Informe del Proyecto Final

Tecnicatura Universitaria en Programación a Distancia

Materia: Programación 2

Integrantes y Roles:

- **Pablo León** → Diseño UML + Entidades (Producto ,CodigoBarras).
- **Andrés Piuzzi** → DAOs + scripts SQL + conexión a base.
- **Pedro Antonio Sota Taier** → Servicios + lógica de transacciones (commit/rollback).
- **Franco Siri** → Menú CRUD + validaciones + integración final.

Índice

Introducción

Este informe describe el desarrollo del proyecto final, incluyendo diseño, arquitectura, persistencia, validaciones, pruebas y conclusiones. El objetivo es implementar un sistema de gestión de productos con códigos de barras.

Elección del dominio y justificación

Se eligió el dominio **Producto → Código de Barras** porque representa un caso real y frecuente en sistemas de gestión comercial. Esta relación permite aplicar de manera práctica los conceptos de **asociación 1→1 unidireccional**, donde cada producto tiene un único código de barras que lo identifica de forma inequívoca en inventarios y procesos de venta.

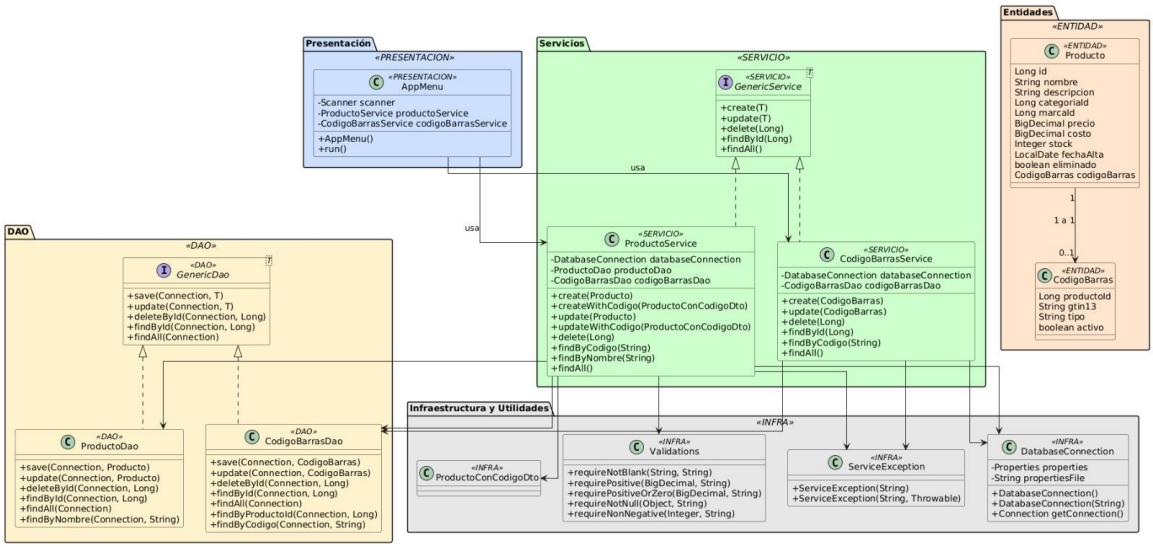
La elección se justifica por varias razones:

- **Relevancia en la industria**: La gestión de productos y códigos de barras es esencial en supermercados, e-commerce y logística.
- **Aplicación de reglas de negocio claras**: Garantizar la unicidad del código (GTIN) y la existencia de un solo código por producto son restricciones comunes en sistemas reales.
- **Escenario ideal para transacciones**: La creación de un producto y su código asociado requiere operaciones coordinadas, lo que permite demostrar el uso de **commit/rollback**.
- **Escalabilidad**: Este dominio puede evolucionar fácilmente para incluir más atributos (stock, proveedor, descuentos) o integrarse con otros módulos (ventas, facturación).
- **Facilidad para pruebas**: Permite realizar búsquedas por nombre o por código, validaciones de formato y pruebas de integridad referencial.

En resumen, este dominio combina **simplicidad conceptual** con **desafíos técnicos reales**, lo que lo convierte en una opción adecuada para aplicar los principios de arquitectura por capas, persistencia y manejo transaccional.

Diseño y UML

El diseño sigue una arquitectura en capas con separación de responsabilidades. Se optó por relaciones 1→1 entre Producto y Código de Barras, utilizando claves primarias compartidas para garantizar integridad referencial.



Arquitectura por capas

La aplicación está organizada en **cuatro capas principales**, cada una con responsabilidades bien definidas para garantizar separación de preocupaciones y facilidad de mantenimiento:

1.

