**ARQUITECTURAS**

1. **TERCER NIVEL:**

La arquitectura de software denomina el tercer nivel como su nombre lo explica se trata de una arquitectura que como indica su nombre lo indica que es una arquitectura de tres niveles

El nivel uno o el nivel más básico es en el que el cliente puede ver como se ve su pagina web o programa

El segundo nivel es en donde esta los servidores remotos el nivel de la lógica

El tercer nivel es en donde se encuentra toda la programación de la pagina web las bases de datos y servidores la parte que el cliente no ve

Es una arquitectura que usa un arquitecto de software para desarrollar un programa

1. **POR CAPAS**

Es una arquitectura en donde se indican los roles de cada persona en donde se otorgan roles a cada una de las personas en donde esta de forma jerarquica y cada persona o cada rol tiene su responsabilidad en donde cada capa representa una responsabilidad mas al nivel de abajo

Es una arquitectura que usa un arquitecto de software para desarrollar un programa

1. **MICROSERVICIOS**

Esta arquitectura sirve para dar soluciones a los problemas inherentes a los sistemas monolíticos es una arquitectura diseñada para empresas pequeñas

Es una arquitectura que usa un arquitecto de software para desarrollar un programa

1. **P2P**

La arquitectura P2P es una arquitectura en donde se ve como se conecta un computador o varios computadores a un servidor para usar un servicio juego o programa que se conecta directamente

Es una arquitectura que usa un arquitecto de software para desarrollar un programa

1. **PIZARRA**

Es una arquitectura de software es en donde un computador celular o algún dispositivo se pueda conectar de forma correcta he independiente es muy parecido al P2P

Es una arquitectura que usa un arquitecto de software para desarrollar un programa

1. **EVENTOS**

Esta arquitectura puede ser definida como un cambio significado de estado en la arquitectura el ejemplo mas claro de este tipo de arquitectura es por ejemplo cuando un cliente vende un coche pasa de “se vende” a “vendido”

Es una arquitectura que usa un arquitecto de software para desarrollar un programa

1. **TUBOS Y FILTROS**

Este tipo de arquitectura es en donde se recibe cierta información y se va filtrando lo que va necesitando el cliente para poder ir sacando la información mas relevante y se da una respuesta dependiendo los filtros o lo que busca específicamente el cliente

Es una arquitectura que usa un arquitecto de software para desarrollar un programa

1. **CLIENTE-SERVIDOR**

Este tipo de arquitectura es en donde una persona se conecta directamente a los servidores para conseguir un servicio ejemplo una pagina web como youtube una persona por medio de un dispositivo a un servidor y consume un servicio

Es una arquitectura que usa un arquitecto de software para desarrollar un programa

1. **PUBLICADOR-SUSCRIPTOR**

Este tipo de arquitectura hace referencia a lo que una persona publica y una determinada cantidad de personas lo ven el ejemplo mas claro seria Instagram o Facebook en donde una persona publica una foto o una información relevante y las personas que lo siguen pueden ver dar me gusta o no me gusta a lo que esta persona publica

Es una arquitectura se usa de las personas que viven de las redes sociales

1. **ABSTRACT FACTORY**

Es la que busca agrupar un conjunto de clases que tienen una función en común llamada familia la cuales son creadas mediante un Factory este tipo de patrón es especialmente útil cuando se requieren causar ciertas familias de clases para resolver un problema

1. **FACTORY METHBOD**

Permite la creación de objetos de un subtipo determinado a trabes de un Factory esto es especialmente útil cuando no sabemos en tiempo de diseño el subtipo que vamos a utilizar o cuando queremos delegar la lógica de creación de los objetos a una clase Factory

1. **BUILDER**

Es uno de los patrones más simples pero más útiles el cual nos permite crear objetos complejos a través de uno mas simple es muy común encontrarnos con situaciones en las cuales tengamos que crear objetos de forma manual y repetidas veces que nos lleva a que establecer cada propiedad del objeto y si además este tiene objetos compuestos dentro

1. **SINGLENTON**

Recibe su nombre y su característica se debe a que solo puede tener una instancia para tosa la aplicación de una determinada clase, este se puede llegar a lograr restringiendo la libre creación de instancias de esta clase mediante un operador nuevo e imponiendo un constructor privado y un método estático para poder obtener la instancia

1. **PROTOTYPE**

Este tipo basa su funcionalidad en la clonación de objetos, esos nuevos objetos son creados mediante un pool de prototipos elaborados previamente y almacenados este patrón es más útil cuando se necesita crear un objeto ya existente y ya establecido

1. **DEPENDENCY**

Este patrón como su nombre lo indica tiene una dependencia de otros objetos ya creados y definido anteriormente

