**Architectural Style**

The architectural style shows how do we organize our code, or how the system will look like from 10000 feet helicopter view to show the highest level of abstraction of our system design. Furthermore, when building the architectural style of our system we focus on layers and modules and how they are communicating with each other.

There are different types of architectural styles, and moreover, we can mix them and produce a hybrid style that consists of a mix between two and even more architectural styles. Below is a list of architectural styles and examples for each category:

* **Structure architectural styles:** such as layered, pipes and filters and component-based styles.
* **Messaging styles:** such as Implicit invocation, asynchronous messaging and publish-subscribe style.
* **Distributed systems:** such as service-oriented, peer to peer style, object request broker, and cloud computing styles.
* **Shared memory styles:** such as role-based, blackboard, database-centric styles.
* **Adaptive system styles:** such as microkernel style, reflection, domain-specific language styles.

**Architectural Patterns**

The architectural pattern shows how a solution can be used to solve a reoccurring problem. In another word, it reflects how a code or components interact with each other. Moreover, the architectural pattern is describing the architectural style of our system and provides solutions for the issues in our architectural style. Personally, I prefer to define architectural patterns as a way to implement our architectural style. For example: how to separate the UI of the data module in our architectural style? How to integrate a third-party component with our system? how many tires will we have in our client-server architecture? Examples of architectural patterns are microservices, message bus, service requester/ consumer, MVC, MVVM, microkernel, n-tier, domain-driven design, and presentation-abstraction-control.

**Design Patterns**

Design patterns are accumulative best practices and experiences that software professionals used over the years to solve the general problem by – trial and error – they faced during software development. The Gang of Four (GOF, refers to Eric Gamma, Richard Helm, Ralf Johnson, and John Vlissides) wrote a book in 1994 titled with “Design Pattern – Elements of reusable object-oriented software” in which they suggested that design patterns are based on two main principles of object-oriented design:

* Develop to an interface, not to an implementation.
* Favor object composition over inheritance.

Also, they presented that the design patterns set contains 23 patterns and categorized into three main sets:

**1. Creational design patterns:**  
Provide a way to create objects while hiding the creation logic. Thus, the object creation is to be done without instantiating objects directly with the “New” keyword to gives the flexibility to decide which objects need to be created for a given use case. The creational design patterns are:

* **Abstract factory pattern:** provide an interface to create objects without specifying the classes.
* **Singleton pattern:** provide only a single instance of the calls and global access to this instance.
* **Builder Pattern:** Separate the construction from representation and allows the same construction to create multiple representations.
* **Prototype pattern:** creating duplicate without affecting the performance and memory. So the duplicate object is built from the skeleton of an existing object.

**2. Structural patterns:**

Concerned with class and object composition. The Structural design patterns are:

* **Adapter pattern:** it works as a bridge between two incompatible interfaces and compines their capabilities.
* **Bridge pattern:** provide a way to decouple the abstraction from its implementation.
* **Filter pattern:** Also known as criteria pattern, it provides a way to filter a set of objects using different criteria and chaining them in a decoupled way through logical operations.
* **Composite pattern:** provide a way to treat a group of objects in a similar way as a single object. It composes objects in term of a tree structure to represent part as well as a whole hierarchy
* **Decorator pattern:** allows adding new functionality to an existing object without altering its structure.
* **Façade pattern:** provide a unified interface to a set of interfaces.it hides the complexities of the system and provides an interface to the client using which the client can access the system.
* **Flyweight pattern:** reduce the number of objects created and to decrease memory footprint and increase performance. It helps in reusing already existing similar kind objects by storing them and creates a new object when no matching object is found.
* **Proxy pattern:** provides a place holder to another object to control access to it. The object has an original object to interface its functionality to the outer world.

**3. Behavioral patterns:**

Behavioral patterns are concerned with communications between objects. The following is the list of behavioral patterns:

* **Responsibility pattern:** creates a chain of receiver objects for a request. This pattern decouples the sender and receiver of a request based on the type of request.
* **Command pattern:** it’s a data-driven pattern in which A request is wrapped under an object as command and passed to an invoker object.
* **Interpreter pattern:** provides a way to evaluate language grammar or expression. It involves implementing an expression interface that tells to interpret a particular context. This pattern is used in SQL parsing, symbol processing engine, etc.
* **Iterator pattern:** provides a way to access the elements of a collection object in a sequential manner without any need to know its underlying representation.
* **Mediator pattern:** used to reduce communication complexity between multiple objects or classes. It provides a mediator class that normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling.
* **Memento pattern:** used to restore the state of an object to a previous state.
* **Observer pattern:** used when there is a one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically.
* **State pattern:** is used to change the class behavior based on its state.
* **Null object pattern:** helps to avoid null references by having a default object.
* **Strategy pattern:** provides a way to change class behavior or its algorithm at run time.
* **Template pattern:** an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class.
* **Visitor pattern:**used to change the executing algorithm of an element class.