Trabajo Final Integrador
Programación II

Tecnicatura Universitaria en Programación
UTN FR San Nicolás

Integrantes: Grupo 125 Producto-CodigoBarras
- Camila Carrión (Comisión 7)
- Pablo Osvaldo Cozzi (Comisión 3)

Año: 2025

1 INTRODUCCIÓN

El presente Trabajo Final Integrador corresponde a la asignatura Programación II y tiene como objetivo desarrollar un sistema en Java basado en arquitectura en capas, aplicando los conceptos de POO, persistencia de datos mediante JDBC y buenas prácticas de diseño.

El sistema implementado consiste en la gestión de productos y sus códigos de barras, manteniendo una relación 1:1 entre ambas entidades. Para ello, se utilizaron clases de dominio, interfaces DAO, implementaciones concretas, servicios con lógica de negocio y un menú interactivo que permite realizar las operaciones CRUD completas.

Durante el desarrollo se aplicaron principios fundamentales como encapsulamiento, separación de responsabilidades, uso de interfaces, manejo de excepciones y acceso seguro a la base de datos a través de consultas parametrizadas. La estructura final organiza el proyecto en paquetes diferenciados (entities, dao, service y main), siguiendo un modelo escalable y mantenible. Este trabajo permite consolidar de manera práctica los contenidos de Programación II, mostrando el funcionamiento completo de un sistema real que conecta Java con MySQL.

El sistema desarrollado tiene como finalidad implementar un módulo completo de gestión de productos utilizando Java y JDBC, demostrando la correcta aplicación de programación orientada a objetos y arquitectura en capas.

Objetivos del proyecto:

- Desarrollar una aplicación funcional en Java que permita administrar productos y sus códigos de barras asociados.
- Aplicar el patrón DAO para separar la lógica de acceso a datos del resto de la aplicación.
- Utilizar una arquitectura modular, escalable y fácil de mantener.
- Implementar clases de dominio que representen correctamente las entidades del sistema (Producto y CodigoBarras).
- Crear interfaces DAO e implementaciones concretas con consultas preparadas, seguras y conectadas a MySQL.
- Definir una capa de servicios que gestione la lógica de negocio, validaciones y reglas.
- Construir un menú por consola que permita realizar las operaciones CRUD: Crear productos con su código de barras, listar los productos almacenados, buscar productos por nombre y realizar una eliminación lógica.
- Validar excepciones, errores de conexión, datos vacíos o inconsistentes.
- Ejecutar la aplicación desde consola, demostrando su funcionamiento paso a paso.

2. MARCO TEÓRICO

El desarrollo del sistema se fundamenta en conceptos centrales de la Programación Orientada a Objetos (POO), el uso de JDBC para la persistencia de datos y la aplicación de una arquitectura en capas. Estos elementos permiten construir aplicaciones robustas, modulares y fáciles de mantener. A continuación, se explican los pilares teóricos que guiaron el diseño del proyecto.

2.1 Programación Orientada a Objetos aplicada al sistema

La POO permite modelar elementos del mundo real mediante clases y objetos. En este sistema, las entidades principales son Producto y CodigoBarras, cada una representada por una clase con atributos, constructores y métodos de acceso. Los principios aplicados fueron:

Encapsulamiento: Todos los atributos se declaran privados, y se exponen mediante getters y setters. Esto garantiza control sobre los valores asignados y evita modificaciones indebidas.

Relaciones entre objetos: Se aplicó una relación uno a uno (1:1), donde cada Producto posee un código de barras único, y cada código pertenece a un único producto. Este vínculo se refleja tanto a nivel lógico (en Java) como físico (en la base de datos).

Constructores y estados válidos: Los constructores aseguran que los objetos puedan ser creados con valores iniciales coherentes. El método toString() facilita la visualización de información durante el listado y la depuración.

