UNIVERSIDAD TECNOLÓGICA NACIONAL

Trabajo Práctico Integrador

Materia:

Programación II

Tema elegido:

Producto–CódigoBarras

Alumno/os:

Noguera, Joana

Fulladoza, Pablo

Lauk, Karen

Profesor/a:

Ramiro Hualpa

N° de grupo:

120

Fecha de Entrega:

17 de Noviembre del 2025

Índice

Introducción

El presente Trabajo Práctico Integrador de la materia Programación II tiene como objetivo aplicar, en un proyecto completo, los principales conceptos trabajados durante el cuatrimestre: programación orientada a objetos, uso de colecciones, manejo de excepciones, interfaces, diseño por capas, UML y acceso a bases de datos mediante JDBC.

La consigna requiere desarrollar una aplicación que combine:

- Modelado UML
- Diseño del dominio
- Implementación de entidades
- Conexión a MySQL mediante JDBC
- Acceso a datos con DAOs
- Capa de servicios con reglas de negocio y transacciones
- Interfaz de usuario por consola

Para este proyecto se seleccionó el dominio **Producto – CódigoBarras**, una relación unidireccional 1→1 que modela un escenario realista y permite aplicar validaciones, integridad referencial y transacciones (ej.: crear producto + código juntos). Este dominio también se integra naturalmente con conceptos vistos en la materia de Bases de Datos.

Desarrollo

1. Diseño

En esta etapa se definió la estructura conceptual del proyecto y el dominio elegido.
Se estableció una relación unidireccional 1→1 entre Producto y CódigoBarras, que permite aplicar validaciones, integridad referencial y transacciones en la capa de servicio. Este diseño sirvió como base para el UML y la construcción de la base de datos.

Elección del dominio

El dominio Producto → Código de Barras fue seleccionado porque:

- Todo producto físico posee un código de barras único.
- La relación 1→1 es simple y realista.
- Permite implementar validaciones (EAN13, EAN8, UPC).
- Requiere unicidad y restricciones en la base.
- Facilita transacciones (alta conjunta de producto + código).
- Soporta baja lógica en ambas entidades.

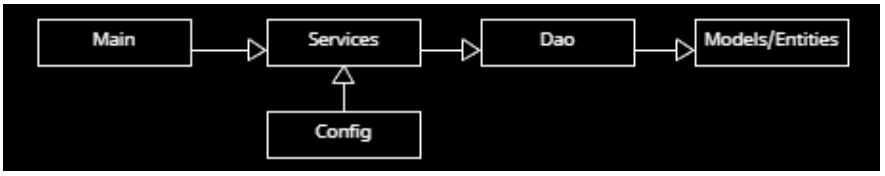
Diseño general

- La relación es unidireccional 1→1: Producto contiene un atributo CodigoBarras.
- La tabla codigo_barras posee producto_id como FK UNIQUE, garantizando la relación 1→1 desde la base.
- CódigoBarras no tiene referencia de vuelta a Producto.
- Ambas entidades incluyen eliminado para implementar baja lógica.

2. Diagrama UML

El UML representa la estructura del sistema y sirvió para verificar la coherencia antes de implementarlo. Incluye entidades, atributos y la relación 1→1, así como la arquitectura por paquetes.

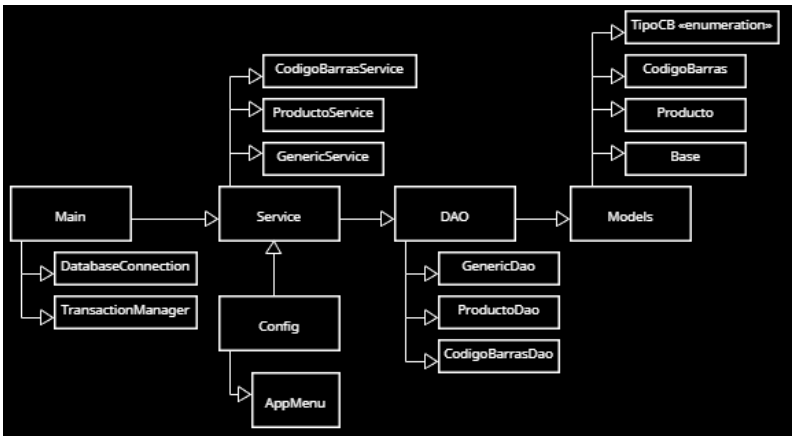
Diagrama UML de paquetes simplificado



Dependencia:

