



TECNOLOGICO NACIONAL DE MEXICO

INSTITUTO TECNOLÓGICO DE OAXACA

Carrera:

Ingeniería en Sistemas Computacionales

Materia:

Diseño e Implementación de software con patrones

Documentación de patrón mediator

Docente:

Espinoza Pérez Jacob

Equipo:

Ordaz Pacheco Ruudvan

Santos Manuel Julia Marlenny

Vera Acevedo Héctor Aramís

Grupo:

7SB

Fecha de entrega:

07/04/2025

Documentación del Patrón Mediator en el Sistema de Ventas

Introducción

El patrón Mediator es un patrón de diseño de comportamiento que **define un objeto que encapsula cómo interactúa un conjunto de objetos**. En el sistema de gestión de clientes, este patrón se implementó para coordinar las interacciones entre la interfaz de usuario (InterCliente), el controlador (Ctrl_Cliente) y el modelo (Cliente), reduciendo el acoplamiento directo entre estos componentes.

Interfaz ClienteMediator

La interfaz ClienteMediator define el contrato que debe implementar el mediador para gestionar las interacciones entre los componentes:

```
import modelo.Cliente;
import vista.InterCliente;

public interface ClienteMediator {
    void registrarCliente(Cliente cliente);
    void notificar(String mensaje, InterCliente origen);
    void setInterfazCliente(InterCliente interfaz);
}
```

Responsabilidades:

- registrarCliente: Coordina el proceso completo de registro
- notificar: Maneja la comunicación de mensajes al usuario
- setInterfazCliente: Establece la referencia a la interfaz gráfica

Clase ClienteMediatorImpl (Mediador Concreto)

Implementa la lógica de mediación entre los componentes:

```
import controlador.Ctrl_Cliente;
import modelo.Cliente;
import vista.InterCliente;
import javax.swing.JOptionPane;

public class ClienteMediatorImpl implements ClienteMediator {
    private InterCliente interfazCliente;
    private final Ctrl_Cliente controladorCliente;

    public ClienteMediatorImpl() {
        this.controladorCliente = new Ctrl_Cliente();
    }

    @Override
    public void registrarCliente(Cliente cliente) {
        if (!controladorCliente.existeCliente(cliente.getCedula())) {
            if (controladorCliente.guardar(cliente)) {
                notificar("Registro Guardado", null);
                interfazCliente.limpiarCampos();
            } else {
                notificar("Error al Guardar", null);
            }
        } else {
            notificar("El cliente ya está registrado", null);
        }
    }

    @Override
    public void notificar(String mensaje, InterCliente origen) {
        JOptionPane.showMessageDialog(null, mensaje);
    }

    @Override
    public void setInterfazCliente(InterCliente interfaz) {
        this.interfazCliente = interfaz;
    }
}
```

Flujo de trabajo:

1. Recibe la solicitud de registro desde la interfaz
2. Valida la existencia del cliente
3. Gestiona el proceso de guardado
4. Notifica resultados
5. Solicita limpieza de campos

Modificaciones en InterCliente

La interfaz gráfica fue adaptada para:

1. Recibir el mediador en su constructor
2. Delegar todas las acciones al mediador

```
public class InterCliente extends javax.swing.JInternalFrame {  
  
    private ClienteMediator mediator;  
  
    public InterCliente(ClienteMediator mediator) {  
        initComponents();  
        this.setSize(new Dimension(400, 300));  
        this.setTitle("Nuevo Cliente");  
        this.mediator = mediator; // Primero asignamos el mediator  
        this.mediator.setInterfazCliente(this);  
        if (mediator == null) {  
            throw new IllegalArgumentException("Mediator no puede ser null");  
        }  
    }  
}
```

Adaptación del Ctrl_Cliente

El controlador mantuvo su funcionalidad, pero ahora es invocado por el mediador:

```
// Verificar que todos los campos estén completos
if (!txt_nombre.getText().isEmpty() && !txt_apellido.getText().isEmpty() &&
    !txt_cedula.getText().isEmpty() && !txt_telefono.getText().isEmpty()) {

    if (!txt_cedula.getText().trim().equals(txt_telefono.getText().trim())) {
        mediator.notificar("Ambos teléfonos deben ser iguales.", this);
        txt_cedula.setBackground(Color.red);
        txt_telefono.setBackground(Color.red);
        return;
    }

    if (txt_cedula.getText().length() != 10 || !txt_cedula.getText().matches("[0-9]+") ||
        txt_telefono.getText().length() != 10 || !txt_telefono.getText().matches("[0-9]+")) {
        mediator.notificar("Ambos teléfonos deben tener exactamente 10 números.", this);
        txt_cedula.setBackground(Color.red);
        txt_telefono.setBackground(Color.red);
        return;
    }

    Cliente cliente = new Cliente();
    cliente.setNombre(txt_nombre.getText().trim());
    cliente.setApellido(txt_apellido.getText().trim());
    cliente.setCedula(txt_cedula.getText().trim());
    cliente.setTelefono(txt_telefono.getText().trim());
    cliente.setDireccion(txt_direccion.getText().trim());
    cliente.setEstado(1);

    mediator.registrarCliente(cliente);
} else {
    mediator.notificar("Completa todos los campos", this);
    txt_nombre.setBackground(Color.red);
    txt_apellido.setBackground(Color.red);
    txt_cedula.setBackground(Color.red);
    txt_telefono.setBackground(Color.red);
    txt_direccion.setBackground(Color.red);
}

// Limpiar los campos después de la validación
this.Limpiar();
```

