Contexto del caso de ejemplo

Vamos a construir una pequeña aplicación de escritorio para **gestionar productos** de una tienda:

Cada producto tendrá:

- `id` (autonumérico)
- `nombre`
- `precio`
- `stock`

Y la aplicación permitirá:

- Agregar productos
- Editar productos
- Eliminar productos
- Listar productos en una tabla
- Buscar producto por ID

Con esto cubren el **CRUD** completo y los indicadores de la pauta (GUI, MVC, CRUD, buenas prácticas).

1. Preparar SQLite

1.1. Crear el archivo de base de datos

1. Crea una carpeta de trabajo, por ejemplo:
   `C:\proyectos\tienda-sqlite\`

2. Dentro, crea un archivo vacío llamado:
   `productos.db`

Puedes crearlo con:

- Clic derecho → Nuevo → Documento de texto → renombrar a `productos.db` (quitando `.txt`).

1.2. Crear la tabla `producto`

Abre la BD con alguna herramienta (SQLiteStudio, DBeaver, etc.) o desde otra app, y ejecuta:

```
CREATE TABLE producto (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nombre TEXT NOT NULL,
    precio REAL NOT NULL,
    stock INTEGER NOT NULL
);

INSERT INTO producto (nombre, precio, stock) VALUES
('Teclado mecánico', 29990, 10),
('Mouse gamer', 19990, 20),
('Monitor 24 pulgadas', 129990, 5);
```

Esto deja la base lista con algunos datos iniciales.

2. Crear el proyecto en NetBeans

1. Abre **NetBeans**.
2. Ve a **File** → **New Project**.
3. Elige:
    - Category: *Java with Ant* o *Java with Maven* (lo que usen en el ramo).
    - Project: *Java Application*.
4. Nombre del proyecto: `TiendaProductosMVC`.
5. Desmarca “Create Main Class” (para crearla tú después).

3. Agregar el driver JDBC de SQLite

1. Descarga el driver desde: `sqlite-jdbc` (Xerial).
2. En NetBeans, clic derecho sobre el proyecto → **Properties** → **Libraries**.
3. Botón **Add JAR/Folder** → selecciona el `sqlite-jdbc-xxx.jar`.
4. Aceptar.

*Sin este paso, la conexión a SQLite no funcionará.

4. Crear paquetes MVC

En **Source Packages**, crea estos paquetes:

- `cl.duoc.tienda.model`
- `cl.duoc.tienda.dao`
- `cl.duoc.tienda.controller`
- `cl.duoc.tienda.view`
- `cl.duoc.tienda.util`

5. Clase de conexión a SQLite

En `cl.duoc.tienda.util`, crea la clase `Conexion.java`:

```
package cl.duoc.tienda.util;

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

public class Conexion {

    // Ruta del archivo productos.db (ajusta la ruta a la ubicación
real)
    private static final String URL =
"jdbc:sqlite:C:/proyectos/tienda-sqlite/productos.db";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL);
    }
}
```

*Si el archivo `.db` está en otra ruta, deben actualizar el `URL`.

6. Modelo: clase `Producto`

En `cl.duoc.tienda.model`, crea `Producto.java`:

```
package cl.duoc.tienda.model;

public class Producto {
    private int id;
    private String nombre;
    private double precio;
    private int stock;

    public Producto() {
    }

    public Producto(int id, String nombre, double precio, int stock)
{
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
        this.stock = stock;
    }

    // Getters y setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public double getPrecio() {
```

```
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }

    public int getStock() {
        return stock;
    }

    public void setStock(int stock) {
        this.stock = stock;
    }
}
```

7. DAO: clase `ProductoDAO` (CRUD completo)

En `cl.duoc.tienda.dao`, crea `ProductoDAO.java`:

```
package cl.duoc.tienda.dao;

import cl.duoc.tienda.model.Producto;
import cl.duoc.tienda.util.Conexion;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ProductoDAO {

    public void insertar(Producto producto) throws SQLException {
        String sql = "INSERT INTO producto (nombre, precio, stock)
VALUES (?, ?, ?)";
        try (Connection cn = Conexion.getConnection();
             PreparedStatement ps = cn.prepareStatement(sql)) {

            ps.setString(1, producto.getNombre());
            ps.setDouble(2, producto.getPrecio());
            ps.setInt(3, producto.getStock());
            ps.executeUpdate();
        }
    }

    public void modificar(Producto producto) throws SQLException {
        String sql = "UPDATE producto SET nombre = ?, precio = ?,
stock = ? WHERE id = ?";
        try (Connection cn = Conexion.getConnection();
             PreparedStatement ps = cn.prepareStatement(sql)) {

            ps.setString(1, producto.getNombre());
            ps.setDouble(2, producto.getPrecio());
            ps.setInt(3, producto.getStock());
            ps.setInt(4, producto.getId());
            ps.executeUpdate();
        }
    }
```

```
public void eliminar(int id) throws SQLException {
    String sql = "DELETE FROM producto WHERE id = ?";
    try (Connection cn = Conexion.getConnection();
         PreparedStatement ps = cn.prepareStatement(sql)) {

        ps.setInt(1, id);
        ps.executeUpdate();
    }
}

public Producto buscarPorId(int id) throws SQLException {
    String sql = "SELECT * FROM producto WHERE id = ?";
    try (Connection cn = Conexion.getConnection();
         PreparedStatement ps = cn.prepareStatement(sql)) {

        ps.setInt(1, id);
        ResultSet rs = ps.executeQuery();
        if (rs.next()) {
            return new Producto(
                    rs.getInt("id"),
                    rs.getString("nombre"),
                    rs.getDouble("precio"),
                    rs.getInt("stock")
            );
        }
        return null;
    }
}

public List<Producto> listar() throws SQLException {
    String sql = "SELECT * FROM producto";
    List<Producto> lista = new ArrayList<>();

    try (Connection cn = Conexion.getConnection();
         PreparedStatement ps = cn.prepareStatement(sql);
         ResultSet rs = ps.executeQuery()) {

        while (rs.next()) {
            Producto p = new Producto(
                    rs.getInt("id"),
```

```
                        rs.getString("nombre"),
                        rs.getDouble("precio"),
                        rs.getInt("stock")
                );
                lista.add(p);
            }
        }
        return lista;
    }
}
```

