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 BUILDER

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 los patrones de diseño State, Facade y potencialmente Builder si se desea construir objetos complejos como facturas o reportes paso a paso.

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.

Posible Patrón Builder:

El patrón Builder podría implementarse para construir objetos de tipo Factura o Venta de manera flexible y modular, aislando la creación de estos objetos complejos del controlador. Por ejemplo, se podría crear un FacturaBuilder que reciba cliente, productos, cantidades, y calcule los totales antes de generar el SQL de inserción.

Método con Patrón State:

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

Otros métodos:

SeleccionarProductoAlquiler(...)

BuscarCliente(...)

SeleccionarClienteAlquiler(...)

pasarproductosVentas(...)

eliminarProductoSeleccionadoResumenVenta(...)

calcularTotalPagar(...)

crearFactura(...)

realizarVenta(...)

limpiarLuegoVenta(...)

MostrarUltimaFactura(...)

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.

Patrón Builder (potencial):

Se podría aplicar el patrón Builder si en el futuro se desea construir objetos Cliente más complejos con múltiples pasos o validaciones antes de guardarlos.

Método con Patrón State:

mostrarClientes(...)

Otros métodos:

agregarCliente(...)

seleccionarCliente(...)

modificarCliente(...)

eliminarCliente(...)

limpiarCampos(...)

3. Clase ControladorProducto

Controlador para la gestión de productos disponibles para el alquiler. Integra los patrones Facade y State.

Patrón Facade:

Proporciona una interfaz centralizada para agregar, modificar, eliminar, seleccionar y consultar productos.

Patrón Builder (potencial):

Útil si se requiere crear productos con configuraciones variables o validaciones antes de agregarlos a la base.

Método con Patrón State:

mostrarProductos(...)

Otros métodos:

agregarProducto(...)

seleccionarProducto(...)

modificarProducto(...)

eliminarProducto(...)

limpiarCampos(...)

4. Clase ControladorReportes

Encargada de consultar reportes, facturas y ventas por fechas. Implementa el patrón Facade y usa State para modularidad.

Patrón Facade:

La clase agrupa todos los reportes posibles (por factura, por producto y por fecha).

Patrón Builder (potencial):

Ideal para generar reportes en objetos complejos que incluyan cliente, factura, detalle de productos y totales, como ReporteVentaBuilder.

Método con Patrón State:

buscarFacturaConDatosCliente(...)

Otros métodos:

buscarProductosPorFactura(...)

mostrarVentasPorFecha(...)

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.

El patrón Builder es recomendado para separar la lógica de construcción de objetos complejos, especialmente facturas o reportes.

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

Recomendaciones:

Aplicar Builder concretamente para Factura, DetalleVenta o ReporteVenta en objetos Java.

Continuar aplicando Facade como patrón estándar en nuevos controladores.

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.