

Reto Java: Modernizando "Akihabara Market"



ENLACE A GITHUB:

<https://github.com/richipa/retoJavaAkihabaraMarketRicardo/tree/main>

Ricardo Pérez Aragonese

13 de junio de 2025.

Campus FP.

1º DAM.

Programación.

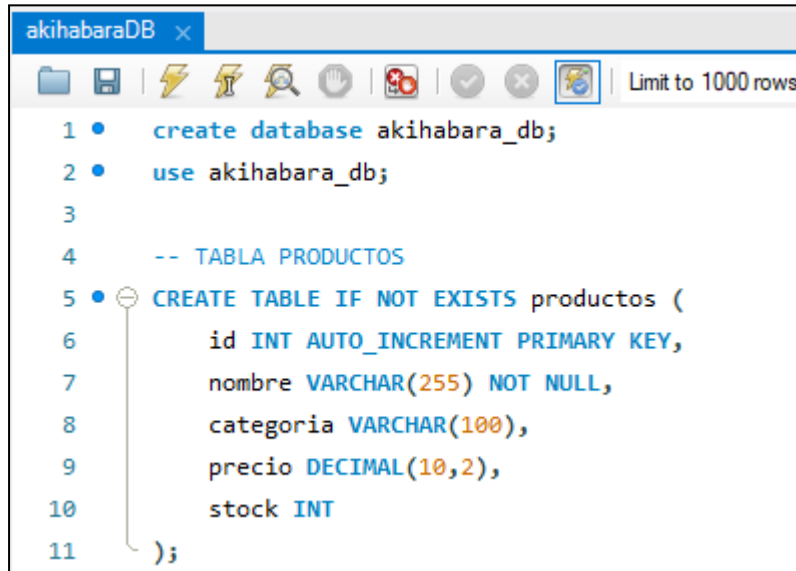
Tabla de Contenidos

Módulo 1: El Corazón del Inventario – Base de Datos y Lógica Principal.....	2
1. Diseño de la Base de Datos "Akihabara DB":.....	2
1.1 La Clase Conexion:.....	4
1.2 La Clase ProductoOtaku:.....	4
1.3 La Clase ProductoDAO:.....	4
1.4 La Clase SetupDatos:.....	5
Módulo 2: La Interfaz del Comerciante – Aplicación de Consola.....	5
2.1 La Clase InterfazConsola:.....	5
2.2 La Clase ControladorMenu:.....	6
2.3 La Clase Main:.....	6
Módulo 3: ¡El Toque de IA! – Integración con LLM vía OpenRouter.....	7
3.1 La Clase LlmService:.....	7
} catch (Exception e) {.....	8
e.printStackTrace();.....	8
return "Error al conectar con la IA.";.....	8
}.....	8
3.2 Archivo configuracion.properties:.....	9
3.3 La Clase Configuracion:.....	9
3.4 Adiciones de la IA a las clases ControladorMenu e InterfazConsola:.....	9
Módulo 4: Calidad y Entrega – Pruebas y Documentación.....	11
4.1 Pruebas exhaustivas:.....	11
BONUS QUEST: Interfaz Gráfica con Swing.....	15
5.1 La Clase MainGrafico:.....	16
5.2 La Clase VentanaGrafica:.....	16
5.3 La Clase EventosVentana:.....	18
5.4 La Clase InterfazProductos e InterfazClientes:.....	18
5.5 Probamos la aplicación por la Interfaz Gráfica:.....	25
BONUS QUEST II: Gestión de clientes.....	28
BIBLIOGRAFÍA.....	30

Módulo 1: El Corazón del Inventario – Base de Datos y Lógica Principal

1. Diseño de la Base de Datos "Akihabara DB":

He creado una base de datos simple, con una sola tabla que va a almacenar los productos de la tienda. Incluye los campos esenciales para organizarlos de la mejor manera posible.



```
1 • create database akihabara_db;
2 • use akihabara_db;
3
4 -- TABLA PRODUCTOS
5 • CREATE TABLE IF NOT EXISTS productos (
6     id INT AUTO_INCREMENT PRIMARY KEY,
7     nombre VARCHAR(255) NOT NULL,
8     categoria VARCHAR(100),
9     precio DECIMAL(10,2),
10    stock INT
11 );
```

Voy a hacer una pequeña tabla que explique la estructura de mi proyecto:

Paquete	Clase	Descripción
modelo	ProductoOtaku.java	Clase entidad que representa un producto otaku. Contiene ID, nombre, categoría, precio y stock.
	ProductoDAO.java	DAO para productos. Gestiona las operaciones CRUD en la base de datos (INSERT, SELECT, UPDATE, DELETE).
	Conexion.java	Proporciona y gestiona la conexión con la base de datos MySQL.
controlador	ControladorMenu.java	Controlador general. Incluye los métodos agregarProducto(), consultarProductoPorId(), actualizarProducto(), eliminarProducto() y menuIA().
	SetupDatos.java	Inserta productos de ejemplo automáticamente si la tabla está vacía.
	Main.java	Punto de entrada de la aplicación. Llama al menú principal del sistema.
vista	InterfazConsola.java	Gestiona la entrada y salida por consola del módulo de productos. Permite leer datos de productos y mostrarlos.

1.1 La Clase Conexion:

La primera clase que he creado, es esencial para que mi programa se conecte con la base de datos. Contiene un método getConnection que es el que se conecta con la BDD. Luego, lo uso en el resto de clases si es necesario.

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

1.2 La Clase ProductoOtaku:

He creado esta clase porque necesitaba tener claro los campos de los productos que se van a gestionar. Esta clase actúa como modelo de datos para el sistema.