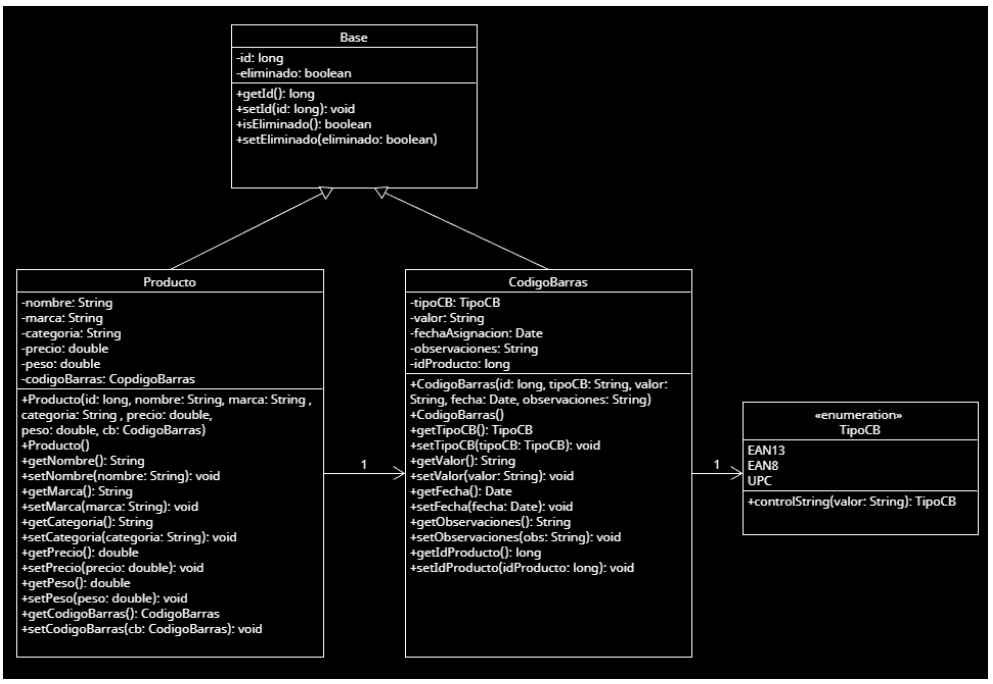
- ➔ Services usa Dao y Models.
- ➔ Dao usa Models y Config (para la conexión).
- ➔ Main usa Services.

