EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Oaxaca

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

CARRERA: INGENIERÍA EN SISTEMAS COMPUTACIONALES

Diseño E Implementacion De Software Con Patrones

Profesora: Espinosa Pérez Jacob

P R E S E N T A:

Luisa Michel Mendez Mendoza

Fernando Girón Pacheco

Mario Alberto Barbosa Santiago

Lourdes Gloria López García

Samuel Pérez Carrasco

PATRON FACADE

Este documento describe el funcionamiento y estructura del conjunto de clases controladoras utilizadas en un sistema de gestión de alquiler, clientes, productos y reportes. A continuación se documentan las siguientes clases:

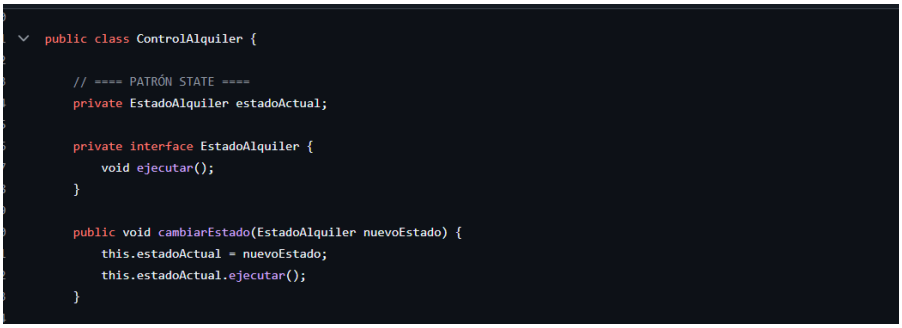
ControlAlquiler

ControladorCliente

ControladorProducto

ControladorReportes

1. Clase ControlAlquiler



```
public class ControlAlquiler {

    // ==== PATRÓN STATE ====
    private EstadoAlquiler estadoActual;

    private interface EstadoAlquiler {
        void ejecutar();
    }

    public void cambiarEstado(EstadoAlquiler nuevoEstado) {
        this.estadoActual = nuevoEstado;
        this.estadoActual.ejecutar();
    }
```

Esta clase se encarga de manejar todo el flujo relacionado al proceso de alquiler de productos por parte de los clientes. Utiliza dos patrones de diseño: State (para modularizar acciones por estados internos) y Facade (para ofrecer una interfaz simplificada al sistema de alquiler).

Patrón Facade:

La clase ControlAlquiler actúa como fachada de todo el proceso de alquiler, concentrando en sí misma el acceso a operaciones como buscar productos, registrar clientes, generar facturas, realizar ventas, calcular totales y limpiar formularios. Esto permite que otras clases o vistas utilicen una interfaz unificada sin preocuparse por la lógica interna.

Patrón State:

Utiliza una versión simplificada del patrón State, implementado mediante clases internas anónimas, para organizar de manera modular el comportamiento según el estado del sistema.

Método con Patrón State:

```java
// ==== MÉTODO CON STATE + FACADE ====
public void buscarProducto(JTextField nombreProducto, JTable tablaProducto) {
    cambiarEstado(() -> {
        configuracion.Conexion conexion = new configuracion.Conexion();
        DefaultTableModel modelo = new DefaultTableModel();

        modelo.addColumn("id");
        modelo.addColumn("Nombre");
        modelo.addColumn("PrecioProducto");
        modelo.addColumn("stock");

        tablaProducto.setModel(modelo);

        try {
            String consulta = "SELECT * FROM producto WHERE producto.nombre LIKE CONCAT('%', ? , '%');";
            PreparedStatement ps = conexion.estableceConexion().prepareStatement(consulta);
            ps.setString(1, nombreProducto.getText());
            ResultSet rs = ps.executeQuery();

            while (rs.next()) {
                modelo.addRow(new Object[]{
                    rs.getInt("idproducto"),
                    rs.getString("nombre"),
                    rs.getDouble("precioProducto"),
                    rs.getInt("stock")
                });
            }

            tablaProducto.setModel(modelo);

        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error al mostrar productos: " + e);
        } finally {
            conexion.cerrarConexion();
        }

        for (int i = 0; i < tablaProducto.getColumnCount(); i++) {
            tablaProducto.setDefaultEditor(tablaProducto.getColumnClass(i), null);
        }
    });
```

buscarProducto(...): Cambia a un estado que ejecuta la búsqueda de productos por nombre. El resultado se muestra en una tabla.