Método	Explicación
ProductoOtaku(...)	Constructor que inicializa todos los atributos del producto.
getId() / setId(...)	Obtiene o actualiza el identificador único del producto.
getNombre() / setNombre()	Obtiene o cambia el nombre del producto.
getCategoria() / setCategoria()	Accede o edita la categoría del producto.
getPrecio() / setPrecio()	Accede o edita el precio del producto.
getStock() / setStock()	Accede o edita la cantidad de stock disponible.

1.3 La Clase ProductoDAO:

A continuación he creado la clase DAO para separar la lógica de la base de datos del resto del programa. Así si tengo que hacer algún tipo de modificación o mantenimiento, sé que tengo todo lo de la BDD en la misma clase.

Método	Explicación
agregarProducto(...)	Inserta un nuevo producto en la base de datos.
obtenerProductoPorId(...)	Recupera un producto concreto a partir de su ID.
obtenerTodosLosProductos()	Devuelve una lista con todos los productos registrados.
actualizarProducto(...)	Modifica los datos de un producto ya existente.
eliminarProducto(...)	Borra un producto por ID de la base de datos.

1.4 La Clase SetupDatos:

Esta clase inserta productos de ejemplo en caso de que la tabla productos esté vacía.

Método	Explicación
PrimerosProductos(...)	Inserta 3 productos por defecto si la tabla está vacía.

Módulo 2: La Interfaz del Comerciante – Aplicación de Consola

2.1 La Clase InterfazConsola:

Es muy importante ya que es la que se encarga de interactuar con el usuario directamente.

Método	Explicación
mostrarMenuPrincipal()	Muestra el menú principal y redirige a los submenús
mostrarMenuProductos()	Muestra el menú de productos con opciones de gestión de productos.
leerDatosProductoNuevo()	Solicita los datos necesarios para registrar un nuevo producto.
leerIdProducto()	Solicita un ID para operaciones de búsqueda, edición o eliminación.
mostrarProducto(...)	Muestra un producto individual en consola.
mostrarListaProductos(...)	Muestra todos los productos en consola.
actualizarProducto(...)	Solicita datos para actualizar un producto ya existente.

2.2 La Clase ControladorMenu:

Otra clase muy importante del programa, ya que es la que coordina todo el programa. Se encarga de recibir las decisiones del usuario (a través de InterfazConsola), llamar a los métodos apropiados del modelo (ProductoDAO) para acceder o modificar los datos en la base de datos, y luego devolver la información a la vista para mostrarla por pantalla.

Método	Explicación
menuPrincipal()	Muestra el menú principal y redirige a cada módulo.
menuProductos()	Muestra el menú de gestión de Productos
agregarProducto()	Lee los datos del nuevo producto y lo guarda en la base de datos.
consultarProductoPorId()	Busca y muestra un producto en base a su ID.
actualizarProducto()	Permite editar y guardar los cambios de un producto existente.
eliminarProducto()	Borra un producto a partir del ID proporcionado.

2.3 La Clase Main:

He creado esta clase como punto de entrada principal del programa. Desde aquí se lanza el controlador del menú general que inicia toda la interacción con el sistema.

Método	Explicación
main(String[] args)	Inicia la aplicación ejecutando el método menuControlador() del controlador principal.

Me he tomado la libertad de separar el main del controlador como pide el enunciado, para que el programa quede más organizado y el main lo más limpio posible.

Módulo 3: ¡El Toque de IA! – Integración con LLM vía OpenRouter

Voy a explicar un poco esta clase porque considero que es algo más complejo que lo anterior.

3.1 La Clase LlmService:

He creado esta clase para poder conectar mi programa con un modelo de inteligencia artificial mediante una API de OpenRouter. Estoy usando un modelo gratuito “mistralai-7B” .

```
//METODO QUE RECIBE EL PROMPT Y DEVUELVE LA RESPUESTA COMO STRING
public static String consultaIA(String prompt) {
    String apiKey = Configuracion.obtenerApiKey();
    if (apiKey == null || apiKey.isEmpty()) {
        return "Error al obtener la api key";
    }
}
```

Este es el método principal que se encarga de enviar una petición a la API de IA. Recibe un texto (prompt) y devuelve la respuesta generada por el modelo como un String. La variable “**String apiKey = Configuracion.obtenerApiKey()**” coge mi API key desde un archivo de configuración a parte llamado **configuración.properties** usando la clase **Configuracion.java**.

```
//CONSTRUIR EL JSON
JsonObject cuerpo = new JsonObject();
cuerpo.addProperty("model", MODELO);

JSONArray mensajes = new JSONArray();
JsonObject mensajeUsuario = new JsonObject();
mensajeUsuario.addProperty("role", "user");
mensajeUsuario.addProperty("content", prompt);
mensajes.add(mensajeUsuario);
cuerpo.add("messages", mensajes);
```

Aquí creo un objeto JSON con el modelo y el mensaje del usuario. Se sigue el formato que la API espera: un campo "model" y un array "messages" con el rol "user" y el contenido del prompt.

```
//CONECTARSE A LA API
URI uri = URI.create(API_URL);
URL url = uri.toURL();

HttpURLConnection conexion = (HttpURLConnection) url.openConnection();
conexion.setRequestMethod("POST");
conexion.setRequestProperty("Content-Type", "application/json");
conexion.setRequestProperty("Accept", "application/json");
conexion.setRequestProperty("Authorization", "Bearer " + Configuracion.obtenerApiKey());
conexion.setRequestProperty("X-Title", "AkihabaraMarket");
conexion.setDoOutput(true);
```


