TECNOLOGICO NACIONAL DE MEXICO

INSTITUTO TECNOLOGICO DE OAXACA

Carrera:

Ingeniería en Sistemas Computacionales

Materia:

Diseño e Implementación de software con patrones

Documentación de patrón Command

Docente:

Espinoza Pérez Jacob

Equipo:

Ordaz Pacheco Ruudvan

Santos Manuel Julia Marlenny

Vera Acevedo Héctor Aramís

Grupo:

7SB

Fecha de entrega:

11/05/2025

Documentación del Patrón Command en el Sistema de Ventas

Introducción

Esta documentación explica la implementación del **patrón de diseño Command** en el sistema de ventas, específicamente en la clase InterFacturacion. El objetivo es **desacoplar** las operaciones de venta (registrar, generar PDF, cancelar) para hacer el código más **flexible, mantenible y extensible**.

¿Qué es el Patrón Command?
El patrón Command es un **patrón de comportamiento** que:

- Encapsula una solicitud como un **objeto independiente**.
- Permite **parametrizar** operaciones (ej: registrar venta, generar PDF).
- Facilita operaciones **reversibles (undo/redo)**.
- Separa el **invocador** (quien ejecuta) del **receptor** (quien realiza la acción).

4. Componentes Clave

4.1. Interfaz ICommand

Define los métodos que todos los comandos deben implementar:

- execute(): Ejecuta la acción principal (ej: guardar venta).
- undo(): Revierte la acción (ej: cancelar venta).

```
 * @author necto
 */
public interface ICommand {
    void execute(); // Ejecuta la acción
    void undo();    // Revierte la acción
}
```

4.2. Comandos Concretos

RegistrarVentaCommand

- Responsabilidad: Registrar una venta en la base de datos y actualizar el stock.
- Métodos:
    - execute(): Guarda la cabecera y detalles de la venta.
    - undo(): Marca la venta como anulada y revierte el stock.

```
import controlador.Ctrl_RegistrarVenta;
import java.util.Iterator;
import javax.swing.JOptionPane;
import modelo.CabeceraVenta;
import modelo.DetalleVenta;
import vista.InterFacturacion;

public class RegistrarVentaCommand implements ICommand {
    private Ctrl_RegistrarVenta controlador;
    private CabeceraVenta cabecera;
    private DetalleVenta detalle;
    private InterFacturacion interfaz;

    public RegistrarVentaCommand(Ctrl_RegistrarVenta controlador, CabeceraVenta cabecera, DetalleVenta detalle, InterFacturacion interfaz) {
        this.controlador = controlador;
        this.cabecera = cabecera;
        this.detalle = detalle;
        this.interfaz = interfaz;
    }

    @Override
    public void execute() {
        if (controlador.guardar(cabecera)) {
            // Guardar detalles y actualizar stock
            for (DetalleVenta item : interfaz.listaProductos) {
                detalle.setIdProducto(item.getIdProducto());
                detalle.setCantidad(item.getCantidad());
                controlador.guardarDetalle(detalle);
                interfaz.RestarStockProductos(item.getIdProducto(), item.getCantidad());
            }
            JOptionPane.showMessageDialog(null, "Venta registrada exitosamente");
        }
    }

    @Override
    public void undo() {
        // Marcar la cabecera como anulada y revertir stock
        cabecera.setEstado(0); // 0 = anulado
        controlador.actualizar(cabecera, cabecera.getIdCabeceraventa());

        for (DetalleVenta item : interfaz.listaProductos) {
            interfaz.RestarStockProductos(item.getIdProducto(), -item.getCantidad()); // Sumar stock
        }
    }
}
```

