write short notes about design patterns.

Design patterns are used to represent some of the best practices adapted by experienced object-oriented software developers. A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples.

1) Creational

These design patterns are all about class instantiation or object creation. These patterns can be further categorized into Class-creational patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

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.

Use case of creational design pattern-

I. Suppose a developer wants to create a simple DB Connection class to connect to a database and wants to access the database at multiple locations from code, generally what the developer will do is create an instance of DB Connection class and use it for doing database operations wherever required. This results in creating multiple connections from the database as each instance of DB Connection class will have a separate connection to the database. To deal with it, we create DB Connection class as a singleton class, so that only one instance of DB Connection is created, and a single connection is established. Because we can manage DB Connection via one instance, we can control load balance, unnecessary connections, etc.

II. Suppose you want to create multiple instances of a similar kind and want to achieve loose coupling then you can go for Factory pattern. A class implementing factory design patterns works as a bridge between multiple classes. Consider an example of using multiple database servers like SQL Server and Oracle. If you are developing an application using SQL Server database as back end, but in the future need to change the database to the oracle, you will need to modify all your code, so as factory design patterns maintain loose coupling and easy implementation, we should go for the factory design pattern to achieve loose coupling and the creation of a similar kind of object.

2) Structural
   These design patterns are about organizing different classes and objects to form larger structures and provide new functionality.

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.

Use Case of Structural Design Pattern.

- When 2 interfaces are not compatible with each other and want to establish a relationship between them through an adapter it's called an adapter design pattern. The adapter pattern converts the interface of a class into another interface or class that the client expects, i.e., adapter lets classes work together that could not otherwise because of incompatibility. So, in these types of incompatible scenarios, we can go for the adapter pattern. Use Case of Structural Design Pattern

3) Behavioral

Behavioral patterns are about identifying common communication patterns between objects and realizing these 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.

Use Case of Behavioral Design Pattern-

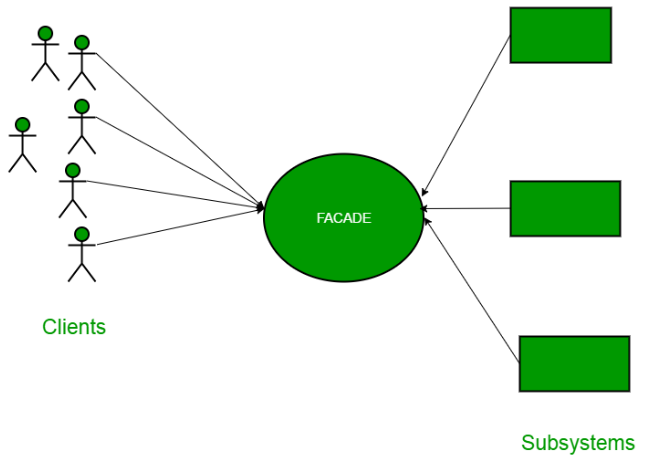
- The template pattern defines the skeleton of an algorithm in an operation deferring some steps to sub-classes. The template method lets subclasses redefine certain steps of an algorithm without changing the algorithm structure. For example, in your project, you want the behavior of the module to be able to extend, such that we can make the module behave in new and different ways as the requirements of the application change, or to meet the needs of new applications. However, no one is allowed to make source code changes to it, i.e., you can add but can't modify the structure in those scenarios a developer can approach template design pattern.

short notes on Facade design patterns

Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

For examples

In Java, the interface JDBC can be called a facade because we as users or clients create connections using the "java.sql.Connection" interface, the implementation of which we are not concerned about. The implementation is left to the vendor of the driver. Another good example can be the startup of a computer. When a computer starts up, it involves the work of CPU, memory, hard drive, etc. To make it easy to use for users, we can add a facade that wraps the complexity of the task and provide one simple interface instead. The same goes for the Facade Design Pattern. It hides the complexities of the system and provides an interface to the client from where the client can access the system.



Facade Design Pattern Diagram

References

- https://www.geeksforgeeks.org/introduction-to-pattern-designing/
- https://www.netsolutions.com/insights/software-design-pattern/
- https://www.geeksforgeeks.org/facade-design-pattern-introduction/
-