



TECNOLOGICO NACIONAL DE MEXICO INSTITUTO TECNOLÓGICO DE OAXACA

Carrera:

Ingeniería en Sistemas Computacionales

Materia:

Diseño e Implementación de software con patrones

Documentación de todos los patrones

Docente:

Espinoza Pérez Jacob

Equipo:

Ordaz Pacheco Ruudvan

Santos Manuel Julia Marlenny

Vera Acevedo Héctor Aramís

Grupo:

7SB

Fecha de entrega:

30/04/2025

Indice

Documentación de todos los patrones en el Sistema de Ventas	4
Introducción.....	4
Proxy-singleton-flyweight	5
TablaCategoriasSingleton	5
ConexionSingleton	6
CategoriaDAOProxy (Proxy)	7
CategoriaFlyweightFactory	9
AbstractFactory-prototype-builder.....	11
ProductoConcretoFactory Responsabilidades:.....	11
Producto.Builder (clase interna en Producto)	12
Producto (implementa Cloneable).....	14
Adapter-Facade-Bridge	15
Interfaz FacturaFormat (Sujeto).....	15
PDFFactura	16
HTMLFactura.....	19
FacturaAdapter (Adapter).....	22
FacturacionFacade (Facade)	23
Mediator	25
Interfaz ClienteMediator.....	25
ClienteMediatorImpl (Mediador Concreto).....	26
InterCliente	27
Ctrl_Cliente	28
Observer.....	29
Interfaz UsuarioObserver.....	29
Ctrl_Usuario (Subject).....	30
UsuarioLogger (Para registro de actividades):.....	32
UsuarioUIUpdater (Para actualizar la interfaz):.....	32
Integración del código a la vista.....	33
Responsability	34
StockHandler.....	34
ProductoSeleccionadoHandler	34
CamposVaciosHandler	35
NumerosValidosHandler	35
CantidadPositivaHandler	35
InterActualizarStock.....	36
FactoryMethod-Memento-Composite	37

ReporteFactory (Interfaz).....	37
ReporteClientesFactory, ReporteProductosFactory, etc. (Implementaciones concretas)	38
Reportes (Clase contexto).....	42
ReporteMemento	47
ReporteOriginator	48
ComponenteReporte (Interfaz)	49
ReporteCompuesto (Composite)	49
Resultados.....	51
Proxy-singleton-flyweight	51
Abstract Factory-Prototype-Build	54
Adapter-Facade-Bridge	56
Mediator	58
Observer.....	59
Responsability	61
FactoryMethod-Memento-Composite	62
Conclusión.....	65

Documentación de todos los patrones en el Sistema de Ventas

Introducción

El presente documento representa un exhaustivo trabajo de implementación de 15 patrones de diseño en el Sistema de Ventas desarrollado para la materia de Diseño e Implementación de Software con Patrones. Esta implementación masiva de patrones demuestra cómo los principios fundamentales del diseño orientado a objetos pueden aplicarse de manera conjunta y complementaria para resolver problemas complejos en sistemas de software empresariales.

Los patrones implementados abarcan las tres categorías principales:

1. **Patrones creacionales:** Abstract Factory, Builder, Prototype, Factory Method, Singleton
2. **Patrones estructurales:** Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy
3. **Patrones de comportamiento:** Chain of Responsibility, Mediator, Observer

Cada patrón fue cuidadosamente seleccionado para abordar necesidades específicas del sistema, desde la generación flexible de reportes (Factory Method, Composite) hasta la gestión de interacciones complejas entre componentes (Mediator, Observer). La implementación muestra cómo estos patrones pueden coexistir y complementarse, manteniendo al mismo tiempo una clara separación de responsabilidades y una alta cohesión en cada módulo.

El documento sigue una estructura que permite comprender tanto la teoría como la práctica de cada patrón, mostrando:

- El problema específico que resuelve
- Su implementación concreta en el sistema
- Las responsabilidades claramente delimitadas de cada componente
- Los beneficios tangibles obtenidos
- Los resultados visuales de la implementación

Este trabajo no solo cumple con los objetivos académicos, sino que también sirve como referencia práctica para futuros desarrollos que requieran aplicar múltiples patrones de diseño de manera integrada.

Proxy-singleton-flyweight

TablaCategoriasSingleton

Responsabilidades:

- Gestionar una única instancia del modelo de tabla de categorías
- Centralizar las operaciones de manipulación de datos en la tabla
- Proporcionar acceso global controlado al modelo de tabla

```
import javax.swing.table.DefaultTableModel;

public class TablaCategoriasSingleton {
    private static TablaCategoriasSingleton instancia;
    private DefaultTableModel model;

    private TablaCategoriasSingleton() {
        model = new DefaultTableModel();
        model.addColumn("idCategoria");
        model.addColumn("descripcion");
        model.addColumn("estado");
    }

    public static synchronized TablaCategoriasSingleton getInstancia() {
        if (instancia == null) {
            instancia = new TablaCategoriasSingleton();
        }
        return instancia;
    }

    public DefaultTableModel getModel() {
        return model;
    }

    public void limpiarTabla() {
        model.setRowCount(0);
    }

    public void agregarFila(Object[] fila) {
        model.addRow(fila);
    }
}
```

ConexionSingleton

Responsabilidades:

- Gestionar una única conexión a la base de datos
- Optimizar el uso de recursos de conexión
- Proporcionar acceso centralizado a la conexión

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Conexion {
    // 1. Instancia única como variable estática
    private static Connection instancia;

    // 2. Constructor privado para evitar instanciación
    private Conexion() {}

    // 3. Método estático para obtener la instancia (versión thread-safe)
    public static synchronized Connection getConexion() {
        if (instancia == null) {
            try {
                // 4. Crear la conexión solo si no existe
                instancia = DriverManager.getConnection(
                    "jdbc:mysql://localhost/bd_sistema_ventas",
                    "root",
                    "1234");
                System.out.println("Conexión establecida");
            } catch (SQLException e) {
                System.err.println("Error al conectar: " + e.getMessage());
                throw new RuntimeException("Error al establecer conexión", e);
            }
        }
        return instancia;
    }

    // 5. Método para cerrar la conexión
    public static void cerrarConexion() {
        if (instancia != null) {
            try {
                instancia.close();
                instancia = null;
                System.out.println("Conexión cerrada");
            } catch (SQLException e) {
                System.err.println("Error al cerrar conexión: " + e.getMessage());
            }
        }
    }
}
```

CategoriaDAOProxy (Proxy)

Implementa las validaciones y controles adicionales:

Responsabilidades:

- Validar datos antes de operaciones críticas
- Implementar logging para auditoría
- Controlar acceso a operaciones sensibles
- Gestionar caché si es necesario
- Delegar al objeto real después de validaciones

```
public class CategoriaDAOProxy implements ICategoriaDAO {
    private CategoriaDAOReal categoriaDAOReal;
    private Logger logger = Logger.getLogger(CategoriaDAOProxy.class.getName());

    public CategoriaDAOProxy() {
        this.categoriaDAOReal = new CategoriaDAOReal();
    }

    @Override
    public boolean guardar(Categoria categoria) {
        if(validarDescripcion(categoria.getDescripcion())) {
            logger.log(Level.INFO, "Intentando guardar categoría: {0}", categoria.getDescripcion());
            boolean resultado = categoriaDAOReal.guardar(categoria);
            if(resultado) {
                logger.log(Level.INFO, "Categoría guardada exitosamente");
            }
            return resultado;
        }
        return false;
    }

    @Override
    public boolean existeCategoria(String descripcion) {
        logger.log(Level.INFO, "Verificando existencia de categoría: {0}", descripcion);
        return categoriaDAOReal.existeCategoria(descripcion);
    }

    @Override
    public boolean actualizar(Categoria categoria, int idCategoria) {
        if(validarDescripcion(categoria.getDescripcion())) {
            logger.log(Level.INFO, "Actualizando categoría ID: {0}", idCategoria);
            return categoriaDAOReal.actualizar(categoria, idCategoria);
        }
        return false;
    }
}
```

```

@Override
public boolean eliminar(int idCategoria) {
    if(!categoriaTieneProductos(idCategoria)) {
        logger.log(Level.INFO, "Eliminando categoría ID: {0}", idCategoria);
        return categoriaDAOReal.eliminar(idCategoria);
    } else {
        JOptionPane.showMessageDialog(null,
            "No se puede eliminar: La categoría tiene productos asociados");
        return false;
    }
}

private boolean validarDescripcion(String descripcion) {
    return descripcion != null && !descripcion.trim().isEmpty();
}

private boolean categoriaTieneProductos(int idCategoria) {
    // Implementar lógica para verificar si hay productos asociados
    return false; // Cambiar por implementación real
}
}

```


CategoriaFlyweightFactory

Responsabilidades:

- Reutilizar objetos Categoria existentes
- Reducir la creación de objetos duplicados
- Centralizar la creación de instancias de Categoria

```
import java.util.HashMap;
import java.util.Map;

public class CategoriaFlyweightFactory {
    private static CategoriaFlyweightFactory instancia;
    private Map<String, Categoria> categoriasCache;

    private CategoriaFlyweightFactory() {
        categoriasCache = new HashMap<>();
    }

    public static synchronized CategoriaFlyweightFactory getInstancia() {
        if (instancia == null) {
            instancia = new CategoriaFlyweightFactory();
        }
        return instancia;
    }

    public Categoria getCategoria(String descripcion) {
        // Verificamos si ya existe una categoría con esta descripción
        if (categoriasCache.containsKey(descripcion)) {
            return categoriasCache.get(descripcion);
        } else {
            // Si no existe, creamos una nueva y la añadimos al cache
            Categoria nuevaCategoria = new Categoria(0, descripcion, 1);
            categoriasCache.put(descripcion, nuevaCategoria);
            return nuevaCategoria;
        }
    }

    public void clearCache() {
        categoriasCache.clear();
    }
}
```

1. Diagrama de Flujo de Operaciones
2. Interfaz gráfica envía solicitud al controlador (Ctrl_Categoria)
3. Controlador delega operación al proxy (CategoriaDAOProxy)
4. Proxy ejecuta validaciones y logging

- 5. Proxy delega operación válida al DAO real (CategoriaDAORreal)**
- 6. DAO real ejecuta operación en base de datos**
- 7. Resultado fluye de vuelta a través de la misma cadena**
- 8. Ventajas de la Implementación**
- 9. Eficiencia: Reduce creación de objetos innecesarios (Flyweight)**
- 10. Control centralizado: Singleton para recursos compartidos**
- 11. Seguridad: Validaciones robustas en el Proxy**
- 12. Extensibilidad: Fácil añadir nuevas validaciones**
- 13. Mantenibilidad: Separación clara de responsabilidades**
- 14. Transparencia: Misma interfaz para componentes existentes**

AbstractFactory-prototype-builder

ProductoConcretoFactory

Responsabilidades:

- Crear productos preconfigurados (crearProductoBasico(), crearProductoPremium()).
- Construir productos personalizados (crearProductoPersonalizado()).
- Centralizar la lógica de creación de objetos Producto.

