



# Manual Completo de Desarrollo – Akihabara Market

Enlace a GitHub: <https://github.com/DiegoCuadrado/PROGRAMACION.git>

Este manual documenta a fondo el desarrollo de la aplicación de consola **Akihabara Market**, un sistema CRUD para gestionar productos otaku, con el valor añadido de inteligencia artificial para generar descripciones y sugerir categorías. Cada módulo se ha implementado con una razón clara y un diseño fundamentado en principios de buenas prácticas en Java.

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

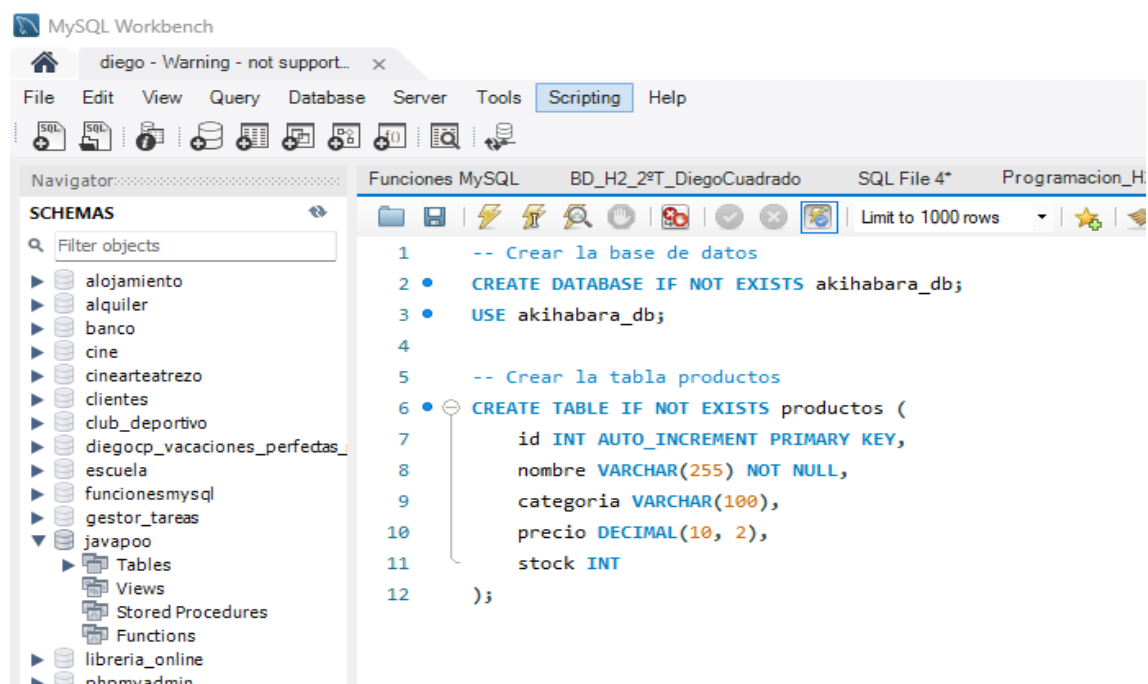
### 1. Diseño de la Base de Datos "AkihabaraDB":

El objetivo principal de este módulo es diseñar una base de datos relacional que sirva como **almacén central de información para los productos otaku** en la aplicación Akihabara Market. Se requiere que la base de datos sea robusta, eficiente, fácil de mantener, y adaptada al crecimiento de datos.

### **Por qué usamos MySQL:**

MySQL es una de las bases de datos relacionales más utilizadas en entornos de desarrollo, tanto profesionales como educativos. Es:

- Gratuita y de código abierto.
- Compatible con JDBC en Java.
- Suficientemente potente para aplicaciones CRUD de tamaño pequeño o medio.
- Rápida, escalable y bien documentada.



**Motivación del nombre:**

El nombre `akihabara_db` representa el núcleo de la app Akihabara Market. Nos permite distinguir esta base de datos en servidores que puedan contener múltiples esquemas.

**Ventajas del diseño:**

-**Simplicidad:** Una tabla sencilla y clara, perfecta para CRUD.

-**Eficiencia:** Columnas bien dimensionadas y optimizadas para consultas.

-**Escalabilidad:** Se pueden añadir nuevas columnas fácilmente (como descripción, proveedor, etc.).

-**Integrabilidad:** Compatible con herramientas externas (JDBC, editores SQL, APIs REST, etc.).

**Conclusion:**

Este diseño de base de datos proporciona una estructura sólida y clara para almacenar productos otaku. Las decisiones tomadas equilibran:

- **Flexibilidad para expansión.**
- **Rendimiento para consultas.**
- **Compatibilidad con Java y JDBC.**

En versiones futuras se puede:

- Añadir una tabla de categorías.
- Introducir usuarios o pedidos.
- Crear relaciones entre entidades.

**2. Modelo de Dominio – La Clase ProductoOtaku:**

Este módulo se centra en la **creación de la representación en código (Java) de un producto otaku**, es decir, en el **modelo de dominio** que conecta la lógica de negocio de la aplicación con los datos almacenados en la base de datos.

La clase `ProductoOtaku` cumple el rol de **entidad o POJO (Plain Old Java Object)** y es el punto central para la gestión de datos en toda la aplicación.

**Principios de diseño aplicados:**

- **POJO limpio y sin lógica:** No hay métodos que mezclen comportamiento ni reglas de negocio. Separa datos de funcionalidad.
- **Reutilizable:** Esta clase puede usarse tanto en consola como en apps gráficas o web.
- **Compatible con persistencia:** Ideal para DAO, Hibernate o frameworks de serialización.

