What are Design Patterns?

Design patterns are reusable solutions to commonly occurring problems in software design.

1. Creational Design Patterns

A creational design pattern deals with object creation and initialization, providing guidance about which objects are created for a given situation. These design patterns are used to increase flexibility and to reuse existing code.

* **Factory Method**: Creates objects with a common interface and lets a class defer instantiation to subclasses.
* **Abstract Factory**: Creates a family of related objects.
* **Builder**: A step-by-step pattern for creating complex objects, separating construction and representation.
* **Prototype**: Supports the copying of existing objects without code becoming dependent on classes.
* **Singleton**: Restricts object creation for a class to only one instance.

2. Structural Design Patterns

A structural design pattern deals with class and object composition, or how to assemble objects and classes into larger structures.

* **Adapter**: How to change or adapt an interface to that of another existing class to allow incompatible interfaces to work together.
* **Bridge**: A method to decouple an interface from its implementation.
* **Composite**: Leverages a tree structure to support manipulation as one object.
* **Decorator**: Dynamically extends (adds or overrides) functionality.
* **Façade**: Defines a high-level interface to simplify the use of a large body of code.
* **Flyweight**: Minimize memory use by sharing data with similar objects.
* **Proxy**: How to represent an object with another object to enable access control, reduce cost and reduce complexity.

3. Behavioral Design Patterns

A behavioral design pattern is concerned with communication between objects and how responsibilities are assigned between objects.

* **Chain of Responsibility**: A method for commands to be delegated to a chain of processing objects.
* **Command**: Encapsulates a command request in an object.
* **Interpreter**: Supports the use of language elements within an application.
* **Iterator**: Supports iterative (sequential) access to collection elements.
* **Mediator**: Articulates simple communication between classes.
* **Memento**: A process to save and restore the internal/original state of an object.
* **Observer**: Defines how to notify objects of changes to other object(s).
* **State**: How to alter the behavior of an object when its stage changes.
* **Strategy**: Encapsulates an algorithm inside a class.
* **Visitor**: Defines a new operation on a class without making changes to the class.
* **Template Method**: Defines the skeleton of an operation while allowing subclasses to refine certain steps.