UML completo

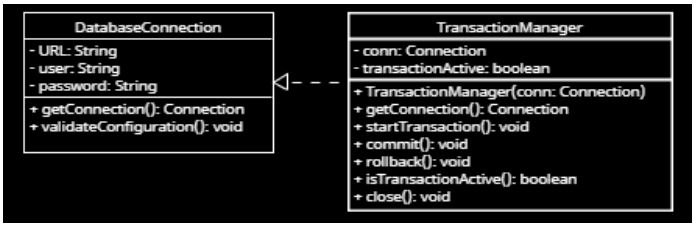


UML por paquetes

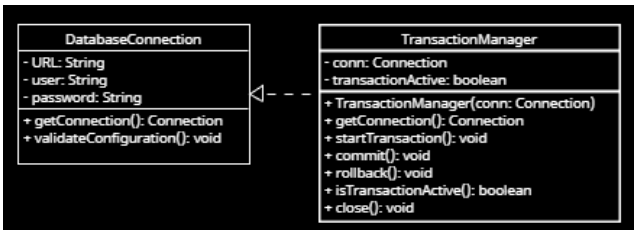
Models



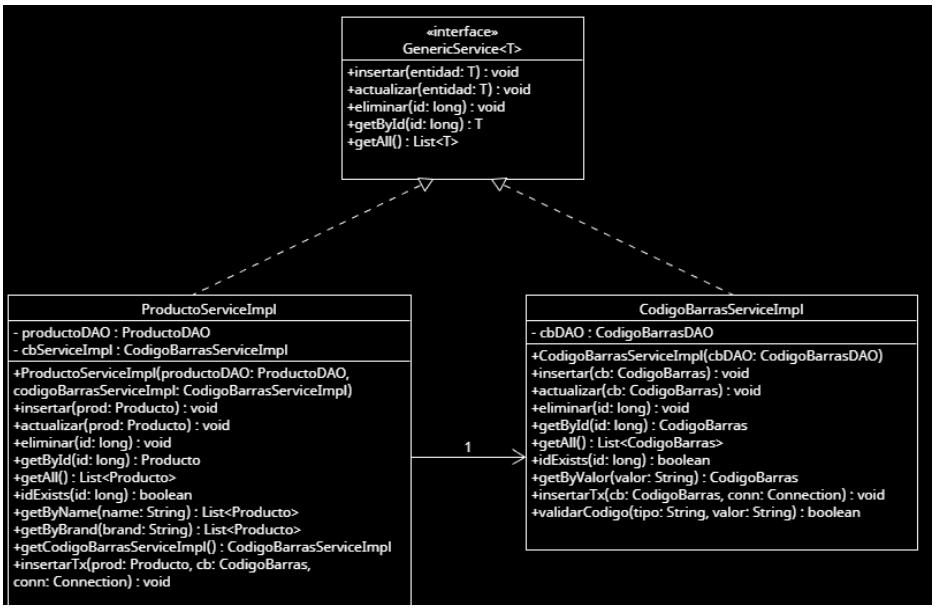
Main



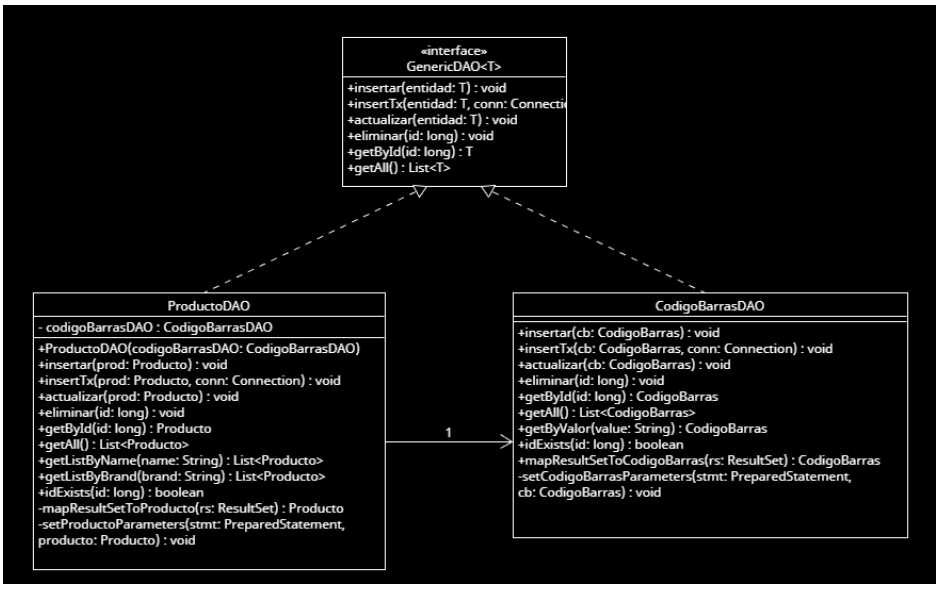
Config



Service



Dao



3. Arquitectura del proyecto

El proyecto sigue una arquitectura por capas, organizada en los paquetes:

- **Models**: entidades del dominio
- **DAO**: acceso a datos mediante JDBC **Service**: reglas de negocio y transacciones.
- **Config**: manejo de la conexión
- **Main/AppMenu**: interacción con el usuario.

