# HITO 1 DEL 3° TRIMESTRE DE PROGRAMACIÓN

Óscar Durán Sánchez

Fecha de entrega: 30/04/2024
CampusFP

# Índice

## Introducción:

En este proyecto, mi tarea es desarrollar una aplicación de gestión de inventario para un negocio dedicado a la venta de productos. Esta aplicación será crucial para asegurar el funcionamiento eficiente del negocio, permitiendo a los usuarios realizar diversas acciones como agregar, modificar y eliminar productos, además de generar informes detallados sobre el inventario actual.

## Funcionalidades principales:

1. **Registro de productos:** Implementaré una función que me permita agregar nuevos productos al inventario, proporcionando detalles como nombre, descripción, cantidad disponible, precio, entre otros.
2. **Modificación de productos:** Desarrollaré la capacidad de actualizar la información de los productos existentes, como cambiar el precio, la cantidad disponible, la descripción, etc.
3. **Eliminación de productos:** Implementaré una función para eliminar productos del inventario que ya no están disponibles o no son necesarios.

## Tecnologías utilizadas:

- **Interfaz de usuario:** Utilizaré el framework Swing en J2SE para crear una interfaz gráfica intuitiva y fácil de usar.
- **Conexión a la base de datos:** Emplearé JDBC en J2EE para establecer una conexión segura y estable con una base de datos MySQL, permitiendo el almacenamiento y la recuperación eficientes de los datos del inventario.

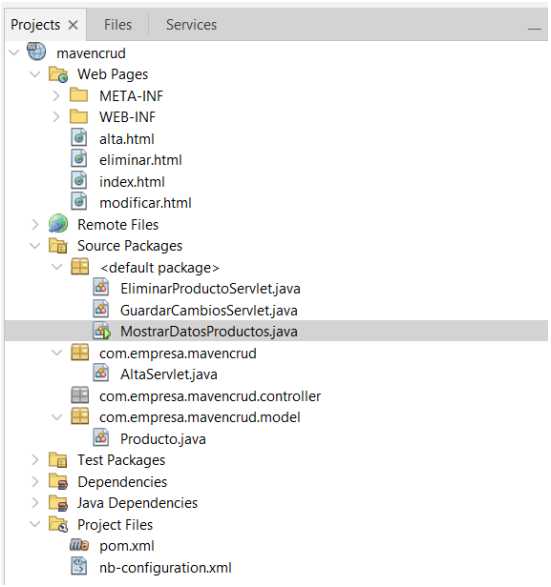
## Objetivo del proyecto:

Mi objetivo principal es desarrollar una aplicación de gestión de inventario que cumpla con los más altos estándares de calidad y funcionalidad. Esta aplicación será esencial para el negocio de venta de productos, ya que permitirá a los usuarios realizar acciones fundamentales, generar informes detallados y mantener un control preciso de todo el inventario.

## 1. Comprensión de los fundamentos de programación en Java

1. **Clases y Objetos en Java:** En Java, todo se implementa a través de clases y objetos. En este proyecto, crearás clases para representar los productos, la interfaz de usuario y la lógica de negocio relacionada con la gestión del inventario.
2. **Estructuras de Control:** Java ofrece diversas estructuras de control como if-else, bucles for, while, y switch-case. Estas estructuras son esenciales para implementar la lógica de negocio de la aplicación, como la validación de entrada de usuario y la iteración sobre los datos del inventario.
3. **Manejo de Excepciones:** El manejo adecuado de excepciones es crucial para garantizar la robustez y la confiabilidad de la aplicación. En este proyecto, deberás manejar excepciones al conectarte a la base de datos, ejecutar consultas SQL y manejar errores en la interfaz de usuario.
4. **Colecciones en Java:** Las colecciones como ArrayList, HashMap, etc., son muy útiles para almacenar y manipular datos en Java. Puedes utilizar estas estructuras de datos para almacenar la información del inventario y facilitar su manipulación dentro de la aplicación.
5. **Programación Orientada a Objetos (POO):** Java es un lenguaje orientado a objetos, lo que significa que se basa en conceptos como encapsulación, herencia y polimorfismo. En este proyecto, puedes aplicar estos conceptos para crear una arquitectura modular y extensible para tu aplicación.
6. **Interfaz de Usuario con Swing:** Swing es un conjunto de bibliotecas en Java para crear interfaces gráficas de usuario (GUI). Deberás familiarizarte con los componentes de Swing, como JFrame, JPanel, JButton, etc., para diseñar la interfaz de usuario de tu aplicación de gestión de inventario de manera efectiva.
7. **Conexión a la Base de Datos con JDBC:** JDBC (Java Database Connectivity) proporciona una API estándar para conectar y ejecutar consultas en bases de datos desde Java. En este proyecto, utilizarás JDBC para establecer una conexión con una base de datos MySQL y ejecutar consultas para manipular los datos del inventario.