2.2 Arquitectura en capas

Para evitar mezclar lógica visual, lógica de negocio y acceso a datos, el sistema se diseñó utilizando una arquitectura en capas, un enfoque ampliamente utilizado en aplicaciones empresariales. Las capas implementadas fueron:

- Capa de Entidades: Contiene los objetos que representan el dominio del problema (Producto, CodigoBarras). Su función es únicamente almacenar datos y definir relaciones entre clases.
- Capa DAO (Data Access Object): Se encarga exclusivamente de la comunicación con la base de datos. Aquí se encuentran interfaces DAO con las operaciones CRUD, implementaciones concretas que utilizan JDBC, consultas SQL seguras mediante PreparedStatement y el archivo de configuración de conexión. Esta capa permite cambiar la base de datos o la forma de persistencia sin modificar el resto del sistema.
- Capa de Servicios (Service Layer): Implementa las reglas y validaciones del negocio, como control de precios, configuración de baja lógica, verificación de datos obligatorios, etc. Esta capa evita que la lógica de negocio quede mezclada con SQL o con la interfaz de usuario.
- Capa de Interfaz (UI por consola): Es la capa externa. Presenta un menú simple que recibe entradas del usuario y muestra información por pantalla. No contiene lógica compleja; su único rol es interactuar con la persona y delegar la ejecución en los servicios.

2.3 Patrón DAO (Data Access Object)

El patrón DAO es fundamental para separar las responsabilidades de acceso a datos del resto de la aplicación. En este proyecto se implementó mediante: una interfaz genérica GenericDao<T>, DAOs específicos (ProductoDao, CodigoBarrasDao), clases que implementan cada método con SQL real.

Este enfoque ofrece tres ventajas principales: Independencia entre Java y SQL, reutilización del código y facilidad de prueba.

2.4 JDBC y consultas seguras

JDBC (Java Database Connectivity) permite conectar Java con bases de datos relacionales. En este trabajo se aplicaron los componentes esenciales:

Connection: Establece la conexión con el motor MySQL. Las credenciales (URL, usuario, contraseña) se manejan desde DataBaseConfig.

PreparedStatement: Permite ejecutar consultas parametrizadas, evitando concatenación de datos. Esto es fundamental para prevenir ataques de inyección SQL.

2.5 Validaciones, reglas de negocio y persistencia

El sistema incorpora reglas claves antes de permitir operaciones:

- El precio debe ser un valor positivo.
- El stock no puede ser negativo.
- El código de barras no puede repetirse.
- Un producto eliminado no se borra físicamente: se marca como eliminado mediante un campo booleano (baja lógica), pero permanecen en la base de datos, evitando pérdidas de información.
- No se crean productos sin un código asociado.

Estas validaciones se ejecutan en la capa de servicios, mientras que la persistencia final se delega al DAO. De este modo, se garantizan datos coherentes, consistencia entre la base y la lógica de negocio, una separación clara entre responsabilidades.

2.6 Operaciones CRUD

El sistema implementa operaciones CRUD, un conjunto de acciones fundamentales en el desarrollo de aplicaciones orientadas a datos. CRUD corresponde a Create (crear), Read (leer o consultar), Update (actualizar) y Delete (eliminar). En este trabajo, dichas operaciones se aplican tanto a productos como a sus códigos de barras, permitiendo insertar nuevos registros, listarlos, buscarlos, actualizarlos y gestionar su eliminación de manera controlada. Este conjunto de funcionalidades constituye la base de cualquier sistema de gestión y refleja la estructura clásica utilizada en aplicaciones empresariales.

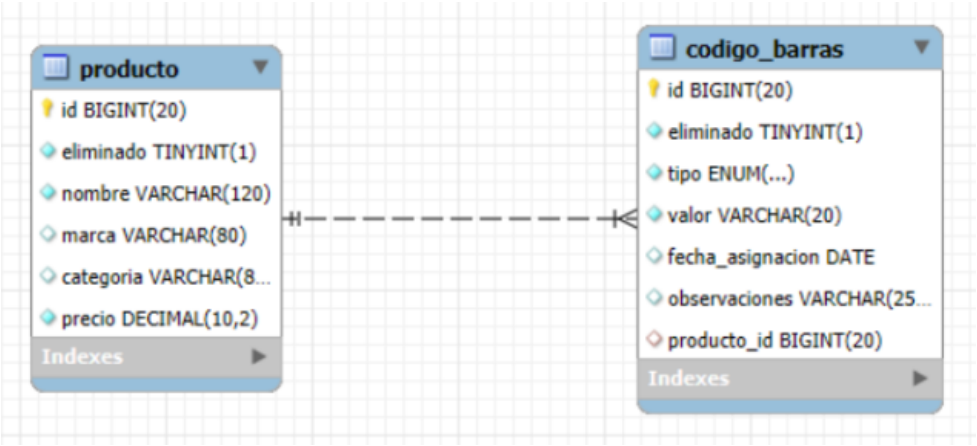
3. DISEÑO DEL SISTEMA

El desarrollo del sistema se basó en una serie de decisiones de diseño que permitieron obtener una aplicación modular, coherente con los principios de POO y ajustada a las buenas prácticas de arquitectura en capas. A continuación, se detallan las principales decisiones y criterios que guiaron la construcción del proyecto, desde la definición del modelo de datos hasta la implementación final de las capas.