GenerarPDFCommand

- **Responsabilidad: Generar un PDF de la factura.**
- **Métodos:**
    - **execute(): Crea el PDF con los datos del cliente y productos.**
    - **undo(): (Opcional) Elimina el PDF generado.**

```
import controlador.VentaPDF;
import vista.InterFacturacion;

public class GenerarPDFCommand implements ICommand {
    private VentaPDF pdfGenerator;
    private InterFacturacion interfaz;

    public GenerarPDFCommand(VentaPDF pdfGenerator, InterFacturacion interfaz) {
        this.pdfGenerator = pdfGenerator;
        this.interfaz = interfaz;
    }

    @Override
    public void execute() {
        pdfGenerator.DatosCliente(interfaz.idCliente);
        pdfGenerator.generarFacturaPDF();
    }

    @Override
    public void undo() {
        // Opcional: Eliminar el PDF generado (si es necesario)
    }
}
```

4.3. Clase Invoker

- **Responsabilidad: Ejecutar y gestionar comandos.**
- **Funcionalidades:**
    - **ejecutarComando(): Ejecuta un comando y lo guarda en el historial.**
    - **undo(): Ejecuta undo() del último comando.**

```java
import java.util.Stack;

public class Invoker {
    private Stack<ICommand> historial = new Stack<>();

    public void ejecutarComando(ICommand comando) {
        comando.execute();
        historial.push(comando);
    }

    public void undo() {
        if (!historial.isEmpty()) {
            ICommand comando = historial.pop();
            comando.undo();
        }
    }
}
```

4.4. Integración en InterFacturacion

- **Cambios principales:**
    - **Se añade un Invoker para gestionar comandos.**
    - **Se reemplaza la lógica directa de registro por comandos.**
    - **Se permite cancelar la última venta con undo().**

```java
private void jButton_RegistrarVentaActionPerformed(java.awt.event.ActionEvent evt) {

    if (!jComboBox_cliente.getSelectedItem().equals("Seleccione cliente:") && !listaProductos.isEmpty()) {
    ObtenerIdCliente();

    // Crear cabecera y detalle
    CabeceraVenta cabecera = new CabeceraVenta(
        0, idCliente,
        Double.parseDouble(txt_total_pagar.getText()),
        new SimpleDateFormat("yyyy/MM/dd").format(new Date()),
        1
    );

    DetalleVenta detalle = new DetalleVenta();

    // Comando para registrar venta
    ICommand registrarVenta = new RegistrarVentaCommand(ctrlVenta, cabecera, detalle, this);
    invoker.ejecutarComando(registrarVenta);

    // Comando para generar PDF
    ICommand generarPDF = new GenerarPDFCommand(pdfGenerator, this);
    invoker.ejecutarComando(generarPDF);

    // Limpiar interfaz
    listaProductos.clear();
    listaTablaProductos();
    txt_total_pagar.setText("0.0");
} else {
    JOptionPane.showMessageDialog(null, "¡Seleccione cliente y productos!");
}
```

```java
public class InterFacturacion extends javax.swing.JInternalFrame {

    //Modelo de los datos
    private DefaultTableModel modeloDatosProductos;
    //lista para el detalle de venta de los productos
    public ArrayList<DetalleVenta> listaProductos = new ArrayList<>();
    private DetalleVenta producto;

    public int idCliente = 0;//id del cliente sleccionado

    private Invoker invoker = new Invoker();
private Ctrl_RegistrarVenta ctrlVenta = new Ctrl_RegistrarVenta();
private VentaPDF pdfGenerator = new VentaPDF();
```

5. Flujo de Trabajo

5.1. Registrar una Venta

1. El usuario hace clic en **"Registrar Venta"**.
2. Se crean los comandos:

ICommand registrarVenta = new RegistrarVentaCommand(ctrlVenta, cabecera, detalle, this);

ICommand generarPDF = new GenerarPDFCommand(pdfGenerator, this);

3. El Invoker ejecuta los comandos:

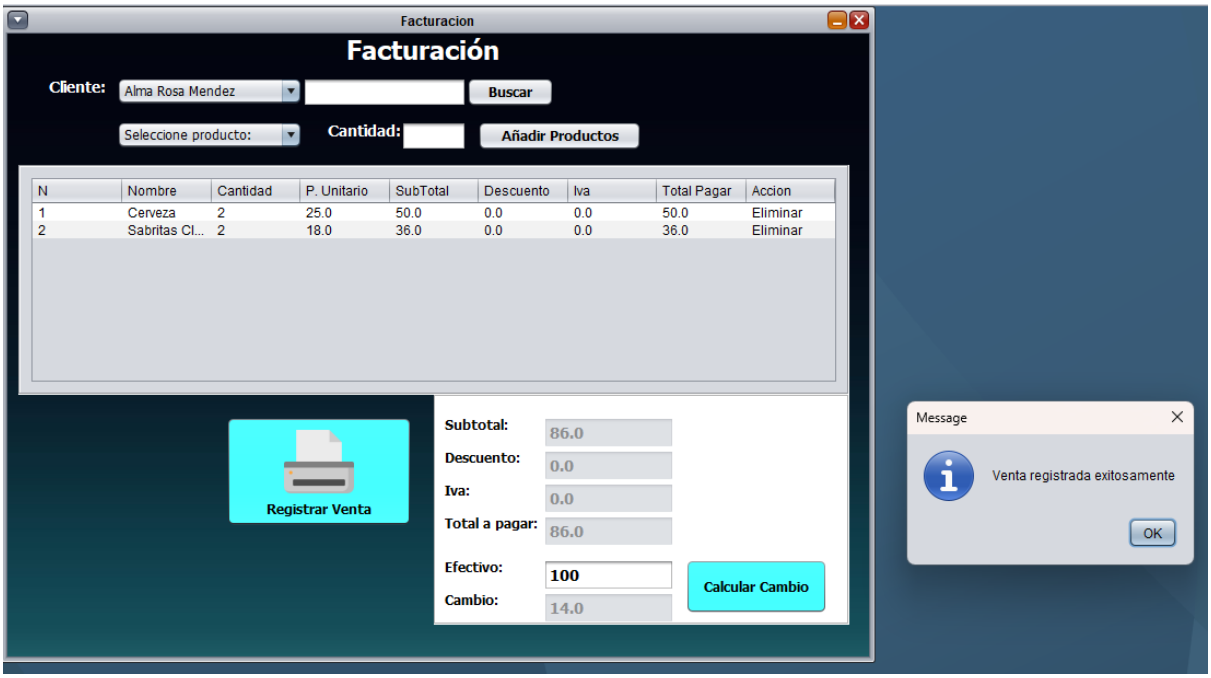
invoker.ejecutarComando(registrarVenta);

invoker.ejecutarComando(generarPDF);

Beneficios:

- **Desacoplamiento**: La interfaz no depende directamente de la lógica de negocio.
- **Flexibilidad**: Se pueden añadir nuevos comandos sin modificar el código existente.
- **Historial de operaciones**: Soporte para **undo/redo**.
- **Mantenibilidad**: Cada comando tiene una única responsabilidad.

Resultados



NOMBRE: Miscelanea Calicanto
TELEFONO: 9547894569
DIRECCION: Av Montoya
RAZON SOCIAL: Apoyando a la economia oaxaqueña

Nota: 020
Fecha: 2025/05/11

Datos del cliente:

Cedula/RUC:	Nombre:	Telefono:	Direccion:
9512156554	Alma Rosa Mendez	9512156554	Privada Texcoco

Cantidad:	Descripcion:	Precio Unitario:	Precio Total:
2	Cerveza	25.0	50.0
2	Sabritas Clasicas	18.0	36.0

Total a pagar: 86.0

¡Gracias por su compra!

Conclusión

- El patrón Command ha mejorado la estructura del sistema de ventas al:
- **Organizar** las operaciones en clases dedicadas.
- **Simplificar** la adición de nuevas funcionalidades.
- **Permitir** operaciones reversibles (útil para correcciones).
- Esta implementación sigue **buenas prácticas de diseño** y facilita el mantenimiento futuro.