**Introduction**

Design patterns are simply defined and validated solutions to reoccurring problems in a given context. To put it simply, Design Patterns are reusable solutions to challenges that developers face daily. Design Patterns are used to overcome difficulties with object generation and integration. Design patterns are best practices that have emerged over time via trial and error among experienced software engineers. Design patterns might be viewed as templates for tackling specific design difficulties rather than final designs that can be converted directly into code.

**Types of Design Patterns**

Design Patterns were classified into three major groups based on the three problem areas of software architecture (Object Creation and Initialization, Structural Changes of Classes and Interfaces, and Class Relationships and Object Communication).

They are as follows.

* Creational Design Pattern
* Structural Design Pattern
* Behavioral Design Pattern

**Creational Design Pattern**

Creational design patterns deal with object creation mechanisms. To put it simply, the Creational Design Pattern is concerned with the creation and initialization of objects. This Design Pattern allows the programmer additional control over which objects should be created for a particular context. The Creational Design Pattern emphasizes object creation in software development. Instead of generating objects directly with the new operator, these patterns provide alternate techniques for object creation. Creational patterns abstract the object instantiation process, resulting in a system that is independent of how objects are produced, assembled, or represented.

* Object-Creational Patterns: Object-Creational Patterns are concerned with object creation. In this case, it defers a portion of the object creation to another object.
* Class-Creational Patterns: Class-Creational Patterns are concerned with class instantiation. It defers object creation to subclasses.

In real applications, projects are generated with a large number of classes. A large number of classes indicates that we are working with a large number of objects. So we must construct several objects (such as new User(), new Item(), new Student(), new Payment(), and so on) for the Application. If the object creation logic based on a condition is implemented in the client code, it results in a large amount of sophisticated logic. The client code is the class that consumes the User Object, Item Object, Student Object, and so on by using its methods and properties.

That is, if the object generation and initialization logic are not centralized, it results in complex client code. The Creational Design Pattern enables us to centralize object creation and initialization logic, and depending on the situation, it will generate, initialize, and return the right object to the client. The client can then use the object by calling the appropriate methods and properties.

Example of Creational Design Patterns

* Singleton Design Pattern
* Factory Method Design Pattern
* Abstract Factory Design Pattern
* Prototype Design Pattern
* Builder Design Pattern

Structural Design Pattern

Structural Design Patterns are design patterns that make design easier by establishing a simple way to represent the link between entities. Simply put, the Structural Design Pattern is used to manage the structure of classes and interfaces, as well as the relationships between classes.

In real applications, we may need to change the structure of a class or the interaction between classes, but we do not want this change to be influenced by the project. For example, suppose we have two classes: User and Product. And the Product class is utilized within the User class to create one-to-many relationships between the User and the Product. Tomorrow, the structure and interactions between these two classes will change. The customer now wishes to separate the Product and User classes so that they can be used independently. This is a structural shift, and we do not want it to impact our project. This is where the Structural Design Pattern can help us.

Example of Structural Design Pattern

* Adapter Design Pattern
* Facade Design Pattern
* Decorator Design Pattern
* Bridge Design Pattern
* Composite Design Pattern
* Proxy Design Pattern
* Flyweight Design Pattern

Behavioral Design Pattern

The Behavioral Design Patterns aim to improve inter-object communication. These patterns are vital for promoting clear and effective communication between objects, making systems easier to comprehend and administer. They concentrate on the duties of objects and how they communicate, hence reducing tight coupling.

In real applications, we occasionally want to change the behavior of a class without affecting other classes in the project. For example, we have an Invoice class that currently applies taxes at 10%. Tomorrow, if we want to impose another tax. That suggests we are altering the behavior of a class. The Behavioral Design Pattern is used to address such behavioral difficulties. Implementing Behavioral Design Patterns can assist manage and optimize the interactions between objects and behaviors in an application.

Example of Behavioral Design Patterns

* Chain of Responsibility Pattern
* Command Pattern
* Interpreter Pattern
* Mediator Pattern
* Memento Pattern