PATRON MEDIATOR

¿Qué es el patrón Mediator?

El patrón **Mediator** (Mediador) es un patrón de diseño **comportamental** que permite definir un objeto central que encapsula la comunicación entre un conjunto de objetos relacionados. Los objetos ya no se comunican directamente entre sí, sino a través del **mediador**, reduciendo así el acoplamiento.

En otras palabras:

- Los objetos no interactúan directamente entre sí.
- Todas las comunicaciones pasan a través de un **objeto intermediario** llamado mediador.

¿Cuándo utilizar el patrón Mediator?

Usa el Mediator cuando:

- Existan múltiples objetos que interactúan y dependan mucho entre sí, creando conexiones complejas.
- Necesites reducir el acoplamiento para que puedas reutilizar o modificar objetos fácilmente.
- Las interacciones entre objetos sean difíciles de entender o mantener debido a que muchos objetos están relacionados directamente entre sí.
- Quieras centralizar el control de la comunicación entre componentes.

Estructura general del Patrón Bridge:

La estructura básica del patrón **Mediator** es:

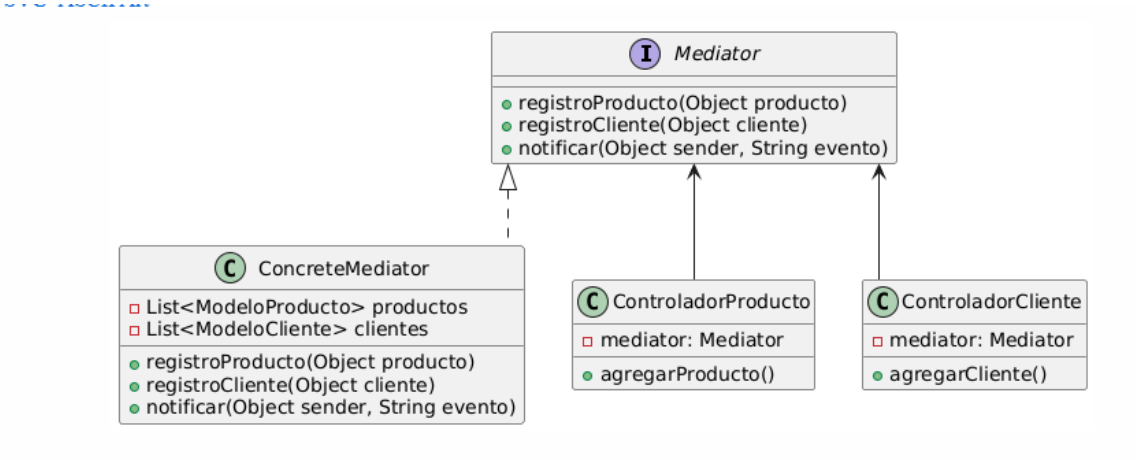
Mediator: Define la interfaz para la comunicación entre objetos (colleagues).

ConcreteMediator: Implementa la interfaz del Mediator. Coordina la comunicación entre objetos concretos.

Colleague: Define la interfaz de objetos que desean comunicarse con otros objetos.

ConcreteColleague: Implementa las operaciones del Colleague y utiliza el mediador para comunicarse con otros objetos.

UML PATRON MEDIATOR



APLICACIÓN DEL PATRON MEDIATOR

Para pasar la instancia del mediador desde fuera de la clase, se modificó el constructor para recibir el objeto Mediator mediante un patrón Builder:

```java
public class ControladorProducto {

    private final boolean habilitarLogs;
    private final boolean mostrarMensajesError;

    // 🔹 Mediator agregado
    private final Mediator mediator;

    // Constructor privado usando Builder
    private ControladorProducto(Builder builder) {
        this.habilitarLogs = builder.habilitarLogs;
        this.mostrarMensajesError = builder.mostrarMensajesError;
        this.mediator = builder.mediator;
    }

    public static class Builder {
        private boolean habilitarLogs = false;
        private boolean mostrarMensajesError = true;
        private Mediator mediator;

        public Builder habilitarLogs(boolean valor) {
            this.habilitarLogs = valor;
            return this;
        }

        public Builder mostrarMensajesError(boolean valor) {
            this.mostrarMensajesError = valor;
            return this;
        }

        // 🔹 Método para asignar el Mediator
        public Builder mediator(Mediator mediator) {
            this.mediator = mediator;
            return this;
        }
```

Uso del Mediator para notificar eventos

En cada método importante que modifica el estado del producto, se añadió una llamada al mediador usando su método notificar(...). Por ejemplo:

- **Agregar producto:**

```
// 🧩 Método para agregar productos con Mediator
public void agregarProducto(JTextField nombres, JTextField precioProducto, JTextField stock) {
    configuracion.Conexion objetoConexion = configuracion.Conexion.getInstancia();
    String consulta = "INSERT INTO producto (nombre, precioProducto, stock) VALUES (?, ?, ?)";

    try {
        CallableStatement cs = objetoConexion.estableceConexion().prepareCall(consulta);
        cs.setString(1, nombres.getText());
        cs.setDouble(2, Double.parseDouble(precioProducto.getText()));
        cs.setInt(3, Integer.parseInt(stock.getText()));
        cs.execute();

        JOptionPane.showMessageDialog(null, "Producto agregado correctamente");
        mediator.notificar(this, "ProductoAgregado"); // 🔔 Notificación Mediator

    } catch (Exception e) {
        manejarError("Error al agregar producto", e);
    } finally {
        objetoConexion.cerrarConexion();
    }
}
```