DOCUMENTACIÓN PROYECTO FINAL

PATRONES DE DISEÑO:

Factory Method

El patrón Factory Method se utiliza para crear objetos sin especificar la clase exacta del objeto que se va a crear. En tu proyecto, la clase ProductoFactory utiliza este patrón:

```
public class ProductoFactory {
    public static Producto crearProducto(String id, String nombre, boolean esImportado, double precio, int cantidad) {
        if (esImportado) {
            return new ProductoImportado(id, nombre, precio, cantidad);
        } else {
            return new ProductoLocal(id, nombre, precio, cantidad);
        }
    }
}
```

Ventaja: Facilita la creación de objetos, permitiendo la adición de nuevos tipos de productos sin modificar el código existente.

Data Access Object (DAO)

El patrón DAO abstrae y encapsula todas las operaciones de acceso a datos, proporcionando una interfaz para la base de datos. En tu proyecto, las clases Inventario, Produccion, e Importacion manejan la conexión a la base de datos y las operaciones CRUD (Create, Read, Update, Delete).

```
public void guardarEnBaseDeDatos(String tabla) {
    try (Connection con = DriverManager.getConnection("jdbc:mariadb://localhost:3306/selvindb", "selvindb", "SHAN1985")) {
        for (Producto producto : productos) {
```

```
            String query = "INSERT INTO " + tabla + " (id, nombre, esImportado, precio,
cantidad, modificado) VALUES (?, ?, ?, ?, ?, ?)";
            try (PreparedStatement pst = con.prepareStatement(query)) {
                pst.setString(1, producto.getId());
                pst.setString(2, producto.getNombre());
                pst.setBoolean(3, producto.esImportado());
                pst.setDouble(4, producto.getPrecio());
                pst.setInt(5, producto.getCantidad());
                pst.setBoolean(6, false);
                pst.executeUpdate();
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Ventaja: Separa la lógica de acceso a datos del resto del código, facilitando el mantenimiento y las pruebas.

Model-View-Controller (MVC)

Este patrón divide una aplicación en tres componentes principales: el modelo, la vista y el controlador. En tu proyecto, las clases de interfaz de usuario (OrdenApp, MenuPrincipal, etc.) actúan como las vistas y controladores, mientras que las clases de lógica de negocio (Producto, Inventario, etc.) actúan como modelos.

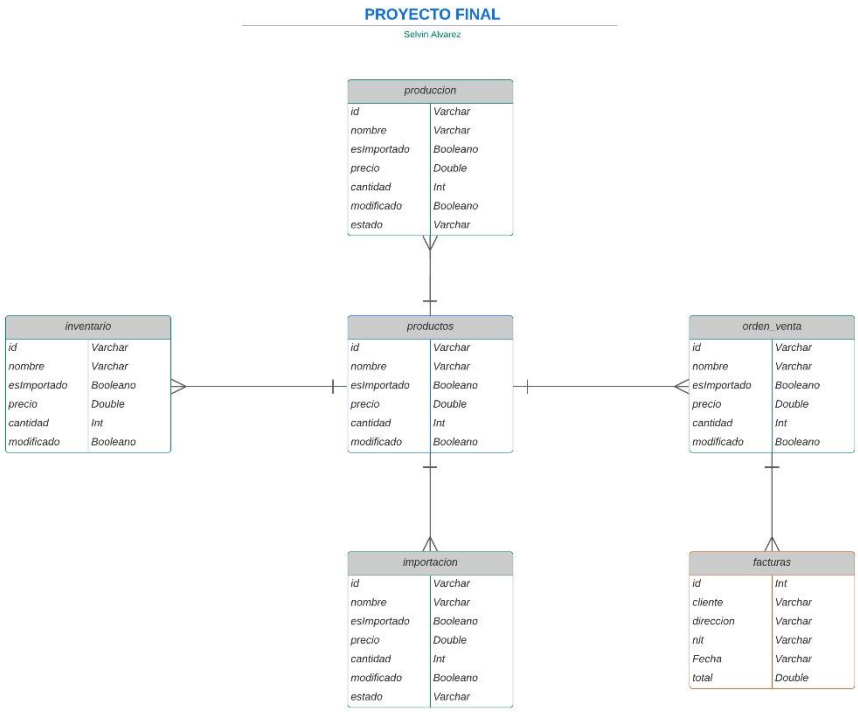
```
public class OrdenApp {
    private JFrame frame;
    private JTextField idField, nombreField, precioField, cantidadField;
    private JCheckBox esImportadoBox;
    private JButton agregarButton, guardarButton;
    private Inventario inventario;
```

```
    public OrdenApp() {
        inventario = new Inventario();
        // Configuración de la interfaz de usuario
        // ...
    }

