



# TECNOLOGICO NACIONAL DE MEXICO INSTITUTO TECNOLOGICO DE OAXACA

Carrera:

Ingeniería en Sistemas Computacionales

Materia:

Diseño e Implementación de software con patrones

Documentación de patrón State

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 State en el Sistema de Ventas

# Introducción

Esta documentación explica la implementación del **patrón State** en el módulo de ventas de un sistema POS (Punto de Venta). El objetivo es manejar el flujo de una venta (agregar/eliminar productos, registrar venta) de manera flexible y mantenible mediante estados.

### ¿Qué es el patrón State?

Es un patrón de diseño que permite que un objeto **cambie su comportamiento** cuando cambia su estado interno. En este caso:

- Una venta puede estar: **Pendiente** (en proceso) o **Pagada** (finalizada).
- Cada estado define qué acciones son válidas (ej: no se pueden agregar productos a una venta ya pagada).

## Problema que resuelve

- Eliminar condicionales complejos (if-else) que verifican el estado de la venta.
- Centralizar la lógica de cada estado en clases separadas.

### Flujo de Trabajo

## **Estados y Comportamiento**

```
public interface EstadoVenta {
    void agregarProducto(InterFacturacion contexto, DetalleVenta producto);
    void eliminarProducto(InterFacturacion contexto, int idProducto);
    void registrarVenta(InterFacturacion contexto);
}
```

Estado	Agregar Producto	Eliminar Producto	Registrar Venta
Pendiente	Permite agregar	Permite eliminar	Cambia a estado "Pagado"
Pagado	Muestra error	Muestra error	Muestra error

## Secuencia Típica

- 1. **Inicio**: La venta se crea en estado Pendiente.
- 2. Agregar producto:
  - o El estado Pendiente añade el producto a la lista.
  - o Actualiza los totales.

```
public class EstadoPendiente implements EstadoVenta {
   @Override
   public void agregarProducto(InterFacturacion contexto, DetalleVenta producto) {
       contexto.getListaProductos().add(producto);
        contexto.actualizarTotales();
       JOptionPane.showMessageDialog(null, "Producto agregado correctamente");
   @Override
   public void eliminarProducto(InterFacturacion contexto, int idProducto) {
       contexto.getListaProductos().remove(idProducto - 1);
        contexto.actualizarTotales():
       JOptionPane.showMessageDialog(null, "Producto eliminado correctamente");
   @Override
   public void registrarVenta(InterFacturacion contexto) {
       if (contexto.getListaProductos().isEmpty()) {
           JOptionPane.showMessageDialog(null, "No hay productos para registrar");
           return;
        // Lógica para registrar la venta
        contexto.setEstado(new EstadoPagado());
       JOptionPane.showMessageDialog(null, "Venta registrada con éxito");
```

## 3. Eliminar producto:

- o El estado Pendiente remueve el producto.
- Actualiza los totales.

#### 4. Registrar venta:

- o El estado Pendiente valida que haya productos.
- Cambia el estado a Pagado.

#### 5. Post-registro:

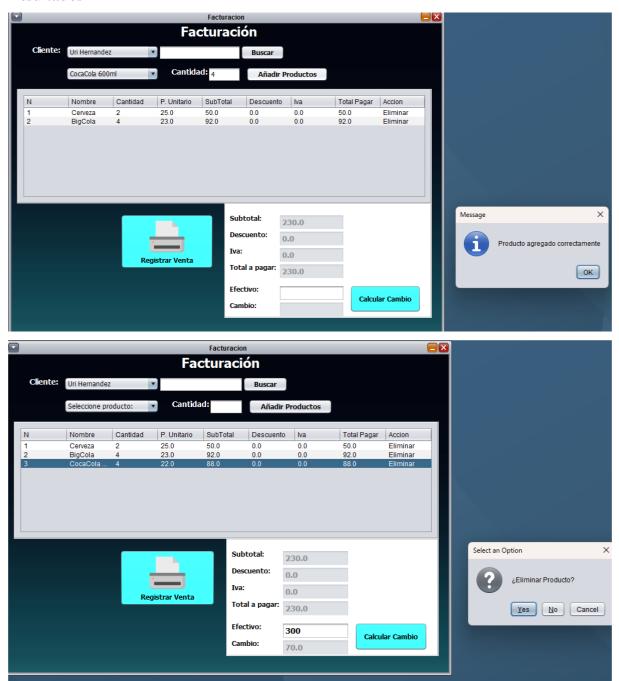
 Cualquier acción adicional (agregar/eliminar) es bloqueada por el estado Pagado.

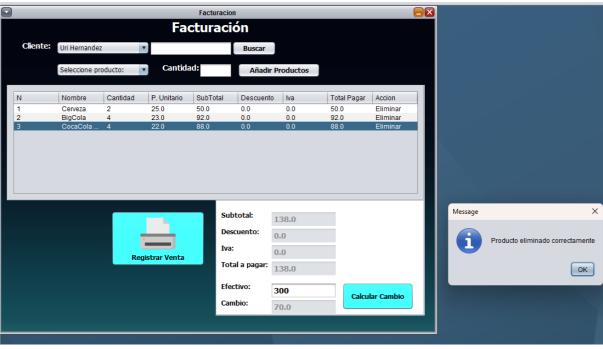
```
public class EstadoPagado implements EstadoVenta {
    @Override
    public void agregarProducto(InterFacturacion contexto, DetalleVenta producto) {
        JOptionPane.showMessageDialog(null, "No se pueden agregar productos a una venta ya pagada");
    }
    @Override
    public void eliminarProducto(InterFacturacion contexto, int idProducto) {
        JOptionPane.showMessageDialog(null, "No se pueden eliminar productos de una venta ya pagada");
    }
    @Override
    public void registrarVenta(InterFacturacion contexto) {
        JOptionPane.showMessageDialog(null, "La venta ya ha sido registrada");
    }
}
```

# **Beneficios:**

Ventaja	Elimina if (ventaPagada) {} en múltiples métodos.		
Código más limpio			
Fácil extensión	Añadir nuevos estados (ej: EstadoCancelado) sin modificar lógica existente.		
Mantenibilidad	Cambiar un estado no afecta a los demás.		

#### Resultados









# Conclusión

El patrón State ha transformado el manejo de ventas:

- Comportamiento claro: Cada estado define sus reglas.
- Escalable: Fácil añadir nuevos estados o modificar existentes.
- Robusto: Previene acciones inválidas (ej: modificar ventas pagadas).