- **Escalable:** Podemos extender fácilmente con más campos, como descripción, proveedor, etc.

### ¿Dónde usamos esta clase?

- ❑ En el ProductoDAO: la usamos para **enviar y recibir datos de la base de datos**.
- ❑ En InterfazConsola: la usamos para **pedir información al usuario y mostrar resultados**.
- ❑ En MainApp: se crean instancias para ser procesadas.
- ❑ En el futuro: puede usarse para serialización JSON o REST APIs.

La clase ProductoOtaku es el **modelo base de toda la aplicación**. Gracias a su diseño:

- Sirve como puente entre la base de datos y la lógica del programa.
- Es clara, reutilizable y compatible con herramientas externas.
- Sigue buenas prácticas de diseño y arquitectura orientada a objetos.

```
ProductoOtaku.java X
30 // Getters y setters
31 public int getId() {
32     return id;
33 }
34
35 public void setId(int id) {
36     this.id = id;
37 }
38
39 public String getNombre() {
40     return nombre;
41 }
42
43 public void setNombre(String nombre) {
44     this.nombre = nombre;
45 }
46
47 public String getCategoria() {
48     return categoria;
49 }
50
51 public void setCategoria(String categoria) {
52     this.categoria = categoria;
53 }
54
55 public double getPrecio() {
56     return precio;
57 }
58
59 public void setPrecio(double precio) {
60     this.precio = precio;
61 }
62
63 public int getStock() {
64     return stock;
65 }
66
67 public void setStock(int stock) {
68     this.stock = stock;
69 }
```

### 3. Acceso a Datos – El Guardián de la Información (ProductoDAO.java):

Este módulo trata de **conectar Java con MySQL** usando JDBC, y de aplicar el patrón DAO (Data Access Object) para acceder a los datos de forma segura, organizada y reutilizable.

La idea es separar **la lógica de acceso a datos** del resto del programa para mantener una arquitectura limpia y escalable.

#### CLASE DATABASECONNECTION

##### ¿Qué hace esta clase?

Encapsula el código necesario para **establecer una conexión con MySQL**. Esto permite que:

- Toda la aplicación use la misma lógica de conexión.
- Si cambia el nombre de la base de datos o las credenciales, solo lo modificamos aquí.
- Evitamos duplicar código.

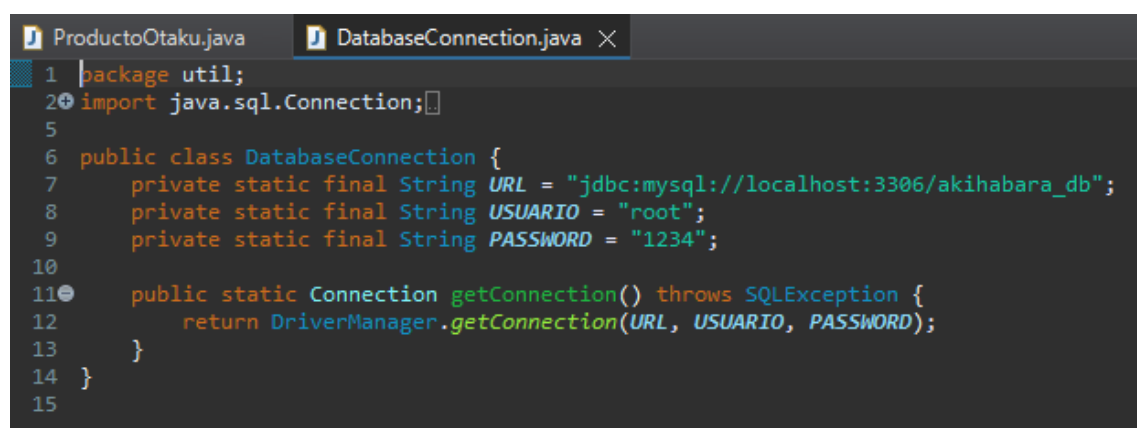
##### ¿Por qué así?

☑ Reutilizable: todas las clases DAO pueden usar `DatabaseConnection.getConnection()`.

☑ Centralización: cambia la configuración una sola vez si se migra la base de datos.

☑ Simplicidad: evitamos copiar el `DriverManager.getConnection()` en cada clase.

☑ Facilidad **de testing**: se puede simular o reemplazar la conexión en pruebas.



```
1 package util;
2 import java.sql.Connection;
3
4
5
6 public class DatabaseConnection {
7     private static final String URL = "jdbc:mysql://localhost:3306/akihabara_db";
8     private static final String USUARIO = "root";
9     private static final String PASSWORD = "1234";
10
11     public static Connection getConnection() throws SQLException {
12         return DriverManager.getConnection(URL, USUARIO, PASSWORD);
13     }
14 }
15
```

## CLASE PRODUCTODAO

¿Qué hace esta clase?

Encapsula todas las operaciones CRUD relacionadas con la tabla productos. Utiliza JDBC para comunicarse con MySQL de forma segura y eficiente.

Métodos implementados:

void agregarProducto(ProductoOtaku producto)

ProductoOtaku obtenerProductoPorId(int id)

List<ProductoOtaku> obtenerTodosLosProductos()

boolean actualizarProducto(ProductoOtaku producto)

boolean eliminarProducto(int id)

Este módulo permite que nuestra aplicación:

- Sea segura, escalable y mantenible.
- Separe correctamente la lógica de negocio de la persistencia.
- Use buenas prácticas con JDBC (como PreparedStatement y manejo de errores).
- Esté preparada para ampliaciones como PostgreSQL, SQLite o acceso desde una web.