```
public class ProductoConcretoFactory implements ProductoFactory {  
    @Override  
    public Producto crearProductoBasico() {  
        return new Producto.Builder()  
            .nombre("Producto Básico")  
            .cantidad(1)  
            .precio(10.0)  
            .descripcion("Producto estándar")  
            .porcentajeIva(12)  
            .idCategoria(1)  
            .estado(1)  
            .build();  
    }  
  
    @Override  
    public Producto crearProductoPremium() {  
        return new Producto.Builder()  
            .nombre("Producto Premium")  
            .cantidad(1)  
            .precio(50.0)  
            .descripcion("Producto de alta calidad")  
            .porcentajeIva(14)  
            .idCategoria(2)  
            .estado(1)  
            .build();  
    }  
  
    @Override  
    public Producto crearProductoPersonalizado(int idProducto, String nombre, int cantidad,  
                                                double precio, String descripcion,  
                                                int porcentajeIva, int idCategoria, int estado) {  
        return new Producto.Builder()  
            .idProducto(idProducto)  
            .nombre(nombre)  
            .cantidad(cantidad)  
            .precio(precio)  
            .descripcion(descripcion)  
            .porcentajeIva(porcentajeIva)  
            .idCategoria(idCategoria)  
            .estado(estado)  
            .build();  
    }  
}
```

Producto.Builder (clase interna en Producto)

Responsabilidades:

- Construir objetos Producto paso a paso.
- Validar atributos antes de la creación (build()).
- Permitir configuraciones flexibles (ej: idProducto(0).nombre("Laptop")).

```
public class Producto implements Cloneable {
    // Atributos
    private int idProducto;
    private String nombre;
    private int cantidad;
    private double precio;
    private String descripcion;
    private int porcentajeIva;
    private int idCategoria;
    private int estado;
    private String descripcionCategoria;

    // Constructor privado para el Builder
    private Producto(Builder builder) {
        this.idProducto = builder.idProducto;
        this.nombre = builder.nombre;
        this.cantidad = builder.cantidad;
        this.precio = builder.precio;
        this.descripcion = builder.descripcion;
        this.porcentajeIva = builder.porcentajeIva;
        this.idCategoria = builder.idCategoria;
        this.estado = builder.estado;
    }

    // Constructor público vacío
    public Producto() {
        this.idProducto = 0;
        this.nombre = "";
        this.cantidad = 0;
        this.precio = 0.0;
        this.descripcion = "";
        this.porcentajeIva = 0;
        this.idCategoria = 0;
        this.estado = 0;
    }
}
```

```
// Builder para Producto
public static class Builder {
    private int idProducto;
    private String nombre;
    private int cantidad;
    private double precio;
    private String descripcion;
    private int porcentajeIva;
    private int idCategoria;
    private int estado;
    private String descripcionCategoria;

    public Builder() {}

    public Builder idProducto(int idProducto) {
        this.idProducto = idProducto;
        return this;
    }

    public Builder nombre(String nombre) {
        this.nombre = nombre;
        return this;
    }

    public Builder descripcionCategoria(String descripcionCategoria) {
        this.descripcionCategoria = descripcionCategoria;
        return this;
    }

    public Producto build() {
        Producto producto = new Producto(this);
        producto.setDescripcionCategoria(this.descripcionCategoria);
        return producto;
    }

    public Builder cantidad(int cantidad) {
        this.cantidad = cantidad;
        return this;
    }

    public Builder precio(double precio) {
        this.precio = precio;
        return this;
    }
}
```

```

public Builder descripcion(String descripcion) {
    this.descripcion = descripcion;
    return this;
}

public Builder porcentajeIva(int porcentajeIva) {
    this.porcentajeIva = porcentajeIva;
    return this;
}

public Builder idCategoria(int idCategoria) {
    this.idCategoria = idCategoria;
    return this;
}

public Builder estado(int estado) {
    this.estado = estado;
    return this;
}
}

```

Producto (implementa Cloneable)

Responsabilidades:

- Clonar productos existentes mediante clone().
- Garantizar copias superficiales rápidas.

```

// Implementación de Prototype
@Override
public Producto clone() {
    try {
        return (Producto) super.clone();
    } catch (CloneNotSupportedException e) {
        throw new AssertionError(); // No debería ocurrir
    }
}

```

Flujo de Operación:

1. **Interfaz gráfica** solicita un tipo de producto (básico, premium, personalizado).
2. ProductoConcretoFactory instancia el objeto usando Producto.Builder.
3. **Ctrl_Producto** inicia construcción con `new Producto.Builder()`.
4. Configura atributos mediante métodos encadenados (ej: `.cantidad(10)`).
5. Finaliza con `build()`, que retorna el `Producto` instanciado.
6. **Interfaz gráfica** selecciona producto a duplicar.

7. `Producto.clone()` crea una copia idéntica.
8. La copia se modifica (opcional) y persiste.

Ventajas:

- Estandariza la creación de productos.
- Facilita la adición de nuevas variantes.
- Reduce acoplamiento entre clases.
- Elimina constructores complejos
- Inmutable tras su creación (`build()`).
- Legibilidad mejorada en código cliente.
- Evita recreación costosa desde cero
- Mantiene independencia del objeto original.
- Útil para plantillas o copias rápidas.

Adapter-Facade-Bridge

Interfaz FacturaFormat (Sujeto)

Define el contrato que deben implementar los adaptadores para diferentes formatos de factura:

Responsabilidades:

- Establecer los métodos fundamentales para generación de facturas
- Servir como contrato común para implementaciones de diferentes formatos
- Permitir la sustitución transparente entre formatos

```
public interface FacturaFormat {  
    void generarFactura(String nombreCliente, String cedulaCliente, String telefonoCliente,  
                        String direccionCliente, String fechaActual, String[][] productos,  
                        String totalPagar);  
    String getExtension();  
}
```

PDFFactura

Implementa la generación de facturas en formato PDF:

Características:

- Utiliza la biblioteca iText para generación de PDF
- Implementa el diseño profesional de facturas
- Cumple con el contrato FacturaFormat

```
import com.itextpdf.text.*;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;
import java.io.FileOutputStream;
import java.io.IOException;

public class PDFFactura implements FacturaFormat {
    private static final Font TITLE_FONT = new Font(Font.FontFamily.HELVETICA, 18, Font.BOLD);
    private static final Font HEADER_FONT = new Font(Font.FontFamily.HELVETICA, 12, Font.BOLD);
    private static final Font NORMAL_FONT = new Font(Font.FontFamily.HELVETICA, 10);

    @Override
    public void generarFactura(String nombreCliente, String cedulaCliente, String telefonoCliente,
                               String direccionCliente, String fechaActual, String[][] productos,
                               String totalPagar) {
        try {
            String nombreArchivo = "Factura_" + nombreCliente.replace(" ", "_") + "_" +
                                   fechaActual.replace("/", "_") + ".pdf";
            String rutaCompleta = "src/pdf/" + nombreArchivo;

            Document doc = new Document();
            PdfWriter.getInstance(doc, new FileOutputStream(rutaCompleta));
            doc.open();

            // Encabezado
            Paragraph title = new Paragraph("FACTURA", TITLE_FONT);
            title.setAlignment(Element.ALIGN_CENTER);
            doc.add(title);

            doc.add(Chunk.NEWLINE);

            // Información de la empresa
            PdfPTable empresaTable = new PdfPTable(2);
            empresaTable.setWidthPercentage(100);
            empresaTable.setWidths(new float[]{30f, 70f});

            empresaTable.addCell(createCell("MISCELÁNEA CALICANTO", HEADER_FONT, Element.ALIGN_LEFT, false));
            empresaTable.addCell(createCell("Fecha: " + fechaActual, NORMAL_FONT, Element.ALIGN_RIGHT, false));
            empresaTable.addCell(createCell("Av. Principal #123", NORMAL_FONT, Element.ALIGN_LEFT, false));
            empresaTable.addCell(createCell("Tel: 555-123456", NORMAL_FONT, Element.ALIGN_RIGHT, false));
            empresaTable.addCell(createCell("Ciudad, País", NORMAL_FONT, Element.ALIGN_LEFT, false));
            empresaTable.addCell(createCell("RUC: 1234567890123", NORMAL_FONT, Element.ALIGN_RIGHT, false));
```



```

doc.add(empresaTable);
doc.add(Chunk.NEWLINE);

// Información del cliente
PdfPTable clienteTable = new PdfPTable(2);
clienteTable.setWidthPercentage(100);
clienteTable.setWidths(new float[]{20f, 80f});

clienteTable.addCell(createCell("CLIENTE:", HEADER_FONT, Element.ALIGN_LEFT, true));
clienteTable.addCell(createCell("", HEADER_FONT, Element.ALIGN_LEFT, false));
clienteTable.addCell(createCell("Nombre:", NORMAL_FONT, Element.ALIGN_LEFT, false));
clienteTable.addCell(createCell(nombreCliente, NORMAL_FONT, Element.ALIGN_LEFT, false));
clienteTable.addCell(createCell("Cédula:", NORMAL_FONT, Element.ALIGN_LEFT, false));
clienteTable.addCell(createCell(cedulaCliente, NORMAL_FONT, Element.ALIGN_LEFT, false));
clienteTable.addCell(createCell("Teléfono:", NORMAL_FONT, Element.ALIGN_LEFT, false));
clienteTable.addCell(createCell(telefonoCliente, NORMAL_FONT, Element.ALIGN_LEFT, false));
clienteTable.addCell(createCell("Dirección:", NORMAL_FONT, Element.ALIGN_LEFT, false));
clienteTable.addCell(createCell(direccionCliente, NORMAL_FONT, Element.ALIGN_LEFT, false));

doc.add(clienteTable);
doc.add(Chunk.NEWLINE);

// Tabla de productos
PdfPTable productosTable = new PdfPTable(5);
productosTable.setWidthPercentage(100);
productosTable.setWidths(new float[]{10f, 40f, 15f, 15f, 20f});

// Encabezados de la tabla
productosTable.addCell(createCell("Cant.", HEADER_FONT, Element.ALIGN_CENTER, true));
productosTable.addCell(createCell("Descripción", HEADER_FONT, Element.ALIGN_LEFT, true));
productosTable.addCell(createCell("P. Unit.", HEADER_FONT, Element.ALIGN_RIGHT, true));
productosTable.addCell(createCell("IVA", HEADER_FONT, Element.ALIGN_RIGHT, true));
productosTable.addCell(createCell("Total", HEADER_FONT, Element.ALIGN_RIGHT, true));

// Agregar productos
for (String[] producto : productos) {
    productosTable.addCell(createCell(producto[1], NORMAL_FONT, Element.ALIGN_CENTER, false));
    productosTable.addCell(createCell(producto[0], NORMAL_FONT, Element.ALIGN_LEFT, false));
    productosTable.addCell(createCell("$" + producto[2], NORMAL_FONT, Element.ALIGN_RIGHT, false));
    productosTable.addCell(createCell("12%", NORMAL_FONT, Element.ALIGN_RIGHT, false));
    productosTable.addCell(createCell("$" + producto[3], NORMAL_FONT, Element.ALIGN_RIGHT, false));
}

doc.add(productosTable);
doc.add(Chunk.NEWLINE);

```

```

// Total
PdfPTable totalTable = new PdfPTable(2);
totalTable.setWidthPercentage(50);
totalTable.setHorizontalAlignment(Element.ALIGN_RIGHT);
totalTable.setWidths(new float[]{30f, 20f});

totalTable.addCell(createCell("TOTAL A PAGAR:", HEADER_FONT, Element.ALIGN_RIGHT, false));
totalTable.addCell(createCell("$" + totalPagar, HEADER_FONT, Element.ALIGN_RIGHT, false));

doc.add(totalTable);
doc.add(Chunk.NEWLINE);