Cada capa cumple un rol específico y evita el acoplamiento entre componentes. Demostración de las capas →

```
UTN-TUPaD-P2-TPIntegrador/
src/
    ├── Config/
    |   ├── DatabaseConnection.java
    |   └── TransactionManager
    ├── Dao/
    |   ├── GenericDao.java
    |   ├── ProductoDaoImpl.java
    |   └── CodigoBarrasDaoImpl.java
    ├── Main/
    |   ├── AppMenu.java
    |   ├── Main.java
    |   ├── MenuDisplay.java
    |   ├── MenuHandler.java
    |   └── TestConexion,java
    ├── Models/Entities/
    |   ├── Base.java
    |   ├── TipoCB.java
    |   ├── Producto.java
    |   └── CodigoBarras.java
    └── Service/
        ├── GenericService.java
        ├── ServiceProducto.java
        └── ServiceCodigoBarras.java
```

4. Entidades

El dominio está formado por Producto, CodigoBarras y el enum TipoCB.

Producto (A)

La entidad Producto representa un producto comercial dentro del sistema.

- ➔ Atributos heredados de Base:
  - ◆ id: Long
  - ◆ Clave primaria de la entidad (mapeada como BIGINT en la base de datos).
  - ◆ eliminado: Boolean
- ➔ Atributos propios del dominio:
  - ◆ nombre: String, marca: String, categoria: String, precio: BigDecimal, peso: Double, CodigoBarras: codigoBarras
    - Relación 1 → 1 con la entidad CodigoBarras, A nivel de modelo: A contiene private B detalle.

CodigoBarras (B)

La entidad CodigoBarras representa el código de barras asociado a un producto.

- ➔ Atributos heredados de Base:
  - ◆ id: Long (clave primaria, tipo BIGINT en la base de datos)
  - ◆ eliminado : Boolean (baja lógica)
- ➔ Atributos propios del dominio:
  - ◆ tipo: String (EAN13 / EAN8 / UPC), valor: String, fechaAsignacion: LocalDate, observaciones: String

Enum TipoCB

El enum TipoCB define los posibles tipos de códigos de barras utilizados en el sistema:

- ➔ EAN13 – Código estándar de 13 dígitos.
- ➔ EAN8 – Variante abreviada utilizada en productos más pequeños.
- ➔ UPC – Código Universal de Producto, común en mercados internacionales.

Constructores, getters, setters y toString()

- ➔ Ambas entidades (Producto y CodigoBarras) incluyen:
  - ◆ Constructor vacío
    - public Producto()
    - public CodigoBarras()
  - ◆ Constructor completo
    - public Producto(Long id, Boolean eliminado, String nombre, String marca, String categoria, BigDecimal precio, Double peso, CodigoBarras codigoBarras)
    - public CodigoBarras(Long id, Boolean eliminado, String tipo, String valor, LocalDate fechaAsignacion, String observaciones)
  - ◆ Getters y setters para todos los atributos, garantizando el encapsulamiento y la manipulación controlada de los datos.
  - ◆ toString() legible

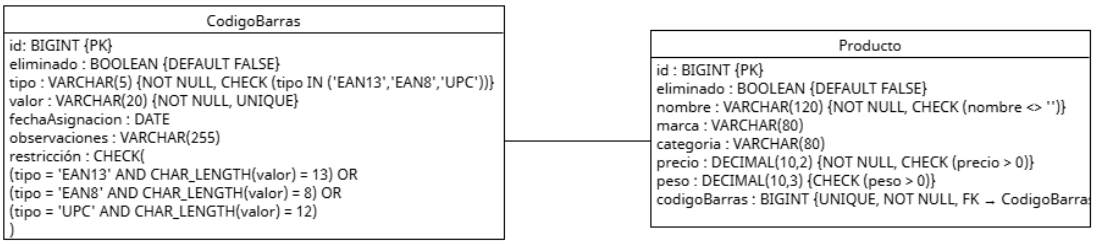
- Cada entidad redefine el método toString() para mostrar sus datos de forma clara por consola, lo que facilita:
  - la visualización de la información en el AppMenu, y
  - las tareas de depuración (debug) durante el desarrollo.

