**Structural Design Patterns**

**FACADE DESIGN PATTERN:**

- It is a cool design pattern that cleans up your interaction with complex systems.

- Provides simple interface to complex system

- Minimizes Communication between dependencies between systems.

- It is more like a helper for the client applications. It doesn't hide subsystem interfaces from the client. Whether to use facade or not is completely dependent on client code.

Real Use Case :

1. Hotel Frontdesk which communicates to other departments without the guest knowing what is happening at the backend of other department.

2. Start Button on Computer which takes care of staring and all the system components are loaded in the memory and ready to use.For example, loading the boot sector from the hard drive, loading the OS into memory, and initializing the CPU

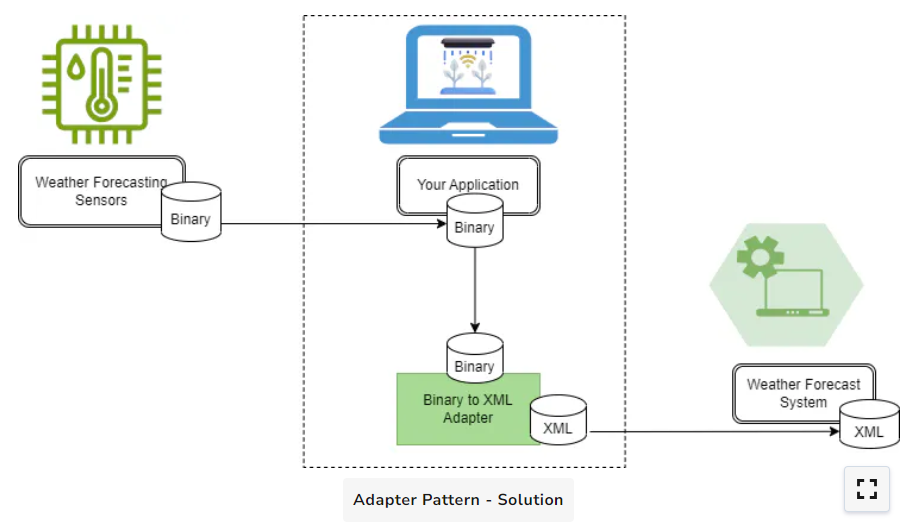
**ADAPTER DESIGN PATTERN:**

- Adapter pattern is used to make two incompatible or unrelated interfaces work together.

- Object that joins this unrelated interfaces is called an Adapter.

**Real-World Example:**

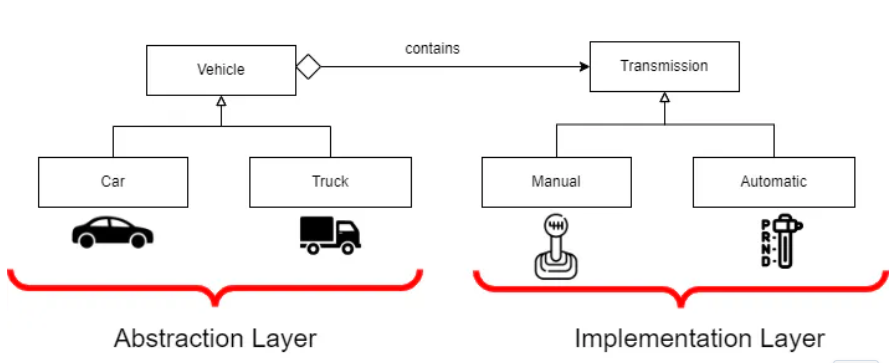
1. In Weather Forecast systems where application collect data from the several sensors in binary format and wanted it to display on the user friendly interface which accepts data in XML format and not binary format.For the sensor and the prediction algorithm to work together seamlessly and be compatible, the Adapter is essential in transforming the binary input into XML.



1. Diplomatic Meeting in which two diplomats from different countries speaking different languages are communicating with each other with the help of translator. Here, the translator translates the speech from both diplomats and acts as an adapter between two incompatible persons.

