



UNIVERSIDAD
Popular del cesar

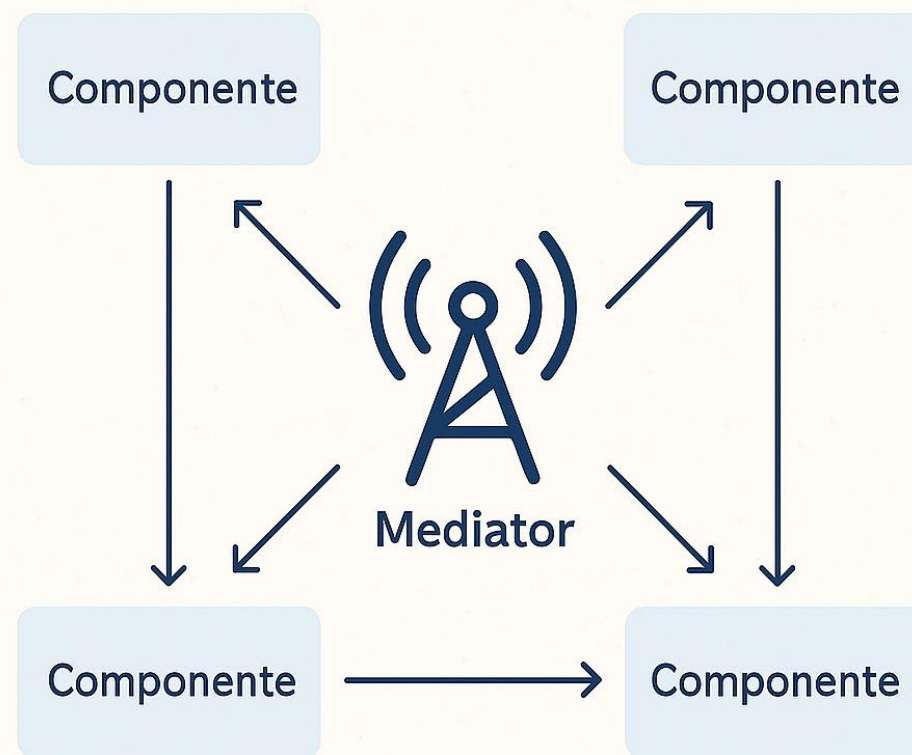
Ingeniería de Sistemas

OLIMPO CASTRO THORNE
JHONNIER JOSE DE LA CRUZ
JESUS DAVID HERNANDEZ R
WILMER GILDARDO HERRERA Y
JUAN DE LA CRUZ LUQUEZ



Mediator

- Es un patrón de diseño de comportamiento cuyo objetivo principal es centralizar la comunicación entre múltiples objetos interdependientes. En lugar de que estos objetos se conozcan y se comuniquen directamente entre sí, lo hacen a través de un componente intermediario: el mediador.

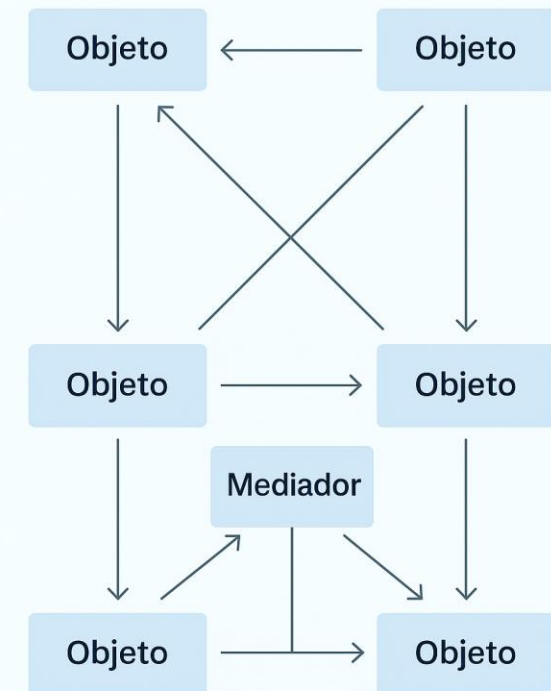


Propósitos

- **Reducir el acoplamiento** entre los objetos participantes, promoviendo su independencia.
- **Simplificar la evolución y el mantenimiento** del sistema, ya que los cambios en las interacciones no requieren modificar los objetos individuales.
- **Concentrar las reglas de negocio o de coordinación** en un solo punto, facilitando el control del flujo de comunicación.

Problemas que Resuelve

- Acoplamiento excesivo: Los objetos dependen demasiado entre sí.
- Complejidad en la comunicación: Las relaciones cruzadas dificultan el mantenimiento.
- Baja reutilización: Los componentes no pueden usarse en otros contextos.
- Propagación de errores: Un cambio puede afectar múltiples partes.
- Código difícil de escalar: Agregar o modificar comportamientos se vuelve riesgoso.



