TECNOLOGICO NACIONAL DE MEXICO

INSTITUTO TECNOLOGICO DE OAXACA

Carrera:

Ingeniería en Sistemas Computacionales

Materia:

Diseño e Implementación de software con patrones

Documentación

Docente:

Espinoza Pérez Jacob

Equipo:

Ordaz Pacheco Ruudvan

Santos Manuel Julia Marlenny

Vera Acevedo Héctor Aramís

Grupo:

7SB

Fecha de entrega:

24/03/2025

Documentación del Patrón Bridge en el Sistema de Ventas

Introducción

El patrón **Bridge** es un patrón de diseño estructural que permite desacoplar una abstracción de su implementación, de modo que ambas puedan variar independientemente. En este caso, se ha aplicado el patrón Bridge para separar la lógica de negocio (abstracción) de la implementación de la base de datos (implementación) en el sistema de gestión de productos.

Estructura del Patrón Bridge

El patrón Bridge se ha aplicado de la siguiente manera:

- **Abstracción**: La clase Ctrl_Producto actúa como la abstracción. Contiene la lógica de negocio y delega las operaciones de base de datos a la implementación.
- **Implementación**: La interfaz ProductoDAO define los métodos que deben ser implementados para interactuar con la base de datos. La clase ProductoDAOImpl es la implementación concreta de esta interfaz.
- **Desacoplamiento**: La abstracción (Ctrl_Producto) y la implementación (ProductoDAOImpl) están desacopladas, lo que permite que ambas evolucionen de manera independiente.

Cambios Realizados

Creación de la Interfaz ProductoDAO

Se creó una interfaz llamada ProductoDAO que define los métodos necesarios para interactuar con la base de datos. Esta interfaz actúa como el "puente" entre la lógica de negocio y la implementación de la base de datos.

```
public interface ProductoDAO {
    boolean guardar(Producto objeto);
    boolean existeProducto(String producto);
    boolean actualizar(Producto objeto, int idProducto);
    boolean eliminar(int idProducto);
    boolean actualizarStock(Producto objeto, int idProducto);
}
```

Implementación de la Clase ProductoDAOImpl

Se creó una clase concreta llamada ProductoDAOImpl que implementa la interfaz ProductoDAO. Esta clase contiene la lógica de acceso a la base de datos.

```java
public class ProductoDAOImpl implements ProductoDAO {

    @Override
    public boolean guardar(Producto objeto) {
        boolean respuesta = false;
        Connection cn = Conexion.conectar();
        try {
            PreparedStatement consulta = cn.prepareStatement("insert into tb_producto values(?,?,?,?,?,?,?,?)");
            consulta.setInt(1, 0); // id
            consulta.setString(2, objeto.getNombre());
            consulta.setInt(3, objeto.getCantidad());
            consulta.setDouble(4, objeto.getPrecio());
            consulta.setString(5, objeto.getDescripcion());
            consulta.setInt(6, objeto.getPorcentajeIva());
            consulta.setInt(7, objeto.getIdCategoria());
            consulta.setInt(8, objeto.getEstado());

            if (consulta.executeUpdate() > 0) {
                respuesta = true;
            }
            cn.close();
        } catch (SQLException e) {
            System.out.println("Error al guardar producto: " + e);
        }
        return respuesta;
    }

    @Override
    public boolean existeProducto(String producto) {
        boolean respuesta = false;
        String sql = "select nombre from tb_producto where nombre = '" + producto + "';";
        Statement st;
        try {
            Connection cn = Conexion.conectar();
            st = cn.createStatement();
            ResultSet rs = st.executeQuery(sql);
            while (rs.next()) {
                respuesta = true;
            }
        } catch (SQLException e) {
            System.out.println("Error al consultar producto: " + e);
        }
        return respuesta;
    }
```

```java
@Override
public boolean actualizar(Producto objeto, int idProducto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement("update tb_producto set nombre=?, cantidad=?, precio=?, descripcion=?, porcentajeIva=?,
        consulta.setString(1, objeto.getNombre());
        consulta.setInt(2, objeto.getCantidad());
        consulta.setDouble(3, objeto.getPrecio());
        consulta.setString(4, objeto.getDescripcion());
        consulta.setInt(5, objeto.getPorcentajeIva());
        consulta.setInt(6, objeto.getIdCategoria());
        consulta.setInt(7, objeto.getEstado());

        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al actualizar producto: " + e);
    }
    return respuesta;
}

@Override
public boolean eliminar(int idProducto) {
    boolean respuesta = false;
    Connection cn = Conexion.conectar();
    try {
        PreparedStatement consulta = cn.prepareStatement("delete from tb_producto where idProducto ='" + idProducto + "'");
        if (consulta.executeUpdate() > 0) {
            respuesta = true;
        }
        cn.close();
    } catch (SQLException e) {
        System.out.println("Error al eliminar producto: " + e);
    }
    return respuesta;
}
```

```java
    @Override
    public boolean actualizarStock(Producto objeto, int idProducto) {
        boolean respuesta = false;
        Connection cn = Conexion.conectar();
        try {
            PreparedStatement consulta = cn.prepareStatement("update tb_producto set cantidad=? where idProducto ='" + idProducto + "'");
            consulta.setInt(1, objeto.getCantidad());

            if (consulta.executeUpdate() > 0) {
                respuesta = true;
            }
            cn.close();
        } catch (SQLException e) {
            System.out.println("Error al actualizar stock del producto: " + e);
        }
        return respuesta;
    }
}
```

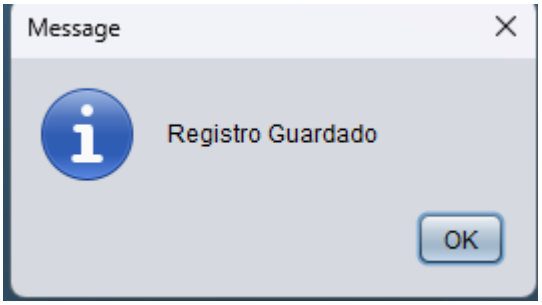
Modificación de la Clase Ctrl_Producto

La clase Ctrl_Producto ahora depende de la interfaz ProductoDAO en lugar de la implementación concreta. Esto permite que la lógica de negocio esté desacoplada de la implementación de la base de datos.

```java
public class Ctrl_Producto {

    private ProductoDAO productoDAO;

    public Ctrl_Producto() {
        this.productoDAO = new ProductoDAOImpl(); // Inyectamos la implementación concreta
    }

    public boolean guardar(Producto objeto) {
        return productoDAO.guardar(objeto);
    }

    public boolean existeProducto(String producto) {
        return productoDAO.existeProducto(producto);
    }

    public boolean actualizar(Producto objeto, int idProducto) {
        return productoDAO.actualizar(objeto, idProducto);
    }

    public boolean eliminar(int idProducto) {
        return productoDAO.eliminar(idProducto);
    }

    public boolean actualizarStock(Producto objeto, int idProducto) {
        return productoDAO.actualizarStock(objeto, idProducto);
    }
}
```

Resultados de la implementación del patrón Composite.

Agregación de productos



26	Doritos	20	12.0	Sabritas pic...	0.0	Sabritas	1

Conclusión

La implementación del patrón Bridge en el sistema de gestión de productos ha permitido desacoplar la lógica de negocio de la implementación de la base de datos, mejorando la flexibilidad, mantenibilidad y extensibilidad del código. Este enfoque facilita futuras modificaciones y pruebas, sin afectar la funcionalidad existente.