3.1 Modelo de datos y relación entre entidades

El núcleo del sistema está compuesto por dos entidades principales: Producto y CodigoBarras. La relación entre ambas se definió como uno a uno (1:1), ya que cada producto posee exactamente un código de barras único, y cada código pertenece a un solo producto. Esta vinculación se reflejó en Java incorporando un atributo de tipo CodigoBarras dentro de la clase Producto, de manera que el vínculo se mantenga tanto en memoria como en la base de datos.

A partir del diseño lógico y físico implementado en la base de datos tp_bdd, se generó el diagrama entidad–relación (DER) mediante la herramienta Reverse Engineer de MySQL Workbench.



Este diagrama representa de forma gráfica las entidades principales, sus atributos y las relaciones que existen entre ellas, permitiendo visualizar la estructura interna del modelo relacional desarrollado en esta primera etapa del trabajo.

3.2 Uso de interfaces, abstracciones y polimorfismo

Una de las decisiones más importantes fue introducir interfaces genéricas y específicas para describir las operaciones que debían implementarse en la capa DAO.

- GenericDao<T> define las operaciones CRUD comunes.
- ProductoDao y CodigoBarrasDao extienden esa estructura agregando búsquedas particulares del dominio.

Este enfoque permitió reducir la repetición de código, asegurando un contrato claro entre capas y asegurando futuras ampliaciones sin romper la arquitectura. El sistema puede agregar nuevas entidades sin alterar las capas superiores, siendo un diseño flexible.

3.3 Separación de responsabilidades entre capas

La aplicación sigue el principio SRP (Single Responsibility Principle), dividiendo la lógica de acuerdo a la función de cada capa:

Capa de Entidades (entities): Contiene clases simples con atributos y métodos de acceso. No ejecutan lógica, no conocen SQL, no validan datos.

Capa DAO (dao): Responsable exclusivamente del acceso a la base de datos. Aquí se ejecutan consultas SQL mediante JDBC y se mapean resultados a objetos Java.

Capa de Servicios (service): Implementa todas las reglas de negocio y validaciones: evitar precios negativos, evitar códigos duplicados, impedir creación de productos sin código asociado, ejecutar la eliminación lógica y controlar excepciones antes de llegar al usuario. Garantizando que la aplicación funcione incluso cargando datos incorrectos.

Capa de Interfaz (main / UI por consola): Recibe entradas desde teclado y muestra resultados. No contiene lógica compleja ni consultas SQL; solo delega en los servicios.

3.4 Manejo de errores y validaciones

El diseño incluye manejo de excepciones tanto en la capa DAO (errores de SQL) como en la capa servicio (errores lógicos). Las principales decisiones fueron:

- Capturar y lanzar excepciones claras hacia las capas superiores.
- Validar campos obligatorios antes de ejecutar consultas.
- Utilizar PreparedStatement para evitar SQL injection.
- Mostrar mensajes específicos al usuario sin revelar información interna del sistema.

Este enfoque asegura robustez y un comportamiento predecible ante datos inválidos, inconsistencias o interrupciones de conexión.

3.5 Diseño del menú y experiencia del usuario

El menú fue diseñado para ser simple, directo y completamente funcional desde consola. Las opciones incluidas responden a las operaciones CRUD más utilizadas, como crear producto con su código de barras, listar productos, buscar por nombre, eliminar producto (baja lógica) y salir del sistema.

La interfaz guía al usuario paso a paso, solicitando valores necesarios y mostrando mensajes claros ante errores o confirmaciones exitosas.