Esto prepara la conexión `URLConnection` con la URL de la API. Dice que el método HTTP será POST y añade los headers necesarios: tipo de contenido, autorización con la API key y un título descriptivo.

```
//ENVIAR EL JSON AL SERVIDOR
try (OutputStreamWriter writer = new OutputStreamWriter(conexion.getOutputStream())) {
    writer.write(gson.toJson(cuerpo));
}
```

“`OutputStreamWriter`” envía el JSON creado al servidor.

```
//RESPUESTA
int status = conexion.getResponseCode();
if (status == 200 || status == 201) {
    BufferedReader reader = new BufferedReader(new InputStreamReader(conexion.getInputStream()));
    JsonObject json = gson.fromJson(reader, JsonObject.class);
    reader.close();
    String respuesta = json
        .getAsJsonArray("choices")
        .get(0).getAsJsonObject()
        .getAsJsonObject("message")
        .get("content").getString();
    return respuesta.trim();
} else {
    return "Error HTTP: " + status;
}
```

Lee la respuesta del servidor SOLO si el estado es 200 o 201. Extrae el texto que genera la IA desde el JSON.

Toda la clase está envuelta en un try catch, si cualquiera de las operaciones anteriores falla, se lanza la excepción

```
} catch (Exception e) {
    e.printStackTrace();
    return "Error al conectar con la IA.";
}
```

```
configuración.properties ×  
1 API_KEY=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
```

Aquí guardo mi API KEY simplemente.

3.3 La Clase Configuración:

He creado esta clase que lee el archivo de arriba y que así, obtiene la API KEY. Entonces la API KEY no queda escrita directamente en el código, con lo que mejora la seguridad del programa.

Método	Explicación
get(String clave)	Carga el archivo configuracion.properties y devuelve el valor asociado a una clave.

3.4 Adiciones de la IA a las clases ControladorMenu e InterfazConsola:

En la clase **ControladorMenu** he añadido un método `menuIA()` y dentro del mismo método, se llama al método `consultaIA()` de la clase “LlmService”. Aquí se le dan los prompts a la IA y se almacenan en una variable.

Método	Explicación
<code>menuIA()</code>	Gestiona un submenú con dos funciones: generar descripciones o sugerir categorías.
<code>LlmService.consultaIA(...)</code>	Se llama desde este menú para obtener respuesta de IA según el prompt generado.

En la clase **InterfazConsola** he añadido simplemente un método para mostrar el menú de la IA. La IA no tiene métodos a parte como puede ser el caso de **productos** ya que esta, no realiza operaciones **CRUD** como tal y por lo tanto, no es necesario que tenga sus métodos.

```
//MENU DE LA IA
public int mostrarMenuIA() {
    System.out.println("\n=== CONSULTA A LA IA ===");
    System.out.println("1. Generar descripción de producto");
    System.out.println("2. Sugerir categoría para nuevo producto");
    System.out.println("3. Salir al menú principal");
    System.out.println("Selecciona una opción: ");

    try {
        return scanner.nextInt();
    } catch (Exception e) {
        scanner.nextLine();
        System.out.println("Entrada no válida, escribe un numero.");
        return -1;
    }
}
```

Este menú al igual que el menú de productos, interactúa directamente con el usuario, por eso pertenece a la vista.

Módulo 4: Calidad y Entrega – Pruebas y Documentación

4.1 Pruebas exhaustivas:

Insertar producto válido (precio positivo y stock positivo):

```
=== AKIHABARA MARKET ===
1. Añadir nuevo producto
2. Consultar producto por ID
3. Ver todos los productos
4. Actualizar producto
5. Eliminar producto
6. Utilizar asistente IA
7. Gestionar clientes
8. Salir
Selecciona una opción: 1
Nombre: Llavero Doraemon
Categoría: Llavero
Precio: 19,99
Stock: 5
Producto añadido con éxito.
```

Insertar con precio negativo (debe fallar):

```
=== AKIHABARA MARKET ===
1. Añadir nuevo producto
2. Consultar producto por ID
3. Ver todos los productos
4. Actualizar producto
5. Eliminar producto
6. Utilizar asistente IA
7. Gestionar clientes
8. Salir
Selecciona una opción: 1
Nombre: Poster Pokemon
Categoría: Posters
Precio: -9,99
El precio no puede ser negativo.
```

Insertar con letras en el campo "precio" (debe mostrar error):

```
=== AKIHABARA MARKET ===
1. Añadir nuevo producto
2. Consultar producto por ID
3. Ver todos los productos
4. Actualizar producto
5. Eliminar producto
6. Utilizar asistente IA
7. Gestionar clientes
8. Salir
Selecciona una opción: 1
Nombre: Poster Pokemon
Categoría: Posters
Precio: 10 euros
Stock: Entrada no válida, producto no creado.
```

Consultar ID existente (mostrar producto correctamente):

```
=== AKIHABARA MARKET ===
1. Añadir nuevo producto
2. Consultar producto por ID
3. Ver todos los productos
4. Actualizar producto
5. Eliminar producto
6. Utilizar asistente IA
7. Gestionar clientes
8. Salir
Selecciona una opción: 2
Introduce el ID del producto: 5
ID: 5 | Nombre: Manga Initial D Vol.1 | Categoría: Manga | Precio: 9,99 € | Stock: 10
```

Consultar ID inexistente (debe mostrar "producto no encontrado"):

```
=== AKIHABARA MARKET ===
1. Añadir nuevo producto
2. Consultar producto por ID
3. Ver todos los productos
4. Actualizar producto
5. Eliminar producto
6. Utilizar asistente IA
7. Gestionar clientes
8. Salir
Selecciona una opción: 2
Introduce el ID del producto: 10
Producto no encontrado.
```