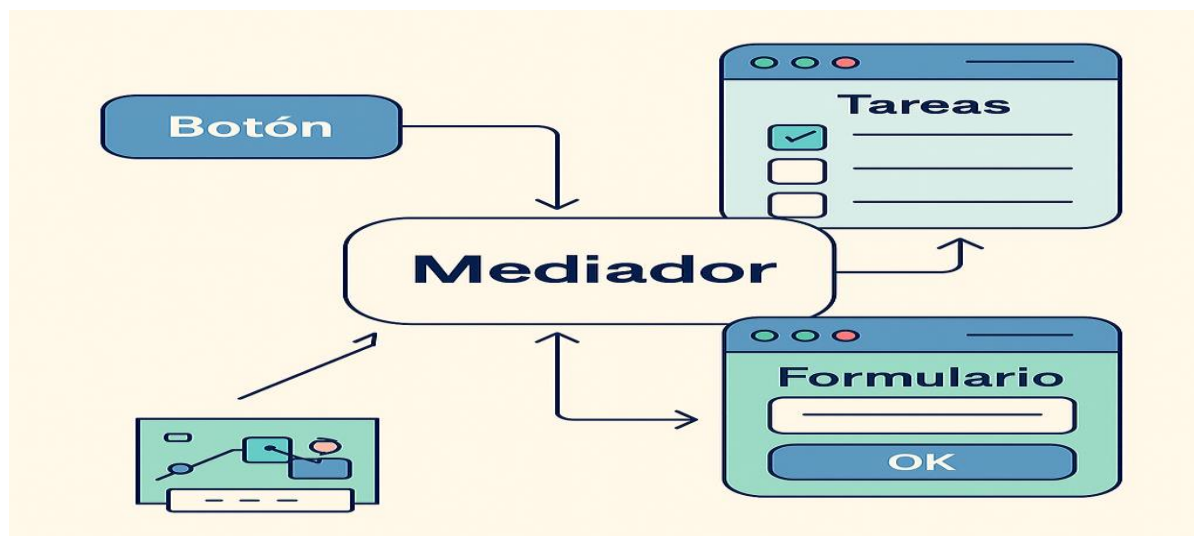
Mediator soluciona esto centralizando

Cuándo Implementarlo

- El patrón **Mediator** resulta especialmente útil cuando el sistema presenta una **comunicación compleja entre múltiples objetos**, generando una red de dependencias difíciles de gestionar.
- **Múltiples componentes necesitan interactuar** entre sí de forma dinámica, pero se desea evitar conexiones directas para mantener bajo acoplamiento.
- **La lógica de coordinación** entre objetos empieza a mezclarse con su lógica interna, afectando la cohesión y dificultando el mantenimiento.

Cuándo Implementarlo

- **El número de relaciones crece de forma desproporcionada** a medida que se añaden nuevos elementos al sistema.
- **Interfaces gráficas complejas (GUI)** donde muchos controles deben coordinarse (como botones, formularios, menús y cuadros de diálogo).
- **Sistemas de mensajería, chats o notificaciones**, donde múltiples participantes requieren una mediación para comunicarse.