5. Base de datos (MySQL)

Se implementaron dos tablas: producto y codigo_barras
Ambas contienen sus claves primarias (id) y el campo eliminado, utilizado para la baja lógica. Los valores de id se gestionan mediante las inserciones definidas en la lógica del sistema, sin utilizar autoincremental.

Estructuras de tablas

DER



Producto

Esta tabla almacena la información básica de cada producto →

```sql
CREATE TABLE Producto (
  id BIGINT PRIMARY KEY,
  eliminado BOOLEAN DEFAULT FALSE,
  nombre VARCHAR(120) NOT NULL,
  marca VARCHAR(80),
  categoria VARCHAR(80),
  precio DECIMAL(10,2) NOT NULL CHECK (precio > 0),
  peso DECIMAL(10,3) CHECK (peso > 0),
  codigoBarras BIGINT UNIQUE,
  FOREIGN KEY (codigoBarras) REFERENCES CodigoBarras(id)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT
) ENGINE = InnoDB;
```

Codigos_barras

Esta tabla guarda el código asociado a un producto.

```sql
CREATE TABLE CodigoBarras (
  id BIGINT PRIMARY KEY,
  eliminado BOOLEAN DEFAULT FALSE,
  tipo VARCHAR(5) NOT NULL,
  valor VARCHAR(20) NOT NULL UNIQUE,
  fechaAsignacion DATE,
  observaciones VARCHAR(255),
  CHECK (tipo IN ('EAN13', 'EAN8', 'UPC'))
) ENGINE = InnoDB;
```

Relación

La relación 1→1 se asegura mediante:

1. Clave foránea → *FOREIGN KEY (id_producto) REFERENCES producto(id)*
2. Restricción UNIQUE → *UNIQUE (id_producto)*

Evita que un producto tenga más de un código de barras.

Baja lógica

Ambas tablas incluyen el atributo eliminado, lo que permite:

- Mantener los registros en la base.
- Evitar pérdidas de información definitiva.
- Aplicar filtros desde el DAO (WHERE eliminado = false)

De esta forma, se cumple con la consigna de implementar baja lógica en lugar de eliminar físicamente los registros.

Consultas SQL utilizadas

Las principales operaciones SQL aplicadas en el proyecto incluyen:

- INSERT para crear productos y sus códigos de barras.
- UPDATE para modificar registros y aplicar la baja lógica (eliminado = TRUE).
- SELECT con JOIN para recuperar un producto junto con su código de barras.
- DELETE no se utiliza debido al esquema de baja lógica.
- Además, se aplican restricciones UNIQUE sobre:
  - el valor del código de barras, y
  - id_producto en codigo_barras,

  Para evitar códigos duplicados y más de un código asociado al mismo producto.

Todas estas consultas se implementan en la capa DAO usando JDBC y PreparedStatement, lo que mejora la seguridad y el manejo de parámetros.