Introducir letras como ID (debe capturar la excepción):

```
=== AKIHABARA MARKET ===
1. Añadir nuevo producto
2. Consultar producto por ID
3. Ver todos los productos
4. Actualizar producto
5. Eliminar producto
6. Utilizar asistente IA
7. Gestionar clientes
8. Salir
Selecciona una opción: 2
Introduce el ID del producto: Ocho
Entrada no válida, escribe un número.
```

Con varios productos insertados (debe listar bien):

```
=== AKIHABARA MARKET ===
1. Añadir nuevo producto
2. Consultar producto por ID
3. Ver todos los productos
4. Actualizar producto
5. Eliminar producto
6. Utilizar asistente IA
7. Gestionar clientes
8. Salir
Selecciona una opción: 3
ID: 2 | Nombre: Figura de Anya Forger | Categoría: Figura | Precio: 59,95 € | Stock: 8
ID: 3 | Nombre: Manga Chainsaw Man Vol.1 | Categoría: Manga | Precio: 9,99 € | Stock: 20
ID: 4 | Nombre: Póster Studio Ghibli Colección | Categoría: Póster | Precio: 15,50 € | Stock: 15
ID: 5 | Nombre: Manga Initial D Vol.1 | Categoría: Manga | Precio: 9,99 € | Stock: 10
ID: 6 | Nombre: Figura de Goku | Categoría: Figura | Precio: 149,99 € | Stock: 25
ID: 7 | Nombre: Llavero Doraemon | Categoría: Llavero | Precio: 19,99 € | Stock: 5
ID: 8 | Nombre: Poster de Pokemon | Categoría: Poster | Precio: -10,00 € | Stock: 10
```

Eliminar un producto existente y confirmar que desaparece:

```
ID: 2 | Nombre: Figura de Anya Forger | Categoría: Figura | Precio: 59,95 € | Stock: 8
ID: 3 | Nombre: Manga Chainsaw Man Vol.1 | Categoría: Manga | Precio: 9,99 € | Stock: 20
ID: 4 | Nombre: Póster Studio Ghibli Colección | Categoría: Póster | Precio: 15,50 € | Stock: 15
ID: 5 | Nombre: Manga Initial D Vol.1 | Categoría: Manga | Precio: 9,99 € | Stock: 10
ID: 6 | Nombre: Figura de Goku | Categoría: Figura | Precio: 149,99 € | Stock: 25
ID: 7 | Nombre: Llavero Doraemon | Categoría: Llavero | Precio: 19,99 € | Stock: 5
ID: 8 | Nombre: Poster de Pokemon | Categoría: Poster | Precio: -10,00 € | Stock: 10
```

```
=== AKIHABARA MARKET ===
1. Añadir nuevo producto
2. Consultar producto por ID
3. Ver todos los productos
4. Actualizar producto
5. Eliminar producto
6. Utilizar asistente IA
7. Gestionar clientes
8. Salir
Selecciona una opción: 5
Introduce el ID del producto: 8
Producto eliminado
```

```
ID: 2 | Nombre: Figura de Anya Forger | Categoría: Figura | Precio: 59,95 € | Stock: 8
ID: 3 | Nombre: Manga Chainsaw Man Vol.1 | Categoría: Manga | Precio: 9,99 € | Stock: 20
ID: 4 | Nombre: Póster Studio Ghibli Colección | Categoría: Póster | Precio: 15,50 € | Stock: 15
ID: 5 | Nombre: Manga Initial D Vol.1 | Categoría: Manga | Precio: 9,99 € | Stock: 10
ID: 6 | Nombre: Figura de Goku | Categoría: Figura | Precio: 149,99 € | Stock: 25
ID: 7 | Nombre: Llavero Doraemon | Categoría: Llavero | Precio: 19,99 € | Stock: 5
```

Eliminar un ID inexistente (debe notificar):

```
=== AKIHABARA MARKET ===
1. Añadir nuevo producto
2. Consultar producto por ID
3. Ver todos los productos
4. Actualizar producto
5. Eliminar producto
6. Utilizar asistente IA
7. Gestionar clientes
8. Salir
Selecciona una opción: 5
Introduce el ID del producto: 10
No se encontró ningún producto con ese ID.
```

Producto existente con nombre y categoría, respuesta generada en español:

```
=== CONSULTA A LA IA ===
1. Generar descripción de producto
2. Sugerir categoría para nuevo producto
3. Salir al menú principal
Selecciona una opción:
1
Introduce el ID del producto: 2
Descripción generada:
"Experiencia inmejorable para fans de anime! Nueva y hermosa figura de colección de Anya Forger"
```

Nombre genérico como “Naruto camiseta”, IA debería sugerir “Ropa”:

```
=== CONSULTA A LA IA ===
1. Generar descripción de producto
2. Sugerir categoría para nuevo producto
3. Salir al menú principal
Selecciona una opción:
2
Introduce el nombre de un nuevo producto para ver de que categoría podría ser: Naruto Camiseta
Categoría sugerida por la IA: Ropa
```

Producto con ID inexistente (caso límite):

```
=== CONSULTA A LA IA ===
1. Generar descripción de producto
2. Sugerir categoría para nuevo producto
3. Salir al menú principal
Selecciona una opción:
1
Introduce el ID del producto: 20
Producto no encontrado.
```