## Organización del proyecto:




## 2. Uso de J2SE con Swing (Archivo: MostrarDatosProductos.java)

1. **Clases y Objetos en Java:**
    - En el código, la clase **MostrarDatosProductos** representa la ventana de la aplicación para mostrar los datos de los productos. Se instancia un objeto de esta clase en el método **main** para crear y mostrar la ventana.

```
public class MostrarDatosProductos extends JFrame {
```

2. **Estructuras de Control:**
    - No hay estructuras de control explícitas en este código, ya que no se necesita realizar ninguna operación condicional o iterativa en este contexto.
3. **Manejo de Excepciones:**
    - Se utiliza un bloque **try-catch** para manejar las excepciones que puedan ocurrir durante la conexión a la base de datos y la ejecución de consultas SQL. Si ocurre una excepción, se imprime la traza de la pila en la consola.

```
try {
    Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "admin", "admin");
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM productos");

} catch (SQLException e) {
    e.printStackTrace();
```

4. **Colecciones en Java:**
    - Se utiliza un objeto de tipo **DefaultTableModel** para almacenar los datos de la consulta SQL y luego asignarlo a la tabla. El **DefaultTableModel** es una implementación de la interfaz **TableModel** que proporciona una estructura de datos tabular.

```
DefaultTableModel modelo = new DefaultTableModel();
```

5. **Programación Orientada a Objetos (POO):**
    - El código muestra la creación de objetos de las clases **JFrame**, **JTable**, **JScrollPane** y **DefaultTableModel**. Estas clases están diseñadas utilizando los principios de la POO, como encapsulamiento, herencia y polimorfismo.
6. **Interfaz de Usuario con Swing:**
    - Se utilizan componentes de Swing, como **JFrame**, **JTable** y **JScrollPane**, para crear la interfaz de usuario. Estos componentes proporcionan la estructura necesaria para mostrar los datos de los productos en una ventana.

```
tabla = new JTable();
scrollPane = new JScrollPane(tabla);
add(scrollPane);
```

7. **Conexión a la Base de Datos con JDBC:**
    - Se establece una conexión a una base de datos MySQL utilizando JDBC en el bloque **try-catch**. Se ejecuta una consulta SQL para seleccionar todos los datos de la tabla **productos**. Los resultados de la consulta se procesan y se muestran en la tabla de la interfaz de usuario.

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "admin", "admin");
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM productos");
```

# 3. Uso de J2EE con JDBC MySQL

## 3.1 Servlet Producto.java

1. **Atributos:**
    - Los atributos **id**, **nombre**, **precio** y **fecha** representan las características de un producto. Estos son ejemplos de variables de instancia que almacenan información específica de cada objeto **Producto**.

2. **POO - Encapsulación:**
    - Se aplican los principios de encapsulación al declarar los atributos de la clase como **private** y proporcionar métodos **getter** y **setter** públicos para acceder y modificar estos atributos de manera controlada.

3. **Constructor:**
    - Se define un constructor para la clase **Producto** que acepta los atributos **id**, **nombre**, **precio** y **fecha** como parámetros. Este constructor se utiliza para crear objetos **Producto** con valores específicos de atributos.

```
public class Producto {

    private int id;
    private String nombre;
    private float precio;
    private Date fecha;

    public Producto(int id, String nombre, float precio, Date fecha) {
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
        this.fecha = fecha;
    }
```

4. **Métodos Getter y Setter:**

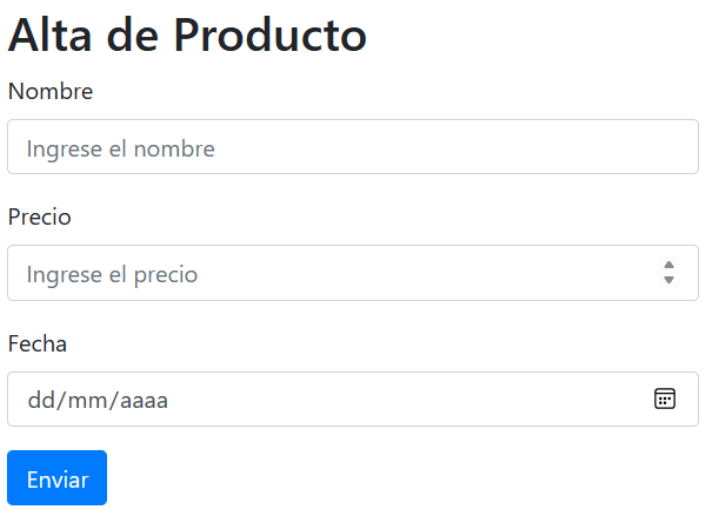
    - Se proporcionan métodos **getter** y **setter** para cada atributo de la clase. Estos métodos permiten acceder y modificar los valores de los atributos de manera controlada, manteniendo la encapsulación.

