



# **TECNOLOGICO NACIONAL DE MEXICO**

## **INSTITUTO TECNOLÓGICO DE OAXACA**

### **Carrera:**

Ingeniería en Sistemas Computacionales

### **Materia:**

Diseño e Implementación de software con patrones

### **Documentación de patrón Observer**

### **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 Observer en el Sistema de Ventas

### Introducción

El patrón Observer es un patrón de diseño de comportamiento que **define una dependencia uno-a-muchos entre objetos**, de modo que cuando un objeto cambia su estado, todos sus dependientes son notificados automáticamente. En el sistema de gestión de usuarios, este patrón se implementó para coordinar las actualizaciones entre la interfaz de usuario (InterGestionarUsuario), el controlador (Ctrl\_Usuario) y otros componentes del sistema.

## Interfaz UsuarioObserver

La interfaz UsuarioObserver define el contrato que deben implementar todos los observadores para recibir notificaciones:

### Responsabilidades:

- Recibir notificaciones cuando ocurren cambios en los usuarios
- Ejecutar acciones específicas según el tipo de cambio (creación, actualización o eliminación)

```
//
public interface UsuarioObserver {
    void update(String mensaje, Usuario usuario);
}

//
public interface UsuarioSubject {
    void registrarObserver(UsuarioObserver observer);
    void removerObserver(UsuarioObserver observer);
    void notificarObservers(String accion, Usuario usuario);
}
```

## Clase Ctrl\_Usuario (Subject)

La clase Ctrl\_Usuario fue modificada para implementar el rol de Subject en el patrón Observer:

### Flujo de trabajo:

1. Los componentes interesados se registran como observadores
2. Cuando ocurre un cambio en los usuarios (creación, actualización o eliminación)
3. El controlador notifica a todos los observadores registrados
4. Cada observador ejecuta su lógica específica

```

public class Ctrl_Usuario implements UsuarioSubject {
    private List<UsuarioObserver> observers = new ArrayList<>();

    // Métodos existentes...

    @Override
    public void registrarObserver(UsuarioObserver observer) {
        observers.add(observer);
    }

    @Override
    public void removerObserver(UsuarioObserver observer) {
        observers.remove(observer);
    }

    @Override
    public void notificarObservers(String accion, Usuario usuario) {
        for (UsuarioObserver observer : observers) {
            observer.update(accion, usuario);
        }
    }

    // Modificar métodos existentes para incluir notificaciones

    public boolean guardar(Usuario objeto) {
        boolean respuesta = false;
        Connection cn = Conexion.getConexion();
        try {
            PreparedStatement consulta = cn.prepareStatement("insert into tb_usuario values(?,?,?,?,?,?,?)");
            consulta.setInt(1, 0); //id
            consulta.setString(2, objeto.getNombre());
            consulta.setString(3, objeto.getApellido());
            consulta.setString(4, objeto.getUsuario());
            consulta.setString(5, objeto.getPassword());
            consulta.setString(6, objeto.getTelefono());
            consulta.setInt(7, objeto.getEstado());
            if (consulta.executeUpdate() > 0) {
                respuesta = true;
                notificarObservers("CREACION", objeto); // Notificar creación
            }
            Conexion.cerrarConexion();
        } catch (SQLException e) {
            System.out.println("Error al guardar usuario: " + e);
        }
        return respuesta;
    }
}