## 4. Puesta a Punto – Carga Inicial de Tesoros:

La **carga inicial de datos** es un procedimiento que se ejecuta una sola vez (o condicionalmente) para **poblar la base de datos con valores predeterminados** que sirven como ejemplo, referencia o contenido mínimo para empezar a usar una aplicación.

En nuestro caso, estamos insertando una pequeña selección de productos otaku reales para que el sistema **no arranque vacío**.

Para simplificar el proceso y evitar duplicar código, esta funcionalidad se colocó directamente en la clase MainApp, dentro del método main().

También podría haberse separado en una clase auxiliar llamada SetupDatos.java, lo cual es recomendable en proyectos más grandes. Sin embargo, en este caso, mantenerlo en MainApp lo hace más accesible y directo para propósitos educativos.

¿Por qué hacemos esta carga inicial?

- ✓ **Usabilidad inmediata:** El usuario puede ver y probar la app sin tener que insertar productos manualmente al principio.
- ✓ **Evita una base de datos vacía:** Mejora la experiencia visual y funcional de la consola desde el primer uso.
- ✓ **Facilita las pruebas:** Permite probar funciones como búsqueda, actualización o eliminación sin tener que crear datos primero.
- ✓ **Inspiración real:** Los productos están basados en merchandising otaku reconocible (Anya Forger, Chainsaw Man, Studio Ghibli), lo que aporta una experiencia más inmersiva.

Esta funcionalidad convierte tu aplicación en un sistema **listo para usar desde el primer segundo**, evitando errores visuales o confusión por una base de datos vacía. Es una práctica estándar en desarrollo profesional y, además, mejora la experiencia de desarrollo, demostración y prueba del sistema.

## **Módulo 2: La Interfaz del Comerciante –Aplicación de Consola**

### **1. La Vista de Consola (InterfazConsola.java o similar):**

En toda aplicación interactiva, **la interfaz de usuario** es el puente entre el sistema y la persona que lo utiliza. En este proyecto, al tratarse de una **aplicación Java de consola**, decidimos separar la lógica de presentación en una clase exclusiva llamada `InterfazConsola.java`.

#### **Objetivo principal:**

Encapsular todas las responsabilidades relacionadas con **mostrar menús, pedir datos y presentar resultados**, para que el resto del sistema (como `MainApp` o `ProductoDAO`) no tenga que preocuparse por cómo se comunican con el usuario.

🔄 Esta separación mejora la legibilidad, facilita pruebas y permite cambiar la interfaz en el futuro (por ejemplo, pasar a una interfaz gráfica o web) sin reescribir la lógica del negocio.

La clase `InterfazConsola` tiene cinco funciones clave:

#### **1. Mostrar el menú principal**

- Presenta todas las opciones posibles al usuario: ver productos, buscar, añadir, editar, eliminar, salir, etc.
- Lo hace de forma clara y numerada, para una navegación sencilla.

#### **2. Solicitar al usuario datos necesarios**

Incluye métodos que piden información como:



- Atributos de un nuevo producto: nombre, categoría, precio, stock.
- ID para búsqueda, edición o eliminación.

Cada dato se recoge con validaciones básicas para evitar errores comunes.

### 3. Mostrar listas de productos de forma legible

Cuando el sistema devuelve una lista de productos desde la base de datos, esta clase se encarga de presentarlos de forma **formateada, clara y bonita**.

### 4. Mostrar mensajes al usuario

Incluye métodos para mostrar:

- ✓ Confirmaciones.
- ⚠ Errores.
- ⓘ Mensajes informativos.

Esto mejora la experiencia del usuario y facilita el seguimiento del sistema.

### 5. Leer opciones del menú

Lee la opción seleccionada por el usuario y devuelve un número entero al controlador (MainApp). Se puede extender para incluir validación de rango o manejo de excepciones.

La clase InterfazConsola es fundamental para que el usuario interactúe con Akihabara Market de forma fluida. Aunque no contiene lógica de negocio ni accede a la base de datos, **es el punto de entrada y salida de toda la experiencia del usuario**. Diseñarla de forma modular, limpia y clara es una de las claves para que una aplicación de consola sea fácil de usar y mantener.

## 2. El Controlador (MainApp.java o similar):

En una arquitectura bien estructurada, la **clase controladora** es la encargada de **coordinar** todos los componentes del sistema: la lógica del negocio, la presentación al usuario y el acceso a datos.

En nuestro caso, la clase MainApp.java es el **cerebro de la aplicación**. No contiene lógica propia de presentación ni persistencia, sino que **actúa como intermediario entre el usuario y los módulos funcionales** del sistema:

- La interfaz de consola (InterfazConsola)
- El acceso a la base de datos (ProductoDAO)
- La inteligencia artificial (LlmService)



✦ Esta separación permite que cada parte del sistema se enfoque en su responsabilidad, haciendo que el código sea más limpio, modular y mantenible.

### ✓ 1. Instanciar los componentes principales

En el método `main`, se crean los objetos necesarios para que funcione la aplicación:

```
ProductoDAO productoDAO = new ProductoDAO();
```

```
InterfazConsola interfaz = new InterfazConsola();
```

```
LlmService asistenteIA = new LlmService(); // si ya está implementado
```

### ✓ 2. Controlar el ciclo de vida del programa

La clase implementa un **bucle principal** (normalmente un `while`) que se repite hasta que el usuario decide salir.

### ✓ 3. Ejecutar la acción adecuada según la opción del usuario

Cada opción del menú se relaciona con una acción específica:

Opción	Acción	Responsable
1	Añadir producto al inventario	ProductoDAO
2	Buscar producto por ID	ProductoDAO
3	Ver todos los productos	ProductoDAO
4	Actualizar producto existente	ProductoDAO
5	Eliminar producto	ProductoDAO
6	Llamar al asistente IA (LlmService)	LlmService
7	Salir del programa	MainApp

¿Por qué es importante esta arquitectura?