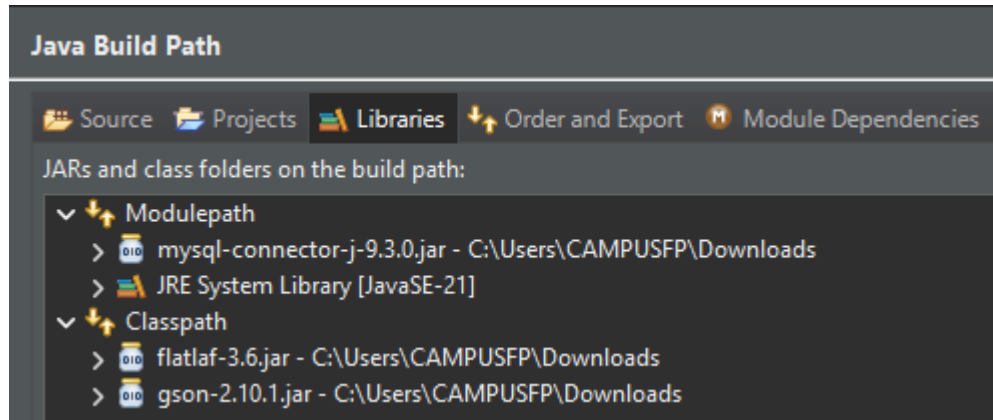
Producto con nombre muy largo o absurdo, observar resultado (caso límite):

```
=== CONSULTA A LA IA ===
1. Generar descripción de producto
2. Sugerir categoría para nuevo producto
3. Salir al menú principal
Selecciona una opción:
2
Introduce el nombre de un nuevo producto para ver de que categoría podría ser: Esternocleidomastoideo
Categoría sugerida por la IA: Otro (En este caso, en relación con el esternocleidomastoideo, estamos hablando de una estructura anatómica y no de un producto material)
```

El README estará incluido en el GITHUB más adelante

BONUS QUEST: Interfaz Gráfica con Swing

Voy a presentar mi interfaz propuesta para la tienda del Maestro Kenji, he utilizado la librería Swing y un .jar externo llamado “FlatLaf”, que es básicamente una extensión de la interfaz para personalizar esta. Primero, he descargado **FlatLaf** y lo he añadido al Build Path de mi proyecto:



He modificado la estructura de mi proyecto añadiendo estas clases para la interfaz con Swing:

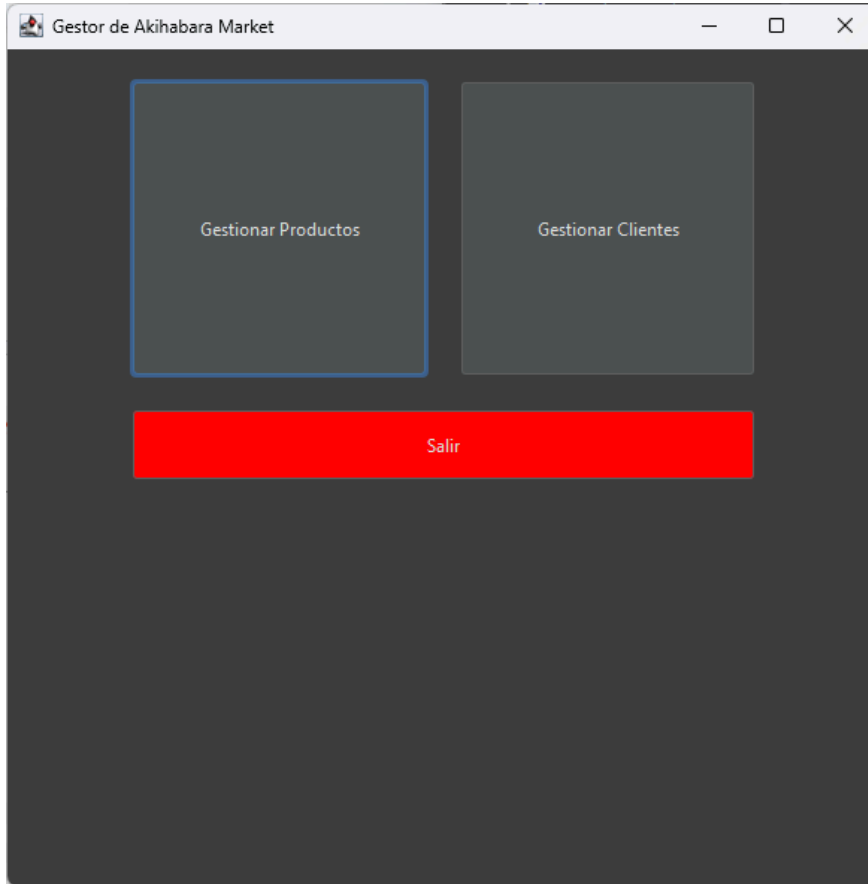
Paquete	Clase	Descripción
controlador	MainGrafico.java	Punto de entrada de la interfaz gráfica. Lanza VentanaGrafica.
vista	VentanaGrafica.java	Ventana principal gráfica con botones de acceso a gestión de productos y clientes.
vista	EventosVentana.java	Asigna los eventos a los botones de VentanaGrafica. Conecta botones con acciones.
vista	VentanaProductos.java	Interfaz gráfica para gestionar productos (formulario y tabla con CRUD).
vista	InterfazClientes.java	Interfaz gráfica para gestionar clientes (formulario y tabla con CRUD).

Lo he separado en varias clases para que si en un futuro quiero expandirlo o cambiar algo, encontrar lo que quiero más fácilmente.

5.1 La Clase MainGrafico:

Simplemente lanza la aplicación desde la interfaz gráfica.

5.2 La Clase VentanaGrafica:



Esta es la ventana principal del programa, se compone de 3 botones. Gestionar Productos, Clientes y Salir de la aplicación.

```
import javax.swing.*.*;
import java.awt.*.*;
import com.formdev.flatlaf.FlatDarculaLaf;
```

Estos son los imports necesarios para utilizar Swing además de FlatLaf.