// Mensaje final
Paragraph gracias = new Paragraph(";Gracias por su compra!", NORMAL_FONT);
gracias.setAlignment(Element.ALIGN_CENTER);
doc.add(gracias);

doc.close();
} catch (DocumentException | IOException e) {
    System.out.println("Error al generar PDF: " + e);
}
}

private PdfPCell createCell(String text, Font font, int alignment, boolean header) {
    PdfPCell cell = new PdfPCell(new Phrase(text, font));
    cell.setHorizontalAlignment(alignment);
    cell.setBorder(header ? PdfPCell.BOTTOM : PdfPCell.NO_BORDER);
    if (header) {
        cell.setBorderWidthBottom(2f);
        cell.setPaddingBottom(5f);
    }
    return cell;
}

@Override
public String getExtension() {
    return "pdf";
}
}

```

HTMLFactura

Implementa la generación de facturas en formato HTML:

Características:

- Genera documentos HTML estilizados
- Fácil visualización en navegadores web
- Cumple con el contrato FacturaFormat

```
import java.io.FileWriter;
import java.io.IOException;
import java.text.NumberFormat;
import java.util.Locale;

public class HTMLFactura implements FacturaFormat {
    @Override
    public void generarFactura(String nombreCliente, String cedulaCliente, String telefonoCliente,
                               String direccionCliente, String fechaActual, String[][] productos,
                               String totalPagar) {
        try {
            String nombreArchivo = "Factura_" + nombreCliente.replace(" ", "_") + "_" +
                                   fechaActual.replace("/", "_") + ".html";
            String rutaCompleta = "src/html/" + nombreArchivo;

            FileWriter writer = new FileWriter(rutaCompleta);

            writer.write("<!DOCTYPE html>\n");
            writer.write("<html lang='es'>\n");
            writer.write("<head>\n");
            writer.write("    <meta charset='UTF-8'>\n");
            writer.write("    <meta name='viewport' content='width=device-width, initial-scale=1.0'>\n");
            writer.write("    <title>Factura " + nombreCliente + "</title>\n");
            writer.write("    <style>\n");
            writer.write("        body { font-family: Arial, sans-serif; margin: 0; padding: 20px; }\n");
            writer.write("        .header { text-align: center; margin-bottom: 20px; }\n");
            writer.write("        .title { font-size: 24px; font-weight: bold; margin-bottom: 10px; }\n");
            writer.write("        .empresa-info { margin-bottom: 30px; }\n");
            writer.write("        .cliente-info { margin-bottom: 20px; }\n");
            writer.write("        table { width: 100%; border-collapse: collapse; margin-bottom: 20px; }\n");
            writer.write("        th, td { padding: 8px; text-align: left; border-bottom: 1px solid #ddd; }\n");
            writer.write("        th { background-color: #f2f2f2; font-weight: bold; }\n");
            writer.write("        .total { text-align: right; font-weight: bold; font-size: 18px; }\n");
            writer.write("        .footer { text-align: center; margin-top: 30px; font-style: italic; }\n");
            writer.write("    </style>\n");
            writer.write("</head>\n");
            writer.write("<body>\n");
```

```

// Encabezado
writer.write("    <div class='header'>\n");
writer.write("        <div class='title'>FACTURA</div>\n");
writer.write("        <div>MISCELÁNEA CALICANTO</div>\n");
writer.write("        <div>Av. Principal #123 - Ciudad, País</div>\n");
writer.write("        <div>Tel: 555-123456 - RUC: 1234567890123</div>\n");
writer.write("    </div>\n");

// Información de la factura y fecha
writer.write("    <div class='empresa-info'>\n");
writer.write("        <div><strong>Fecha:</strong> " + fechaActual + "</div>\n");
writer.write("    </div>\n");

// Información del cliente
writer.write("    <div class='cliente-info'>\n");
writer.write("        <h3>DATOS DEL CLIENTE</h3>\n");
writer.write("        <div><strong>Nombre:</strong> " + nombreCliente + "</div>\n");
writer.write("        <div><strong>Cédula:</strong> " + cedulaCliente + "</div>\n");
writer.write("        <div><strong>Teléfono:</strong> " + telefonoCliente + "</div>\n");
writer.write("        <div><strong>Dirección:</strong> " + direccionCliente + "</div>\n");
writer.write("    </div>\n");

// Tabla de productos
writer.write("    <table>\n");
writer.write("        <thead>\n");
writer.write("            <tr>\n");
writer.write("                <th>Cant.</th>\n");
writer.write("                <th>Descripción</th>\n");
writer.write("                <th>P. Unit.</th>\n");
writer.write("                <th>IVA</th>\n");
writer.write("                <th>Total</th>\n");
writer.write("            </tr>\n");
writer.write("        </thead>\n");
writer.write("        <tbody>\n");

```

```

// Productos
for (String[] producto : productos) {
    writer.write("                <tr>\n");
    writer.write("                <td>" + producto[1] + "</td>\n");
    writer.write("                <td>" + producto[0] + "</td>\n");
    writer.write("                <td>$" + producto[2] + "</td>\n");
    writer.write("                <td>12%\n");
    writer.write("                <td>$" + producto[3] + "</td>\n");
    writer.write("            </tr>\n");
}

writer.write("        </tbody>\n");
writer.write("    </table>\n");

// Total
writer.write("    <div class='total'>\n");
writer.write("        TOTAL A PAGAR: $" + totalPagar + "\n");
writer.write("    </div>\n");

// Pie de página
writer.write("    <div class='footer'>\n");
writer.write("        ;Gracias por su compra!\n");
writer.write("    </div>\n");

writer.write("</body>\n");
writer.write("</html>");

writer.close();
} catch (IOException e) {
    System.out.println("Error al generar HTML: " + e);
}
}

@Override
public String getExtension() {
    return "html";
}
}

```

FacturaAdapter (Adapter)

Adapta las implementaciones concretas a la interfaz común:

Responsabilidades:

- Actuar como intermediario entre el cliente y las implementaciones
- Traducir llamadas a métodos específicos de cada formato
- Ocultar complejidades de cada implementación

```
public class FacturaAdapter {
    private final FacturaFormat facturaFormat;

    public FacturaAdapter(FacturaFormat facturaFormat) {
        this.facturaFormat = facturaFormat;
    }

    public void generar(String nombreCliente, String cedulaCliente, String telefonoCliente,
        String direccionCliente, String fechaActual, String[][] productos,
        String totalPagar) {
        facturaFormat.generarFactura(nombreCliente, cedulaCliente, telefonoCliente,
            direccionCliente, fechaActual, productos, totalPagar);
    }

    public String getFormato() {
        return facturaFormat.getExtension().toUpperCase();
    }

    // Añadir este nuevo método
    public String getExtension() {
        return facturaFormat.getExtension();
    }
}
```

FacturacionFacade (Facade)

Proporciona una interfaz simplificada para el subsistema de facturación:

Responsabilidades:

- Ocultar la complejidad del subsistema
- Gestionar la creación de carpetas necesarias
- Coordinar el proceso completo de generación
- Proporcionar un punto único de acceso

```
import java.io.File;
import modelo.HTMLFactura;
import modelo.PDFFactura;

public class FacturacionFacade {
    private FacturaAdapter adapter;

    public FacturacionFacade(String formato) {
        crearCarpetasSiNoExisten(); // Asegurar que las carpetas existan
        setFormato(formato);
    }

    public void setFormato(String formato) {
        if (formato.equalsIgnoreCase("PDF")) {
            this.adapter = new FacturaAdapter(new PDFFactura());
        } else if (formato.equalsIgnoreCase("HTML")) {
            this.adapter = new FacturaAdapter(new HTMLFactura());
        }
    }

    private void crearCarpetasSiNoExisten() {
        File carpetaPdf = new File("src/pdf");
        File carpetaHtml = new File("src/html");

        if (!carpetaPdf.exists()) {
            carpetaPdf.mkdirs();
            System.out.println("Carpeta PDF creada");
        }

        if (!carpetaHtml.exists()) {
            carpetaHtml.mkdirs();
            System.out.println("Carpeta HTML creada");
        }
    }
}
```

```

public String generarFacturaCompleta(String nombreCliente, String cedulaCliente, String telefonoCliente,
String direccionCliente, String fechaActual, String[][] productos,
String totalPagar) {
    if (adapter == null) {
        System.err.println("Error: No se ha configurado un adaptador de factura");
        return null;
    }

    try {
        System.out.println("Generando factura en formato " + adapter.getFormato());
        String nombreArchivo = "Factura_" + nombreCliente.replace(" ", "_") + "_" +
            fechaActual.replace("/", "_") + "." + adapter.getExtension();

        String rutaCompleta = "src/" + adapter.getExtension() + "/" + nombreArchivo;

        File carpeta = new File("src/" + adapter.getExtension());
        if (!carpeta.exists()) {
            if (!carpeta.mkdirs()) {
                System.err.println("Error al crear directorio: " + carpeta.getAbsolutePath());
                return null;
            }
        }

        adapter.generar(nombreCliente, cedulaCliente, telefonoCliente,
            direccionCliente, fechaActual, productos, totalPagar);

        System.out.println("Documento generado en: " + rutaCompleta);
        return rutaCompleta;
    } catch (Exception e) {
        System.err.println("Error en FacturacionFacade: " + e.getMessage());
        e.printStackTrace();
        return null;
    }
}

```

Bridge entre Generación y Formatos

El patrón Bridge se manifiesta en:

1. **Abstracción:** La interfaz FacturaFormat
2. **Implementación:** Las clases PDFFactura y HTMLFactura
3. **Desacoplamiento:** El cliente trabaja con la abstracción sin conocer detalles

Beneficios:

- Cambios en formatos no afectan al cliente
- Fácil adición de nuevos formatos
- Implementaciones intercambiables

Mediator

Interfaz ClienteMediator

La interfaz ClienteMediator define el contrato que debe implementar el mediador para gestionar las interacciones entre los componentes:

```
import modelo.Cliente;
import vista.InterCliente;

public interface ClienteMediator {
    void registrarCliente(Cliente cliente);
    void notificar(String mensaje, InterCliente origen);
    void setInterfazCliente(InterCliente interfaz);
}
```

Responsabilidades:

- registrarCliente: Coordina el proceso completo de registro
- notificar: Maneja la comunicación de mensajes al usuario
- setInterfazCliente: Establece la referencia a la interfaz gráfica

ClienteMediatorImpl (Mediador Concreto)

Implementa la lógica de mediación entre los componentes:

```
import controlador.Ctrl_Cliente;
import modelo.Cliente;
import vista.InterCliente;
import javax.swing.JOptionPane;

public class ClienteMediatorImpl implements ClienteMediator {
    private InterCliente interfazCliente;
    private final Ctrl_Cliente controladorCliente;

    public ClienteMediatorImpl() {
        this.controladorCliente = new Ctrl_Cliente();
    }

    @Override
    public void registrarCliente(Cliente cliente) {
        if (!controladorCliente.existeCliente(cliente.getCedula())) {
            if (controladorCliente.guardar(cliente)) {
                notificar("Registro Guardado", null);
                interfazCliente.limpiarCampos();
            } else {
                notificar("Error al Guardar", null);
            }
        } else {
            notificar("El cliente ya está registrado", null);
        }
    }

    @Override
    public void notificar(String mensaje, InterCliente origen) {
        JOptionPane.showMessageDialog(null, mensaje);
    }

    @Override
    public void setInterfazCliente(InterCliente interfaz) {
        this.interfazCliente = interfaz;
    }
}
```

Flujo de trabajo:

1. Recibe la solicitud de registro desde la interfaz
2. Valida la existencia del cliente
3. Gestiona el proceso de guardado
4. Notifica resultados
5. Solicita limpieza de campos

InterCliente

La interfaz gráfica fue adaptada para:

1. Recibir el mediador en su constructor
2. Delegar todas las acciones al mediador