Otros métodos:

SeleccionarProductoAlquiler(...): Llena campos con los datos del producto seleccionado.

BuscarCliente(...): Realiza una consulta SQL para obtener clientes filtrados por nombre.

SeleccionarClienteAlquiler(...): Llena campos con los datos del cliente seleccionado.

pasarproductosVentas(...): Agrega productos seleccionados a la tabla de resumen de venta.

eliminarProductoSeleccionadoResumenVenta(...): Elimina un producto de la tabla de resumen.

calcularTotalPagar(...): Calcula el total con IVA.

crearFactura(...): Inserta una nueva factura a la base de datos.

realizarVenta(...): Registra el detalle de productos vendidos.

limpiarLuegoVenta(...): Limpia todos los campos y tablas luego de una venta.

MostrarUltimaFactura(...): Muestra el último ID de factura registrado.

2. Clase ControladorCliente

Controlador encargado de gestionar los clientes del sistema. Aplica el patrón Facade para agrupar todas las operaciones CRUD del cliente, y utiliza el patrón State para modularizar la consulta principal.

Patrón Facade:

Agrupa en una sola clase todos los métodos necesarios para registrar, editar, eliminar y consultar clientes, facilitando la interacción con la interfaz gráfica.

Método con Patrón State:

```
// === MÉTODO CON STATE + FACADE ===
public void mostrarClientes(JTable tablaTotalClientes) {
    cambiarEstado(() -> {
        configuracion.Conexion conexion = new configuracion.Conexion();
        ModeloCliente cliente = new ModeloCliente();
        DefaultTableModel modelo = new DefaultTableModel();

        modelo.addColumn("id");
        modelo.addColumn("nombres");
        modelo.addColumn("appaterno");
        modelo.addColumn("apmaterno");

        tablaTotalClientes.setModel(modelo);

        String sql = "SELECT idcliente, nombres, appaterno, apmaterno FROM cliente";

        try {
            Statement st = conexion.estableceConexion().createStatement();
            ResultSet rs = st.executeQuery(sql);

            while (rs.next()) {
                cliente.setIdCliente(rs.getInt("idcliente"));
                cliente.setNombres(rs.getString("nombres"));
                cliente.setApPaterno(rs.getString("appaterno"));
                cliente.setApMaterno(rs.getString("apmaterno"));

                modelo.addRow(new Object[]{
                    cliente.getIdCliente(),
                    cliente.getNombres(),
                    cliente.getApPaterno(),
                    cliente.getApMaterno()
                });
```

mostrarClientes(...): Realiza una consulta SQL para listar todos los clientes y mostrarlos en una tabla.

Otros métodos:

agregarCliente(...): Inserta un nuevo cliente en la base de datos.

seleccionarCliente(...): Llena campos con los datos del cliente seleccionado.

modificarCliente(...): Actualiza los datos del cliente.

eliminarCliente(...): Elimina un cliente por ID.

limpiarCampos(...): Limpia los campos de texto.

3. Clase ControladorProducto

Controlador para la gestión de productos disponibles para el alquiler. Integra los patrones Facade y State de forma similar a las otras clases.

Patrón Facade:

Proporciona una interfaz centralizada para agregar, modificar, eliminar, seleccionar y consultar productos, ideal para ser utilizada desde la vista o controladores externos.

Método con Patrón State:

```
27
28        // === MÉTODO CON STATE + FACADE ===
29        public void mostrarProductos(JTable tablaTotalProductos) {
30            cambiarEstado(() -> {
31                configuracion.Conexion conexion = new configuracion.Conexion();
32                ModeloProducto producto = new ModeloProducto();
33                DefaultTableModel modelo = new DefaultTableModel();
34
35                modelo.addColumn("id");
36                modelo.addColumn("nombresProd");
37                modelo.addColumn("Precio");
38                modelo.addColumn("Stock");
39
40                tablaTotalProductos.setModel(modelo);
41
42                String sql = "SELECT idproducto, nombre, precioProducto, stock FROM producto";
43
44                try {
45                    Statement st = conexion.estableceConexion().createStatement();
46                    ResultSet rs = st.executeQuery(sql);
47
48                    while (rs.next()) {
49                        producto.setIdProducto(rs.getInt("idproducto"));
50                        producto.setNombreProducto(rs.getString("nombre"));
51                        producto.setPrecioProducto(rs.getDouble("precioProducto"));
52                        producto.setStockProducto(rs.getInt("stock"));
53
54                        modelo.addRow(new Object[]{
55                            producto.getIdProducto(),
56                            producto.getNombreProducto(),
57                            producto.getPrecioProducto(),
58                            producto.getStockProducto()
59                        });
```