```

```

public boolean actualizar(Usuario objeto, int idUsuario) {
    boolean respuesta = false;
    Connection cn = Conexion.getConexion();
    try {
        PreparedStatement consulta = cn.prepareStatement("update tb_usuario set nombre=?, apellido = ?, usuario = ?, password= ?, telefono = ?, estado
        consulta.setString(1, objeto.getNombre());
        consulta.setString(2, objeto.getApellido());
        consulta.setString(3, objeto.getUsuario());
        consulta.setString(4, objeto.getPassword());
        consulta.setString(5, objeto.getTelefono());
        consulta.setInt(6, objeto.getEstado());
        if (consulta.executeUpdate() > 0) {
            respuesta = true;
            notificarObservers("ACTUALIZACION", objeto); // Notificar actualización
        }
        Conexion.cerrarConexion();
    } catch (SQLException e) {
        System.out.println("Error al actualizar usuario: " + e);
    }
    return respuesta;
}

public boolean eliminar(int idUsuario) {
    boolean respuesta = false;
    Connection cn = Conexion.getConexion();
    try {
        Usuario usuarioEliminado = obtenerUsuarioPorId(idUsuario); // Necesitarás implementar este método
        PreparedStatement consulta = cn.prepareStatement(
            "delete from tb_usuario where idUsuario = " + idUsuario + " ";
        consulta.executeUpdate();
        if (consulta.executeUpdate() > 0) {
            respuesta = true;
            notificarObservers("ELIMINACION", usuarioEliminado); // Notificar eliminación
        }
        Conexion.cerrarConexion();
    } catch (SQLException e) {
        System.out.println("Error al eliminar usuario: " + e);
    }
    return respuesta;
}
}

```

```

public boolean existeUsuario(String usuario) {
    boolean respuesta = false;
    String sql = "select usuario from tb_usuario where usuario = '" + usuario + "'";
    Statement st;
    try {
        Connection cn = Conexion.getConnection();
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }
    } catch (SQLException e) {
        System.out.println("Error al consultar usuario: " + e);
    }
    return respuesta;
}

/**
 * *****
 * metodo para Iniciar Sesion
 * *****
 */
public boolean loginUser(Usuario objeto) {
    boolean respuesta = false;
    Connection cn = Conexion.getConnection();
    String sql = "select usuario, password from tb_usuario where usuario = '" + objeto.getUsuario() + "' and password = '" + objeto.getPassword() + "'";
    Statement st;
    try {
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }
    } catch (SQLException e) {
        System.out.println("Error al Iniciar Sesion");
        JOptionPane.showMessageDialog(null, "Error al Iniciar Sesion");
    }
    return respuesta;
}

public Usuario obtenerUsuarioPorId(int idUsuario) {
    Usuario usuario = null;
    Connection cn = Conexion.getConnection();
    String sql = "SELECT idUsuario, nombre, apellido, usuario, password, telefono, estado FROM tb_usuario WHERE idUsuario = ?";

    try {
        PreparedStatement consulta = cn.prepareStatement(sql);
        consulta.setInt(1, idUsuario);
        ResultSet rs = consulta.executeQuery();

        if (rs.next()) {
            usuario = new Usuario();
            usuario.setIdUsuario(rs.getInt("idUsuario"));
            usuario.setNombre(rs.getString("nombre"));
            usuario.setApellido(rs.getString("apellido"));
            usuario.setUsuario(rs.getString("usuario"));
            usuario.setPassword(rs.getString("password"));
            usuario.setTelefono(rs.getString("telefono"));
            usuario.setEstado(rs.getInt("estado"));
        }

        Conexion.cerrarConexion();
    } catch (SQLException e) {
        System.out.println("Error al obtener usuario por ID: " + e);
    }

    return usuario;
}
}

```

## Implementaciones concretas de Observers

UsuarioLogger (Para registro de actividades):

```
import modelo.Usuario;
import modelo.UsuarioObserver;

public class UsuarioLogger implements UsuarioObserver {
    @Override
    public void update(String accion, Usuario usuario) {
        System.out.println("[LOG] Acción: " + accion +
            " | Usuario: " + usuario.getUsuario() +
            " | Nombre: " + usuario.getNombre());
    }
}
```

UsuarioUIUpdater (Para actualizar la interfaz):

```
import modelo.Usuario;
import modelo.UsuarioObserver;

public class UsuarioUIUpdater implements UsuarioObserver {
    private InterGestionarUsuario gestionarUsuario;

    public UsuarioUIUpdater(InterGestionarUsuario gestionarUsuario) {
        this.gestionarUsuario = gestionarUsuario;
    }

    @Override
    public void update(String accion, Usuario usuario) {
        if (accion.equals("CREACION") || accion.equals("ACTUALIZACION") || accion.equals("ELIMINACION")) {
            gestionarUsuario.CargarTablaUsuarios(); // Refrescar tabla
        }
    }
}
```