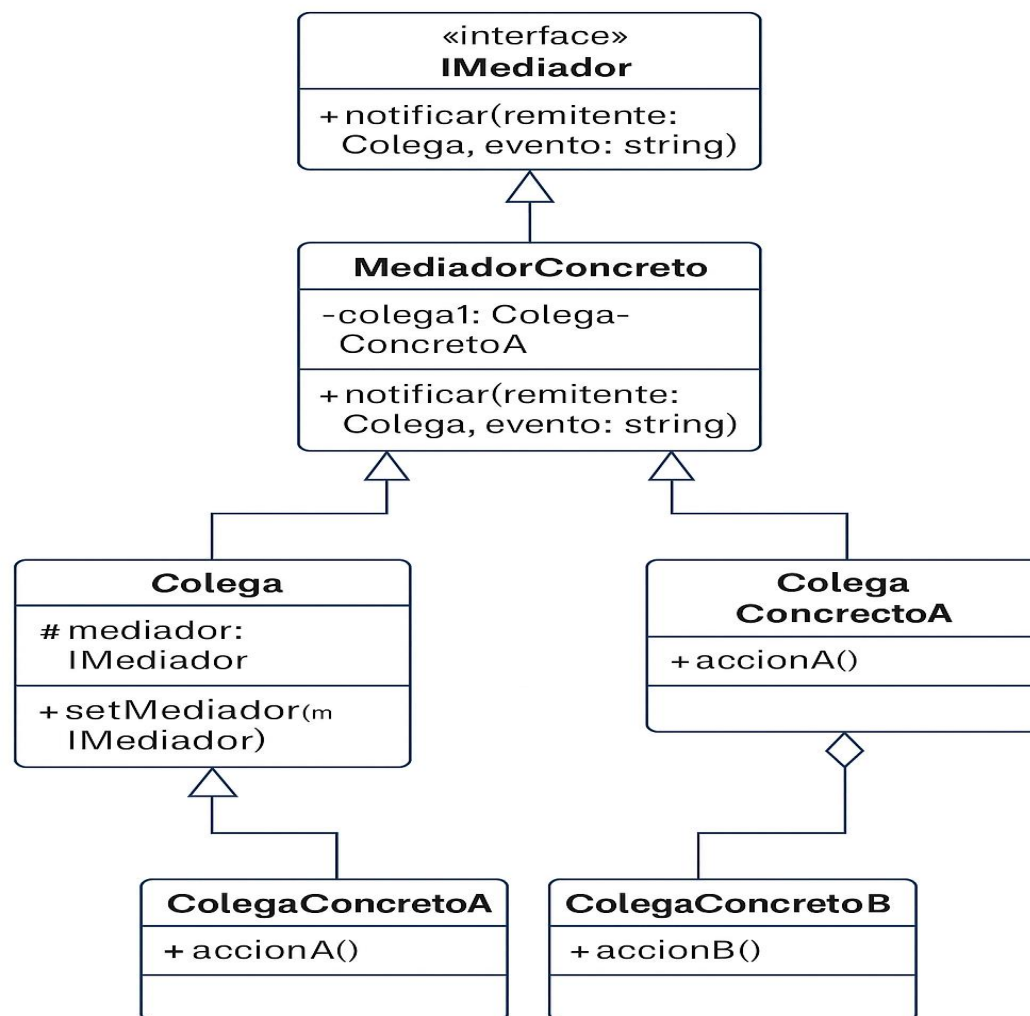
Ventajas

- **Desacoplamiento entre objetos**
- **Mayor claridad en la comunicación**
- **Facilita la escalabilidad y mantenimiento**
- **Promueve la reutilización de componentes**
- **Aumenta la cohesión**

Desventajas


- **Riesgo de sobrecarga en el Mediador**
- **Dependencia centralizada**
- **Curva de diseño inicial más elevada**
- **Menor flexibilidad directa**

Estructura





Ejemplo aplicado: Sala de Clases Virtual

- En una sala virtual (tipo Classroom o Teams), los usuarios (profesor y estudiantes) se comunican a través de una clase mediadora llamada SalaDeClase.
 - Cuando un participante envía un mensaje, no lo dirige directamente a los demás, sino que lo envía al mediador, quien decide cómo y a quién reenviarlo. Esto evita dependencias directas entre usuarios y centraliza la lógica de interacción.
- 



Beneficios

- Comunicación ordenada y desacoplada.
- Fácil de extender con reglas, filtros o historial.
- Mejor mantenibilidad del sistema.