8. Controlador: `ProductoController`

En `cl.duoc.tienda.controller`, crea `ProductoController.java`:

```
package cl.duoc.tienda.controller;

import cl.duoc.tienda.dao.ProductoDAO;
import cl.duoc.tienda.model.Producto;

import java.sql.SQLException;
import java.util.List;

public class ProductoController {

    private ProductoDAO productoDAO;

    public ProductoController() {
        this.productoDAO = new ProductoDAO();
    }

    public void guardarProducto(String nombre, double precio, int
stock) throws SQLException {
        Producto p = new Producto();
        p.setNombre(nombre);
        p.setPrecio(precio);
        p.setStock(stock);
        productoDAO.insertar(p);
    }

    public void actualizarProducto(int id, String nombre, double
precio, int stock) throws SQLException {
        Producto p = new Producto(id, nombre, precio, stock);
        productoDAO.modificar(p);
    }

    public void eliminarProducto(int id) throws SQLException {
        productoDAO.eliminar(id);
    }

    public Producto buscarProducto(int id) throws SQLException {
        return productoDAO.buscarPorId(id);
```

```
    }

    public List<Producto> listarProductos() throws SQLException {
        return productoDAO.listar();
    }
}
```

*El controlador será el “puente” entre la **vista (Swing)** y la **lógica/DAO**.

9. Vista: formulario Swing `FrmProductos`

En `cl.duoc.tienda.view`, crea un **JFrame Form** llamado `FrmProductos`.

En el diseñador, agrega:

- **JLabel + JTextField** para:
    - `txtId`
    - `txtNombre`
    - `txtPrecio`
    - `txtStock`
- **JButtons**:
    - `btnNuevo`
    - `btnGuardar`
    - `btnActualizar`
    - `btnEliminar`
    - `btnBuscar`
    - `btnListar`
- **JTable** dentro de un `JScrollPane`:
    - `tblProductos`

Luego, en el código de `FrmProductos`, conecta con el controlador:

```
package cl.duoc.tienda.view;

import cl.duoc.tienda.controller.ProductoController;
import cl.duoc.tienda.model.Producto;
import java.sql.SQLException;
import java.util.List;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

public class FrmProductos extends javax.swing.JFrame {

    private ProductoController controller;

    public FrmProductos() {
        initComponents();
        controller = new ProductoController();
        cargarTabla();
    }

    private void cargarTabla() {
        try {
            List<Producto> productos = controller.listarProductos();
            DefaultTableModel modelo = (DefaultTableModel)
tblProductos.getModel();
            modelo.setRowCount(0); // limpia la tabla

            for (Producto p : productos) {
                Object[] fila = {
                    p.getId(),
                    p.getNombre(),
                    p.getPrecio(),
                    p.getStock()
                };
                modelo.addRow(fila);
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Error al listar
productos: " + ex.getMessage());
        }
    }
```

```
    // Ejemplo de acción del botón Guardar
    private void
btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {
        String nombre = txtNombre.getText();
        double precio = Double.parseDouble(txtPrecio.getText());
        int stock = Integer.parseInt(txtStock.getText());

        try {
            controller.guardarProducto(nombre, precio, stock);
            JOptionPane.showMessageDialog(this, "Producto guardado
correctamente");
            cargarTabla();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Error al guardar: "
+ ex.getMessage());
        }
    }

    // Ejemplo de botón Buscar por ID
    private void btnBuscarActionPerformed(java.awt.event.ActionEvent
evt) {
        int id = Integer.parseInt(txtId.getText());
        try {
            Producto p = controller.buscarProducto(id);
            if (p != null) {
                txtNombre.setText(p.getNombre());
                txtPrecio.setText(String.valueOf(p.getPrecio()));
                txtStock.setText(String.valueOf(p.getStock()));
            } else {
                JOptionPane.showMessageDialog(this, "Producto no
encontrado");
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Error al buscar: "
+ ex.getMessage());
        }
    }

    // Implementar de forma similar btnActualizar y btnEliminar...
```

```
}
```

> En la tabla, asegúrate de configurar el **modelo** con columnas: `ID`, `Nombre`, `Precio`, `Stock`.

10. Clase `Main` para iniciar la aplicación

En algún paquete (por ejemplo `cl.duoc.tienda`), crea `Main.java`:

```
package cl.duoc.tienda;

import cl.duoc.tienda.view.FrmProductos;

public class Main {

    public static void main(String[] args) {
        java.awt.EventQueue.invokeLater(() -> {
            new FrmProductos().setVisible(true);
        });
    }
}
```