3.6 Conexión a la base de datos y configuración independiente

Para evitar mezclar valores sensibles en el código, se creó la clase DataBaseConfig, encargada de almacenar URL, usuario y contraseña, administrar la conexión mediante DriverManager y centralizar cambios futuros sin alterar otras capas. Esta decisión permite que si la base cambia de servidor o si se migra a otro motor de SQL, la aplicación pueda adaptarse rápidamente modificando una única clase.

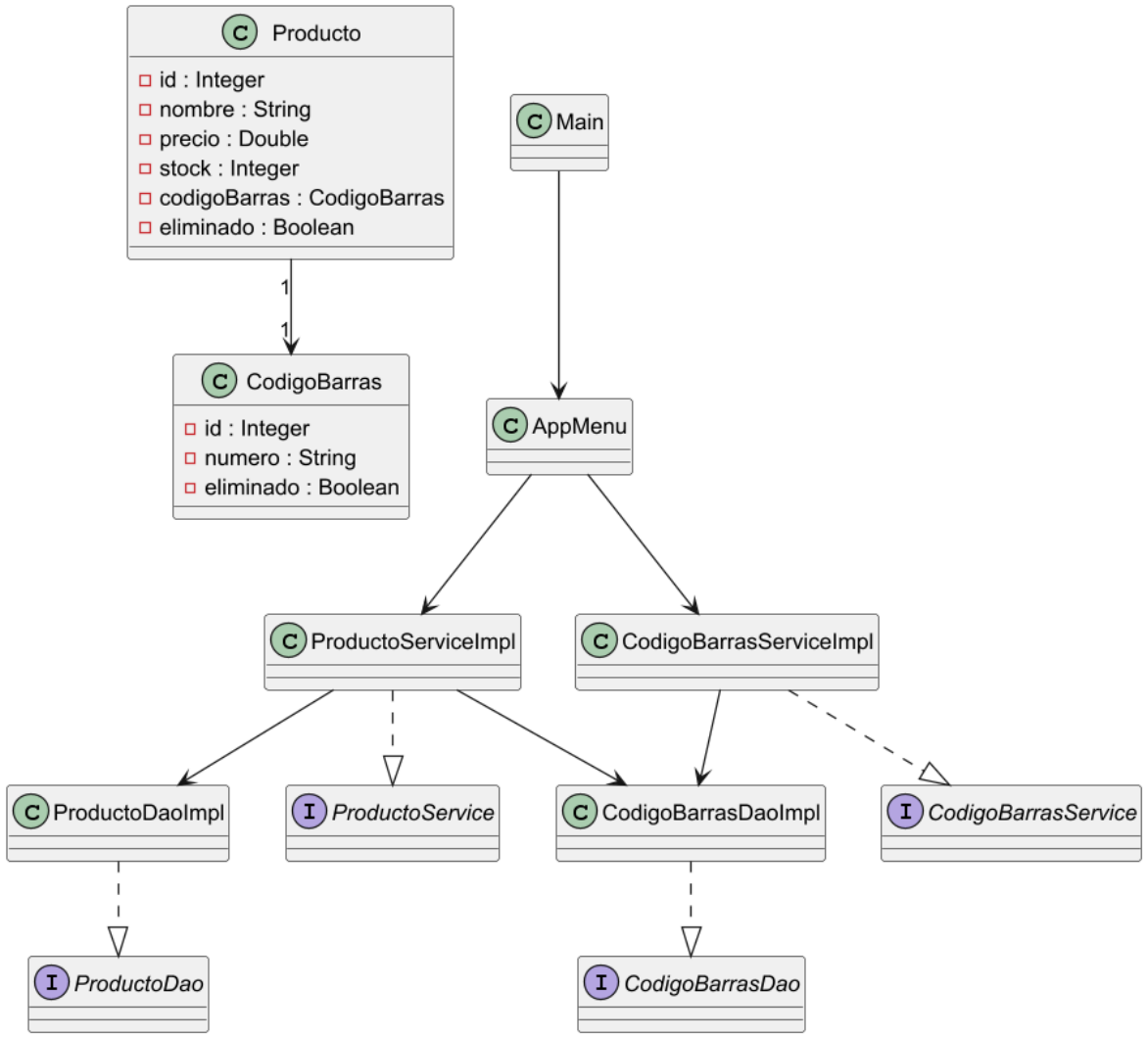
3.7 Consideraciones de seguridad y buenas prácticas

En la implementación se aplicaron prácticas fundamentales, como el uso obligatorio de PreparedStatement, la prohibición de concatenación directa de datos, el cierre automático de conexiones con try-with-resources, la validaciones en servicios antes de acceder al DAO y evitar objetos nulos mediante verificaciones previas.