## Integración del código a la vista

```
public class InterGestionarUsuario extends javax.swing.JInternalFrame {

    private int idUsuario = 0;
    private Ctrl_Usuario ctrlUsuario;

    public InterGestionarUsuario() {
        initComponents();
        this.setSize(new Dimension(900, 500));
        this.setTitle("Gestionar Usuarios");
        //Cargar tabla
        this.CargarTablaUsuarios();
        ctrlUsuario = new Ctrl_Usuario();
        // Registrar observadores
        ctrlUsuario.registrarObserver(new UsuarioLogger());
        ctrlUsuario.registrarObserver(new UsuarioUIUpdater(this));

        //insertar imagen en nuestro JLabel
        ImageIcon wallpaper = new ImageIcon("src/img/fondo3.jpg");
        Icon icono = new ImageIcon(wallpaper.getImage().getScaledInstance(900, 500, WIDTH));
        jLabel_wallpaper.setIcon(icono);
        this.repaint();
    }
}
```

### Ventajas del Patrón Observer en este Código

1. **Desacoplamiento:** Los componentes no necesitan conocerse entre sí directamente
2. **Extensibilidad:** Fácil agregar nuevos observadores sin modificar el código existente
3. **Actualizaciones en tiempo real:** Cambios se reflejan inmediatamente en todos los componentes interesados
4. **Mantenibilidad:** Cada observador tiene una única responsabilidad clara
5. **Consistencia:** Todos los componentes reciben la misma información actualizada

## Resultados

Gestionar Usuarios

Administrar Usuarios

N°	nombre	apellido	usuario	password	telefono	estado
1	Ruudvan	Ordaz	Orda1	123	9514401726	1
2	Uri	Hernandez	Uri1	123	9513436751	1
3	Teresa	Hernandez	Herna1	123	9581732455	1
4	Uriel	Hernandez	637279	7279	9513436751	1

Actualizar

Eliminar

Nombre:

Uri

Apellido:

Fernandez

Usuario:

Uri1

Password:

1234

Telefono:

9513436722

Message

i

¡Actualizacion Exitosa!

OK

Gestionar Usuarios

Administrar Usuarios

N°	nombre	apellido	usuario	password	telefono	estado
1	Ruudvan	Ordaz	Orda1	123	9514401726	1
2	Uri	Fernandez	Uri1	1234	9513436722	1
3	Teresa	Hernandez	Herna1	123	9581732455	1
4	Uriel	Hernandez	637279	7279	9513436751	1

Actualizar

Eliminar

Nombre:

Apellido:

Usuario:

Password:

Telefono:



**Nuevo Usuario**

**Nombre:**

**Apellido:**


**Usuario:**

**Password:**  ☒

**Telefono:**

**Guardar**

Message

 ¡Usuario Registrado!

**OK**

**Gestionar Usuarios**

**Administrar Usuarios**

N°	nombre	apellido	usuario	password	telefono	estado
1	Ruudvan	Ordaz	Orda1	123	9514401726	1
2	Uri	Fernandez	Uri1	1234	9513436722	1
3	Teresa	Hernandez	Herna1	123	9581732455	1
4	Uriel	Hernandez	637279	7279	9513436751	1
5	Hector	Vera	Hec	123	9512923456	1

**Actualizar**  
**Eliminar**

**Nombre:**  **Apellido:**  **Usuario:**

**Password:**  **Telefono:**

## Conclusión

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

1. **Organizar** el flujo de notificaciones entre componentes
2. **Reducir** el acoplamiento entre las diferentes capas del sistema
3. **Facilitar** el mantenimiento y la extensión del código
4. **Centralizar** la lógica de notificaciones

Este enfoque permite que el sistema evolucione de manera controlada, haciendo más sencilla la incorporación de nuevas funcionalidades que requieran reaccionar a cambios en los usuarios.