-Separa **responsabilidades**: Cada clase hace su trabajo.

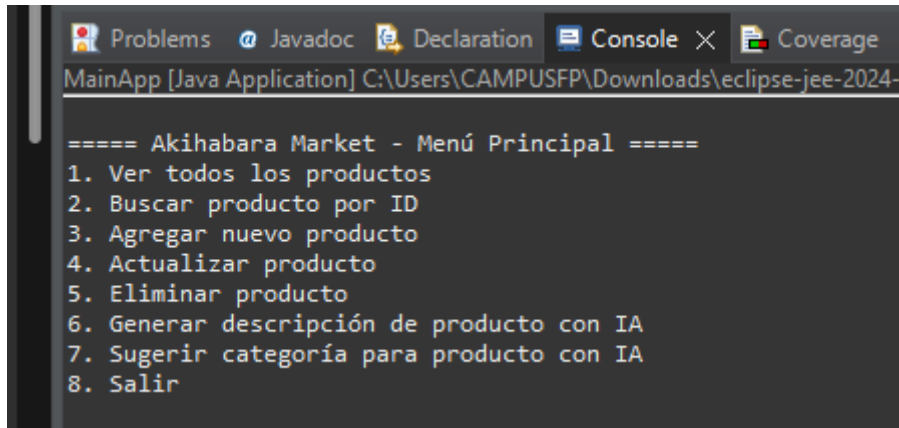
-Facilita **mantenimiento**: Si cambia la interfaz, no se toca la lógica ni la base de datos.

-Permite **escalar**: Si agregamos una interfaz gráfica o una API REST, no hay que modificar el controlador.

-Hace **el código más limpio y testeable**.

La clase `MainApp` es el **nexo de unión de todos los módulos**. Aunque no hace nada por sí misma, **hace que todo lo demás funcione en armonía**. Si cada módulo fuese una pieza de ajedrez, `MainApp` sería el tablero que les da sentido.

Gracias a ella, la aplicación funciona de manera organizada, modular y escalable. Es una pieza indispensable del sistema y un excelente ejemplo de buenas prácticas en programación Java.



```
==== Akihabara Market - Menú Principal ====
1. Ver todos los productos
2. Buscar producto por ID
3. Agregar nuevo producto
4. Actualizar producto
5. Eliminar producto
6. Generar descripción de producto con IA
7. Sugerir categoría para producto con IA
8. Salir
```

### **3. Operaciones del Inventario (a través de InterfazConsola y MainApp):**

Este apartado se centra en cómo dotar de **funcionalidad completa al menú principal** de la aplicación:

- Implementando la lógica real detrás de cada opción.
- Asegurando que las responsabilidades estén correctamente separadas:
  - InterfazConsola: entrada/salida y validación inicial.
  - MainApp: flujo de control.
  - ProductoDAO: interacción con la base de datos.

#### **Validación en InterfazConsola**

Todos los datos ingresados por el usuario se deben validar antes de ser enviados al ProductoDAO. Esto incluye:

- Nombres no vacíos.
- Categorías dentro de una lista predefinida.
- Precio como double positivo.
- Stock como int mayor o igual a 0.
- ID como entero válido.

#### **¿Por qué aquí?**

- Si los datos son inválidos, no tiene sentido contactar con la base de datos.
- Evita excepciones (NumberFormatException, SQLException, etc.).
- Mejora la experiencia del usuario al detectar errores inmediatamente.

Gracias a esta implementación:

- Se logra un flujo claro y coherente en toda la aplicación.
- Se garantiza que los datos sean válidos antes de persistirlos.

- Se respeta el principio de responsabilidad única en cada clase.
- El usuario tiene una experiencia consistente y segura.

¿Te gustaría que te genere una plantilla completa del switch con todos estos casos listos para copiar/pegar en MainApp.java?

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

#### **1. Conexión al Mundo de los LLMs:**

El objetivo de este módulo es incorporar inteligencia artificial generativa (IA de lenguaje o LLM) a la aplicación **Akihabara Market** utilizando el servicio de **OpenRouter.ai**. Para ello, debemos:

1. Registrar una cuenta.
2. Obtener y proteger una API Key personal.
3. Elegir un modelo LLM gratuito.
4. Integrar esa clave de forma segura en nuestro proyecto Java.
5. Visita <https://openrouter.ai>.
6. Crea una cuenta gratuita.
7. Accede a tu panel de usuario.
8. En la sección “API Keys”, genera una nueva clave.

**🔒 Importante:** Esta clave es **personal** y debe ser protegida como si fuera una contraseña. No la compartas ni la subas a GitHub u otros repositorios públicos.

Integrar OpenRouter.ai de forma profesional implica:

- Registrarse y obtener tu clave personal.
- Elegir modelos gratuitos adecuados.
- Gestionar la clave como un secreto privado.
- Separar configuración y lógica en tu aplicación.

De este modo, tu aplicación no solo accede a IA de forma segura, sino que sigue buenas prácticas de ingeniería de software. ¿Quieres que te prepare también un ejemplo real de cómo estructurar la petición HTTP con HttpClient y la respuesta con Gson?

#### **2. El Oráculo Digital (LlmService.java):**

Crear una clase Java que permita **comunicarse con modelos de lenguaje (LLM)** usando la **API de OpenRouter.ai**, con el fin de obtener descripciones inteligentes o sugerencias categorizadas para los productos de Akihabara Market.