6. DAO

Se utiliza MySQL con JDBC y PreparedStatement para prevenir inyección SQL y manejar parámetros de forma segura.

a) CodigoBarrasDao

Esta clase se encarga de la persistencia de la entidad CodigoBarras.
Características principales:

- Implementa el CRUD completo:
  - Inserción, actualización, baja lógica, obtención por id y listado general.
- PreparedStatement en todas las operaciones
- Permite recibir una Connection externa
- La baja de registros se realiza de forma lógica, actualizando el campo eliminado en lugar de ejecutar DELETE.

b) ProductoDao

Esta clase administra el acceso a datos para la entidad Producto.
Características principales:

- Implementa el CRUD completo para productos.
- Utiliza consultas con LEFT JOIN para hidratar el objeto Producto junto con su correspondiente CodigoBarras, recuperando toda la información en una sola consulta.
- Acepta una Connection externa, permitiendo que las operaciones se ejecuten dentro de una misma transacción cuando es necesario.
- También aplica baja lógica sobre los productos mediante el atributo eliminado.

c) GenericDao

Para evitar duplicación de código y favorecer la reutilización, se define una interfaz genérica GenericDAO<T> con las operaciones básicas de acceso a datos. En esta capa se:

- ProductoDAO implementa GenericDAO<Producto>.
- CodigoBarrasDAO implementa GenericDAO<CodigoBarras>.

De esta forma se mantiene una estructura homogénea en todos los DAOs del sistema.

7. Service

Esta etapa aborda la lógica de negocio del sistema. Aquí se aplican validaciones, se controla la consistencia de los datos y se gestionan transacciones completas (commit y rollback). Los servicios coordinan las llamadas a los DAOs y garantizan que la relación 1→1 y las reglas del dominio se cumplan correctamente.

a. GenericService<T, K>

Se define la interfaz GenericService<T, K> con las operaciones de negocio básicas:

- Insertar, actualizar, eliminar, getById y getAll

Esto permite mantener una estructura uniforme para los distintos servicios del sistema.

b. CodigoBarrasServiceImpl

Esta encapsula la lógica de negocio asociada a los códigos de barras. Además del CRUD, se encarga de:

- Validar el tipo de código de barras.
- Verificar la longitud según el estándar (por ejemplo, EAN13, EAN8, UPC).
- Controlar que se completen los campos obligatorios antes de persistir los datos.

Se trata de un CRUD más simple, pero con validaciones específicas del dominio de código de barras

c. ProductoServiceImpl

La clase ProductoService es la encargada de coordinar la creación y modificación de productos junto con su código de barras, implementando una transacción completa.
Flujo simplificado de la operación de alta:

A. setAutoCommit(false) sobre la conexión.
B. Insertar primero el Código de Barras a través de CodigoBarrasDAO.
C. Insertar el Producto asociado mediante ProductoDAO.
D. Ejecutar commit() si ambas operaciones fueron exitosas.
E. Ejecutar rollback() en caso de que ocurra algún error en cualquiera de los pasos.

Garantiza atomicidad: o se crean ambos registros, o ninguno

8. AppMenu

En esta parte se presenta la interfaz de interacción con el usuario. El menú de consola permite ejecutar las operaciones CRUD del sistema, gestionando entradas, validaciones y mensajes claros ante errores o casos especiales. Esta capa integra todas las anteriores en un flujo simple y funcional.

Interfaz por consola (AppMenu)

La interacción con el usuario se realiza a través de un menú por consola, implementado en la clase AppMenu. Esta capa actúa como punto de entrada para las operaciones CRUD del sistema, gestionando:

- Captura de datos ingresados por el usuario,

- Validaciones básicas
- Presentación de mensajes claros ante errores o casos especiales.

De esta manera, el AppMenu integra las capas de servicios y DAO en un flujo simple y funcional.

Funcionalidad

El menú permite al usuario realizar las siguientes operaciones

Producto:

- **Crear productos con su código de barras** → Permite dar de alta un producto junto a su código, utilizando lógica transaccional de ProductoService.
- **Listar productos** → muestra el listado, incluyendo información básica y el código asociado.
- **Buscar producto por ID** → recupera un producto específico a partir del identificador.
- **Buscar producto por nombre** → búsqueda filtrado por el nombre.
- **Buscar producto por marca o categoría** → localización de productos aplicando filtros.
- **Editar producto** → modificación de datos de producto existente, respetando validaciones definidas en la capa de service.
- **Eliminar producto** → aplica la baja lógica del producto, marcándolo como eliminado en lugar de borrarlo fácilmente de la base de datos.