**Bridge Pattern**

* Separate abstraction (high-level control logic) from its implementation (low-level functional logic) allowing both to act independently.
* This separation increases modularity and improves code maintainability.
* The Bridge Pattern defines abstraction as the high-level layer with which the client interacts. It defines the abstract interface and keeps a reference to an implementation layer object.



* The class Vehicle is an abstract one. It is a representation of a generic vehicle that has a reference to an implementation layer Transmission object. Subclasses of Vehicle, such as Truck, Bus, and Car, are also included in it. Although they may have additional features or behaviors of their own, they nonetheless inherit high-level behaviors.
* The implementation layer consists of the Transmission interface and its implementations (Manual, Automatic, SemiAutomatic). These classes offer particular functions for different transmission systems, like gear shifting.

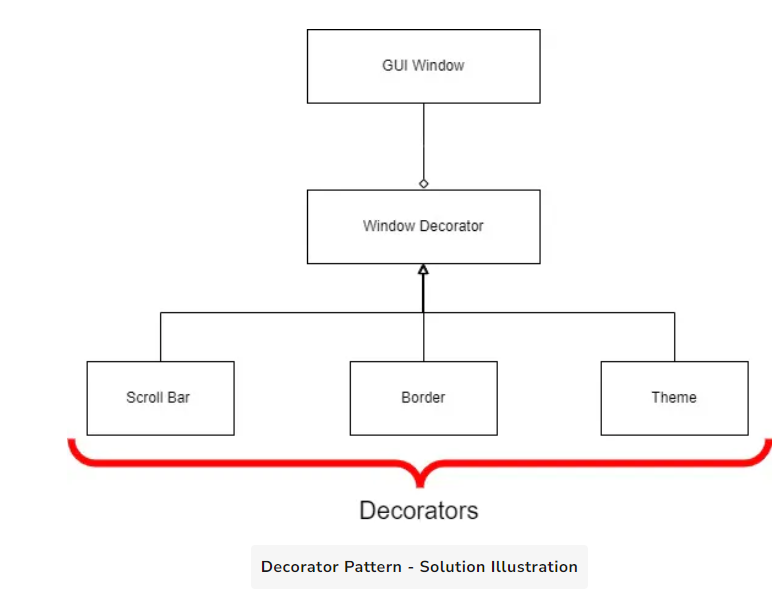
Real-World Example:

* Remote Control System with multiple electronic devices. (universal remote control system)
* The universal remote control system acts as an abstraction layer of the Bridge Pattern. It provides some of the basic functionalities like play, stop, volume up or volume down, etc, but does not directly handle the specific operations of each device.
* The electronic devices acts as an Implementation layer where each device has its own particular way of handling different functionalities

This configuration exemplifies the fundamental idea of the Bridge Pattern: you may add new devices or modify current ones without changing the design of the remote control. The remote (abstraction) can operate many devices (implementations). Similar to this, the Bridge Pattern in software architecture enables an abstraction to communicate with different implementations, promoting both layers' autonomous evolution and adaptability.

Decorator Pattern:

* Closed for modification and Open for extension.
* Enables to dynamically add new functionalities to objects without changing the structure.
* A new decorator class is generated that wraps the original class and adds new functionality. This means the original class code remains unchanged
* Create new decorator classes that encapsulate the original class and add new behaviors.



Proxy Pattern:

* Proxy means ‘in place of’ representing or ‘on behalf of’
* It control access to other object.
* Can be used to manage complex operations, controlling resource access or handling Network Communication
* We use proxy pattern in situation in which you have a highly valuable or complex object. You wouldn't want everyone to have direct access to it. Perhaps you just want to add some extra logic, like access control or lazy initialization, or it's too sensitive or heavy.
* Consider a heavy Java Object (like JDBC connection or session Factory) that requires some initial configuration. We only want such objects to be initialize only on demand and once thy are we want to reuse then for all calls.