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:

Mendez Mendoza Luisa Michel

Girón Pacheco Fernando

Mario Alberto Barbosa Santiago

Lourdes Gloria López García

Samuel Pérez Carrasco

DOCUMENTACIÓN DE CLASES DE CONTROLADORES EN EL SISTEMA DE ALQUILER

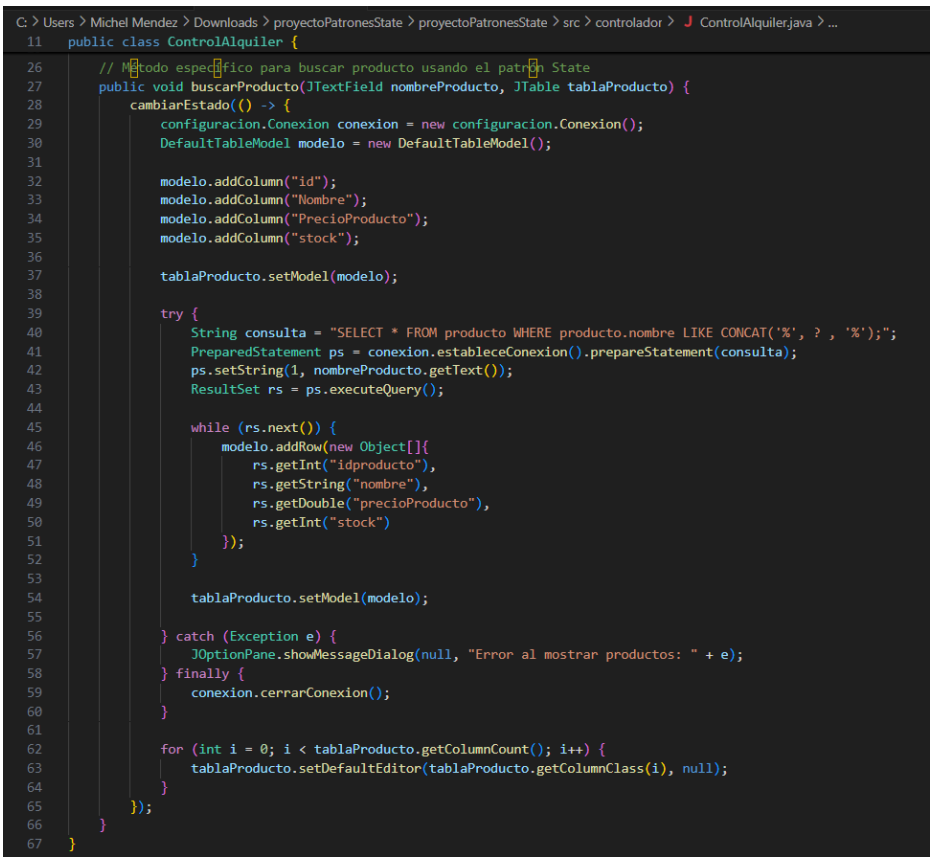
Este documento describe el funcionamiento y estructura del conjunto de clases controladoras utilizadas en un sistema de gestión de alquiler, clientes, productos y reportes. A continuación se documentan las siguientes clases:

- ControlAlquiler
- ControladorCliente
- ControladorProducto
- ControladorReportes

1. Clase ControlAlquiler

Esta clase se encarga de manejar todo el flujo relacionado al proceso de alquiler de productos por parte de los clientes. Utiliza una versión simplificada del patrón de diseño State, implementado mediante clases internas anónimas, para organizar de manera modular el comportamiento según el estado del sistema.

Método con Patrón State:




```java
public class ControlAlquiler {

    // Método específico para buscar producto usando el patrón State
    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. Uno de sus métodos, Mostrarclientes(...), ha sido adaptado al patrón State usando una clase interna anónima.
Método con Patrón State:

```java
public class ControladorCliente {

    private EstadoCliente estadoActual;

    // Interfaz interna del patrón State
    private interface EstadoCliente {
        void ejecutar();
    }

    // Método público para cambiar y ejecutar el estado
    private void cambiarEstado(EstadoCliente nuevoEstado) {
        this.estadoActual = nuevoEstado;
        this.estadoActual.ejecutar();
    }
```

```java
public class ControladorCliente {
    public void Mostrarclientes(JTable tablaTotalClientes) {
        cambiarEstado(() -> {
            configuracion.Conexion objetoConexion = new configuracion.Conexion();
            ModeloCliente objetocliente = 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 = objetoConexion.estableceConexion().createStatement();
                ResultSet rs = st.executeQuery(sql);

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

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

                tablaTotalClientes.setModel(modelo);

            } catch (Exception e) {
                JOptionPane.showMessageDialog(null, "Error al mostrar usuarios: " + e.toString());
            } finally {
                objetoConexion.cerrarConexion();
            }
        });
    }
```

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.
Seleccionar(...): Llena campos con los datos del cliente seleccionado.
ModificarCliente(...): Actualiza los datos del cliente.
limpiarCamposClientes(...): Limpia los campos de texto.
EliminarClientes(...): Elimina un cliente por ID.

3. Clase ControladorProducto

Controlador para la gestión de productos disponibles para el alquiler
Método con Patrón State:

```
import modelo.ModeloProducto;

public class ControladorProducto {

    private EstadoProducto estadoActual;

    // Interfaz interna del patrón State
    private interface EstadoProducto {
        void ejecutar();
    }

    // Método para cambiar y ejecutar el estado
    private void cambiarEstado(EstadoProducto nuevoEstado) {
        this.estadoActual = nuevoEstado;
        this.estadoActual.ejecutar();
    }
```

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

Otros métodos:

AgregarProducto(...): Inserta un nuevo producto.
Seleccionar(...): Llena campos de texto con datos del producto seleccionado.
ModificarProducto(...): Actualiza los datos del producto.
limpiarCamposproductos(...): Limpia los campos.
EliminarProductos(...): Elimina un producto por ID.

4. Clase ControladorReportes

Encargada de consultar reportes, facturas y ventas por fechas. Se adaptó el método BuscarFacturaMostrarDatosclientes(...) con el patrón State.

Método con Patrón State:

```java
public class ControladorReportes {

    private EstadoReporte estadoActual;

    // Interfaz interna del patrón State
    private interface EstadoReporte {
        void ejecutar();
    }

    // Método para cambiar y ejecutar el estado
    private void cambiarEstado(EstadoReporte nuevoEstado) {
        this.estadoActual = nuevoEstado;
        this.estadoActual.ejecutar();
    }
```

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

Otros métodos:

BucarFacturaDatosProductos(...): Muestra los productos vendidos en una factura.
MostrarTotalVentaPorFecha(...): 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.
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 para una mayor organización.
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.