3.8 Diagrama de Clases del Sistema

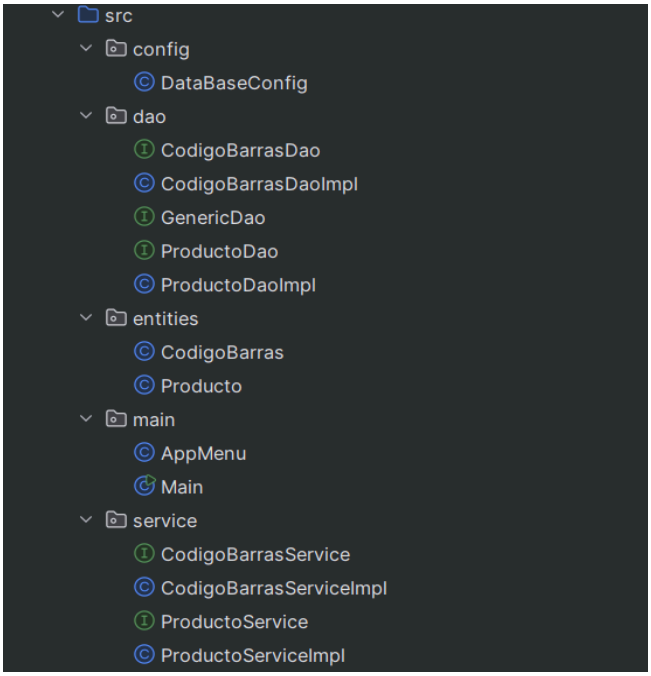
A continuación se presenta el Diagrama de Clases correspondiente al sistema implementado. El modelo refleja la arquitectura en capas definida para el proyecto, destacando la separación entre entidades, servicios, capa DAO y la interfaz de usuario por consola.

El diagrama muestra la relación 1:1 entre Producto y CodigoBarras, las dependencias entre los servicios y los objetos DAO, así como la interacción de la clase AppMenu con la lógica de negocio. Cada clase incluye sus atributos principales, métodos relevantes y las interfaces que implementa, permitiendo visualizar de manera clara la estructura interna del sistema y cómo se comunican sus componentes.



4. IMPLEMENTACIÓN DEL SISTEMA

La implementación se llevó a cabo respetando la arquitectura en capas definida previamente. En esta sección se describe cómo se construyó cada una de las capas y cómo interactúan entre sí para conformar un sistema funcional de gestión de productos.



4.1 Capa de Configuración (config)

Para conectar la aplicación Java con MySQL se creó la clase DataBaseConfig, responsable de: almacenar los parámetros de conexión, registrar el driver JDBC, abrir la conexión mediante DriverManager y centralizar cualquier modificación futura de la configuración.

4.2 Capa DAO (Acceso a Datos)

Esta capa implementa todo el acceso a MySQL utilizando JDBC, un DAO con 3 componentes:

- GenericDao: Define la estructura común de CRUD, siendo insertar, actualizar, eliminar (baja lógica), obtenerPorId y obtenerTodos.
- DAOs específicos: ProductoDao y CodigoBarrasDao. Incluyen operaciones propias del dominio.
- Implementaciones concretas (DAOImpl): Se ejecutan las consultas SQL con PreparedStatement. La clave es usar consultas parametrizadas, cerrar conexiones automáticamente con try-with-resources, mapear cada fila de la BD a un objeto Java.

4.3 Capa de Entidades (entities)

Las entidades modelan los objetos principales del dominio: Producto y CodigoBarras. Ambas clases son simples, sin lógica adicional, respetando el principio de responsabilidad única.

- Producto: Representa un artículo del inventario y contiene atributos básicos (nombre, precio, stock), la relación 1:1 con CodigoBarras, la marca de baja lógica (eliminado) y métodos getter/setter y toString() para su visualización.
- CodigoBarras: Contiene identificador único, código y la marca de eliminado.

4.4 Capa de Interfaz (AppMenu)

Se implementó una interfaz por consola simple pero funcional, con un menú que ofrece crear producto con código de barras, listar productos, buscar productos por nombre, eliminar producto (baja lógica) y salir del sistema. El usuario puede ejecutar desde la consola de IntelliJ o desde PowerShell.

4.5 Clase principal (Main)

El entry point se encarga únicamente de iniciar la ejecución del menú.

4.6 Capa de Servicios (service)

La capa de servicios actúa como puente entre la interfaz de usuario y los DAOs. Aquí se ubicó la lógica de negocio, las validaciones y los mensajes significativos. Las funciones principales son: controlar que el producto tenga nombre válido, precio positivo y stock coherente, garantizar que cada código de barras se cree antes que el producto, prevenir operaciones sobre IDs inexistentes y manejar excepciones y traducirlas en mensajes claros para el usuario. Esta capa asegura que el sistema funcione aunque el usuario ingrese datos incompletos o erróneos.

5. EJECUCIÓN DEL SISTEMA Y DEMOSTRACIÓN PRÁCTICA

En esta sección se documenta el funcionamiento real del sistema desarrollado. Esta parte del informe valida que el proyecto funciona en su totalidad y coincide con lo que se mostrará en el video. A continuación se describe cada funcionalidad, cómo se ejecuta y las capturas obligatorias.

5.1 Configuración previa y ejecución

Para ejecutar la aplicación, se debe tener la base de datos creada en MySQL (nombre sugerido: tp_programacion) y verificar que las tablas producto y codigo_barras estén vacías o inicializadas correctamente. Ejecutar el proyecto desde: IntelliJ IDEA o PowerShell.

5.2 Crear un producto (inserción con código de barras):

El usuario ingresa: nombre del producto, precio, stock y valor del código de barras. Esta operación crea primero el CódigoBarras, luego crear el Producto asociado y aplica validaciones (precio > 0, nombre no vacío).

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\J

SISTEMA DE GESTIÓN DE PRODUCTOS
1. Crear producto
2. Listar productos
3. Buscar producto por nombre
4. Eliminar producto
5. Salir
Elija una opcion: 1
Nombre: Short Nike
Precio: 28000
Stock: 9
Codigo de barras: 456789
Producto creado con exito.
```

Trabajando a la vez con sql, se muestra como se agrega un producto de prueba:

```
26 •    INSERT INTO productos (nombre, precio, stock, codigo_barras_id)
27      VALUES ('Producto test', 100, 5, 1);
```

5.3 Listar productos

Esta opción muestra todos los productos activos (que no fueron eliminados lógicamente). Cabe destacar que también lista los productos que fueron creados directamente desde sql.

```
SISTEMA DE GESTIÓN DE PRODUCTOS
1. Crear producto
2. Listar productos
3. Buscar producto por nombre
4. Eliminar producto
5. Salir
Elija una opcion: 2
Producto{id=1, nombre='Producto test', precio=100.0, stock=5, codigoBarras=12345, eliminado=false}
Producto{id=2, nombre='Short Nike', precio=28000.0, stock=9, codigoBarras=456789, eliminado=false}
Producto{id=3, nombre='Remera Adidas', precio=25000.0, stock=12, codigoBarras=963852, eliminado=false}
Producto{id=4, nombre='Medias Topper', precio=9000.0, stock=23, codigoBarras=741963, eliminado=false}
```

5.4 Buscar productos por nombre

Permite buscar productos por coincidencia parcial del nombre:

```
SISTEMA DE GESTIÓN DE PRODUCTOS
1. Crear producto
2. Listar productos
3. Buscar producto por nombre
4. Eliminar producto
5. Salir
Elija una opcion: 3
Ingrese nombre o parte del nombre: Nike
Producto{id=2, nombre='Short Nike', precio=28000.0, stock=9, codigoBarras=456789, eliminado=false}
```

5.5 Eliminación lógica de un producto

En este caso eliminamos el producto prueba creado desde sql:

```
SISTEMA DE GESTIÓN DE PRODUCTOS
1. Crear producto
2. Listar productos
3. Buscar producto por nombre
4. Eliminar producto
5. Salir
Elija una opcion: 4
Ingrese ID del producto a eliminar: 1
Producto eliminado correctamente.

