

**Example Use Cases:**

**Behavioural Design Patterns:**

* **Chain of Responsibility:** Processing a request through a chain of handlers (e.g., technical support system).
* **Observer:** Updating UI components when data changes (e.g., MVC architecture).
* **Strategy:** Switching between different algorithms (e.g., different sorting methods).

**Structural Design Patterns:**

* **Adapter:** Allowing incompatible interfaces to work together (e.g., integrating a third-party library).
* **Composite:** Representing part-whole hierarchies (e.g., a tree structure for a file system).
* **Proxy:** Controlling access to an object (e.g., lazy loading of large objects).

**Creational Design Patterns:**

* **Factory Method:** Creating objects without specifying the exact class (e.g., creating different types of documents).
* **Builder:** Constructing complex objects step by step (e.g., constructing a complex configuration).
* **Singleton:** Ensuring a class has only one instance (e.g., configuration settings, logging).

**Summary:**

* **Behavioural Patterns:** Focus on object interaction and responsibility delegation.
* **Structural Patterns:** Emphasize composing objects and classes to form larger structures.
* **Creational Patterns:** Centre on object creation mechanisms, providing flexibility and reuse in object instantiation.

These patterns are fundamental in software design, each serving distinct purposes and enhancing various aspects of object-oriented programming. Choosing the appropriate pattern depends on the specific design needs and challenges faced during software development.