Código de barras:

- **Listar códigos de barras** → muestra todos los códigos activos obtenidos desde la capa de service.
- **Buscar código de barra por ID** → recupera un código específico por su identificador.
- **Buscar código de barra por valor** → permite encontrar un código según su valor (cadena numérica)
- **Agregar observaciones a un código de barra** → actualiza el campo de observaciones para un código existente, permitiendo registrar comentarios adicionales sobre su uso o estado.

Control del flujo

- **Salir del sistema** → La opción 0 finaliza la ejecución del programa y cierra el menú principal.

Manejo de errores

El AppMenu realiza distintas validaciones y controles para mejorar la experiencia del usuario:

- **Validación de entradas** → Control de valores nulos o vacíos y verificación de opciones de menú válidas.
- **Manejo de números/fechas** → Conversión segura de cadenas a tipos numéricos y manejo de fechas con el formato esperado, mostrando mensajes claros cuando el formato es incorrecto.
- **Errores de BD** → Captura de excepciones provenientes de la capa DAO/servicio y mensajes informativos ante fallos en la conexión o en la ejecución de consultas.
- **IDs inexistentes** → Verificación previa de la existencia de registros antes de operar sobre ellos y a visos cuando se intenta consultar, modificar o eliminar un registro que no existe.

Integración con Main

La clase AppMenu se integra con el punto de entrada de la aplicación:

- El método AppMenu.iniciar() es el responsable de iniciar toda la ejecución del sistema, mostrando el menú principal y gestionando el ciclo de interacción con el usuario hasta que éste decide salir.

9. Mejoras futuras

a. Autoincremental del ID.
b. Reactivación de productos eliminados.
c. Validaciones más estrictas.
d. Interfaz gráfica.
e. Eliminación empleando transacciones.

Conclusión

A lo largo de este trabajo pudimos integrar los principales contenidos de la materia y aplicarlos en un proyecto concreto. La relación elegida (Producto → Código de Barras) nos permitió trabajar con una estructura clara, con una relación 1 a 1 sencilla de entender pero lo suficientemente completa como para aplicar conceptos de modelado, validaciones y persistencia de datos.

Durante el desarrollo implementamos una arquitectura por capas (modelo, DAO, servicios y menú), lo que facilitó la organización del código y la separación de responsabilidades. Además, utilizamos JDBC para conectarnos a MySQL, trabajar con consultas preparadas, manejar excepciones y realizar una transacción completa al momento de crear un producto junto con su código de barras. También se incorporó la baja lógica mediante el atributo eliminado, evitando la pérdida definitiva de información y cumpliendo con las consignas del TPI.

El sistema final permite crear, listar y buscar productos y códigos de barras, asociarlos entre sí y gestionar sus estados, aplicando validaciones desde la capa de servicio y manejando los posibles errores en la interacción con el usuario y la base de datos. Este proceso reforzó conceptos como POO, uso de clases e interfaces genéricas, diseño de DAOs y servicios, validaciones de dominio y manejo de errores.

En resumen, este trabajo integrador permitió unir teoría y práctica, y brindó una visión más completa de cómo se desarrolla un sistema desde cero: desde el modelo de entidades y la base de datos, pasando por la capa de acceso a datos y servicios, hasta la interfaz por consola, aplicando buenas prácticas de programación y consolidando los contenidos de la materia.

Bibliografía

Link del repositorio GitHub:
https://github.com/Fulla1996/UTN-TUPaD-P2-TPIntegrador/tree/main

Link del video:
https://youtu.be/t1uFYBuqA5M

Link de la carpeta digital (completa):
https://drive.google.com/drive/folders/1zvLOClQhh2N6vJW-yv-dY8SRE9Qdmf84?usp=sharing

Link de presentación Canva:
https://www.canva.com/design/DAG5P9hMAVQ/F5rgdOZM8U0ii0GjwDJalg/edit?utm_content=DAG5P9hMAVQ&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton