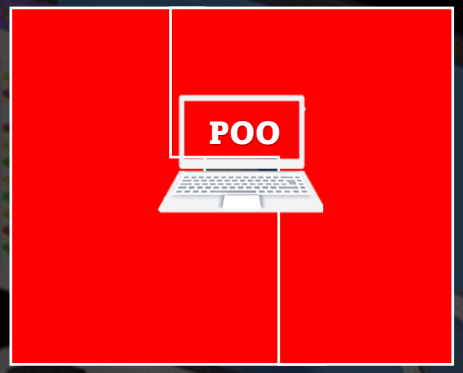
Modelo Vista Controlador (MVC) y Patrones de Diseño



NRC: 1322

Integrantes:

- Arequipa Tigasi Edwin Fernando
- López Morales Dylan Joel
- Pugachi Suarez Karol Elizabeth
- Yar Zumba Evelyn Cristina

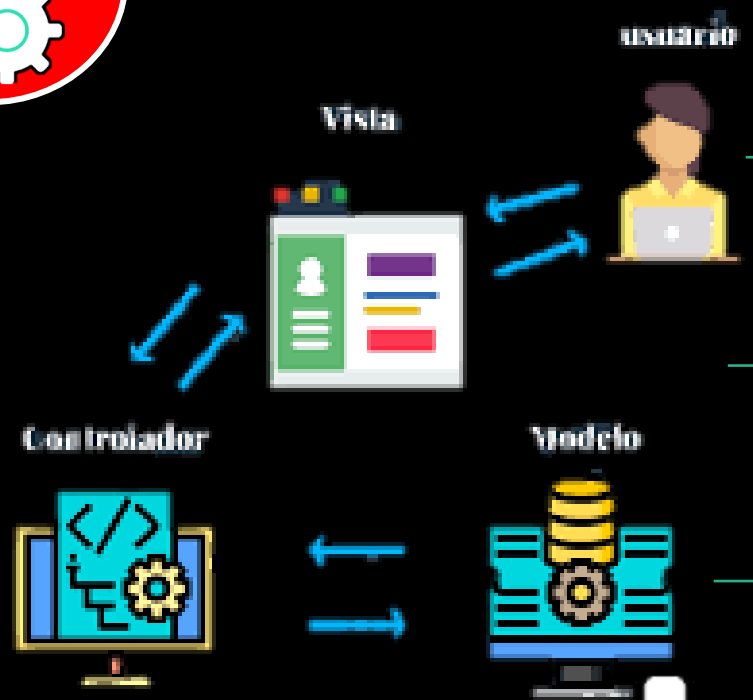
PROGRAMACIÓN ORIENTADA A OBJETOS

Ing. Luis Enrique Jaramillo Montaño

ESPE

POO

PATRÓN MODELO VISTA CONTROLADOR



MODELO: Representa los datos y la lógica de negocio.

VISTA: Es el diseño y presentación de los datos que el usuario visualiza.

CONTROLADOR: Controla, recibe las órdenes del usuario y se encarga de solicitar los datos al modelo y de comunicárselos a la vista.

Para qué sirve?

Se utiliza para separar las responsabilidades de una aplicación en tres componentes: el modelo, la vista y el controlador

POO

PATRÓN MODELO VISTA CONTROLADOR

Cómo mejora el código?

- Separación de responsabilidades
- Reutilización de componentes
- Facilita la incorporación de nuevas características
- Aumenta la capacidad de prueba del código

Cuándo se utiliza?

- En aplicaciones web complejas, que requieren un alto control sobre el comportamiento.
- En aplicaciones web con equipos grandes de desarrolladores

Cuándo no se utiliza?

- En aplicaciones donde se desea crear una SPA o un cliente de una API
- En aplicaciones donde una arquitectura de dos o tres niveles es más adecuada

Dylan López

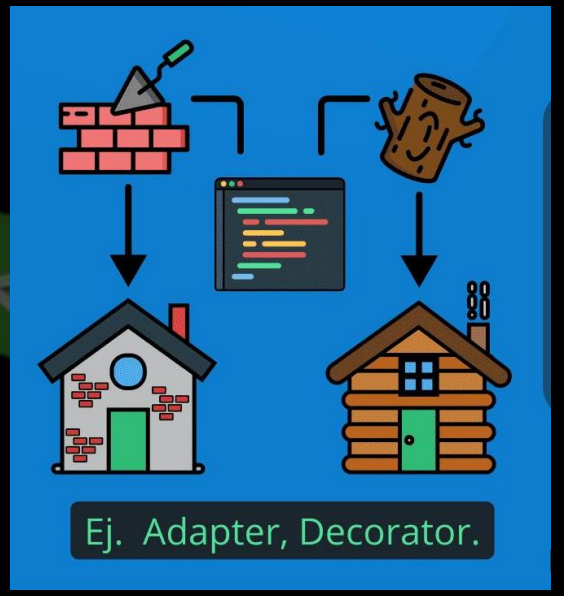
ESPE

PATRÓN ESTRUCTURAL: Adapter Decorator

Los patrones estructurales son un tipo de patrón de diseño en programación que se enfocan en la composición de clases y objetos para facilitar la reutilización y la flexibilidad del código.

1. ¿Cómo mejoran la organización del código?
- Promueven un código modular y bien estructurado.
- Facilitan la reutilización al definir relaciones claras entre clases y objetos.
- Mejoran la mantenibilidad al separar responsabilidades y reducir dependencias acopladas.
- Hacen que el código sea más fácil de leer y comprender.

2. ¿Para qué sirven?
- Para definir relaciones eficientes entre clases y objetos sin modificar su implementación.
- Para facilitar la interacción entre estructuras complejas.
- Para mejorar la extensibilidad del sistema, permitiendo agregar nuevas funcionalidades sin alterar el código existente.



Evelyn Yar

PATRÓN ESTRUCTURAL

3. ¿Cuándo se aplican y cuándo no?

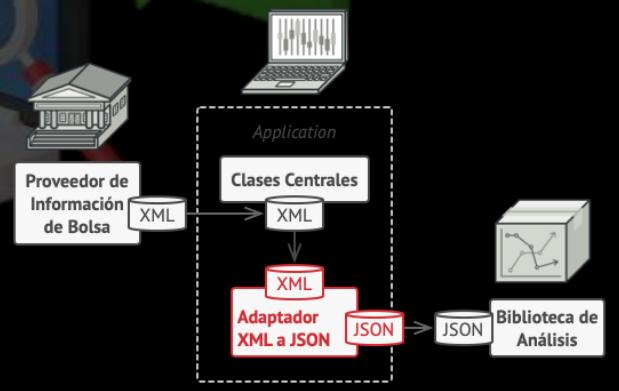
Se aplican cuando:

- Se necesita organizar múltiples clases u objetos de manera eficiente.
- Se requiere reutilización de código sin alterar su implementación original.
- Se quiere mantener un bajo acoplamiento entre los componentes del sistema.

No se aplican cuando:

- La solución es simple y no justifica la sobrecarga de un patrón.
- No hay necesidad de cambiar la estructura de clases u objetos a futuro.
- Se puede resolver el problema con una solución más directa y menos abstracta.

Ejemplo:
Imagina que tienes un enchufe europeo, pero viajas a EE.UU. Necesitas un adaptador. Lo mismo ocurre en la programación cuando dos clases no son compatibles, y ahí es donde entra el patrón Adaptador."