## 🔑 Variables y Configuración

En el constructor, cargamos parámetros sensibles desde un archivo `config.properties`, como ya se explicó en el paso anterior:

```
private final String apiKey;  
private final String modelo;  
private final String apiUrl = "https://openrouter.ai/api/v1/chat/completions";  
  
public LlmService() {  
    Properties config = new Properties();  
    try (InputStream input = new FileInputStream("config.properties")) {  
        config.load(input);  
        this.apiKey = config.getProperty("openrouter.api_key");  
        this.modelo = config.getProperty("openrouter.model");  
    } catch (IOException e) {  
        throw new RuntimeException("Error cargando configuración de OpenRouter", e);  
    }  
}
```

---

### ✉ Método principal: `obtenerRespuesta(String promptUsuario)`

#### ✓ Entrada:

- String `promptUsuario`: la instrucción o pregunta para el LLM, como:  
  
“Describe el producto llamado ‘Figura de Luffy Gear 5’ en tono creativo y breve.”

#### ✓ Salida:

- String: texto generado por el modelo (respuesta del LLM).
- 

## 🔗 Construcción del cuerpo JSON

OpenRouter espera un cuerpo en este formato:

```
{  
  "model": "mistralai/mistral-7b-instruct:free",  
  "messages": [  
    {  
      "role": "user",  
      "content": "Describe este producto..."  
    }  
  ]  
}
```

Esto se construye en Java con Gson:

```
JsonObject mensaje = new JsonObject();
mensaje.addProperty("role", "user");
mensaje.addProperty("content", promptUsuario);
```

```
JsonArray mensajes = new JsonArray();
mensajes.add(mensaje);
```

```
JsonObject cuerpo = new JsonObject();
cuerpo.addProperty("model", this.modelo);
cuerpo.add("messages", mensajes);
```

---

## Envío de la petición HTTP POST

Utilizamos HttpClient (nativo desde Java 11):

```
HttpClient cliente = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create(apiUrl))
    .header("Authorization", "Bearer " + apiKey)
    .header("Content-Type", "application/json")
    .POST(HttpRequest.BodyPublishers.ofString(cuerpo.toString()))
    .build();
```

---

## Manejo de la respuesta

La respuesta de OpenRouter viene en formato JSON. Un ejemplo es:

```
json
CopiarEditar
{
  "choices": [
    {
      "message": {
        "content": "¡Este póster de Ghibli es una joya visual para cualquier fan!"
      }
    }
  ]
}
```

### Código para procesar la respuesta:

```
try {
    HttpResponse<String> response = cliente.send(request, HttpResponse.BodyHandlers.ofString());

    if (response.statusCode() == 200) {
        JsonObject json = JsonParser.parseString(response.body()).getAsJsonObject();
        JsonArray choices = json.getAsJsonArray("choices");
        JsonObject mensajeRespuesta = choices.get(0).getAsJsonObject().getAsJsonObject("message");
        return mensajeRespuesta.get("content").getString();
    } else {
        System.err.println("Error del API (status " + response.statusCode() + "): " + response.body());
        return "⚠ No se pudo generar la respuesta del asistente IA.";
    }
}
```

```
} catch (IOException | InterruptedException e) {  
    e.printStackTrace();  
    return "⚠ Error al comunicarse con OpenRouter.";  
}
```

---

## ✂ Manejo de errores

- Se comprueba el código HTTP (debe ser 200).
  - Se capturan excepciones como `IOException` y `InterruptedException`.
  - Se devuelve un mensaje de error entendible para el usuario final.
- 

## 📖 Ventajas del uso de `HttpClient` y `Gson`

Tecnología	Razón de elección
<code>HttpClient</code>	Es nativa desde Java 11, sin necesidad de librerías externas. Sencilla y potente.
<code>Gson</code>	Librería ligera de Google para manipular JSON, ideal para extraer respuestas de la API.
Formato POST con JSON	Requisito del endpoint de OpenRouter para una comunicación estandarizada.

---

## ✓ Resultado final

Al invocar `obtenerRespuesta(prompt)`, obtienes texto generado automáticamente por IA, por ejemplo:

**Entrada:** "Sugiere una categoría para el producto 'Peluches Pokémon colección Kanto'."

**Salida:** "Categoría sugerida: Peluches."

Esto se puede usar para:

- Enriquecer descripciones.
- Asistir en categorización.
- Ofrecer interacción creativa con el usuario.

## 3. Funciones Asistidas por IA en la Consola (gestionado por `MainApp` y `InterfazConsola`):

Ampliar las funcionalidades del sistema CRUD con opciones avanzadas que aprovechan un modelo de lenguaje (LLM) a través de la API de `OpenRouter.ai`, para ofrecer dos servicios:

1. **Generación de descripciones de productos.**
2. **Sugerencia de categorías basadas en el nombre del producto.**

Esto transforma una app de consola tradicional en una herramienta **más inteligente, dinámica y moderna.**

En la clase MainApp.java, el menú principal se amplía con dos nuevas opciones:

**6. Generar descripción de producto con IA**

**7. Sugerir categoría con IA**

Detalle de la Opción 6 – Descripción con IA

#### **Propósito**

Ofrecer una **descripción creativa y atractiva** de marketing para un producto existente usando IA.

#### **Flujo completo:**

1. InterfazConsola solicita al usuario el **ID del producto** deseado.
2. MainApp usa ProductoDAO para buscar ese producto.
3. Si existe, se **construye un prompt personalizado** para el modelo LLM, como:

“Genera una descripción de marketing breve y atractiva para el producto otaku: *Figura de Anya Forger* de la categoría *Figura*.”