SISTEMA DE GESTIÓN DE PRODUCTOS
1. Crear producto
2. Listar productos
3. Buscar producto por nombre
4. Eliminar producto
5. Salir
Elija una opcion: 2
Producto{id=2, nombre='Short Nike', precio=28000.0, stock=9, codigoBarras=456789, eliminado=false}
Producto{id=3, nombre='Remera Adidas', precio=25000.0, stock=12, codigoBarras=963852, eliminado=false}
Producto{id=4, nombre='Medias Topper', precio=9000.0, stock=23, codigoBarras=741963, eliminado=false}
```

5.6 Validaciones y manejo de errores

El sistema incorpora validaciones en la capa de servicio:

nombre vacío ⟶ error

```
SISTEMA DE GESTIÓN DE PRODUCTOS
1. Crear producto
2. Listar productos
3. Buscar producto por nombre
4. Eliminar producto
5. Salir
Elija una opcion: 1
Nombre:
El nombre no puede estar vacio.
```

precio ≤ 0 ⟶ error

```
SISTEMA DE GESTIÓN DE PRODUCTOS
1. Crear producto
2. Listar productos
3. Buscar producto por nombre
4. Eliminar producto
5. Salir
Elija una opcion: 1
Nombre: Remera
Precio: -2000
El precio debe ser un numero valido.
```

stock negativo → error

```
SISTEMA DE GESTIÓN DE PRODUCTOS
1. Crear producto
2. Listar productos
3. Buscar producto por nombre
4. Eliminar producto
5. Salir
Elija una opcion: 1
Nombre: Remera
Precio: 2000
Stock: -5
El stock debe ser un numero entero valido.
```

5.7 Resumen de ejecución

La ejecución integral del sistema permitió validar el correcto funcionamiento de todos los módulos implementados. En primer lugar, se comprobó que las operaciones CRUD para ambas entidades se ejecutan de manera consistente, persistiendo los datos en MySQL y reflejando los cambios en tiempo real. La relación 1:1 entre Producto y CodigoBarras se aplicó correctamente tanto a nivel lógico (clases Java) como a nivel físico (modelo relacional), verificándose la unicidad del código de barras y su vinculación directa con el producto asociado.

Asimismo, durante las pruebas se confirmó que las validaciones implementadas en la capa de servicios responden de forma adecuada ante entradas inválidas, mostrando mensajes de error, evitando la propagación de inconsistencias hacia la capa DAO o hacia la base de datos.

La eliminación lógica, mediante el atributo eliminado, funcionó de manera correcta: los productos marcados como inactivos dejan de visualizarse en listados y búsquedas, pero permanecen registrados en la base para auditorías futuras. Finalmente, se comprobó que el sistema mantiene su robustez ante errores de conexión, consultas no válidas y entradas inesperadas del usuario, garantizando estabilidad y un comportamiento predecible.

En conclusión, el sistema cumple con todos los requisitos establecidos en la consigna del Trabajo Final Integrador, demostrando un diseño modular, una arquitectura en capas correctamente aplicada y un funcionamiento confiable en cada una de las funcionalidades implementadas.

6. CONCLUSIÓN

El desarrollo de este sistema de gestión de productos permitió integrar de manera práctica los principales contenidos de Programación II, combinando POO, JDBC y arquitectura en capas dentro de una aplicación funcional y mantenible. A lo largo del trabajo se diseñaron e implementaron las entidades, los DAOs, los servicios y la interfaz de usuario, asegurando una correcta separación de responsabilidades y un flujo de interacción claro entre cada componente.

La conexión con MySQL y la aplicación del patrón DAO demostraron la importancia de trabajar con consultas seguras, estructuras escalables y un modelo de datos coherente. La relación uno a uno entre Producto y CodigoBarras se implementó de forma correcta, tanto en código como en la base de datos, garantizando integridad y consistencia en las operaciones realizadas.

En términos funcionales, todas las operaciones CRUD fueron ejecutadas y verificadas, incluyendo inserción de productos con su código, listados, búsquedas y eliminación lógica. Las validaciones aplicadas evitaron errores comunes y aseguraron la confiabilidad del sistema frente a datos inválidos o inconsistentes.

En conjunto, el trabajo permitió consolidar conocimientos teóricos y aplicarlos en un proyecto real, evidenciando un proceso completo que va desde el diseño conceptual hasta la implementación final y pruebas. El resultado es un sistema estable, organizado y preparado para futuras extensiones, cumpliendo plenamente con los objetivos planteados para este integrador.