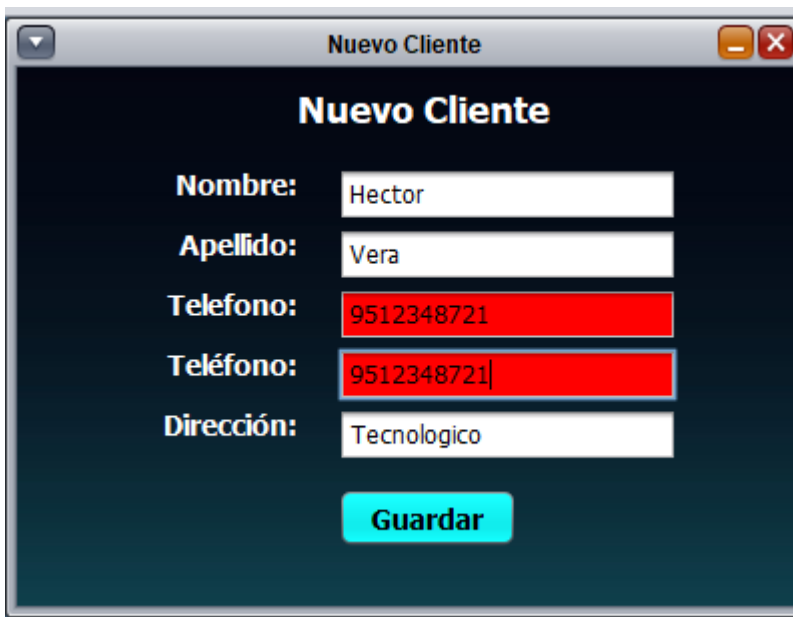
Flujo de Registro de Cliente

1. Usuario ingresa datos y hace clic en "Guardar"
2. InterCliente crea objeto Cliente y lo envía al Mediador
3. Mediador:
 - Verifica existencia mediante Ctrl_Cliente
 - Intenta guardar el registro
 - Notifica resultados
4. InterCliente muestra visualmente.

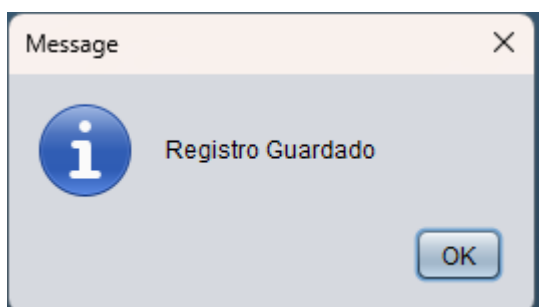
Ventajas del Patrón Singleton en este Código

1. **Control centralizado:** Todas las partes del sistema utilizan la misma instancia de conexión
2. **Eficiencia en recursos:** Evita la creación múltiple de conexiones a la BD
3. **Consistencia:** Garantiza que todas las operaciones trabajen con el mismo estado de conexión
4. **Facilidad de mantenimiento:** Cambios en la configuración de conexión se realizan en un solo lugar
5. **Thread-safe:** La implementación sincronizada previene problemas en entornos concurrentes

Resultados



A screenshot of a Windows-style application window titled "Nuevo Cliente". The window has a dark blue background. At the top, the title bar says "Nuevo Cliente" with standard minimize, maximize, and close buttons. Below the title bar, the text "Nuevo Cliente" is displayed in a large, bold, white font. The form contains five labels and text boxes: "Nombre:" with the value "Hector", "Apellido:" with the value "Vera", "Telefono:" with the value "9512348721" (the text box has a red background), "Teléfono:" with the value "9512348721|" (the text box has a red background), and "Dirección:" with the value "Tecnologico". At the bottom center of the form is a blue button with the text "Guardar" in white.



Gestionar Clientes

Administrar Clientes

N°	nombre	apellido	telefono	telefono	direccion	estado
1	Uri	Hernandez	9514401725	9514401725	Av Montoya	1
2	Teresa	Hernandez	9581732455	9581732455	Privada Texcoco	1
3	Alma Rosa	Mendez	9512156554	9512156554	Privada Texcoco	1
5	Alberto	Mendez	9511111212	9511111212	Texcoco 6	1
6	abdiel	Diaz	9514565656	9514565656	Av montealban	1
7	Hector	Vera	9512348721	9512348721	Tecnologico	1

Actualizar

Eliminar

Nombre:

Apellido:

Teléfono:

Teléfono:

Dirección:

Conclusión

La implementación del patrón Mediator en el sistema de gestión de clientes ha demostrado ser una solución efectiva para:

1. **Organizar** las interacciones entre componentes
2. **Reducir** las dependencias directas
3. **Facilitar** el mantenimiento y extensión del sistema
4. **Centralizar** la lógica de coordinación

Este enfoque permite que el sistema evolucione de manera más controlada, haciendo más sencilla la incorporación de nuevas funcionalidades o la modificación de las existentes sin afectar múltiples componentes.