Esta es la configuración de la ventana principal:

```
getContentPane().setBackground(Color.DARK_GRAY);
setTitle("Gestor de Akihabara Market");
setSize(600, 600);
setDefaultCloseOperation(EXIT_ON_CLOSE);
setLocationRelativeTo(null);
setLayout(new GridLayout(4, 1, 10, 10));
```

Esta configuración contiene color de fondo, título, tamaño de la ventana, cerrar la aplicación al cerrar la ventana, la posición de los elementos, etc...

```
// INICIALIZACIÓN DE BOTONES
setLayout(new FlowLayout(FlowLayout.CENTER, 20, 20));

//BOTON PRODUCTOS Y TAMAÑO
btnGestionarProductos = new JButton("Gestionar Productos");
btnGestionarProductos.setPreferredSize(new Dimension(200, 200));

//BOTON CLIENTES Y TAMAÑO
btnGestionarClientes = new JButton("Gestionar Clientes");
btnGestionarClientes.setPreferredSize(new Dimension(200, 200));

//BOTON SALIR Y TAMAÑO Y COLOR
btnSalir = new JButton("Salir");
btnSalir.setPreferredSize(new Dimension(420, 50));
btnSalir.setBackground(Color.red);
// AÑADIR BOTONES A LA VENTANA
add(btnGestionarProductos);
add(btnGestionarClientes);
add(btnSalir);
// VINCULAR EVENTOS A LOS BOTONES
EventosVentana.asignarEventos(this);
setVisible(true);
```

Inicio los botones, les doy tamaño y color, los añado y finalmente los vinculo a sus respectivos eventos (a la clase de EventosVentana).

5.3 La Clase EventosVentana:

Esta clase asigna los listeners a los botones de VentanaGrafica (por ejemplo, cerrar ventana o abrir formularios). Lo he dividido en cada botón para que se entienda mejor.

```
// ASIGNAR EVENTOS A LOS BOTONES DE VENTANA PRINCIPAL
public static void asignarEventos(VentanaGrafica ventana) {
    // BOTÓN PRODUCTOS
    ventana.btnGestionarProductos.addActionListener(e -> {
        InterfazProductos.mostrarVentanaProductos();
    });
    // BOTÓN CLIENTES
    ventana.btnGestionarClientes.addActionListener(e -> {
        InterfazClientes.mostrarVentanaClientes();
    });
    // BOTÓN SALIR
    ventana.btnSalir.addActionListener(e -> ventana.dispose()); //DISPOSE CIERRA LA VENTANA PERO
    NO LA APLICACIÓN ENTERA
}
}
```

5.4 La Clase InterfazProductos e InterfazClientes:

Estas dos clases son prácticamente idénticas, lo que hacen es crear y mostrar la ventana de gestión de clientes y productos con formulario, tabla y botones CRUD. Vamos a desglosar una de ellas, ya que lo único que cambia entre ellas es que una maneja a los productos y otra a los clientes:

ID	Nombre	Categoría	Precio	Stock
2	Figura de Anya Forger	Figura	59.95	8
3	Manga Chainsaw Man Vol.1	Manga	9.99	20
4	Póster Studio Ghibli Colección	Póster	15.5	15
5	Manga Initial D Vol.1	Manga	9.99	10
6	Figura de Goku	Figura	149.99	25
7	Llavero Doraemon	Llavero	19.99	5

Esta ventana se compone de unos campos de texto para introducir los datos de los productos o clientes, justo debajo, los 4 botones que realizan las operaciones CRUD. Y por último, debajo, una tabla que se actualiza en tiempo real que contiene los productos / clientes. Además de un botón que vuelve al menú principal.

```
// MOSTRAR VENTANA PARA GESTIÓN DE Productos
public static void mostrarVentanaProductos() {
    JFrame ventana = new JFrame("Gestión de Productos");
    ventana.setSize(600, 600);
    ventana.setLocationRelativeTo(null);
    ventana.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    ventana.setLayout(new BorderLayout());
    ProductoDAO dao = new ProductoDAO();
    // PANEL FORMULARIO PRODUCTOS
    JPanel panelFormulario = new JPanel(new GridLayout(6, 3));
    JTextField txtNombre = new JTextField();
    JTextField txtCategoria = new JTextField();
    JTextField txtPrecio = new JTextField();
    JTextField txtStock = new JTextField();
    panelFormulario.add(new JLabel("Nombre:"));
    panelFormulario.add(txtNombre);
    panelFormulario.add(new JLabel("Categoria:"));
    panelFormulario.add(txtCategoria);
    panelFormulario.add(new JLabel("Precio:"));
    panelFormulario.add(txtPrecio);
    panelFormulario.add(new JLabel("Stock:"));
    panelFormulario.add(new JTextField());
}
```

Creamos la ventana y el formulario.

```
//BOTON AGREGAR
JButton btnAgregar = new JButton("Agregar Producto");
panelFormulario.add(btnAgregar);

//BOTON CONSULTAR
JButton btnConsultar = new JButton("Consultar Producto por ID");
panelFormulario.add(btnConsultar);

//BOTON ACTUALIZAR
JButton btnActualizar = new JButton("Actualizar Producto");
panelFormulario.add(btnActualizar);

//BOTON ELIMINAR
JButton btnEliminar = new JButton("Eliminar Producto");
panelFormulario.add(btnEliminar);
ventana.add(panelFormulario, BorderLayout.NORTH);

//BOTON VOLVER AL MENU PRINCIPAL
JButton btnVolver = new JButton("Volver al menú principal");
btnVolver.addActionListener(e -> ventana.dispose());
ventana.add(btnVolver, BorderLayout.SOUTH);
```