Evelyn Yar

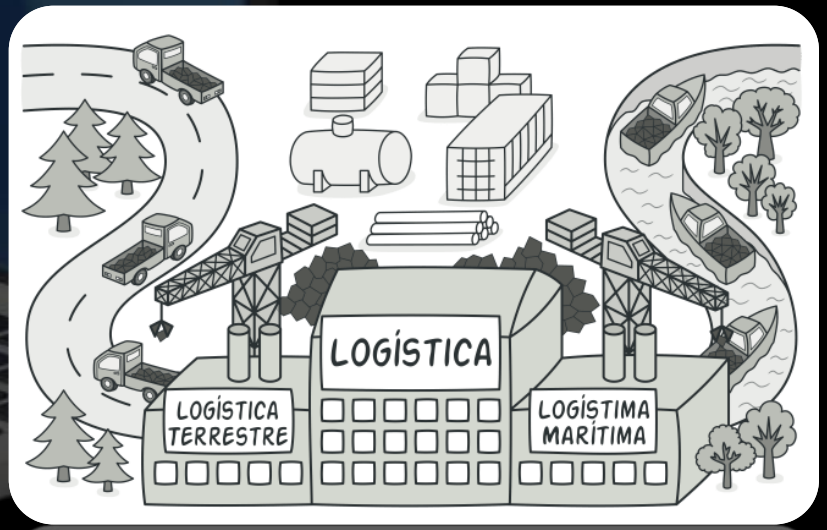
POO

PATRÓN CREACIONAL: Factory Method – Método de Fabrica

Los patrones creacionales proporcionan mecanismos de creación de objetos que incrementan la flexibilidad y reutilización de código existente.

Factory Method – Método de Fabrica.

Es un patrón de diseño creacional que proporciona una interfaz para crear objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.
Permite la creación de objetos de un subtipo determinado a través de una clase Factory.



Edwin Arequipa

ESPE

PATRÓN CREACIONAL: Factory Method

El patrón Factory Method mejora la organización del código al separar el código de construcción de productos del código que hace uso del producto. Es decir que el patrón ayuda a facilitar la extensión del código de construcción de forma independiente al resto del código.

¿Para qué sirve?	Permite la creación de objetos de un subtipo determinado a través de una clase Factory.
	Permite añadir nuevos tipos de objetos sin modificar el código existente.
	Permite alterar el tipo de objetos que se crearan por medio de las subclases.
	Permite manejar de forma genérica diferentes tipos de objetos.
	Facilita la creación de código modular y reutilizable.

PATRÓN CREACIONAL: Factory Method

¿Cuándo se aplica y cuando no?	
• Se aplica o se utiliza el método cuando no se conoce de antemano las dependencias y los tipos exactos de los objetos con los que deba funcionar el código. • También se aplica cuando se quiere ofrecer al usuario una biblioteca o framework para poder extender los componentes internos.	• No se aplica el método cuando no se conoce el subtipo que se va a utilizar en un tiempo de diseño. • Cuando no es necesario desacoplar la creación de objetos. • Para las aplicaciones donde las clases no van a cambiar, un Factory Method podría ser un exceso. • Si la creación de los objetos es simple y no requiere distintas variables, el método seria innecesario.

PATRÓN CREACIONAL: Factory Method

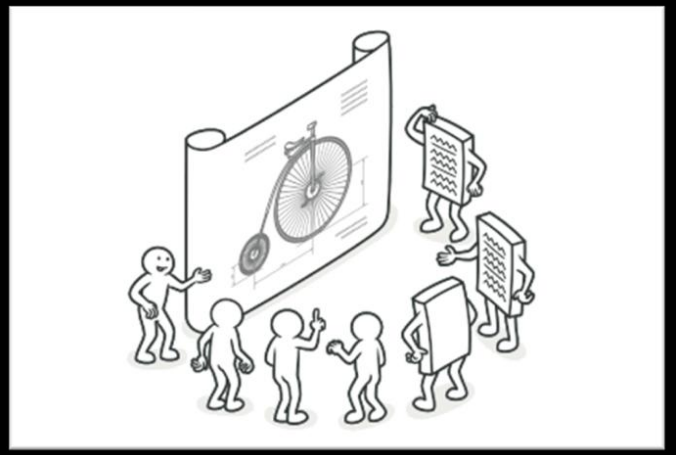
Ejemplo en JAVA

```java
// Interfaz Transporte
public interface Transporte {
    void mover();
}
// Clase concreta: Coche
public class Coche implements Transporte{
    @Override
    public void mover() {
        System.out.println("El coche se está moviendo en la carretera.");
    }
}
// Clase concreta: Bicicleta
public class Bicicleta implements Transporte {
    @Override
    public void mover() {
        System.out.println("La bicicleta se está moviendo por el carril bici.");
    }
}
// Clase abstracta que define el Factory Method
public abstract class Fabrica {
    public abstract Transporte crearTransporte();
}
// Fábrica concreta: Coche
public class FabricaCoche extends Fabrica {
    @Override
    public Transporte crearTransporte() {
        return new Coche();
    }
}
// Fábrica concreta: Bicicleta
public class FabricaBicicleta extends Fabrica {
    @Override
    public Transporte crearTransporte() {
        return new Bicicleta();
    }
}
// Cliente que utiliza la fábrica
public class Cliente {
    public static void main(String[] args) {
        Fabrica fabricaCoche = new FabricaCoche();
        Fabrica fabricaBicicleta = new FabricaBicicleta();

        System.out.println("Coche:");
        Transporte coche = fabricaCoche.crearTransporte();
        coche.mover();

        System.out.println("\nBicicleta:");
        Transporte bicicleta = fabricaBicicleta.crearTransporte();
        bicicleta.mover();
    }
}
```


Edwin Arequipa

PATRÓN COMPORTAMIENTO: Interpreter

Los patrones de comportamiento se enfocan en la comunicación y la interacción entre objetos. El Patrón Observador permite establecer una relación de dependencia entre un objeto (sujeto) y múltiples observadores, de manera que todos sean notificados automáticamente cuando el estado del sujeto cambie.



Se utiliza para definir la gramática de un lenguaje y proporcionar un intérprete para procesar las instrucciones en ese lenguaje.

```
namespace Interprete21
{
    class Contexto
    {
        private string expresion;
        private int valor;

        public string Expresion { get => expresion; set => expresion = value; }
        public int Valor { get => valor; set => valor = value; }

        // Colocamos la expresion a interpretar
        public Contexto(string pExpresion)
        {
            expresion = pExpresion;

            Console.ForegroundColor = ConsoleColor.White;
            Console.WriteLine("Se evaluara {0}", expresion);
        }
```

POO

PATRÓN COMPORTAMIENTO: Interpreter

Los patrones de comportamiento se enfocan en la comunicación y la interacción entre objetos. El Patrón Observador permite establecer una relación de dependencia entre un objeto (sujeto) y múltiples observadores, de manera que todos sean notificados automáticamente cuando el estado del sujeto cambie.

¿Cómo mejora la organización del código?

- ☑ Facilita un código modular y flexible.
- ☑ Reduce el acoplamiento entre clases, permitiendo que trabajen de forma independiente.
- ☑ Permite la reactividad en los sistemas, notificando cambios en tiempo real.
- ☑ Mejora la mantenibilidad y escalabilidad del software

¿Para que sirve?

- ✔ Se usa para establecer comunicación eficiente entre objetos sin acoplarlos directamente.
- ✔ Es ideal para sistemas de eventos, interfaces gráficas y modelos de publicación/suscripción.
- ✔ Permite manejar múltiples suscriptores sin modificar la lógica del sujeto.

PATRÓN COMPORTAMIENTO: INTERPRETER

¿Cuándo no se aplica?

❌ No es necesario que varios objetos respondan a un mismo evento.
❌ El número de observadores es pequeño y conocido de antemano.
❌ Se busca una solución más simple sin sobrecarga de patrones.

¿Cuándo se aplica?

☑ Se necesita que múltiples objetos reaccionen a cambios en otro objeto.
☑ Se diseñan sistemas dinámicos y escalables (notificaciones, juegos, interfaces).
☑ Se trabaja con modelos publicador-suscriptor en arquitectura de software.