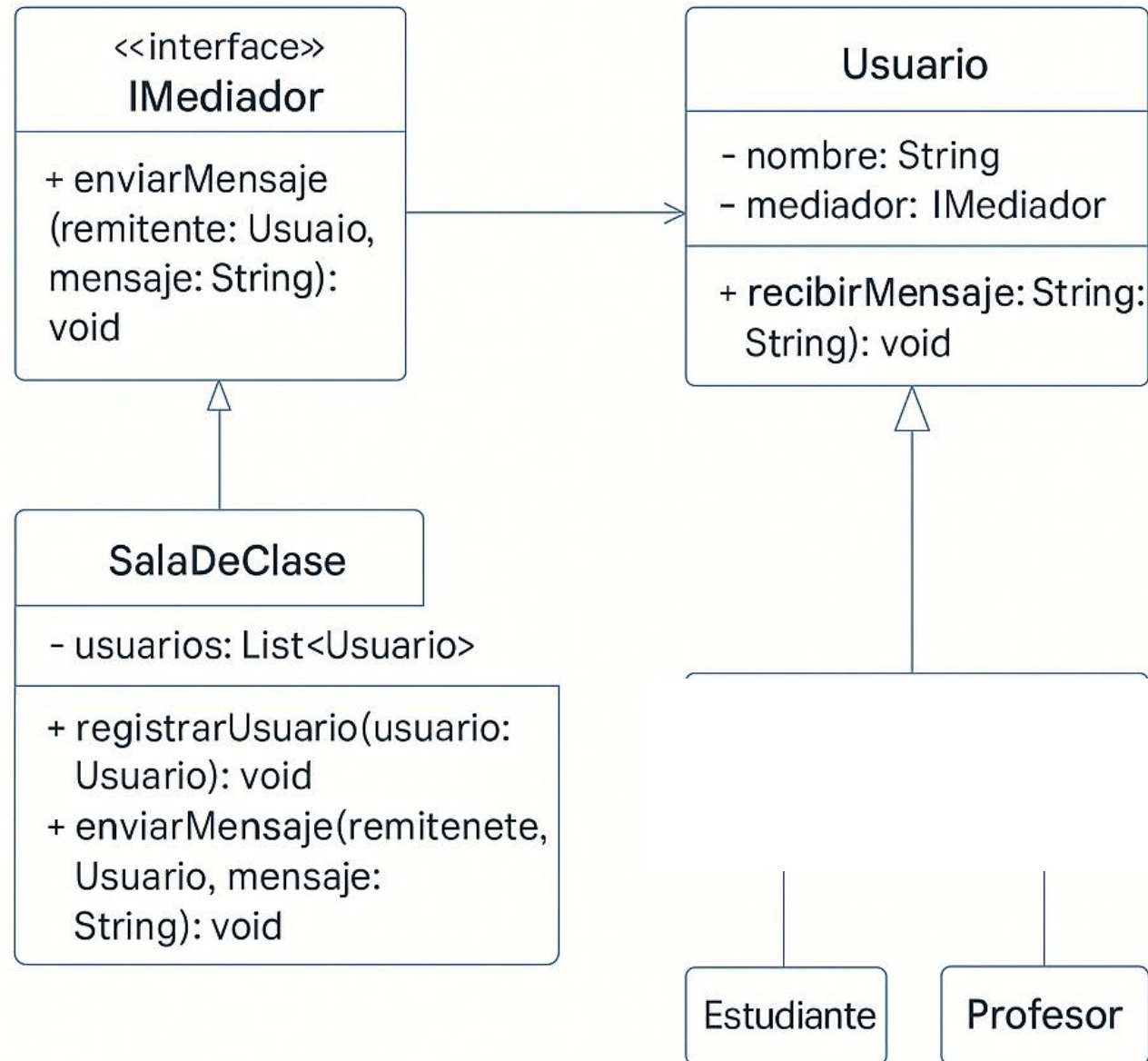




Participantes:

- Usuario (Estudiante / Profesor)
- SalaDeClase (mediador)





Implementación del Patrón Mediator – Código Java

```
public abstract class Usuario {
    protected String nombre;
    protected IMediator mediador;

    public Usuario(String nombre, IMediator mediador){
        this.nombre = nombre;
        this.mediador = mediador;
    }

    public abstract void recibirMensaje(String mensaje);

    public String getNombre() {
        return nombre;
    }
}
```

```
public class Estudiante extends Usuario {
    public Estudiante(String nombre, IMediator mediador) {
        super(nombre, mediador);
    }

    @Override
    public void recibirMensaje(String mensaje) {
        System.out.println("[Estudiante " + nombre + " recibe]: " + mensaje);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        SalaDeClase sala = new SalaDeClase();
        Profesor profe = new Profesor("Carlos", sala);
        Estudiante est1 = new Estudiante("Ana", sala);
        Estudiante est2 = new Estudiante("Luis", sala);

        sala.registrarUsuario(profe);
        sala.registrarUsuario(est1);
        sala.registrarUsuario(est2);

        sala.enviarMensaje("Tengo una duda sobre el patrón Mediator.", est1);
        profe.enviarMensaje("Claro Ana, dime tu pregunta.");
    }
}
```

```
public interface IMediator {
    void enviarMensaje(String mensaje, Usuario remitente);
}
```

```
public class Profesor extends Usuario {
    public Profesor(String nombre, IMediator mediador) {
        super(nombre, mediador);
    }

    @Override
    public void recibirMensaje(String mensaje) {
        System.out.println("[Profesor " + nombre + " recibe]: " + mensaje);
    }
}
```

```
public class SalaDeClase implements IMediator {
    private List<Usuario> usuarios = new ArrayList<>();

    public void registrarUsuario(Usuario usuario) {
        usuarios.add(usuario);
    }

    @Override
    public void enviarMensaje(String mensaje, Usuario remitente) {
        for (Usuario u : usuarios) {
            if (!u.equals(remitente)) {
                u.recibirMensaje(mensaje);
            }
        }
    }
}
```


Conclusión

- El patrón Mediator permite **organizar la comunicación entre objetos** de forma centralizada, reduciendo el acoplamiento y facilitando el mantenimiento.
Es ideal para sistemas con múltiples componentes que deben coordinarse, como **interfaces gráficas, chats o microservicios**.
Su aplicación mejora la **claridad, escalabilidad y flexibilidad** del software.



UNIVERSIDAD
Popular del cesar