Aquí creamos los botones

```
// TABLA PRODUCTOS
String[] columnas = {"ID", "Nombre", "Categoria", "Precio", "Stock"};
DefaultTableModel modeloTabla = new DefaultTableModel(columnas, 0);
JTable tabla = new JTable(modeloTabla);
ventana.add(new JScrollPane(tabla), BorderLayout.CENTER);

//SE RECARGA LA TABLA
Runnable cargarProductos = () -> {
    modeloTabla.setRowCount(0);
    for (ProductoOtaku p : dao.obtenerTodosLosProductos()) {
        modeloTabla.addRow(new Object[]{
            p.getId(), p.getNombre(), p.getCategoria(), p.getPrecio(), p.getStock()
        });
    }
};
cargarProductos.run();
```

Creamos la tabla y creamos un método ejecutable que la actualice en tiempo real.

```
// ACCIÓN BOTÓN AGREGAR PRODUCTO
btnAgregar.addActionListener(e -> {
    try {
        String nombre = txtNombre.getText().trim();
        String categoria = txtCategoria.getText().trim();
        String precio = txtPrecio.getText().trim();
        String stock = txtStock.getText().trim();
        if (nombre.isEmpty() || categoria.isEmpty() || precio.isEmpty() || stock.isEmpty()) {
            JOptionPane.showMessageDialog(ventana, "Todos los campos deben estar completos.");
            return;
        }

        //PARSEO A DOUBLE E INTEGER
        double precioParseado = Double.parseDouble(precio);
        int stockParseado = Integer.parseInt(stock);

        ProductoOtaku nuevo = new ProductoOtaku(nombre, categoria, precioParseado, stockParseado);
        dao.agregarProducto(nuevo);
        JOptionPane.showMessageDialog(ventana, "Producto añadido correctamente.");
        txtNombre.setText("");
        txtCategoria.setText("");
        txtPrecio.setText("");
        txtStock.setText("");
        cargarProductos.run();
    } catch (Exception e1) {
        JOptionPane.showMessageDialog(ventana, "Error al agregar Producto: " + e1.getMessage());
    }
});
```

Este código es el evento que se ejecuta al pulsar el botón "Agregar Producto". Primero, recoge los datos de los campos de texto (nombre, categoría, precio y stock) y comprueba que no estén vacíos. Luego, convierte el precio a número decimal (double) y el stock a entero (int). Con estos datos, crea un nuevo objeto de tipo ProductoOtaku y lo guarda en la base de datos usando un objeto DAO.

```
// ACCIÓN BOTÓN CONSULTAR Producto POR ID
btnConsultar.addActionListener(e -> {
    String input = JOptionPane.showInputDialog(ventana, "Introduce el ID del Producto:");
    try {
        int id = Integer.parseInt(input);
        ProductoOtaku Producto = dao.obtenerProductoPorId(id);
        if (Producto != null) {
            JOptionPane.showMessageDialog(ventana,
                "ID: " + Producto.getId() + "\n" +
                "Nombre: " + Producto.getNombre() + "\n" +
                "Categoria: " + Producto.getCategoria() + "\n" +
                "Precio: " + Producto.getPrecio() + "\n" +
                "Teléfono: " + Producto.getStock()
            );
        } else {
            JOptionPane.showMessageDialog(ventana, "Producto no encontrado.");
        }
    } catch (NumberFormatException e2) {
        JOptionPane.showMessageDialog(ventana, "ID no válido.");
    }
});
```

Este código maneja la acción del botón "Consultar Producto por ID". Cuando se pulsa, muestra un cuadro de diálogo para que el usuario introduzca un ID. Luego intenta convertir ese ID a número entero (int) y busca el producto correspondiente en la base de datos usando el DAO.

```
// ACCIÓN BOTÓN ACTUALIZAR
btnActualizar.addActionListener(e -> {
    String input = JOptionPane.showInputDialog(ventana, "Introduce el ID del Producto a actualizar:");
    try {
        int id = Integer.parseInt(input);
        ProductoOtaku Producto = dao.obtenerProductoPorId(id);
        if (Producto != null) {

            String nuevoNombre = JOptionPane.showInputDialog(ventana, "Nuevo nombre:",
Producto.getNombre());
            String nuevaCategoria = JOptionPane.showInputDialog(ventana, "Nueva categoría:",
Producto.getCategoria());
            String nuevoPrecio = JOptionPane.showInputDialog(ventana, "Nuevo precio:",
Producto.getPrecio());
            String nuevoStock = JOptionPane.showInputDialog(ventana, "Nuevo stock:", Producto.getPrecio());

            //PARSEO A DOUBLE E INTEGER
            double precio = Double.parseDouble(nuevoPrecio);
            int stock = Integer.parseInt(nuevoStock);

            // ACTUALIZAR EL OBJETO PRODUCTO
            Producto.setNombre(nuevoNombre);
            Producto.setCategoria(nuevaCategoria);
            Producto.setPrecio(precio);
            Producto.setStock(stock);
            boolean actualizado = dao.actualizarProducto(Producto);
            if (actualizado == true) {
                JOptionPane.showMessageDialog(ventana, "Producto actualizado correctamente.");
                cargarProductos.run();
            } else {
                JOptionPane.showMessageDialog(ventana, "No se pudo actualizar el Producto.");
            }
        } else {
            JOptionPane.showMessageDialog(ventana, "Producto no encontrado.");
        }
    } catch (NumberFormatException e3) {
        JOptionPane.showMessageDialog(ventana, "ID no válido.");
    }
});
```

