TECNOLOGICO NACIONAL DE MEXICO

INSTITUTO TECNOLOGICO DE OAXACA

Carrera:

Ingeniería en Sistemas Computacionales

Materia:

Diseño e Implementación de software con patrones

Documentación

Docente:

Espinoza Pérez Jacob

Equipo:

Ordaz Pacheco Ruudvan

Santos Manuel Julia

Marlenny

Vera Acevedo Héctor Aramís

Grupo:

7SB

Fecha de entrega:

24/03/2025

Documentación del Patrón Facade en el Sistema de Ventas

Introducción

El patrón **Facade** (o **Fachada**) es un patrón de diseño estructural que proporciona una interfaz simplificada para un conjunto de interfaces más complejas en un subsistema. En este caso, se aplicó el patrón Facade para simplificar las operaciones relacionadas con las ventas, como la generación de facturas, la creación de reportes y la obtención de folios.

El objetivo principal de esta implementación es:

- **Simplificar** el uso de las operaciones de ventas.
- **Reducir el acoplamiento** entre las clases que manejan ventas.
- **Mejorar la mantenibilidad** del código.
- **Personalización eficiente**: Permite crear variantes de productos más fácilmente

Cambios Realizados

Creación de la Clase VentaFacade

Se creó la clase VentaFacade, que actúa como una interfaz única para las operaciones relacionadas con ventas. Esta clase encapsula la lógica de las siguientes operaciones:

1. Generación de facturas.
2. Generación de reportes de ventas.
3. Obtención y actualización de folios de venta.

```java
import modelo.Venta;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import conexion.Conexion;

public class VentaFacade {

    private Venta venta;
    private VentaPDF ventaPDF;
    private Reportes reportes;

    public VentaFacade() {
        this.venta = new Venta();
        this.ventaPDF = new VentaPDF();
        this.reportes = new Reportes();
    }

    // Método para generar la factura de venta
    public void generarFacturaPDF(int idCliente) {
        ventaPDF.DatosCliente(idCliente);
        ventaPDF.generarFacturaPDF();
    }

    // Método para generar el reporte de ventas
    public void generarReporteVentas() {
        reportes.ReportesVentas();
    }

    // Método para generar el reporte de ventas del día
    public void generarReporteVentasDia() {
        reportes.ReportesVentasDia();
    }

    // Método para obtener y actualizar el folio de venta
    public int obtenerYActualizarFolio() {
        return venta.obtenerYActualizarFolio();
    }
}
```

Modificación de la Clase VentaPDF

La clase VentaPDF ahora utiliza el VentaFacade para obtener el folio de venta en lugar de interactuar directamente con la clase Venta.

```
public class VentaPDF {

    private String nombreCliente;
    private String cedulaCliente;
    private String telefonoCliente;
    private String direccionCliente;

    private String fechaActual = "";
    private String nombreArchivoPDFVenta = "";
    private VentaFacade ventaFacade = new VentaFacade();

    /*Venta venta = new Venta();
    int folioVenta = venta.obtenerYActualizarFolio();
    String folioStr = String.format("%03d", folioVenta); // Formato 001, 002, 003*/
```

```
//metodo para generar la factura de venta
public void generarFacturaPDF() {
    try {

        int folioVenta = ventaFacade.obtenerYActualizarFolio();
        String folioStr = String.format("%03d", folioVenta);
        //cargar la fecha actual
        Date date = new Date();
        fechaActual = new SimpleDateFormat("yyyy/MM/dd").format(date);
        //cambiar el formato de la fecha de / a _
        String fechaNueva = "";
        for (int i = 0; i < fechaActual.length(); i++) {
            if (fechaActual.charAt(i) == '/') {
                fechaNueva = fechaActual.replace("/", "_");
            }
        }
```

Modificación de la Clase Reportes
La clase Reportes ahora utiliza el VentaFacade para generar reportes de ventas.

```
public class Reportes {

    private VentaFacade ventaFacade = new VentaFacade();
    /* ************************************************************************
    ************************************************************************ */
public void ReportesVentas() {
    Document documento = new Document();
    try {
        String ruta = System.getProperty("user.home");
        PdfWriter.getInstance(documento, new FileOutputStream(ruta + "/Desktop/Reporte_Ventas.pdf"));
        Image header = Image.getInstance("src/img/header1.jpg");
        header.scaleToFit(650, 1000);
        header.setAlignment(Chunk.ALIGN_CENTER);
        //formato al texto
        Paragraph parrafo = new Paragraph();
        parrafo.setAlignment(Paragraph.ALIGN_CENTER);
        parrafo.add("Reporte creado por \nMiscelanea Calicanto\n\n");
        parrafo.setFont(FontFactory.getFont("Tahoma", 18, Font.BOLD, BaseColor.DARK_GRAY));
        parrafo.add("Reporte de Ventas \n\n");
```

```
public void ReportesVentasDia() {
    Document documento = new Document();
    try {
        // Obtener la ruta del usuario
        String ruta = System.getProperty("user.home");

        // Crear el archivo PDF en el escritorio
        PdfWriter.getInstance(documento, new FileOutputStream(ruta + "/Desktop/Reporte_Ventas_Dia.pdf"));

        // Agregar la imagen de cabecera
        Image header = Image.getInstance("src/img/header1.jpg");
        header.scaleToFit(650, 1000);
        header.setAlignment(Chunk.ALIGN_CENTER);

        // Formato del texto del reporte
        Paragraph parrafo = new Paragraph();
        parrafo.setAlignment(Paragraph.ALIGN_CENTER);
        parrafo.add("Reporte creado por \nMiscelanea Calicanto\n\n");
        parrafo.setFont(FontFactory.getFont("Tahoma", 18, Font.BOLD, BaseColor.DARK_GRAY));
        parrafo.add("Reporte de Ventas del Día \n\n");

        documento.open();
```

Conclusión

La implementación del patrón **Facade** en el sistema de ventas permite simplificar y organizar mejor las operaciones relacionadas con ventas. Al encapsular la lógica compleja en una única interfaz (VentaFacade), se ha logrado reducir el acoplamiento entre las clases y mejorar la mantenibilidad del código. Este enfoque es especialmente útil en sistemas grandes donde se requiere una interacción simplificada con subsistemas complejos.