4. Se llama al método obtenerRespuesta(prompt) de LlmService.
5. La respuesta generada se muestra usando InterfazConsola.

#### **¿Por qué así?**

- La IA permite a los usuarios no redactar manualmente descripciones de productos.
- Mejora la experiencia de venta (si el sistema se integrara a un e-commerce).
- El ID garantiza que se obtiene el producto correcto.

Detalle de la Opción 7 – Sugerencia de Categoría

#### **Propósito**

Asistir al usuario al momento de agregar un nuevo producto, sugiriendo una **categoría lógica** basada en el nombre del producto, usando IA.

#### **Flujo completo:**

1. InterfazConsola solicita al usuario el **nombre de un nuevo producto**.
2. MainApp construye el siguiente prompt:



“Para un producto otaku llamado '[nombre]', sugiere una categoría adecuada de esta lista: Figura, Manga, Póster, Llavero, Ropa, Videojuego, Otro.”

3. Se envía el prompt al LlmService.
4. Se recibe la sugerencia y se muestra al usuario.

#### 🔍 ¿Por qué así?

- Ayuda a los usuarios indecisos o sin experiencia a elegir categorías coherentes.
- El LLM tiene conocimiento semántico de términos otaku y puede hacer inferencias contextuales.
- La lista cerrada de categorías asegura consistencia.

===== Akihabara Market - Menú Principal =====

1. Ver todos los productos
2. Buscar producto por ID
3. Agregar nuevo producto
4. Actualizar producto
5. Eliminar producto
6. Generar descripción de producto con IA
7. Sugerir categoría para producto con IA
8. Salir

1

--- Lista de Productos ---

ID: 1

Nombre: Poster Manga

Categoría: Manga

Precio: \$13.99

Stock: 2

-----

ID: 2

Nombre: Poster

Categoría: Manga

Precio: \$12.0

Stock: 12

-----

ID: 3

Nombre: Naruto

Categoría: Figura

Precio: \$23.0

Stock: 34

-----

===== Akihabara Market - Menú Principal =====

1. Ver todos los productos
2. Buscar producto por ID
3. Agregar nuevo producto
4. Actualizar producto
5. Eliminar producto
6. Generar descripción de producto con IA
7. Sugerir categoría para producto con IA
8. Salir

2

Ingrese el ID del producto: 3

ID: 3

Nombre: Naruto

Categoría: Figura

Precio: \$23.0

Stock: 34

-----

```
===== Akihabara Market - Menú Principal =====
1. Ver todos los productos
2. Buscar producto por ID
3. Agregar nuevo producto
4. Actualizar producto
5. Eliminar producto
6. Generar descripción de producto con IA
7. Sugerir categoría para producto con IA
8. Salir
3
Nombre del producto: Fernando
Categoría (Figura, Manga, Póster, Llavero, Ropa): Figura
Precio: 23.00
Stock: 23
Producto agregado con éxito.
[INFO] Producto añadido correctamente.

===== Akihabara Market - Menú Principal =====
1. Ver todos los productos
2. Buscar producto por ID
3. Agregar nuevo producto
4. Actualizar producto
5. Eliminar producto
6. Generar descripción de producto con IA
7. Sugerir categoría para producto con IA
8. Salir
1

--- Lista de Productos ---
ID: 1
Nombre: Poster Manga
Categoría: Manga
Precio: $13.99
Stock: 2
-----
ID: 2
Nombre: Poster
Categoría: Manga
Precio: $12.0
Stock: 12
-----
ID: 3
Nombre: Naruto
Categoría: Figura
Precio: $23.0
Stock: 34
-----
ID: 4
Nombre: Fernando
Categoría: Figura
Precio: $23.0
Stock: 23
-----
```

```

-----

===== Akihabara Market - Menú Principal =====
1. Ver todos los productos
2. Buscar producto por ID
3. Agregar nuevo producto
4. Actualizar producto
5. Eliminar producto
6. Generar descripción de producto con IA
7. Sugerir categoría para producto con IA
8. Salir
5
Ingrese el ID del producto: 4
[INFO] Producto eliminado.

===== Akihabara Market - Menú Principal =====
1. Ver todos los productos
2. Buscar producto por ID
3. Agregar nuevo producto
4. Actualizar producto
5. Eliminar producto
6. Generar descripción de producto con IA
7. Sugerir categoría para producto con IA
8. Salir
1

--- Lista de Productos ---
ID: 1
Nombre: Poster Manga
Categoría: Manga
Precio: $13.99
Stock: 2

-----
ID: 2
Nombre: Poster
Categoría: Manga
Precio: $12.0
Stock: 12

-----
ID: 3
Nombre: Naruto
Categoría: Figura
Precio: $23.0
Stock: 34

-----

```

```

===== Akihabara Market - Menú Principal =====
1. Ver todos los productos
2. Buscar producto por ID
3. Agregar nuevo producto
4. Actualizar producto
5. Eliminar producto
6. Generar descripción de producto con IA
7. Sugerir categoría para producto con IA
8. Salir
Introduce el ID del producto para generar su descripción con IA: 3
--- Descripción Generada por IA ---
¡Somos tu tienda oficial de votos, donde encontrarás la figura del ansiado personaje Naruto en su forma easy-mode de la serie clásica Naruto! Captura la pasión del universo Naruto en esta estatua detallada, iluminada por un fino acabado láctico. ¡Además,
-----

===== Akihabara Market - Menú Principal =====
1. Ver todos los productos
2. Buscar producto por ID
3. Agregar nuevo producto
4. Actualizar producto
5. Eliminar producto
6. Generar descripción de producto con IA
7. Sugerir categoría para producto con IA
8. Salir
Introduce el nombre del producto para sugerir categoría con IA: 3
--- Descripción Generada por IA ---
Categoría sugerida: Figura
-----

===== Akihabara Market - Menú Principal =====
1. Ver todos los productos
2. Buscar producto por ID
3. Agregar nuevo producto
4. Actualizar producto
5. Eliminar producto
6. Generar descripción de producto con IA
7. Sugerir categoría para producto con IA
8. Salir
[INFO] ¡Mesa pronto, Maestro Kenji!

