What types of problems Design pattern solves

Design patterns are reusable solutions to common software design problems that developers encounter during the software development process. These patterns provide structured and proven approaches to designing and organizing code, improving code quality, maintainability, and scalability. Here's a detailed overview of the types of problems that design patterns solve:

1. **Creational Patterns:**

Creational patterns deal with object creation mechanisms, abstracting the instantiation process and hiding the details of object creation.

- **Singleton:** Ensures that a class has only one instance and provides a global point of access to that instance.

- **Factory Method:** Provides an interface for creating objects but allows subclasses to decide which class to instantiate.

- **Abstract Factory:** Offers an interface for creating families of related or dependent objects without specifying their concrete classes.

- **Builder:** Separates the construction of a complex object from its representation, allowing the same construction process to create different representations.

- **Prototype:** Creates new objects by copying an existing object, allowing efficient object creation.

2. **Structural Patterns:**

Structural patterns focus on the composition of classes and objects to form larger structures while keeping the system flexible and efficient.

- **Adapter:** Allows incompatible interfaces to work together by providing a wrapper that translates one interface to another.

- **Bridge:** Separates an abstraction from its implementation, allowing them to evolve independently.

- **Composite:** Composes objects into tree structures to represent part-whole hierarchies, enabling clients to treat individual objects and compositions uniformly.

- **Decorator:** Dynamically adds responsibilities to objects without modifying their code, providing a flexible alternative to subclassing.

- **Facade:** Provides a unified interface to a set of interfaces in a subsystem, simplifying interaction for clients.

- **Flyweight:** Shares objects to reduce memory usage when many instances of the same data are needed.

3. **Behavioral Patterns:**

Behavioral patterns define how objects communicate and interact with each other, promoting better collaboration and flexibility.

- **Chain of Responsibility:** Passes a request along a chain of handlers, allowing each handler to decide whether to process the request or pass it to the next handler.

- **Command:** Encapsulates a request as an object, allowing parameterization of clients with different requests and supporting undoable operations.

- **Interpreter:** Defines a grammar for interpreting a language and provides an interpreter to execute sentences in the language.

- **Iterator:** Provides a way to access elements of an aggregate object sequentially without exposing its underlying representation.

- **Mediator:** Reduces direct connections between objects by using a mediator to control their interactions.

- **Memento:** Captures and restores an object's internal state without exposing its internal structure, allowing the object to return to a previous state.

- **Observer:** Defines a one-to-many dependency between objects, ensuring that when one object changes state, all its dependents are notified and updated.

- **State:** Allows an object to change its behavior when its internal state changes, enabling more dynamic behavior.

- **Strategy:** Defines a family of algorithms, encapsulates each one, and makes them interchangeable.

- **Template Method:** Defines the structure of an algorithm, allowing its steps to be implemented by subclasses.

- **Visitor:** Separates an algorithm from the object structure on which it operates, allowing new operations to be added without modifying the objects.

4. **Architectural Patterns:**

Architectural patterns address larger-scale design decisions, guiding the overall structure and organization of a software system.

- **Model-View-Controller (MVC):** Separates an application into three components: Model (data and business logic), View (presentation), and Controller (user interaction).

- **Model-View-ViewModel (MVVM):** Similar to MVC, MVVM separates concerns by adding a ViewModel that represents the state and behavior of the View.

- **Observer/Observable Pattern:** Allows objects to publish and subscribe to changes, often used for communication between components.

Design patterns provide solutions to recurring design challenges, promote code reusability, and enhance software architecture. By understanding and applying these patterns, developers can create more maintainable, modular, and flexible software systems.