1. **INJECTION**

Siempre he sido uno de los conceptos que cuesta entender en el mundo del desarrollo de software sobre todo a la gente que esta empezando normalmente cuando nosotros programamos día a día con la programación orientada objetos nos encontramos a construyendo objetos y relacionando objetos utilizando dependencia

1. **ADEPTER**

Es cuando tenemos interfaces de software incompatibles las cuales a pesar de su incompatibilidad tiene una funcionalidad similar este patrón es implementado cuando se desea poner de forma compatibles la interfaz que son incompatibles

1. **BRIDGE**

Es un patrón de diseño utilizado para desacoplar una abstracción de su implementación de manera que las dos puedan ser modificadas por separado sin necesidad de modificar la otra; dicho de otra manera, se desacopla una abstracción de su implementación para que puedan variar independientemente.

1. **COMPOSITE**

El patrón de diseño Composite nos sirve para construir estructuras complejas partiendo de otras mucho más simples; dicho de otra manera, podemos crear estructuras compuestas que están conformadas por otras estructuras más pequeñas.

1. **DECORATOR**

El patrón decorator está diseñado para solucionar problemas donde la jerarquía con subclasificación no puede ser aplicada, o se requiere de un gran impacto en todas las clases de la jerarquía con el fin de poder lograr el comportamiento esperado. Decorator permite al usuario añadir nuevas funcionalidades a un objeto existente sin alterar su estructura, mediante la adición de nuevas clases que envuelven a la anterior dándole funcionamiento extra.

1. **FACADE**

El patrón Facade (fachada) tiene la característica de ocultar la complejidad de interactuar con un conjunto de subsistemas proporcionando una interface de alto nivel, la cual se encarga de realizar la comunicación con todos los subsistemas necesarios

1. **FLY WEITGHT**

Flyweight es un patrón que centra su atención en la construcción de objetos ligeros, mediante la abstracción de las partes reutilizables que pueden ser compartidas con otros objetos, esto con el fin de que en lugar de crear objetos cada vez que sea requerido, podamos reutilizar objetos creados por otras instancias logrando con ello reducir en gran medida la capacidad de memoria requerida por la aplicación

1. **FRONT CONTROLER**

Patrón que centraliza el acceso de las peticiones provenientes del cliente.

1. **CHAIN OF RESPONSABILITY**

El patrón de diseño Cadena de responsabilidad (Chain of Responsability) es un patrón que se distingue por su versatilidad, permitiendo resolver problemas donde no estamos muy seguros de qué objeto deberá procesar una solicitud concreta; este patrón de diseño resuelve problemas fácilmente donde la herencia no puede. Apoyándose de una estructura en forma de cadena donde una secuencia de objetos tratan de atender una petición.

1. **COMMAND**

El patrón de diseño Command nos permite ejecutar operaciones sin conocer los detalles de la implementación de la misma. Las operaciones son conocidas como comandos y cada operación es implementada como una clase independiente que realiza una acción muy concreta, para lo cual, puede o no recibir parámetros para realizar su tarea.

1. **INTERPRETER**

El patrón de diseño interpreter es utilizado para evaluar un lenguaje definido como Expresiones, este patrón nos permite interpretar un lenguaje como Java, C#, SQL o incluso un lenguaje inventado por nosotros el cual tiene un significado; y darnos una respuesta tras evaluar dicho lenguaje

1. **INTERATOR**

Este patrón de diseño permite recorrer una estructura de datos sin que sea necesario conocer la estructura interna de la misma. Es especialmente útil cuando trabajamos con estructuras de datos complejas, ya que nos permite recorrer sus elementos mediante un Iterador, el Iterador es una interface que proporciona los métodos necesarios para recorrer los elementos de la estructura de datos, los métodos más comunes son:

hasNext: Método que regresa un booleano para indicar si existen más elementos en la estructura por recorrer. True si existen más y false si hemos llegado al final y no hay más elementos por recorrer.

next: Regresa el siguiente elemento de la estructura de datos.

1. **MEMENTO**

Memento es un patrón diseñado que nos permite capturar el estado de un objeto en un momento determinado con la finalidad de regresar a este estado en cualquier momento. Este patrón es utilizado cuando tenemos objetos que cambian en el tiempo y por alguna razón necesitamos restaurar su estado en un momento determinado

1. **NULL OBJECT**

El patrón de diseño Null Object nace de la necesidad de evitar los valores nulos que puedan originar error en tiempo de ejecución. Básicamente lo que este patrón propone es utilizar instancias que implementen la interface requerida pero con un cuerpo vacío en lugar de regresar un valor null

1. **OBSERVER**

El patrón de diseño Observer permite observar los cambios producidos por un objeto, de esta forma, cada cambio que afecte el estado del objeto observado lanzará una notificación a los observadores; a esto se le conoce como Publicador-Suscriptor