```
public class InterCliente extends javax.swing.JInternalFrame {  
  
    private ClienteMediator mediator;  
  
    public InterCliente(ClienteMediator mediator) {  
        initComponents();  
        this.setSize(new Dimension(400, 300));  
        this.setTitle("Nuevo Cliente");  
        this.mediator = mediator; // Primero asignamos el mediator  
        this.mediator.setInterfazCliente(this);  
        if (mediator == null) {  
            throw new IllegalArgumentException("Mediator no puede ser null");  
        }  
    }  
}
```

Ctrl_Cliente

El controlador mantuvo su funcionalidad, pero ahora es invocado por el mediador:

```
// Verificar que todos los campos estén completos
if (!txt_nombre.getText().isEmpty() && !txt_apellido.getText().isEmpty() &&
    !txt_cedula.getText().isEmpty() && !txt_telefono.getText().isEmpty()) {

    if (!txt_cedula.getText().trim().equals(txt_telefono.getText().trim())) {
        mediator.notificar("Ambos teléfonos deben ser iguales.", this);
        txt_cedula.setBackground(Color.red);
        txt_telefono.setBackground(Color.red);
        return;
    }

    if (txt_cedula.getText().length() != 10 || !txt_cedula.getText().matches("[0-9]+") ||
        txt_telefono.getText().length() != 10 || !txt_telefono.getText().matches("[0-9]+")) {
        mediator.notificar("Ambos teléfonos deben tener exactamente 10 números.", this);
        txt_cedula.setBackground(Color.red);
        txt_telefono.setBackground(Color.red);
        return;
    }

    Cliente cliente = new Cliente();
    cliente.setNombre(txt_nombre.getText().trim());
    cliente.setApellido(txt_apellido.getText().trim());
    cliente.setCedula(txt_cedula.getText().trim());
    cliente.setTelefono(txt_telefono.getText().trim());
    cliente.setDireccion(txt_direccion.getText().trim());
    cliente.setEstado(1);

    mediator.registrarCliente(cliente);
} else {
    mediator.notificar("Completa todos los campos", this);
    txt_nombre.setBackground(Color.red);
    txt_apellido.setBackground(Color.red);
    txt_cedula.setBackground(Color.red);
    txt_telefono.setBackground(Color.red);
    txt_direccion.setBackground(Color.red);
}

// Limpiar los campos después de la validación
this.Limpiar();
```

Flujo de Registro de Cliente

1. Usuario ingresa datos y hace clic en "Guardar"
2. InterCliente crea objeto Cliente y lo envía al Mediador
3. Mediador:
 - Verifica existencia mediante Ctrl_Cliente
 - Intenta guardar el registro
 - Notifica resultados

4. InterCliente muestra visualmente.

Ventajas del Patrón Singleton en este Código

1. **Control centralizado:** Todas las partes del sistema utilizan la misma instancia de conexión
2. **Eficiencia en recursos:** Evita la creación múltiple de conexiones a la BD
3. **Consistencia:** Garantiza que todas las operaciones trabajen con el mismo estado de conexión
4. **Facilidad de mantenimiento:** Cambios en la configuración de conexión se realizan en un solo lugar
5. **Thread-safe:** La implementación sincronizada previene problemas en entornos concurrentes

Observer

Interfaz UsuarioObserver

La interfaz UsuarioObserver define el contrato que deben implementar todos los observadores para recibir notificaciones:

Responsabilidades:

- Recibir notificaciones cuando ocurren cambios en los usuarios
- Ejecutar acciones específicas según el tipo de cambio (creación, actualización o eliminación)

```
//  
public interface UsuarioObserver {  
    void update(String mensaje, Usuario usuario);  
}  
  
//  
public interface UsuarioSubject {  
    void registrarObserver(UsuarioObserver observer);  
    void removerObserver(UsuarioObserver observer);  
    void notificarObservers(String accion, Usuario usuario);  
}
```

Ctrl_Usuario (Subject)

La clase Ctrl_Usuario fue modificada para implementar el rol de Subject en el patrón Observer:

Flujo de trabajo:

1. Los componentes interesados se registran como observadores
2. Cuando ocurre un cambio en los usuarios (creación, actualización o eliminación)
3. El controlador notifica a todos los observadores registrados
4. Cada observador ejecuta su lógica específica

```
public class Ctrl_Usuario implements UsuarioSubject {
    private List<UsuarioObserver> observers = new ArrayList<>();

    // Métodos existentes...

    @Override
    public void registrarObserver(UsuarioObserver observer) {
        observers.add(observer);
    }

    @Override
    public void removerObserver(UsuarioObserver observer) {
        observers.remove(observer);
    }

    @Override
    public void notificarObservers(String accion, Usuario usuario) {
        for (UsuarioObserver observer : observers) {
            observer.update(accion, usuario);
        }
    }

    // Modificar métodos existentes para incluir notificaciones

    public boolean guardar(Usuario objeto) {
        boolean respuesta = false;
        Connection cn = Conexion.getConexion();
        try {
            PreparedStatement consulta = cn.prepareStatement("insert into tb_usuario values(?, ?, ?, ?, ?, ?, ?)");
            consulta.setInt(1, 0); //id
            consulta.setString(2, objeto.getNombre());
            consulta.setString(3, objeto.getApellido());
            consulta.setString(4, objeto.getUsuario());
            consulta.setString(5, objeto.getPassword());
            consulta.setString(6, objeto.getTelefono());
            consulta.setInt(7, objeto.getEstado());
            if (consulta.executeUpdate() > 0) {
                respuesta = true;
                notificarObservers("CREACION", objeto); // Notificar creación
            }
            Conexion.cerrarConexion();
        } catch (SQLException e) {
            System.out.println("Error al guardar usuario: " + e);
        }
        return respuesta;
    }
}
```

```

public boolean actualizar(Usuario objeto, int idUsuario) {
    boolean respuesta = false;
    Connection cn = Conexion.getConnection();
    try {
        PreparedStatement consulta = cn.prepareStatement("update tb_usuario set nombre=?, apellido = ?, usuario = ?, password= ?, telefono = ?, estado
        consulta.setString(1, objeto.getNombre());
        consulta.setString(2, objeto.getApellido());
        consulta.setString(3, objeto.getUsuario());
        consulta.setString(4, objeto.getPassword());
        consulta.setString(5, objeto.getTelefono());
        consulta.setInt(6, objeto.getEstado());
        if (consulta.executeUpdate() > 0) {
            respuesta = true;
            notificarObservers("ACTUALIZACION", objeto); // Notificar actualización
        }
        Conexion.cerrarConexion();
    } catch (SQLException e) {
        System.out.println("Error al actualizar usuario: " + e);
    }
    return respuesta;
}

public boolean eliminar(int idUsuario) {
    boolean respuesta = false;
    Connection cn = Conexion.getConnection();
    try {
        Usuario usuarioEliminado = obtenerUsuarioPorId(idUsuario); // Necesitarás implementar este método
        PreparedStatement consulta = cn.prepareStatement(
            "delete from tb_usuario where idUsuario =" + idUsuario + "''");
        consulta.executeUpdate();
        if (consulta.executeUpdate() > 0) {
            respuesta = true;
            notificarObservers("ELIMINACION", usuarioEliminado); // Notificar eliminación
        }
        Conexion.cerrarConexion();
    } catch (SQLException e) {
        System.out.println("Error al eliminar usuario: " + e);
    }
    return respuesta;
}

public boolean existeUsuario(String usuario) {
    boolean respuesta = false;
    String sql = "select usuario from tb_usuario where usuario = '" + usuario + "'";
    Statement st;
    try {
        Connection cn = Conexion.getConnection();
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }
    } catch (SQLException e) {
        System.out.println("Error al consultar usuario: " + e);
    }
    return respuesta;
}

/**
 * *****
 * metodo para Iniciar Sesion
 * *****
 */
public boolean loginUser(Usuario objeto) {
    boolean respuesta = false;
    Connection cn = Conexion.getConnection();
    String sql = "select usuario, password from tb_usuario where usuario = '" + objeto.getUsuario() + "' and password = '" + objeto.getPassword() +
    Statement st;
    try {
        st = cn.createStatement();
        ResultSet rs = st.executeQuery(sql);
        while (rs.next()) {
            respuesta = true;
        }
    } catch (SQLException e) {
        System.out.println("Error al Iniciar Sesion");
        JOptionPane.showMessageDialog(null, "Error al Iniciar Sesion");
    }
    return respuesta;
}

```

```

    public Usuario obtenerUsuarioPorId(int idUsuario) {
        Usuario usuario = null;
        Connection cn = Conexion.getConnection();
        String sql = "SELECT idUsuario, nombre, apellido, usuario, password, telefono, estado FROM tb_usuario WHERE idUsuario = ?";

        try {
            PreparedStatement consulta = cn.prepareStatement(sql);
            consulta.setInt(1, idUsuario);
            ResultSet rs = consulta.executeQuery();

            if (rs.next()) {
                usuario = new Usuario();
                usuario.setIdUsuario(rs.getInt("idUsuario"));
                usuario.setNombre(rs.getString("nombre"));
                usuario.setApellido(rs.getString("apellido"));
                usuario.setUsuario(rs.getString("usuario"));
                usuario.setPassword(rs.getString("password"));
                usuario.setTelefono(rs.getString("telefono"));
                usuario.setEstado(rs.getInt("estado"));
            }

            Conexion.cerrarConexion();
        } catch (SQLException e) {
            System.out.println("Error al obtener usuario por ID: " + e);
        }

        return usuario;
    }
}

```

UsuarioLogger (Para registro de actividades):

```

import modelo.Usuario;
import modelo.UsuarioObserver;

public class UsuarioLogger implements UsuarioObserver {
    @Override
    public void update(String accion, Usuario usuario) {
        System.out.println("[LOG] Acción: " + accion +
            " | Usuario: " + usuario.getUsuario() +
            " | Nombre: " + usuario.getNombre());
    }
}

```

UsuarioUIUpdater (Para actualizar la interfaz):

```

import modelo.Usuario;
import modelo.UsuarioObserver;

public class UsuarioUIUpdater implements UsuarioObserver {
    private InterGestionarUsuario gestionarUsuario;

    public UsuarioUIUpdater(InterGestionarUsuario gestionarUsuario) {
        this.gestionarUsuario = gestionarUsuario;
    }

    @Override
    public void update(String accion, Usuario usuario) {
        if (accion.equals("CREACION") || accion.equals("ACTUALIZACION") || accion.equals("ELIMINACION")) {
            gestionarUsuario.CargarTablaUsuarios(); // Refrescar tabla
        }
    }
}

```


Integración del código a la vista

```
public class InterGestionarUsuario extends javax.swing.JInternalFrame {

    private int idUsuario = 0;
    private Ctrl_Usuario ctrlUsuario;

    public InterGestionarUsuario() {
        initComponents();
        this.setSize(new Dimension(900, 500));
        this.setTitle("Gestionar Usuarios");
        //Cargar tabla
        this.CargarTablaUsuarios();
        ctrlUsuario = new Ctrl_Usuario();
        // Registrar observadores
        ctrlUsuario.registrarObserver(new UsuarioLogger());
        ctrlUsuario.registrarObserver(new UsuarioUIUpdater(this));

        //insertar imagen en nuestro JLabel
        ImageIcon wallpaper = new ImageIcon("src/img/fondo3.jpg");
        Icon icono = new ImageIcon(wallpaper.getImage().getScaledInstance(900, 500, WIDTH));
        jLabel_wallpaper.setIcon(icono);
        this.repaint();
    }
}
```

Ventajas del Patrón Observer en este Código

1. **Desacoplamiento:** Los componentes no necesitan conocerse entre sí directamente
2. **Extensibilidad:** Fácil agregar nuevos observadores sin modificar el código existente
3. **Actualizaciones en tiempo real:** Cambios se reflejan inmediatamente en todos los componentes interesados
4. **Mantenibilidad:** Cada observador tiene una única responsabilidad clara
5. **Consistencia:** Todos los componentes reciben la misma información actualizada

Responsability

StockHandler

La interfaz StockHandler define el contrato que deben implementar los manejadores para procesar validaciones:

Responsabilidades:

- handle(): Ejecuta la validación específica y decide si pasa la solicitud al siguiente eslabón.
- setNext(): Establece el siguiente manejador en la cadena.