    private void agregarProducto() {
        String id = idField.getText();
        String nombre = nombreField.getText();
        boolean esImportado = esImportadoBox.isSelected();
        double precio = Double.parseDouble(precioField.getText());
        int cantidad = Integer.parseInt(cantidadField.getText());
        Producto producto = ProductoFactory.crearProducto(id, nombre, esImportado, precio,
cantidad);
        inventario.agregarProducto(producto);
        JOptionPane.showMessageDialog(frame, "¡Producto agregado a la orden!");
        limpiarCampos();
    }
}
```

Ventaja: Separa claramente la presentación de la lógica de negocio, facilitando la gestión y escalabilidad del código.

Diagrama Entidad Relación



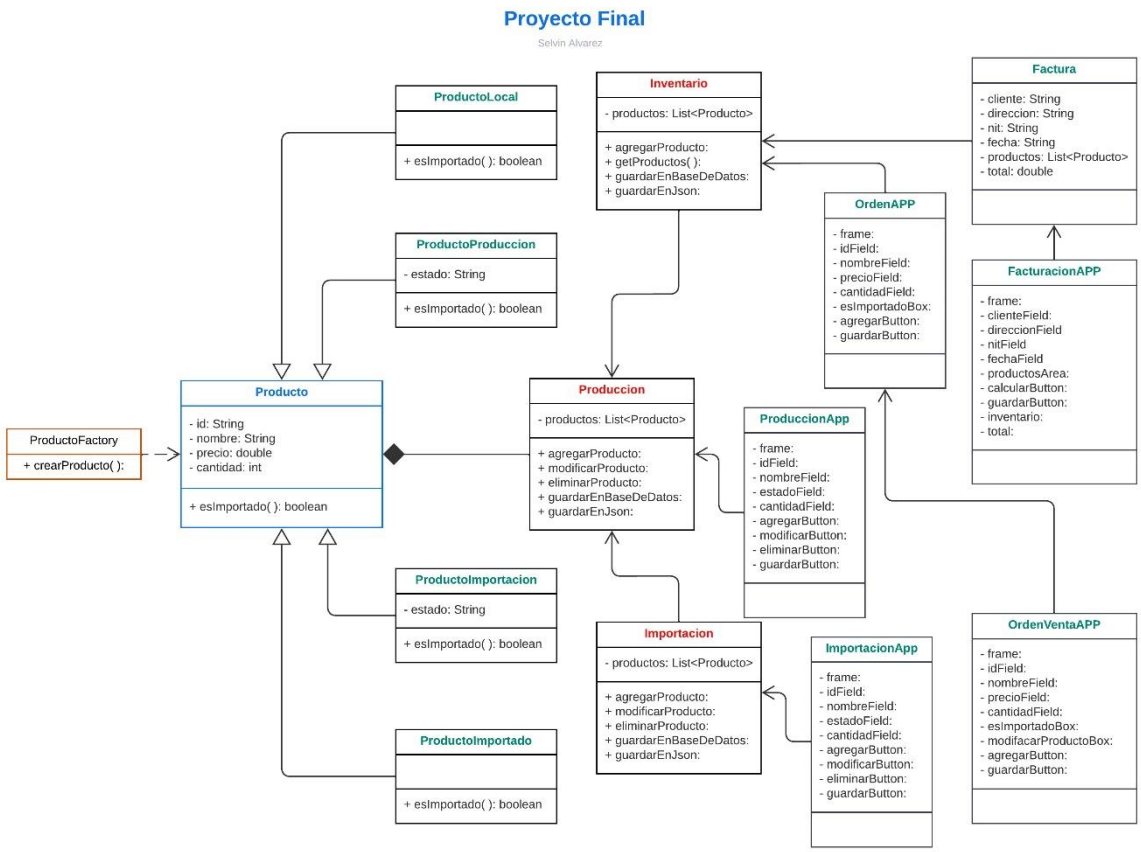
Relaciones

Productos tiene una relación de uno a muchos con Inventario, Orden_Venta, Produccion, e Importacion.

Factura puede estar relacionada con Orden_Venta para detallar los productos vendidos.

En resumen: productos actúa como la tabla central de la cual las otras tablas (inventario, orden_venta, produccion, importacion). La tabla factura está vinculada a las órdenes de venta para detallar las transacciones. Esta estructura asegura que los datos estén bien organizados y las relaciones sean claras.

Diagrama de Clases



Relaciones entre Clases

Producto es la clase base abstracta, de la cual heredan ProductoLocal, ProductoImportado y ProductoProduccion.

ProductoFactory se utiliza para crear instancias de Producto según el tipo y los parámetros proporcionados.

Inventario, Produccion, e Importacion utilizan listas de Producto para gestionar sus respectivos elementos.

Factura contiene una lista de Producto para detallar los productos incluidos en una factura.
Producto (Clase abstracta)

ProductoFactory utiliza Producto como el tipo de retorno para el método de creación.

ProductoLocal (Extiende Producto)

ProductoFactory puede crear una instancia de ProductoLocal si el parámetro esImportado es false.

ProductoImportado (Extiende Producto)

ProductoFactory puede crear una instancia de ProductoImportado si el parámetro esImportado es true.

ProductoProduccion (Extiende Producto)

ProductoImportacion (Extiende Producto)

Factura:
Contiene los detalles de la factura y una lista de productos incluidos en la factura.

Se relaciona con Producto y sus subclases para detallar los productos en la factura.

FacturacionApp:
Maneja la interfaz de usuario para crear y guardar facturas.
Se relaciona con Inventario para obtener los productos y calcular el total de la factura.

Guarda los datos de la factura en la base de datos y en un archivo JSON.

OrdenApp
Relaciona con:

Inventario: Mantiene y gestiona los productos. (Composición)

ProductoFactory: Crea instancias de productos. (Dependencia)
Producto: Los productos que se crean y gestionan. (Dependencia)

OrdenVentaApp
Relaciona con:

Inventario: Similar a OrdenApp, maneja los productos. (Composición)

ProductoFactory: Crea productos para la venta. (Dependencia)

Producto: Los productos que se añaden a la orden de venta. (Dependencia)

ProduccionApp interactúa con Produccion para gestionar productos en producción.

Produccion mantiene una lista de ProductoProduccion, similar a cómo Inventario mantiene una lista de Producto.

ProductoProduccion es una clase específica para la producción, que puede tener atributos adicionales como estado.

ImportacionApp interactúa con Importacion para gestionar productos de importación.

Importacion mantiene una lista de ProductoImportacion, similar a cómo Inventario y Produccion mantienen listas de sus respectivos productos.

ProductoImportacion es una clase específica para productos de importación, similar a ProductoProduccion.