Trabajo Práctico Integrador

Sistema de gestión de productos
Producto → CódigoBarras

Alumnos - Grupo 128

Diego Raúl Montes – Cristian A. Morales – Ramiro Morales – Sebastián Ríos

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Programación 2

Docente Titular

Ariel Enferrel

Docente Tutor

David López

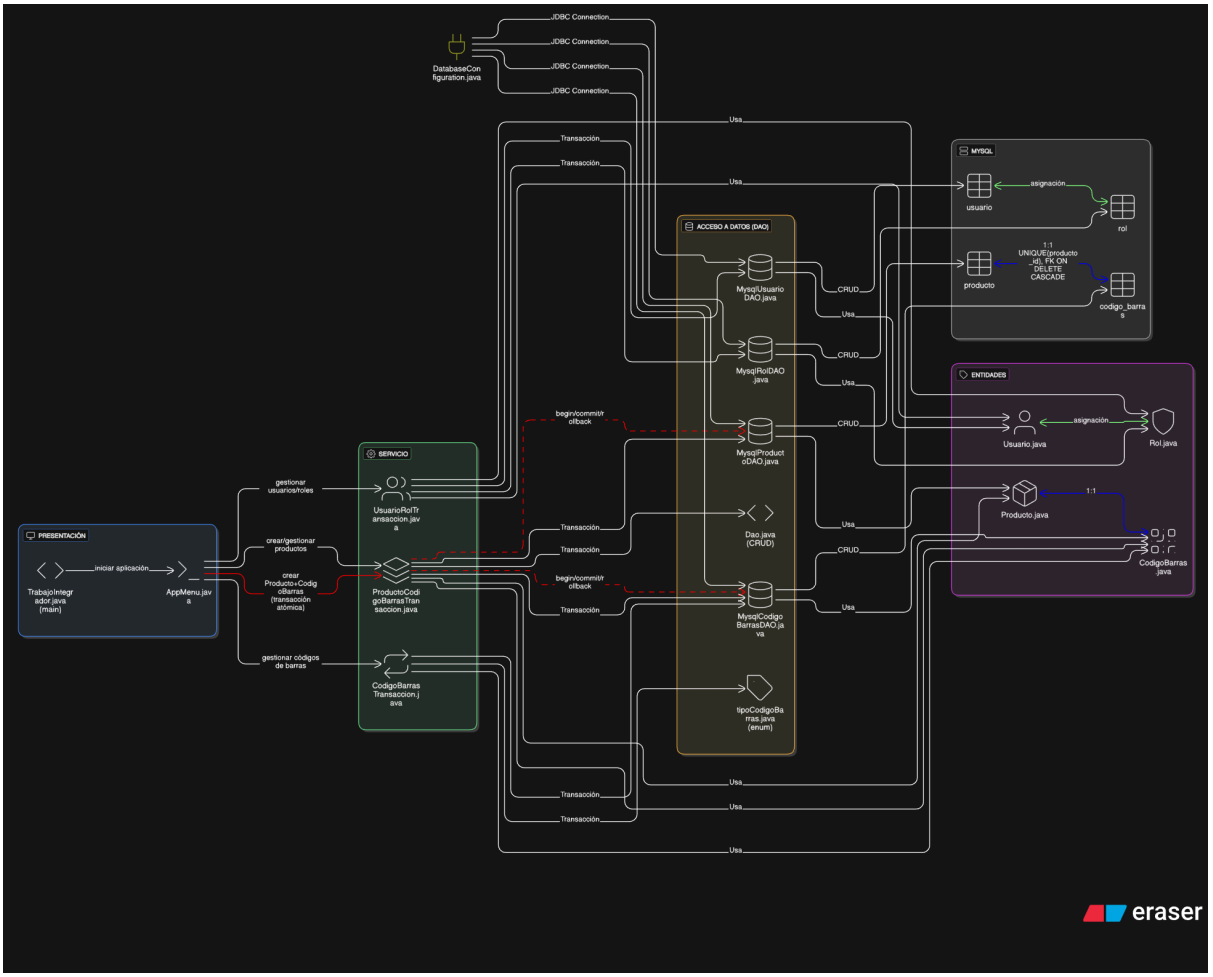
16 de Noviembre de 2025

ÍNDICE

Introducción

Como equipo del Grupo 128, presentamos el desarrollo del Trabajo Final Integrador de la materia Programación II. Nuestro objetivo fue diseñar e implementar una aplicación Java robusta y modular que gestione la relación unidireccional 1→1 entre Producto y CódigoBarras, aplicando patrones de diseño arquitectónicos y buenas prácticas de desarrollo.

El sistema implementa una arquitectura en capas claramente definida, separando responsabilidades entre entidades, acceso a datos, lógica de negocio y presentación. Utilizamos JDBC para la persistencia, implementamos transacciones manuales para garantizar consistencia, y aplicamos validaciones de negocio centralizadas.



Repositorio GitHub

Link repositorio

Video

Link del video

Desarrollo por etapas

Etapa 1: Análisis y Diseño

Como equipo, analizamos los requerimientos y definimos el dominio Producto ⟶ CódigoBarras. Diseñamos el diagrama UML de clases identificando atributos, tipos de datos, visibilidad y relaciones. Decidimos implementar la relación 1⟶1 como unidireccional, donde Producto referencia a CódigoBarras pero no viceversa, simplificando el modelo y manteniendo la independencia de las entidades.

Etapa 2: Implementación de Entidades

Desarrollamos las clases Producto y CodigoBarras en el paquete entities, incluyendo constructores completos y vacíos, getters, setters, y el campo eliminado para implementar bajas lógicas. Como equipo, acordamos usar tipos de datos específicos como LocalDate para fechas y BigDecimal para precios, garantizando precisión en cálculos financieros.