```
//  
public interface StockHandler {  
    void setNext(StockHandler next);  
    boolean handle(InterActualizarStock frame);  
}
```

Clases Concretas de Manejadores

Cada manejador implementa una validación específica:

ProductoSeleccionadoHandler

- Verifica que se haya seleccionado un producto en el JComboBox.

```
import javax.swing.JOptionPane;  
import vista.InterActualizarStock;  
  
public class ProductoSeleccionadoHandler implements StockHandler {  
    private StockHandler next;  
  
    @Override  
    public void setNext(StockHandler next) {  
        this.next = next;  
    }  
  
    @Override  
    public boolean handle(InterActualizarStock frame) {  
        if (frame.jComboBox_producto.getSelectedItem().equals("Seleccione producto:")) {  
            JOptionPane.showMessageDialog(null, "Seleccione un producto");  
            return false;  
        }  
        return next == null || next.handle(frame);  
    }  
}
```

CamposVaciosHandler

- Valida que el campo de cantidad no esté vacío.

```
public class CamposVaciosHandler implements StockHandler {
    private StockHandler next;

    @Override
    public void setNext(StockHandler next) {
        this.next = next;
    }

    @Override
    public boolean handle(InterActualizarStock frame) {
        if (frame.txt_cantidad_nueva.getText().isEmpty()) {
            JOptionPane.showMessageDialog(null, "Ingrese una nueva cantidad para sumar el stock del producto");
            return false;
        }
        return next == null || next.handle(frame);
    }
}
```

NumerosValidosHandler

- Comprueba que el valor ingresado sea numérico.

```
public class NumerosValidosHandler implements StockHandler {
    private StockHandler next;

    @Override
    public void setNext(StockHandler next) {
        this.next = next;
    }

    @Override
    public boolean handle(InterActualizarStock frame) {
        try {
            Integer.parseInt(frame.txt_cantidad_nueva.getText().trim());
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "En la cantidad no se admiten caracteres no numéricos");
            return false;
        }
        return next == null || next.handle(frame);
    }
}
```

CantidadPositivaHandler

- Asegura que la cantidad sea mayor a cero.

```
public class CantidadPositivaHandler implements StockHandler {
    private StockHandler next;

    @Override
    public void setNext(StockHandler next) {
        this.next = next;
    }

    @Override
    public boolean handle(InterActualizarStock frame) {
        int cantidad = Integer.parseInt(frame.txt_cantidad_nueva.getText().trim());
        if (cantidad <= 0) {
            JOptionPane.showMessageDialog(null, "La cantidad no puede ser cero ni negativa");
            return false;
        }
        return next == null || next.handle(frame);
    }
}
```

Flujo de trabajo:

1. La interfaz InterActualizarStock envía la solicitud al primer manejador.
2. Cada manejador procesa su validación y decide si continúa la cadena.
3. Si alguna validación falla, se muestra un mensaje de error y se detiene el proceso.

InterActualizarStock

La interfaz gráfica fue adaptada para:

1. **Configurar la cadena** de manejadores en el método jButton1ActionPerformed.
2. **Delegar las validaciones** a la cadena antes de actualizar el stock.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    //validamos seleccion del producto  
    StockHandler productoSeleccionado = new ProductoSeleccionadoHandler();  
    StockHandler camposVacios = new CamposVaciosHandler();  
    StockHandler numerosValidos = new NumerosValidosHandler();  
    StockHandler cantidadPositiva = new CantidadPositivaHandler();  
  
    productoSeleccionado.setNext(camposVacios);  
    camposVacios.setNext(numerosValidos);  
    numerosValidos.setNext(cantidadPositiva);  
  
    // Ejecutar la cadena de validaciones  
    if (!productoSeleccionado.handle(this)) {  
        return; // Si alguna validación falla, salir del método  
    }  
  
    // Si todas las validaciones pasan, proceder con la actualización  
    Producto producto = new Producto();  
    Ctrl_Producto controlProducto = new Ctrl_Producto();  
    int stockActual = Integer.parseInt(txt_cantidad_actual.getText().trim());  
    int stockNuevo = Integer.parseInt(txt_cantidad_nueva.getText().trim());  
  
    stockNuevo = stockActual + stockNuevo;  
    producto.setCantidad(stockNuevo);  
  
    if (controlProducto.actualizarStock(producto, idProducto)) {  
        JOptionPane.showMessageDialog(null, "Stock Actualizado");  
        jComboBox_producto.setSelectedItem("Seleccione producto:");  
        txt_cantidad_actual.setText("");  
        txt_cantidad_nueva.setText("");  
        this.CargarComboProductos();  
    } else {  
        JOptionPane.showMessageDialog(null, "Error al Actualizar Stock");  
    }  
}
```

FactoryMethod-Memento-Composite

ReporteFactory (Interfaz)

Responsabilidades:

1. Definir el contrato común para todas las fábricas concretas de reportes
2. Establecer los métodos obligatorios para generación de reportes
3. Garantizar consistencia en la estructura de todos los reportes
4. Permitir la sustitución transparente entre diferentes tipos de reportes

```
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import java.io.FileNotFoundException;
import java.sql.SQLException;
import java.io.IOException;

public interface ReporteFactory {
    Document crearDocumento() throws DocumentException, FileNotFoundException;
    void agregarContenido(Document documento) throws DocumentException, SQLException, IOException;
    String obtenerNombreArchivo();
}
```

ReporteClientesFactory, ReporteProductosFactory, etc. (Implementaciones concretas)

Responsabilidades:

1. Implementar la lógica específica para cada tipo de reporte
2. Gestionar la conexión y consultas a la base de datos para su dominio
3. Definir la estructura de tablas y contenido específico
4. Proporcionar el nombre de archivo adecuado para cada reporte
5. Mantener consistencia en el formato de presentación

```
public class ReporteClientesFactory implements ReporteFactory {
    @Override
    public Document crearDocumento() throws DocumentException, FileNotFoundException {
        String ruta = System.getProperty("user.home");
        Document documento = new Document();
        PdfWriter.getInstance(documento, new FileOutputStream(ruta + "/Desktop/Reporte_Clientes.pdf"));
        return documento;
    }

    @Override
    public void agregarContenido(Document documento) throws DocumentException, SQLException, IOException {
        // Código para agregar contenido específico de clientes
        Image header = Image.getInstance("src/img/header1.jpg");
        header.scaleToFit(650, 1000);
        header.setAlignment(Chunk.ALIGN_CENTER);

        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 Clientes \n\n");

        documento.add(header);
        documento.add(parrafo);

        PdfPTable tabla = new PdfPTable(5);
        tabla.addCell("Codigo");
        tabla.addCell("Nombres");
        tabla.addCell("Cedula");
        tabla.addCell("Telefono");
        tabla.addCell("Direccion");

        Connection cn = Conexion.getConexion();
        PreparedStatement pst = cn.prepareStatement(
            "select idCliente, concat(nombre, ' ', apellido) as nombres, cedula, telefono, direccion from tb_cliente");
        ResultSet rs = pst.executeQuery();
        if (rs.next()) {
            do {
                tabla.addCell(rs.getString(1));
                tabla.addCell(rs.getString(2));
                tabla.addCell(rs.getString(3));
                tabla.addCell(rs.getString(4));
                tabla.addCell(rs.getString(5));
            } while (rs.next());
            documento.add(tabla);
        }
    }
}
```

```

@Override
public String obtenerNombreArchivo() {
    return "Reporte_Clientes.pdf";
}

public void generarReporte(ReporteFactory factory) {
    try {
        Document documento = factory.crearDocumento();
        documento.open();
        factory.agregarContenido(documento);
        documento.close();
        JOptionPane.showMessageDialog(null, "Reporte " + factory.obtenerNombreArchivo() + " creado");
    } catch (DocumentException | FileNotFoundException e) {
        System.out.println("Error al generar documento: " + e);
    } catch (IOException | SQLException e) {
        System.out.println("Error al procesar contenido: " + e);
    }
}

// Método original modificado para usar el Factory
public void ReportesClientes() {
    generarReporte(new ReporteClientesFactory());
}
}

```

```

private class ReporteUsuariosFactory implements ReporteFactory {
    @Override public String obtenerNombreArchivo() { return "Reporte_Usuarios.pdf"; }

    @Override
    public Document crearDocumento() throws DocumentException, FileNotFoundException {
        return crearDocumentoBase(obtenerNombreArchivo());
    }

    @Override
    public void agregarContenido(Document documento) throws DocumentException, SQLException, IOException {
        agregarEncabezado(documento, "Reporte de Usuarios", "Listado de usuarios del sistema");

        PdfPTable tabla = new PdfPTable(4);
        agregarCeldaEncabezado(tabla, "ID");
        agregarCeldaEncabezado(tabla, "Usuario");
        agregarCeldaEncabezado(tabla, "Nombre Completo");
        agregarCeldaEncabezado(tabla, "Teléfono");

        Connection cn = Conexion.getConexion();
        PreparedStatement pst = cn.prepareStatement(
            "SELECT idUsuario, usuario, CONCAT(nombre, ' ', apellido), telefono FROM tb_usuario");
        llenarTabla(tabla, pst);

        documento.add(tabla);
    }
}

```

```

private class ReporteProductosFactory implements ReporteFactory {
    @Override public String obtenerNombreArchivo() { return "Reporte_Productos.pdf"; }

    @Override
    public Document crearDocumento() throws DocumentException, FileNotFoundException {
        return crearDocumentoBase(obtenerNombreArchivo());
    }

    @Override
    public void agregarContenido(Document documento) throws DocumentException, SQLException, IOException {
        agregarEncabezado(documento, "Reporte de Productos", "Inventario de productos");

        PdfPTable tabla = new PdfPTable(7);
        float[] widths = {1, 3, 2, 2, 3, 2, 3};
        tabla.setWidths(widths);

        agregarCeldaEncabezado(tabla, "ID");
        agregarCeldaEncabezado(tabla, "Nombre");
        agregarCeldaEncabezado(tabla, "Cantidad");
        agregarCeldaEncabezado(tabla, "Precio");
        agregarCeldaEncabezado(tabla, "Descripción");
        agregarCeldaEncabezado(tabla, "IVA %");
        agregarCeldaEncabezado(tabla, "Categoría");

        Connection cn = Conexion.getConnection();
        PreparedStatement pst = cn.prepareStatement(
            "SELECT p.idProducto, p.nombre, p.cantidad, p.precio, p.descripcion, p.porcentajeIva, c.descripcion " +
            "FROM tb_producto p JOIN tb_categoria c ON p.idCategoria = c.idCategoria");
        llenarTabla(tabla, pst);

        documento.add(tabla);
    }
}

private class ReporteCategoriasFactory implements ReporteFactory {
    @Override public String obtenerNombreArchivo() { return "Reporte_Categorias.pdf"; }

    @Override
    public Document crearDocumento() throws DocumentException, FileNotFoundException {
        return crearDocumentoBase(obtenerNombreArchivo());
    }

    @Override
    public void agregarContenido(Document documento) throws DocumentException, SQLException, IOException {
        agregarEncabezado(documento, "Reporte de Categorías", "Listado de categorías de productos");

        PdfPTable tabla = new PdfPTable(3);
        agregarCeldaEncabezado(tabla, "ID");
        agregarCeldaEncabezado(tabla, "Descripción");
        agregarCeldaEncabezado(tabla, "Estado");

        Connection cn = Conexion.getConnection();
        PreparedStatement pst = cn.prepareStatement(
            "SELECT idCategoria, descripcion, estado FROM tb_categoria");
        llenarTabla(tabla, pst);

        documento.add(tabla);
    }
}