mostrarProductos(...): Obtiene todos los productos de la base de datos y los muestra en una tabla.

Otros métodos:

agregarProducto(...): Inserta un nuevo producto.

seleccionarProducto(...): Llena campos de texto con datos del producto seleccionado.

modificarProducto(...): Actualiza los datos del producto.

eliminarProducto(...): Elimina un producto por ID.

limpiarCampos(...): Limpia los campos.

4. Clase ControladorReportes

Encargada de consultar reportes, facturas y ventas por fechas. Implementa el patrón Facade para exponer funciones de reporteo y utiliza el patrón State para modularizar la consulta principal.

Patrón Facade:

La clase agrupa todos los reportes posibles (por factura, por producto y por fecha), permitiendo que la interfaz gráfica o el backend simplemente invoquen un método por cada necesidad de consulta.

Método con Patrón State:

```
// === MÉTODO CON STATE + FACADE ===
public void buscarFacturaConDatosCliente(JTextField numeroFacturaInput, JLabel idFactura, JLabel fecha, JLabel nombre, JLabel apPaterno, JLabel apMaterno) {
    cambiarEstado(() -> {
        if (numeroFacturaInput == null || numeroFacturaInput.getText().trim().isEmpty()) {
            JOptionPane.showMessageDialog(null, "Ingrese un número de factura válido.");
            return;
        }

        configuracion.Conexion conexion = new configuracion.Conexion();

        try {
            String sql = """
                SELECT factura.idfactura, factura.fechaFactura, cliente.nombres, cliente.appaterno, cliente.apmaterno
                FROM factura
                INNER JOIN cliente ON cliente.idcliente = factura.fkcliente
                WHERE factura.idfactura = ?;
                """;

            PreparedStatement ps = conexion.estableceConexion().prepareStatement(sql);
            ps.setInt(1, Integer.parseInt(numeroFacturaInput.getText().trim()));
            ResultSet rs = ps.executeQuery();

            if (rs.next()) {
                idFactura.setText(String.valueOf(rs.getInt("idfactura")));
                fecha.setText(rs.getDate("fechaFactura").toString());
                nombre.setText(rs.getString("nombres"));
                apPaterno.setText(rs.getString("appaterno"));
                apMaterno.setText(rs.getString("apmaterno"));
            } else {
                JOptionPane.showMessageDialog(null, "No se encontró la factura.");
            }
```

buscarFacturaConDatosCliente(...): Realiza una búsqueda de factura por ID, mostrando también los datos del cliente asociado.

Otros métodos:

buscarProductosPorFactura(...): Muestra los productos vendidos en una factura.

mostrarVentasPorFecha(...): Muestra todas las ventas realizadas entre dos fechas.

Consideraciones generales:

Las conexiones a base de datos se manejan mediante una clase Conexion del paquete configuracion.

El patrón de diseño State se utiliza para modularizar comportamiento específico dentro de cada clase.

El patrón Facade permite ofrecer una interfaz unificada y simplificada a cada módulo funcional del sistema.

La interfaz gráfica se apoya en componentes Swing (JTable, JTextField, JLabel, etc).

Todas las consultas y operaciones están protegidas con manejo de excepciones para evitar caídas del sistema.

Recomendaciones:

Aplicar el patrón State a más métodos donde se requiera modularidad por estados.

Aplicar el patrón Facade como estándar en nuevos controladores para mantener una arquitectura limpia.

Separar la lógica de negocio del acceso a datos en futuras refactorizaciones.

Utilizar prepared statements siempre, incluso en los updates y deletes, para prevenir inyecciones SQL.