```

## BONUS QUEST: Interfaz Gráfica con Swing

Ejecutar MainGUIApp para lanzar la interfaz Grafica:

### 1. Separación de responsabilidades (MVC-like)

 *modelo*

- **ProductoOtaku**: clase que representa un producto (POJO).
- **ProductoDAO**: clase que maneja el acceso a los datos (simula una base de datos, usa listas o puede acceder a archivos/bd).

### 2. Adaptación del servicio de IA con OpenRouter

 *servicio/LlmService.java*

- Se implementó un servicio para conectar con un modelo LLM vía API (OpenRouter).
- Se usa para dos funciones:
  - Generar descripciones de marketing.
  - Sugerir categorías para nuevos productos.

### 3. Diseño de la Interfaz Gráfica (GUI)

 *gui/AkihabaraGUI.java*

- Se creó una clase AkihabaraGUI que **extiende JFrame**.
- Componentes principales:
  - JTable para mostrar productos.
  - JButton para CRUD e interacción IA.
  - JTextField para entradas de usuario.
  - JOptionPane para formularios y alertas.
- Cada botón tiene su ActionListener que llama a los métodos del DAO y LLM.

### 4. Inicialización del sistema con Swing

 *app/MainGUIApp.java*

```
public class MainGUIApp {  
    public static void main(String[] args) {  
        SwingUtilities.invokeLater(() -> {  
            new AkihabaraGUI();  
        });  
    }  
}
```

### 5. Desactivación o reemplazo de la versión de consola

❓ MainApp y InterfazConsola se mantienen solo si quieres conservar la versión CLI.

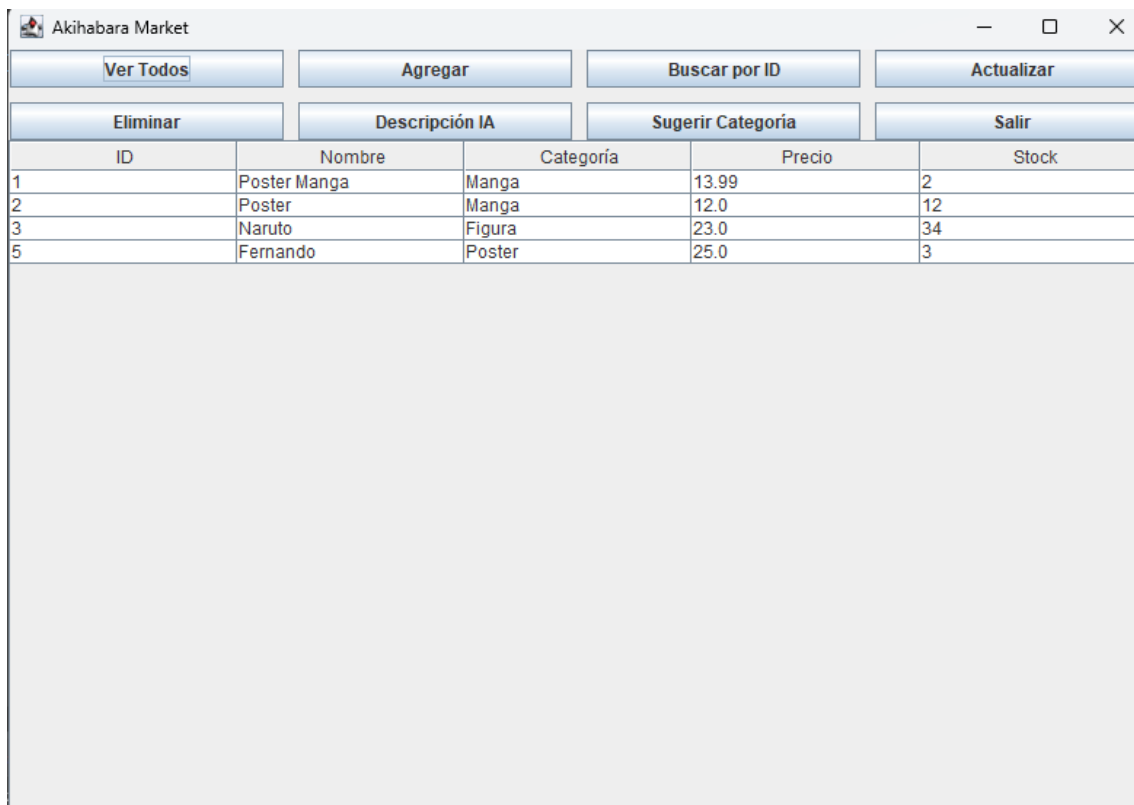
❓ Ahora la interfaz completa la maneja AkihabaraGUI.