```



```

private class ReporteVentasFactory implements ReporteFactory {
    @Override public String obtenerNombreArchivo() { return "Reporte_Ventas.pdf"; }

    @Override
    public Document crearDocumento() throws DocumentException, FileNotFoundException {
        return crearDocumentoBase(obtenerNombreArchivo());
    }

    @Override
    public void agregarContenido(Document documento) throws DocumentException, SQLException, IOException {
        agregarEncabezado(documento, "Reporte de Ventas", "Historial completo de ventas");

        PdfPTable tabla = new PdfPTable(5);
        float[] widths = {1, 4, 2, 2, 1};
        tabla.setWidths(widths);

        agregarCeldaEncabezado(tabla, "ID");
        agregarCeldaEncabezado(tabla, "Cliente");
        agregarCeldaEncabezado(tabla, "Total");
        agregarCeldaEncabezado(tabla, "Fecha");
        agregarCeldaEncabezado(tabla, "Estado");

        Connection cn = Conexion.getConnection();
        PreparedStatement pst = cn.prepareStatement(
            "SELECT cv.idCabeceraVenta, CONCAT(c.nombre, ' ', c.apellido), " +
            "cv.valorPagar, cv.fechaVenta, cv.estado " +
            "FROM tb_cabecera_venta cv JOIN tb_cliente c ON cv.idCliente = c.idCliente");
        llenarTabla(tabla, pst);

        documento.add(tabla);
    }
}

private class ReporteVentasDiaFactory implements ReporteFactory {
    @Override public String obtenerNombreArchivo() { return "Reporte_Ventas_Dia.pdf"; }

    @Override
    public Document crearDocumento() throws DocumentException, FileNotFoundException {
        return crearDocumentoBase(obtenerNombreArchivo());
    }

    @Override
    public void agregarContenido(Document documento) throws DocumentException, SQLException, IOException {
        agregarEncabezado(documento, "Reporte de Ventas del Dia", "Ventas realizadas hoy");

        PdfPTable tabla = new PdfPTable(5);
        float[] widths = {1, 4, 2, 2, 1};
        tabla.setWidths(widths);

        agregarCeldaEncabezado(tabla, "ID");
        agregarCeldaEncabezado(tabla, "Cliente");
        agregarCeldaEncabezado(tabla, "Total");
        agregarCeldaEncabezado(tabla, "Fecha");
        agregarCeldaEncabezado(tabla, "Estado");

        Connection cn = Conexion.getConnection();
        PreparedStatement pst = cn.prepareStatement(
            "SELECT cv.idCabeceraVenta, CONCAT(c.nombre, ' ', c.apellido), " +
            "cv.valorPagar, cv.fechaVenta, cv.estado " +
            "FROM tb_cabecera_venta cv JOIN tb_cliente c ON cv.idCliente = c.idCliente " +
            "WHERE DATE(cv.fechaVenta) = CURRENT_DATE");

        double totalVentas = llenarTablaConTotal(tabla, pst);

        documento.add(tabla);

        // Agregar total del día
        Paragraph total = new Paragraph("\n\nTotal de ventas del día: " + totalVentas,
            new Font(Font.Family.HELVETICA, 14, Font.BOLD));
        total.setAlignment(Element.ALIGN_RIGHT);
        documento.add(total);
    }
}

```

Reportes (Clase contexto)

Responsabilidades:

1. Coordinar el proceso de generación de reportes
2. Proveer los métodos base para creación de documentos
3. Ofrecer métodos auxiliares compartidos (encabezados, tablas)
4. Gestionar los recursos comunes (conexiones, configuración)
5. Solicitar la creación de mementos cuando sea necesario
6. Decidir cuándo restaurar estados anteriores
7. Gestionar el ciclo de vida de los mementos
8. Proporcionar contexto para las operaciones de guardado/restauración
9. Construir la estructura de componentes
10. Configurar las relaciones entre componentes
11. Iniciar el proceso de generación
12. Proporcionar los recursos compartidos necesarios
13. Gestionar el documento PDF final

```

/* ===== MÉTODOS BASE PARA FACTORY ===== */

private Document crearDocumentoBase(String nombreArchivo) throws DocumentException, FileNotFoundException {
    String ruta = System.getProperty("user.home");
    Document documento = new Document();
    PdfWriter.getInstance(documento, new FileOutputStream(ruta + "/Desktop/" + nombreArchivo));
    return documento;
}

private void agregarEncabezado(Document documento, String titulo, String subtitulo)
    throws DocumentException, IOException {
    Image header = Image.getInstance("src/img/header1.jpg");
    header.scaleToFit(650, 1000);
    header.setAlignment(Element.ALIGN_CENTER);

    Paragraph parrafo = new Paragraph();
    parrafo.setAlignment(Element.ALIGN_CENTER);
    parrafo.add("Reporte creado por \nMiscelanea Calicanto\n\n");
    parrafo.setFont(FontFactory.getFont("Tahoma", 18, Font.BOLD, BaseColor.DARK_GRAY));
    parrafo.add(titulo + "\n\n");
    parrafo.add(subtitulo + "\n\n");

    documento.add(header);
    documento.add(parrafo);
}

private void agregarCeldaEncabezado(PdfPTable tabla, String texto) {
    PdfPCell celda = new PdfPCell(new Phrase(texto));
    celda.setBackgroundColor(BaseColor.LIGHT_GRAY);
    celda.setHorizontalAlignment(Element.ALIGN_CENTER);
    tabla.addCell(celda);
}

private void llenarTabla(PdfPTable tabla, PreparedStatement pst) throws SQLException {
    ResultSet rs = pst.executeQuery();
    while (rs.next()) {
        for (int i = 1; i <= tabla.getNumberOfColumns(); i++) {
            tabla.addCell(rs.getString(i) != null ? rs.getString(i) : "");
        }
    }
}

```

```

private double llenarTablaConTotal(PdfPTable tabla, PreparedStatement pst) throws SQLException {
    double total = 0;
    ResultSet rs = pst.executeQuery();
    while (rs.next()) {
        for (int i = 1; i <= tabla.getNumberOfColumns(); i++) {
            tabla.addCell(rs.getString(i) != null ? rs.getString(i) : "");
        }
        total += rs.getDouble(3); // Sumar la columna de total
    }
    return total;
}

/* ===== MÉTODOS PÚBLICOS PARA GENERAR REPORTES ===== */

public void ReportesClientes() {
    generarReporteConMemento(new ReporteClientesFactory());
}

public void ReportesUsuarios() {
    generarReporteConMemento(new ReporteUsuariosFactory());
}

public void ReportesProductos() {
    generarReporteConMemento(new ReporteProductosFactory());
}

public void ReportesCategorias() {
    generarReporteConMemento(new ReporteCategoriasFactory());
}

public void ReportesVentas() {
    generarReporteConMemento(new ReporteVentasFactory());
}

public void ReportesVentasDia() {
    generarReporteConMemento(new ReporteVentasDiaFactory());
}

/* ===== IMPLEMENTACIÓN MEMENTO ===== */

public void generarReporteConMemento(ReporteFactory factory) {
    ReporteOriginator originator = new ReporteOriginator();
    originator.setEstado("Generando", factory.getClass().getSimpleName(),
        System.getProperty("user.home") + "/Desktop/" + factory.obtenerNombreArchivo());

    ReporteMemento memento = originator.guardarEstado();

    try {
        Document documento = factory.crearDocumento();
        documento.open();
        factory.agregarContenido(documento);
        documento.close();

        originator.setEstado("Completado", factory.getClass().getSimpleName(),
            System.getProperty("user.home") + "/Desktop/" + factory.obtenerNombreArchivo());

        JOptionPane.showMessageDialog(null, "Reporte generado exitosamente: " + factory.obtenerNombreArchivo());
    } catch (Exception e) {
        originator.restaurarEstado(memento);
        JOptionPane.showMessageDialog(null,
            "Error al generar reporte. Estado restaurado.",
            "Error", JOptionPane.ERROR_MESSAGE);
        System.out.println("Error al generar reporte: " + e);
    }
}

```

```

/* ===== IMPLEMENTACIÓN COMPOSITE ===== */

public class EncabezadoComponente implements ComponenteReporte {
    private final String titulo;
    private final String subtítulo;

    public EncabezadoComponente(String titulo, String subtítulo) {
        this.titulo = titulo;
        this.subtítulo = subtítulo;
    }

    @Override
    public void agregar(Document documento) throws DocumentException, IOException {
        agregarEncabezado(documento, titulo, subtítulo);
    }
}

public class TablaComponente implements ComponenteReporte {
    private final String consulta;
    private final String[] encabezados;

    public TablaComponente(String consulta, String[] encabezados) {
        this.consulta = consulta;
        this.encabezados = encabezados;
    }

    @Override
    public void agregar(Document documento) throws DocumentException, SQLException {
        PdfPTable tabla = new PdfPTable(encabezados.length);

        for (String encabezado : encabezados) {
            agregarCeldaEncabezado(tabla, encabezado);
        }

        Connection cn = Conexion.getConnection();
        PreparedStatement pst = cn.prepareStatement(consulta);
        llenarTabla(tabla, pst);

        documento.add(tabla);
    }
}

```

```

public void generarReporteCompleto() {
    try {
        String ruta = System.getProperty("user.home");
        Document documento = new Document();
        PdfWriter.getInstance(documento, new FileOutputStream(ruta + "/Desktop/Reporte_Completo.pdf"));

        documento.open();

        // Crear reporte compuesto
        ReporteCompuesto reporte = new ReporteCompuesto();

        // Agregar componentes
        reporte.agregarComponente(new EncabezadoComponente(
            "Reporte Completo del Sistema",
            "Resumen general de todas las áreas"));

        reporte.agregarComponente(new TablaComponente(
            "SELECT COUNT(*) as total, 'Clientes' as tipo FROM tb_cliente " +
            "UNION SELECT COUNT(*), 'Productos' FROM tb_producto " +
            "UNION SELECT COUNT(*), 'Usuarios' FROM tb_usuario " +
            "UNION SELECT COUNT(*), 'Ventas Hoy' FROM tb_cabecera_venta WHERE DATE(fechaVenta) = CURRENT_DATE",
            new String[]{"Cantidad", "Tipo"}));

        reporte.agregarComponente(new TablaComponente(
            "SELECT c.descripcion as Categoria, COUNT(p.idProducto) as Productos, " +
            "SUM(p.cantidad) as Existencia FROM tb_categoria c " +
            "LEFT JOIN tb_producto p ON c.idCategoria = p.idCategoria " +
            "GROUP BY c.descripcion",
            new String[]{"Categoría", "Productos", "Existencia"}));

        reporte.agregarComponente(new TablaComponente(
            "SELECT DATE(fechaVenta) as Fecha, COUNT(*) as Ventas, " +
            "SUM(valorPagar) as Total FROM tb_cabecera_venta " +
            "GROUP BY DATE(fechaVenta) ORDER BY Fecha DESC LIMIT 7",
            new String[]{"Fecha", "Ventas", "Total"}));

        // Generar el reporte compuesto
        reporte.agregar(documento);

        documento.close();

        JOptionPane.showMessageDialog(null, "Reporte completo generado exitosamente");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null,
            "Error al generar reporte completo",
            "Error", JOptionPane.ERROR_MESSAGE);
        System.out.println("Error al generar reporte completo: " + e);
    }
}
}