Etapa 3: Acceso a Datos (DAO)

Implementamos el patrón DAO con interfaces genéricas y clases concretas para cada entidad. Utilizamos PreparedStatement en todas las operaciones CRUD para prevenir inyecciones SQL. Los DAOs aceptan conexiones externas para participar en transacciones compartidas, facilitando el control transaccional desde la capa Service.

Etapa 4: Capa Service

Creamos servicios que orquestan operaciones complejas y aplican reglas de negocio. Implementamos validaciones como precio positivo, unicidad de códigos de barras, y garantía de la relación 1⟶1. El control transaccional se maneja aquí, con setAutoCommit(false), commit() y rollback() para operaciones atómicas.

Etapa 5: Interfaz y Pruebas

Desarrollamos AppMenu con un menú de consola interactivo que permite todas las operaciones CRUD. Implementamos manejo robusto de excepciones, validación de entradas de usuario, y conversión a mayúsculas donde aplica. Realizamos pruebas exhaustivas de flujos felices y alternativos, incluyendo simulación de errores para verificar el rollback transaccional.

Decisiones de diseño y razonamiento técnico

Arquitectura en Capas

Optamos por una separación clara de responsabilidades:

- **Entities**: Modelan el dominio sin lógica de persistencia
- **DAO**: Encapsulan el acceso a datos usando JDBC directamente
- **Service**: Contienen lógica de negocio y control transaccional
- **Main**: Maneja la interacción con el usuario

Patrón DAO con GenericDao

Implementamos GenericDao<T> con operaciones CRUD básicas, permitiendo reutilización y consistencia. Las clases concretas especializan comportamientos específicos de cada entidad.

Control Manual de Transacciones

Decidimos manejar transacciones manualmente en lugar de depender del autocommit, ya que necesitábamos garantizar atomicidad en operaciones que involucran múltiples entidades.

Baja Lógica vs Física

Implementamos el campo eliminado para mantener el historial de datos y permitir recuperación, siguiendo prácticas empresariales estándar.

Control de transacciones

Estrategia Implementada

Como equipo, diseñamos un sistema de transacciones que garantiza consistencia en operaciones compuestas:

```
// Ejemplo de transacción en ProductoService
public boolean crearProductoCompleto(Producto producto, CodigoBarras
codigoBarras) {
    Connection conn = null;
    try {
        conn = DatabaseConnection.getConnection();
        conn.setAutoCommit(false);

        // 1. Insertar código de barras
        CodigoBarrasDao codigoDao = new CodigoBarrasDao(conn);
        Long codigoId = codigoDao.crear(codigoBarras);

        // 2. Asociar y insertar producto
        producto.setCodigoBarras(codigoBarras);
        ProductoDao productoDao = new ProductoDao(conn);
        Long productoId = productoDao.crear(producto);

        conn.commit();
        return true;

    } catch (SQLException e) {
        if (conn != null) {
            try { conn.rollback(); } catch (SQLException ex) {}
        }
        return false;
    } finally {
        if (conn != null) {
            try {
                conn.setAutoCommit(true);
                conn.close();
            } catch (SQLException e) {}
        }
    }
}
```

Flujo Transaccional

1. **Inicio**: Obtener conexión y desactivar autocommit
2. **Ejecución**: Operaciones DAO secuenciales
3. **Confirmación**: Commit si todas las operaciones son exitosas
4. **Recuperación**: Rollback ante cualquier error

5. **Limpieza**: Restaurar autocommit y cerrar recursos

Validaciones y reglas de negocio

Validaciones Implementadas

- **Precio positivo**: producto.getPrecio().compareTo(BigDecimal.ZERO) > 0
- **Campos obligatorios**: nombre, categoría, tipo de código
- **Unicidad**: código de barras único en el sistema
- **Formato**: validación de tipos de código EAN13, EAN8, UPC
- **Relación 1⟶1**: imposibilidad de asignar múltiples códigos a un producto

Reglas de Negocio Aplicadas

- **RN-001**: Todo producto debe tener exactamente un código de barras
- **RN-002**: Los precios deben ser mayores a cero
- **RN-003**: Solo bajas lógicas, nunca eliminación física
- **RN-004**: Códigos de barras únicos en todo el sistema
- **RN-005**: Integridad referencial mantenida transaccionalmente

Estructura de carpetas del proyecto

```
tpi-programacion2/
│
├── src/
│   ├── config/
│   │   └── DatabaseConnection.java
│   │
│   ├── entities/
│   │   ├── Producto.java
│   │   └── CodigoBarras.java
│   │
│   ├── dao/
│   │   ├── GenericDao.java
│   │   ├── ProductoDao.java
│   │   └── CodigoBarrasDao.java
│   │
│   ├── service/
│   │   ├── ProductoService.java
│   │   └── CodigoBarrasService.java
│   │
│   └── main/
│       └── AppMenu.java
│
├── sql/
│   ├── 01_esquema.sql
│   ├── 02_datos_prueba.sql
│   └── 03_consultas.sql
│
├── docs/
│   ├── diagrama_uml.png
│   └── README.md
│
└── lib/
    └── mysql-connector-java-8.0.33.jar
```

Referencias

UTN. (2025). *Trabajo Práctico Integrador*. Material de Cátedra – Programación II, Tecnicatura Universitaria en Programación. Universidad Tecnológica Nacional.

ChatGPT. (2025). *Asistencia en el desarrollo del Trabajo Práctico Integrador*. OpenAI. Comunicación personal, 16 de Noviembre de 2025.