes objects into tree structures to represent part-whole hierarchies.
- **Example:** A file system where directories contain files and sub-directories.

### d. Decorator Pattern

- **Definition:** Adds functionality to an object dynamically without modifying its structure.
- **Example:** A text editor that adds formatting features like bold and italic dynamically.

### e. Facade Pattern

- **Definition:** Provides a simplified interface to a complex system.
- **Example:** A computer startup process that involves multiple subsystems like CPU, Memory, and Hard Drive.

### f. Flyweight Pattern

- **Definition:** Reduces memory usage by sharing common objects instead of creating new ones.
- **Example:** Text editor storing shared character objects instead of individual instances.

### g. Proxy Pattern

- **Definition:** Provides a surrogate or placeholder for another object to control access to it.
- **Example:** A virtual proxy that delays the loading of an image until it's needed.

---

## 3. Behavioral Design Patterns

Behavioral patterns focus on communication between objects and the delegation of responsibilities.

### a. Chain of Responsibility Pattern

- **Definition:** Passes a request along a chain of handlers until one handles it.
- **Example:** A logging system where different log levels (info, warning, error) pass through handlers.

### b. Command Pattern

- **Definition:** Encapsulates a request as an object, allowing parameterization and queuing of requests.
- **Example:** A remote control system that executes commands like turning the TV on/off.

### c. Interpreter Pattern

- **Definition:** Defines a grammar for a language and provides an interpreter to process statements.
- **Example:** A simple arithmetic expression parser.

### d. Iterator Pattern

- **Definition:** Provides a way to traverse elements of a collection without exposing its underlying representation.
- **Example:** Iterating through a list of songs in a music playlist.

### e. Mediator Pattern

- **Definition:** Defines an intermediary object to handle communication between multiple objects, reducing dependencies.
- **Example:** A chatroom where a mediator facilitates communication between users.

### f. Memento Pattern

- **Definition:** Captures and restores an object's state to support undo functionality.
- **Example:** A text editor with an undo/redo feature.

### g. Observer Pattern

- **Definition:** Defines a dependency between objects so that when one changes state, all its dependents are notified.
- **Example:** A stock market application that updates investors when stock prices change.

### h. State Pattern

- **Definition:** Allows an object to change its behavior when its internal state changes.
- **Example:** A traffic light system switching between red, yellow, and green states.

### i. Strategy Pattern

- **Definition:** Defines a family of algorithms and lets the algorithm vary independently from clients using it.
- **Example:** A payment system supporting multiple payment methods (credit card, PayPal, Bitcoin).

### j. Template Method Pattern

- **Definition:** Defines the program skeleton in a base class but lets subclasses implement specific steps.
- **Example:** A game engine where different games share a common game loop but have unique mechanics.

### k. Visitor Pattern

- **Definition:** It separates operations from objects so new operations can be added without modifying the object structure.
- **Example:** A tax calculation system where different tax rules apply to different product types.

---

# Conclusion

Design patterns provide standard solutions to recurring design problems.
Understanding and applying them correctly

can lead to better-structured, more maintainable, and flexible code.