```

ReporteMemento

Responsabilidades:

1. Almacenar el estado interno del originador de manera inmutable
2. Preservar la información crítica (estado, tipo, ruta)
3. Proteger la integridad de los datos almacenados
4. Proveer acceso controlado a los datos almacenados

```
*/  
public class ReporteMemento {  
    private final String estado;  
    private final String tipoReporte;  
    private final String rutaArchivo;  
  
    public ReporteMemento(String estado, String tipoReporte, String rutaArchivo) {  
        this.estado = estado;  
        this.tipoReporte = tipoReporte;  
        this.rutaArchivo = rutaArchivo;  
    }  
  
    // Getters  
    public String getEstado() { return estado; }  
    public String getTipoReporte() { return tipoReporte; }  
    public String getRutaArchivo() { return rutaArchivo; }  
}
```

ReporteOriginator

Responsabilidades:

1. Crear snapshots del estado actual (mementos)
2. Restaurar su estado a partir de un memento
3. Gestionar los cambios de estado internos
4. Mantener la lógica de negocio asociada a los estados
5. Coordinar con el caretaker para guardar/restaurar

```
public class ReporteOriginator {  
    private String estado;  
    private String tipoReporte;  
    private String rutaArchivo;  
  
    public void setEstado(String estado, String tipoReporte, String rutaArchivo) {  
        this.estado = estado;  
        this.tipoReporte = tipoReporte;  
        this.rutaArchivo = rutaArchivo;  
    }  
  
    public ReporteMemento guardarEstado() {  
        return new ReporteMemento(estado, tipoReporte, rutaArchivo);  
    }  
  
    public void restaurarEstado(ReporteMemento memento) {  
        this.estado = memento.getEstado();  
        this.tipoReporte = memento.getTipoReporte();  
        this.rutaArchivo = memento.getRutaArchivo();  
    }  
}
```


ComponenteReporte (Interfaz)

Responsabilidades:

1. Definir la interfaz común para todos los componentes
2. Establecer el método básico de agregación al documento
3. Permitir el tratamiento uniforme de hojas y compuestos
4. Facilitar la extensión con nuevos tipos de componentes

```
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import java.sql.SQLException;
import java.io.IOException;

public interface ComponenteReporte {
    void agregar(Document documento) throws DocumentException, SQLException, IOException;
}
```

ReporteCompuesto (Composite)

Responsabilidades:

1. Gestionar la colección de componentes hijos
2. Delegar operaciones a los componentes hijos
3. Coordinar el orden de renderizado
4. Proporcionar interfaz para agregar/remove componentes
5. Mantener la estructura jerárquica del reporte

```
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import java.sql.SQLException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import modelo.ComponenteReporte;

public class ReporteCompuesto implements ComponenteReporte {
    private List<ComponenteReporte> componentes = new ArrayList<>();

    public void agregarComponente(ComponenteReporte componente) {
        componentes.add(componente);
    }

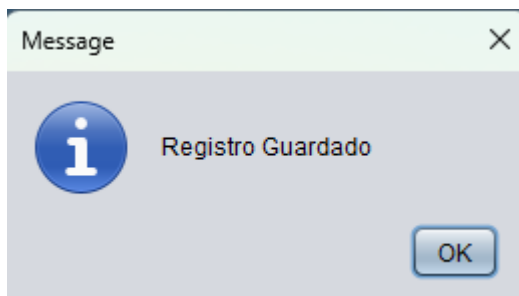
    @Override
    public void agregar(Document documento) throws DocumentException, SQLException, IOException {
        for (ComponenteReporte componente : componentes) {
            componente.agregar(documento);
        }
    }
}
```

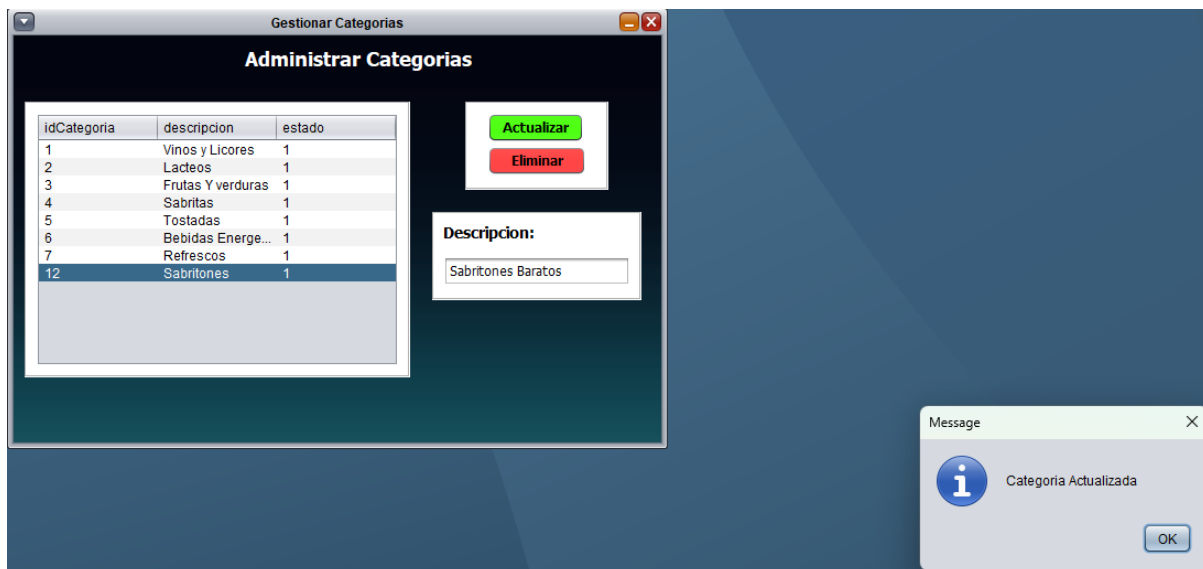
Para todos los patrones:

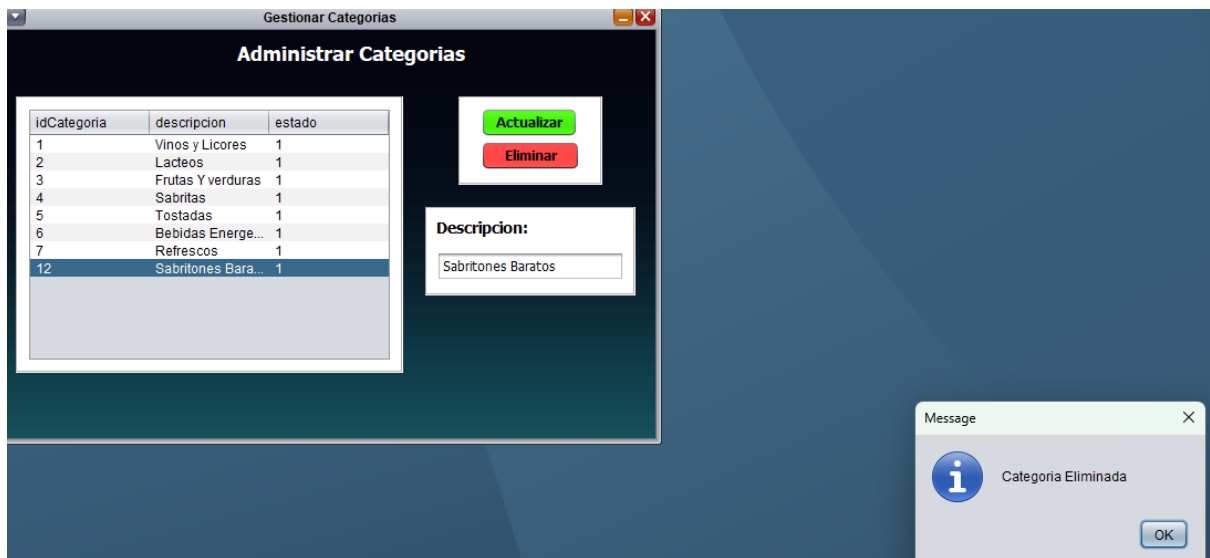
1. Mantener el principio de responsabilidad única
2. Preservar la encapsulación
3. Garantizar la consistencia en el manejo de errores
4. Proporcionar logs adecuados para auditoría
5. Mantener interfaces claras y documentadas

Resultados

Proxy-singleton-flyweight







Abstract Factory-Prototype-Build

Nuevo Producto

Nombre:

Cantidad:

Precio:


Descripcion:

IVA:

Categorias:

Guardar

Message

 Registro Guardado

OK

40	Chetos	20	37.0	hola	0.0	Sabritas	1
7	Tostadas Mi...	5	38.0	Tostadas Mi...	0.0	Tostadas	1
25	Tacos	50	10.0	Tacos de ca...	1.2	Tostadas	1
28	Tostitos	26	10.0	Milpa	1.2	Tostadas	1
32	perros	12	8.0	Mascotas	0.0	Tostadas	1
10	Volt	0	20.0	Volt	0.0	Bebidas En...	1
11	Amper	10	20.0	Amper	0.0	Bebidas En...	1
12	Monster En...	157	32.0	Monster En...	0.0	Bebidas En...	1

Nombre:

Cantidad:

Precio:

Descripcion:

IVA:

Categoria:

8	Coca Cola ...	22	38.0	CocaCpla R...	0.0	Refrescos	1
9	CocaCola 6...	27	22.0	CocaCola 6...	0.0	Refrescos	1
13	BigCola	124	23.0	imitacion C...	0.0	Refrescos	1
24	dwjdj	32	12.0	dawdwa	0.0	Refrescos	1
39	vaca	24	2.0	awdaw	0.24	Refrescos	1

Nombre:

Cantidad:

Precio:

Descripcion:

IVA:

Categoria:

24	PruebaPatron	32	80.0	esto es pru...	11.2	Refrescos	1
39	vaca	24	2.0	awdaw	0.24	Refrescos	1

Adapter-Facade-Bridge

Facturación

Ciente: Teresa Hernandez

Buscar

Seleccione producto:

Cantidad:

Añadir Productos

N	Nombre	Cantidad	P. Unitario	SubTotal	Descuento	Iva	Total Pagar	Accion
1	Cerveza	20	25.0	500.0	0.0	0.0	500.0	Eliminar
2	Tostadas ...	5	38.0	190.0	0.0	0.0	190.0	Eliminar

PDF

Registrar Venta

Subtotal: 690.0

Descuento: 0.0

Iva: 0.0

Total a pagar: 690.0

Efectivo: 700

Cambio: 10.0

Calcular Cambio

Message

 ¡Venta Registrada!