Capa Entities (`entities/`)
Contiene las clases del dominio: `Producto` y `CodigoBarras`.

2.

1. **Relación 1→1 unidireccional**: `Producto` tiene una referencia a `CodigoBarras`, pero no al revés.
2. Incluye atributos de negocio (nombre, marca, categoría, precio) y el campo `eliminado` para bajas lógicas.
3. Se implementan constructores, getters/setters y `toString()` para facilitar la interacción con el menú.

3.

Capa DAO (`dao/`)
Encapsula el acceso a la base de datos usando **JDBC** y `PreparedStatement`.

4.

1. Cada DAO implementa operaciones CRUD: crear, leer por ID, listar, actualizar y eliminar (baja lógica).

2. Se incluyen búsquedas específicas (por nombre en `ProductoDao`, por valor en `CodigoBarrasDao`).
3. Los métodos aceptan una `Connection` externa para participar en transacciones coordinadas.

5.

Capa Service (`service/`)
Contiene la lógica de negocio y la orquestación de transacciones.

6.

1. Valida reglas como unicidad del GTIN y que cada producto tenga un solo código.
2. Controla las transacciones: desactiva auto-commit, ejecuta varias operaciones DAO y decide `commit()` o `rollback()` según el resultado.
3. Maneja excepciones y asegura que los recursos se cierren correctamente.

7.

Capa Presentación (`main/`)
Implementa el menú de consola (`AppMenu`) para la interacción con el usuario.

8.

1. Captura opciones y datos desde teclado.
2. Delegación: cada opción invoca métodos de la capa Service.
3. Muestra mensajes claros de éxito o error y maneja entradas inválidas.

Este diseño permite que cada capa cumpla una función específica: **Entities modela el negocio, DAO gestiona la persistencia, Service aplica reglas y transacciones, y AppMenu ofrece la interfaz de usuario**.

Persistencia

La persistencia se implementa utilizando **JDBC** y el **patrón DAO**, garantizando la separación entre la lógica de negocio y el acceso a datos. La base de datos está compuesta por dos tablas: **Producto** y **CodigoBarras**, relacionadas mediante una asociación **1→1 unidireccional**, donde la tabla `CodigoBarras` contiene una clave foránea única que referencia a `Producto`. Esto asegura que cada producto tenga un solo código de barras y mantiene la integridad referencial.

El flujo de las operaciones es el siguiente:

1. **Apertura de conexión**: Se obtiene una conexión desde la clase `DatabaseConnection` y se desactiva el auto-commit (`setAutoCommit(false)`).
2. **Ejecución de operaciones**:
    1. Se inserta primero el código de barras en su tabla.
    2. Luego se inserta el producto, asociándolo al código de barras mediante la FK.
    3. Ambas operaciones se realizan usando DAOs que aceptan la misma conexión externa para participar en la transacción.
3. **Commit o rollback**:

    1. Si todas las operaciones se ejecutan correctamente, se realiza `commit()`.
    2. Ante cualquier error (validación, SQL, integridad), se ejecuta `rollback()` para revertir los cambios y evitar inconsistencias.

4. **Cierre y limpieza**: Se restablece el auto-commit y se cierran los recursos (PreparedStatement y Connection).

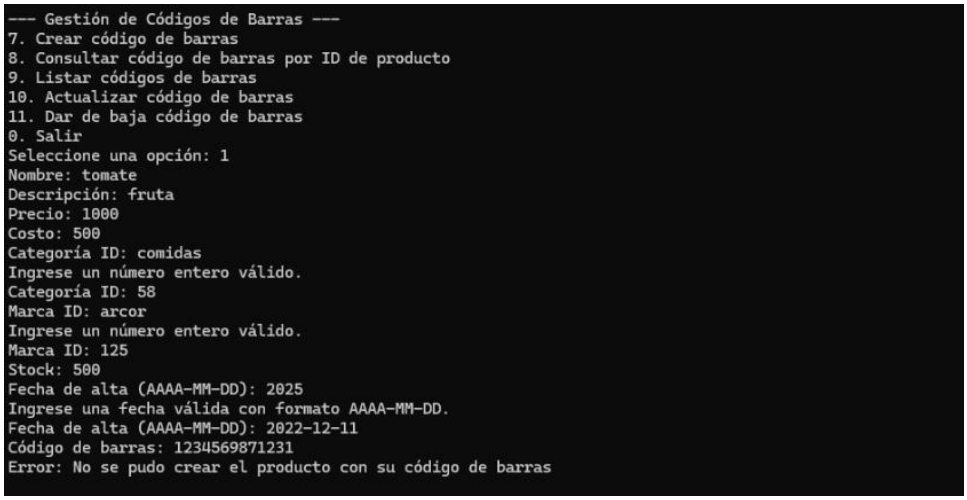
Este mecanismo asegura que la creación de un producto y su código de barras se realice de forma consistente, evitando registros incompletos. Además, se aplican validaciones previas (formato de fecha, unicidad, tipos de datos) para reducir errores antes de llegar a la base.

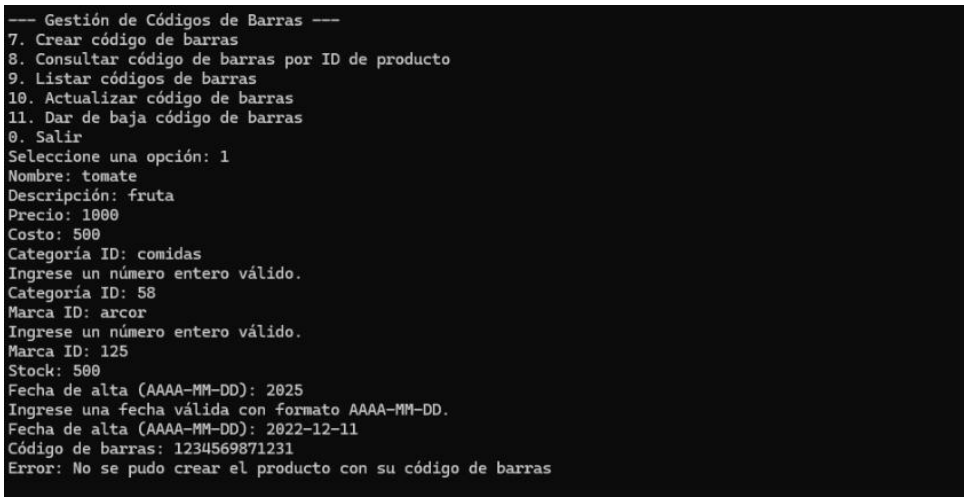
Validaciones y reglas de negocio

Se validan campos obligatorios, tipos de datos y formato de fecha. Ejemplo: al crear un producto, si el codigo de barra no comple el formato, se rechaza la operación. También se valida que IDs sean enteros.

Pruebas realizadas

Se realizaron pruebas funcionales del menú y creación de productos. A continuación, capturas del menú y error por validación:





Conclusiones y mejoras futuras

El desarrollo del sistema permitió aplicar los conceptos fundamentales de Programación Orientada a Objetos, persistencia con JDBC, y arquitectura por capas, logrando una solución funcional para la gestión de productos y códigos de barras con relación 1→1 unidireccional. Se implementaron validaciones, manejo de excepciones y transacciones para garantizar la consistencia de los datos.

El proyecto cumple con los objetivos planteados: CRUD completo, separación de responsabilidades, y control de errores mediante rollback. Además, se evidenció la importancia de diseñar correctamente la base de datos y la lógica de negocio para evitar inconsistencias y mejorar la experiencia del usuario.

Como futuras mejoras se podria:

- **Interfaz gráfica (GUI)**

Sustituir el menú de consola por una interfaz gráfica amigable, utilizando **JavaFX** o **Swing**, para mejorar la usabilidad.

- **Validaciones más robustas**

Implementar validaciones avanzadas (por ejemplo, expresiones regulares para formatos, rangos numéricos, fechas coherentes) y mensajes más claros para el usuario.

- **Gestión de excepciones centralizada**

Crear un manejador global de errores que permita registrar logs y mostrar mensajes uniformes.

- **Pruebas automatizadas**

Incorporar **JUnit** para pruebas unitarias y de integración, asegurando la calidad del código y reduciendo errores en futuras modificaciones.

- **Optimización de la persistencia**

Evaluar el uso de **pool de conexiones** para mejorar el rendimiento y considerar frameworks como **Hibernate** para reducir código repetitivo.

- **Escalabilidad y modularidad**

Preparar la arquitectura para soportar más entidades y relaciones, permitiendo que el sistema evolucione hacia un ERP básico.

- **Internacionalización y configuración externa**

Permitir que los textos y parámetros se gestionen desde archivos externos para facilitar la adaptación a distintos entornos.

Fuentes y herramientas utilizadas

Lenguaje: Java
Base de datos: MySQL
IDE: IntelliJ
Herramientas: GitHub, UML Designer
IA: Se utilizó para organización de tareas y apoyo en la redacción del informe.