## 🔄 FUNCIONALIDADES MIGRADAS (de consola → Swing)

Funcionalidad	Consola (MainApp)	Swing (AkihabaraGUI)
Ver todos los productos	mostrarTodosLosProductos()	cargarProductos()
Agregar producto	agregarProducto()	mostrarFormularioAgregar()
Consultar por ID	consultarProductoPorId()	buscarPorId()
Actualizar producto	actualizarProducto()	actualizarProducto()
Eliminar producto	eliminarProducto()	eliminarProducto()
Generar descripción con IA	generarDescripcionConIA()	generarDescripcionIA()
Sugerir categoría con IA	sugerirCategoriaConIA()	sugerirCategorialA()

Has completado con éxito la transformación de **una app CLI a una app GUI profesional** en Java con Swing, integrando:

- Interfaz de usuario moderna.
- Persistencia de datos modular (DAO).
- Inteligencia Artificial (via LLM).
- Buenas prácticas de arquitectura (MVC-like).
- Separación de responsabilidades.



Akihabara Market

Ver Todos

Agregar

Buscar por ID

Actualizar

Eliminar

Descripción IA

Sugerir Categoría

Salir

ID	Nombre	Categoría	Precio	Stock
1	Poster Manga	Manga	13.99	2
2	Poster	Manga	12.0	12
3	Naruto	Figura	23.0	34
5	Fernando	Poster	25.0	3

Agregar Producto

?

Nombre:

Categoría:

Precio:

Stock:

Aceptar

Cancelar

Akihabara Market

Ver Todos

Agregar

Buscar por ID

Actualizar

Eliminar

Descripción IA

Sugerir Categoría

Salir

ID	Nombre	Categoría	Precio	Stock
1	Poster Manga	Manga	13.99	2
2	Poster	Manga	12.0	12
3	Naruto	Figura	23.0	34
5	Fernando	Poster	25.0	3

Entrada

?

ID del producto a eliminar:

Aceptar

Cancelar

Mensaje

?

Descripción IA:

Título: Experimenta el espíritu del manga con nuestra magnífica colección de posters!

¿Buscas algo exclusivo para adornar tu habitación? Nuestra remarcable colección de posters manga es la que estarías buscando. Disfruta de una alta calidad en tu experiencia olfativa, visible y tangible con estos posters espectaculares.

Nuestras obras maestras de arte son graduadas de Kirakira, cada uno con una impresión detallada y vibrante del mundo manga de tus sueños. Nuestras elecciones de manga más populares no dejarán indiferente a ningún fanático de manga.

¡Adorná tu habitación con estos posters contemporáneos inspirados en los mangas más imprescindibles para crear un ambiente único que te hará sentir inmerso en tu mundo manga favorito! ¡Te lo garantizamos!

Sóloamos con brindarte la oportunidad de cerrar la brecha entre tu espacio físico y tu espacio mental en el que se desenvuelve la magia manga. Adquiere antes tu poster de manga preferido antes de que se acabe esta oferta especial! Ordena ahora y experimenta una verdadera transformación en tu hogar!

Aceptar

Mensaje

?

Sugerencia IA:

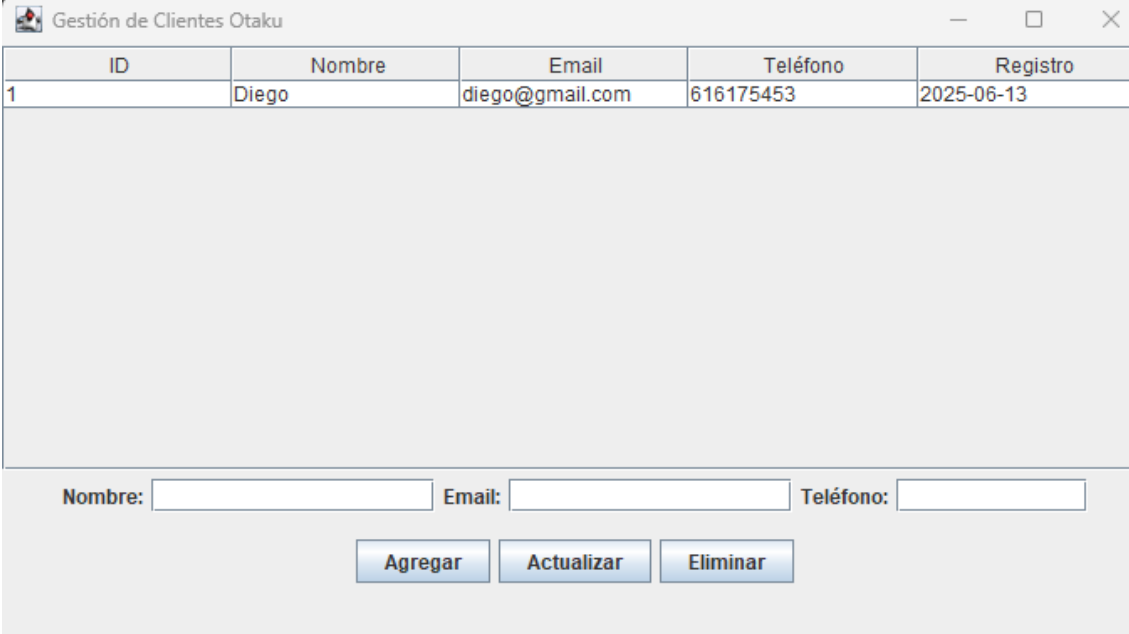
La categoría adecuada para el producto otaku llamado "I" podría ser "Figura", ya que figuras son una popular opción de colección entre los aficionados al manga y anime, y posiblemente se refiere a una figurita o escultura representativa de un personaje del manga o anime. Sin embargo, es importante tener

Aceptar



## BONUS QUEST II: Gestión de clientes

Ejecutar TestClientesGUI para lanzar la interfaz gráfica:



ID	Nombre	Email	Teléfono	Registro
1	Diego	diego@gmail.com	616175453	2025-06-13

Nombre:  Email:  Teléfono:

Enlace a GitHub: <https://github.com/DiegoCuadrado/PROGRAMACION.git>