```
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 float getPrecio() {
    return precio;
}

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

public Date getFecha() {
    return fecha;
}

public void setFecha(Date fecha) {
    this.fecha = fecha;
```

## 3.2 Servlet AltaServlet.java

El método **processRequest** en un servlet Java se encarga de manejar las solicitudes HTTP entrantes y generar una respuesta. En este caso, establece el tipo de contenido como HTML UTF-8 y crea una página HTML simple que muestra el contexto de la solicitud.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet AltaServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet AltaServlet at " + request.getContextPath() + "</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```


## 3.3 Servlet EliminarProductoServlet.java

1. **Método doPost:**

Este método maneja las solicitudes POST enviadas al servlet. Aquí es donde se procesa la solicitud para eliminar un producto de la base de datos.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
```

2. **Conexion a la Base de Datos:**

Estas líneas definen la URL de conexión a la base de datos y crean objetos para la conexión y la preparación de la consulta SQL.

```
String url = "jdbc:mysql://localhost:3306/test";
Connection conn = null;
PreparedStatement ps = null;
```

### 3. Bloque Try-Catch:

```
} catch (ClassNotFoundException ex) {
    Logger.getLogger(EliminarProductoServlet.class.getName()).log(Level.SEVERE, null, ex);
} catch (SQLException ex) {
    Logger.getLogger(EliminarProductoServlet.class.getName()).log(Level.SEVERE, null, ex);
} finally {
    // Cerrar recursos
    try {
        if (ps != null) {
            ps.close();
        }
        if (conn != null) {
            conn.close();
        }
    } catch (SQLException ex) {
        Logger.getLogger(EliminarProductoServlet.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Este bloque maneja las excepciones que puedan ocurrir durante la ejecución del código, como errores de conexión a la base de datos o errores en la consulta SQL. También se asegura de cerrar correctamente los recursos, como la conexión y la declaración preparada, en el bloque **finally**.

### 4. Eliminación del Producto:

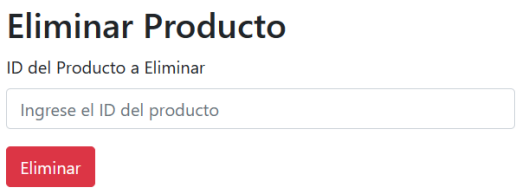
```
String eliminar = "DELETE FROM productos WHERE id = ?";
ps = conn.prepareStatement(eliminar);

int idProducto = Integer.parseInt(request.getParameter("codigo"));

ps.setInt(1, idProducto);

int filasAfectadas = ps.executeUpdate();
```

Estas líneas preparan y ejecutan una consulta SQL para eliminar un producto de la base de datos. El ID del producto se obtiene de los parámetros de la solicitud HTTP.

### 5. Respuesta al Cliente:

```
if (filasAfectadas > 0) {
    response.getWriter().println("El producto con ID " + idProducto + " ha sido eliminado correctamente.");
} else {
    response.getWriter().println("No se pudo encontrar el producto con ID " + idProducto + " para eliminarlo.")
}
```

Dependiendo de si se eliminó con éxito el producto o no, se envía una respuesta adecuada al cliente.

## 3.4 Servlet GuardarCambiosServlet.java

1. **doPost(HttpServletRequest request, HttpServletResponse response)**:
    - Este método maneja la solicitud POST enviada al servlet. Aquí hay un ejemplo de cómo se vería un método **doPost** en un servlet:

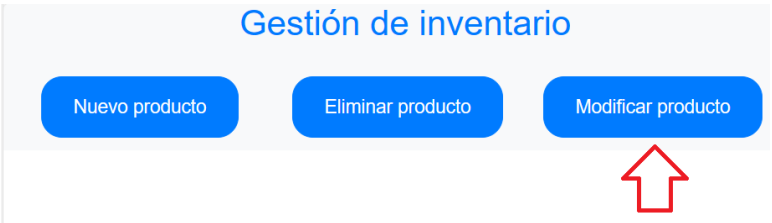
```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
```

2. **Extracción de parámetros**:
    - Aquí hay un ejemplo de cómo se extraen los parámetros del cuerpo de la solicitud:

```
String nombre = request.getParameter("nombre");
double precio = Double.parseDouble(request.getParameter("precio"));
```

4. **Escritura de la respuesta**:
    - Aquí hay un ejemplo de cómo se escribe la respuesta HTML:

```
try (PrintWriter out = response.getWriter()) {
    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Éxito</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Los cambios se han guardado correctamente.</h1>");
    out.println("</body>");
    out.println("</html>");
```




## Bibliografía

**Caules, C. Á.** (2022, enero 22). *Java Collections Framework y su estructura*. Arquitectura Java. https://www.arquitecturajava.com/java-collections-framework-y-su-estructura/

***ChatGPT*.** (s/f). Openai.com. Recuperado el 30 de abril de 2024, de https://chat.openai.com/

***Java downloads*.** (s/f). Oracle.com. Recuperado el 30 de abril de 2024, de https://www.oracle.com/java/technologies/downloads/

**Lichtmaier, N.** (s/f). *Collections en Java*. Com.ar. Recuperado el 30 de abril de 2024, de http://www.reloco.com.ar/prog/java/collections.html

**(S/f). Apache.org.** Recuperado el 30 de abril de 2024, de https://netbeans.apache.org/tutorial/main/kb/docs/java/quickstart-gui.html