OK

FACTURA

MISCELÁNEA CALICANTO

Av. Principal #123
Ciudad, País

Fecha: 2025/04/15

Tel: 555-123456
RUC: 1234567890123

CLIENTE:

Nombre: Teresa Hernandez
Cédula: 9581732455
Teléfono: 9581732455
Dirección: Privada Texcoco

Cant.	Descripción	P. Unit.	IVA	Total
20	Cerveza	\$25.0	12%	\$500.0
5	Tostadas Mi Reyna	\$38.0	12%	\$190.0

TOTAL A PAGAR: \$690.0

¡Gracias por su compra!

Facturación

Ciente:

Hector Vera

Buscar

Selección producto:

Cantidad:

Añadir Productos

N	Nombre	Cantidad	P. Unitario	SubTotal	Descuento	Iva	Total Pagar	Acción
1	Cerveza	15	25.0	375.0	0.0	0.0	375.0	Eliminar
2	Sabritas Cl...	10	18.0	180.0	0.0	0.0	180.0	Eliminar
3	Monster En...	5	32.0	160.0	0.0	0.0	160.0	Eliminar

HTML

Registrar Venta

Subtotal:

715.0

Descuento:

0.0

Iva:

0.0

Total a pagar:

715.0

Efectivo:

1000

Cambio:

285.0

Calcular Cambio

Message

i

¡Venta Registrada!

OK

FACTURA
 MISCELÁNEA CALICANTO
 Av. Principal #123 - Ciudad. País
 Tel: 555-123456 - RUC: 1234567890123

Fecha: 2025/04/15

DATOS DEL CLIENTE

Nombre: Hector Vera
 Cédula: 9512348721
 Teléfono: 9512348721
 Dirección: Tecnológico

Cant.	Descripción	P. Unit.	IVA	Total
15	Cerveza	\$25.0	12%	\$375.0
10	Sabritas Clasicas	\$18.0	12%	\$180.0
5	Monster Energy	\$32.0	12%	\$160.0

TOTAL A PAGAR: \$715.0

¡Gracias por su compra!

Mediator

Nuevo Cliente

Nombre:

Apellido:


Telefono:

Teléfono:

Dirección:

Guardar

Message

 Registro Guardado

OK

Gestionar Clientes

Administrar Clientes

N°	nombre	apellido	telefono	telefono	direccion	estado
1	Uri	Hernandez	9514401725	9514401725	Av Montoya	1
2	Teresa	Hernandez	9581732455	9581732455	Privada Texcoco	1
3	Alma Rosa	Mendez	9512156554	9512156554	Privada Texcoco	1
5	Alberto	Mendez	9511111212	9511111212	Texcoco 6	1
6	abdiel	Diaz	9514565656	9514565656	Av montealban	1
7	Hector	Vera	9512348721	9512348721	Tecnologico	1

Actualizar

Eliminar

Nombre: **Apellido:** **Teléfono:**

Teléfono: **Dirección:**

Observer

Gestionar Usuarios

Administrar Usuarios

N°	nombre	apellido	usuario	password	telefono	estado
1	Ruudvan	Ordaz	Orda1	123	9514401726	1
2	Uri	Hernandez	Uri1	123	9513436751	1
3	Teresa	Hernandez	Herna1	123	9581732455	1
4	Uriel	Hernandez	637279	7279	9513436751	1

Actualizar

Eliminar

Nombre:

Uri

Apellido:

Fernandez

Usuario:

Uri1

Password:

1234

Telefono:

9513436722

Message

i

¡Actualizacion Exitosa!

OK

Gestionar Usuarios

Administrar Usuarios

N°	nombre	apellido	usuario	password	telefono	estado
1	Ruudvan	Ordaz	Orda1	123	9514401726	1
2	Uri	Fernandez	Uri1	1234	9513436722	1
3	Teresa	Hernandez	Herna1	123	9581732455	1
4	Uriel	Hernandez	637279	7279	9513436751	1

Actualizar

Eliminar

Nombre:

Apellido:

Usuario:

Password:

Telefono:

Nuevo Usuario

Nombre:

Apellido:


Usuario:

Password: ☒

Telefono:

Guardar

Message

 ¡Usuario Registrado!

OK

Gestionar Usuarios

Administrar Usuarios

N°	nombre	apellido	usuario	password	telefono	estado
1	Ruudvan	Ordaz	Orda1	123	9514401726	1
2	Uri	Fernandez	Uri1	1234	9513436722	1
3	Teresa	Hernandez	Herna1	123	9581732455	1
4	Uriel	Hernandez	637279	7279	9513436751	1
5	Hector	Vera	Hec	123	9512923456	1

Actualizar
Eliminar

Nombre: **Apellido:** **Usuario:**

Password: **Telefono:**

Responsability

Actualizar Stock de los Productos

Actualizar Stock de Productos


Producto: Cerveza

Stock Actual: 80

Stock Nuevo: 120

Actualizar

Message

 Stock Actualizado

OK

Actualizar Stock de los Productos

Actualizar Stock de Productos

Producto: Cerveza

Stock Actual: 200

Stock Nuevo:

Actualizar

FactoryMethod-Memento-Composite



Reporte_Categorías	15/04/2025 07:20 p. m.	Brave HTML Docu...	31 KB
Reporte_Clientes	15/04/2025 07:18 p. m.	Brave HTML Docu...	31 KB
Reporte_Productos	15/04/2025 07:20 p. m.	Brave HTML Docu...	34 KB
Reporte_Ventas	15/04/2025 07:20 p. m.	Brave HTML Docu...	33 KB
Reporte_Ventas_Día	15/04/2025 07:20 p. m.	Brave HTML Docu...	31 KB

Reporte creado por
Miscelanea Calicanto

Reporte de Clientes

Listado completo de clientes

ID	Nombre	Cédula	Teléfono	Dirección
1	Uri Hernandez	9514401725	9514401725	Av Montoya
2	Teresa Hernandez	9581732455	9581732455	Privada Texcoco
3	Alma Rosa Mendez	9512156554	9512156554	Privada Texcoco
6	abdiel Tomas Diaz	9514565656	9514565656	Av Tecnologico
7	Hector Vera	9512348721	9512348721	Tecnologico

Reporte creado por
Miscelanea Calicanto

Reporte de Categorías

Listado de categorías de productos

ID	Descripción	Estado
1	Vinos y Licores	1
2	Lacteos	1
3	Frutas Y verduras	1
4	Sabritas	1
5	Tostadas	1
6	Bebidas Energetizantes	1
7	Refrescos	1

Reporte de Productos

Inventario de productos

ID	Nombre	Cantidad	Precio	Descripción	IVA %	Categoría
1	Cerveza	145	25.00	Corona	0	Vinos y Licores
2	Leche Nutri Deslactosa	4	28.00	Leche Deslactosada	0	Lacteos
3	Sabritas Clasicas	4	18.00		0	Sabritas
4	Paquetaxo Azul Ch	0	18.00		0	Sabritas
5	Cheetos Torciditos Ch	0	16.00		0	Sabritas
6	Yoghurt Oikos	0	19.00		0	Vinos y Licores
7	Tostadas Mi Reyna	0	38.00	Tostadas Mi Reyna	0	Tostadas
8	Coca Cola Ret 2.5L	22	38.00	CocaCpla Retornable 2.5L	0	Refrescos
9	CocaCola 600ml	27	22.00	CocaCola 600ml	0	Refrescos
10	Volt	0	20.00	Volt	0	Bebidas Energetizantes
11	Amper	10	20.00	Amper	0	Bebidas Energetizantes
12	Monster Energy	152	32.00	Monster Energy	0	Bebidas Energetizantes
13	BigCola	124	23.00	imitacion Coca	0	Refrescos
14	sabritas	2	56.00		0	Lacteos
15	dwed	34	23.40	adwda	0	Lacteos
17	dawdaw	123	32.00	adwdwa	0	Frutas Y verduras

Reporte creado por
Miscelanea Calicanto

Reporte de Ventas

Historial completo de ventas

ID	Cliente	Total	Fecha	Estado
1	Uri Hernandez	1254.00	2024-10-24	1
2	Uri Hernandez	50.16	2024-10-24	1
3	Uri Hernandez	50.16	2024-10-25	1
4	Uri Hernandez	125.40	2024-10-28	1
5	Uri Hernandez	125.40	2024-10-28	1
6	Teresa Hernandez	390.96	2024-11-02	1
7	Alma Rosa Mendez	101.00	2024-11-02	1
8	Teresa Hernandez	84.00	2024-11-07	1
9	Uri Hernandez	168.00	2024-11-07	1
10	Uri Hernandez	194.00	2024-11-07	1
11	Uri Hernandez	72.00	2024-11-07	1
12	Teresa Hernandez	149.00	2024-12-01	1
14	Alma Rosa Mendez	74.00	2024-12-02	1
15	Teresa Hernandez	98.00	2024-12-02	1
17	Uri Hernandez	76.00	2024-12-02	1
18	Teresa Hernandez	56.00	2024-12-02	1
19	Uri Hernandez	294.00	2024-12-05	1
20	Teresa Hernandez	116.00	2024-12-05	0
21	abdiel Tomas Diaz	402.00	2024-12-05	1
23	Teresa Hernandez	303.00	2024-12-06	1
24	Alma Rosa Mendez	959.00	2024-12-06	1
25	Uri Hernandez	56.00	2025-03-22	1
26	Alma Rosa Mendez	50.00	2025-03-22	1
27	Teresa Hernandez	25.00	2025-03-22	1
28	Teresa Hernandez	25.00	2025-03-22	1
29	Teresa Hernandez	25.00	2025-03-22	1

Reporte creado por
Miscelanea Calicanto

Reporte de Ventas del Día

Ventas realizadas hoy

ID	Cliente	Total	Fecha	Estado
45	Teresa Hernandez	690.00	2025-04-15	1
46	Hector Vera	715.00	2025-04-15	1

Total de ventas del día: 1405.0

Conclusión

La implementación de 15 patrones de diseño en el Sistema de Ventas ha demostrado ser un éxito tanto en aspectos técnicos como conceptuales. Este ejercicio ha permitido comprobar en la práctica cómo los patrones de diseño, cuando se aplican adecuadamente, pueden transformar un sistema convencional en una arquitectura robusta, mantenible y extensible.

Los principales logros de esta implementación incluyen:

1. **Modularidad extremo:** Cada componente del sistema tiene responsabilidades claramente definidas y acotadas, facilitando el mantenimiento y la evolución futura.
2. **Flexibilidad sin precedentes:** La capacidad de adaptar el sistema a nuevos requisitos se ha multiplicado gracias a patrones como Abstract Factory y Strategy.
3. **Reducción de acoplamiento:** Patrones como Mediator y Observer han permitido desacoplar componentes que originalmente tenían dependencias directas.
4. **Eficiencia mejorada:** El uso de Flyweight y Singleton ha optimizado el uso de recursos críticos en el sistema.
5. **Consistencia arquitectónica:** A pesar de la diversidad de patrones implementados, se ha mantenido una coherencia general en el diseño.
6. **Preparación para escalamiento:** La arquitectura resultante está preparada para crecer en funcionalidad sin comprometer su estabilidad.

Esta experiencia ha validado el poder transformador de los patrones de diseño cuando se aplican de manera consciente y sistemática. El sistema resultante no solo cumple con sus requisitos funcionales actuales, sino que está preparado para adaptarse a futuras necesidades con cambios mínimos en su estructura central.