Este código maneja la acción del botón "Actualizar": pide al usuario el ID del producto a modificar, busca el producto en la base de datos y, si existe, solicita los nuevos datos (nombre, categoría, precio y stock) mostrando los valores actuales como referencia. Luego convierte el precio a double y el stock a int, actualiza el objeto y lo guarda en la base de datos.


```
// ACCIÓN BOTÓN ELIMINAR
btnEliminar.addActionListener(e -> {
    String input = JOptionPane.showInputDialog(ventana, "Introduce el ID del Producto a eliminar:");
    try {
        int id = Integer.parseInt(input);
        ProductoOtaku Producto = dao.obtenerProductoPorId(id);
        if (Producto != null) {
            boolean eliminado = dao.eliminarProducto(id);
            if (eliminado == true) {
                JOptionPane.showMessageDialog(ventana, "Producto eliminado.");
                cargarProductos.run();
            } else {
                JOptionPane.showMessageDialog(ventana, "No se pudo eliminar el Producto.");
            }
        } else {
            JOptionPane.showMessageDialog(ventana, "Producto no encontrado.");
        }
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(ventana, "ID no válido.");
    }
});

ventana.setVisible(true);
}
```

Este código controla la acción del botón "Eliminar": primero pide al usuario el ID del producto a borrar mediante un cuadro de diálogo. Luego intenta convertir ese ID a número entero (int) y busca el producto en la base de datos. Finalmente lo elimina si no ocurre ningún error.

5.5 Probamos la aplicación por la Interfaz Gráfica:

Ahora mostraré clientes ya que anteriormente he mostrado los productos. Vamos a hacer el CRUD entero con los clientes.

Agregar cliente:

Gestión de Clientes

Nombre: Sergio

Correo: sergio@mail.com

Teléfono: +34 999 99 99

Agregar cliente Consultar cliente por ID

Actualizar cliente Eliminar cliente

ID	Nombre	Correo	Teléfono
4	Ian Fernandez	ianfdez@mail.com	+34 877 90 56

Volver al menú principal

Mensaje

Cliente añadido correctamente.

Aceptar

ID	Nombre	Correo	Teléfono
4	Ian Fernandez	ianfdez@mail.com	+34 877 90 56
6	Sergio	sergio@mail.com	+34 999 99 99

Consultar Cliente por su ID (buscamos a Ian):

Entrada

Introduce el ID del cliente:

4

Aceptar Cancelar

Mensaje

ID: 4
Nombre: Ian Fernandez
Correo: ianfdez@mail.com
Teléfono: +34 877 90 56

Aceptar

Actualizar un cliente (a Sergio por ejemplo):

Entrada

Introduce el ID del cliente a actualizar:

6

Aceptar Cancelar

Entrada

Nuevo nombre:

Sergio Gomez

Aceptar Cancelar


Así con todos los campos

ID	Nombre	Correo	Teléfono
4	Ian Fernandez	ianfdez@mail.com	+34 877 90 56
6	Sergio Gomez	sergiogomez@mail.com	+34 999 99 99

Eliminamos a un cliente (Ian):

Entrada

×


 Introduce el ID del cliente a eliminar:

Aceptar

Cancelar

Mensaje

×

 Cliente eliminado.

Aceptar

ID	Nombre	Correo	Teléfono
6	Sergio Gomez	sergiogomez@mail.com	+34 999 99 99

BONUS QUEST II: Gestión de clientes

Este apartado es prácticamente idéntico al de gestión de productos. He añadido una tabla de clientes a mi base de datos, y después he ido creando cada clase correspondiente. Simplemente voy a mostrar las clases que he creado para los clientes y dónde están organizadas en una tabla.

```
13  -- TABLA CLIENTES
14  • CREATE TABLE clientes (
15      id INT AUTO_INCREMENT PRIMARY KEY,
16      nombre VARCHAR(255) NOT NULL,
17      email VARCHAR(255) NOT NULL UNIQUE,
18      telefono VARCHAR(20),
19      fecha_registro DATE DEFAULT (CURRENT_DATE)
20  );
```

Aquí va la tabla con las clases que he añadido para los clientes:

Paquete	Clase	Descripción
modelo	ClienteOtaku	Modelo de datos que representa a un cliente (ID, nombre, email, teléfono y fecha de registro). Incluye constructores y getters/setters.
modelo	ClienteDAO	Clase de acceso a datos que gestiona operaciones CRUD con la base de datos (insertar, consultar, actualizar y eliminar clientes).

Además, ahora la clase InterfazConsola y ControladorMenu, incluyen (lógicamente) las funcionalidades de Clientes.

ControladorMenu	menuClientes()	Muestra el submenú de gestión de clientes y redirige a las operaciones CRUD.
	agregarCliente()	Recoge datos desde la vista y llama a ClienteDAO.agregarCliente().
	consultarClientePorId()	Busca un cliente por ID y lo muestra en la vista.
	actualizarCliente()	Actualiza los datos de un cliente existente usando el DAO.
	eliminarCliente()	Elimina un cliente usando su ID a través del DAO.
InterfazConsola	mostrarMenuClientes()	Muestra el menú de opciones para gestionar clientes (CRUD).
	leerDatosClienteNuevo()	Solicita al usuario los datos básicos de un nuevo cliente (nombre, email, teléfono).
	leerIdCliente()	Pide el ID del cliente para operaciones específicas (consulta, edición, etc.).
	mostrarCliente(...)	Muestra los detalles de un cliente individual en consola.
	mostrarListaClientes(...)	Lista todos los clientes registrados en formato tabla.

BIBLIOGRAFÍA

Java tutorial. (s/f). W3schools.com. Recuperado el 13 de junio de 2025, de
<https://www.w3schools.com/java/default.asp>

Java platform SE 8. (s/f). Oracle.com. Recuperado el 13 de junio de 2025, de
<https://docs.oracle.com/javase/8/docs/api/index.html?javax/swing/package-summary.html>

FlatLaf - flat look and feel. (s/f). Formdev.com. Recuperado el 13 de junio de 2025, de
<https://www.formdev.com/flatlaf/>