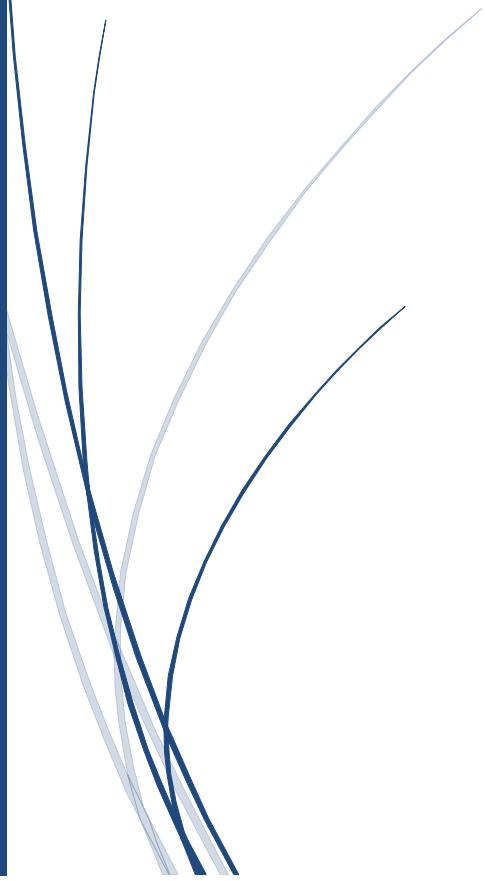
UNIVERSIDAD TECNOLÓGICA NACIONAL

Programación II

Trabajo Práctico 6:

Colecciones y Sistema de Stock

Alumno: Lara Malena Pérez Hernández
Legajo: 101063
Comisión: 19
Fecha de entrega: 21/11/2025



Índice

1. Objetivo

El objetivo de este trabajo práctico es desarrollar estructuras de datos dinámicas en Java mediante el uso de colecciones (ArrayList) y enumeraciones (enum), implementando tres sistemas diferentes que refuerzan conceptos clave de la programación orientada a objetos:

2. Repositorio GitHub

Link al repositorio: https://github.com/laraphernandez/TPS-Prog2-2025/tree/main/TP6_Colecciones%20y%20Sistema%20de%20Stock

Estructura del repositorio:

/ejercicio1_stock - Sistema de gestión de inventario
/ejercicio2_biblioteca - Sistema de gestión de biblioteca
/ejercicio3_universidad - Sistema académico universidad

3. Ejercicios Implementados

3.1. Ejercicio 1: Sistema de Stock

Descripción:

Sistema de gestión de inventario que permite controlar productos en una tienda, manejando disponibilidad, precios y categorías mediante colecciones dinámicas.

Relaciones:

- Relación 1 a N: Inventario (1) → Productos (N)
- Cada producto tiene una categoría (enum)

Clases implementadas:

CategoriaProducto (enum):

- Valores: ALIMENTOS, ELECTRONICA, ROPA, HOGAR
- Atributo: descripcion (String)
- Método: getDescripcion()

Producto:

- id (String) - Identificador único
  nombre (String) - Nombre del producto
- precio (double) - Precio del producto
- cantidad (int) - Cantidad en stock
- categoria (CategoriaProducto) - Categoría
- Método: mostrarInfo()

Inventario:

- productos (ArrayList<Producto>) - Colección de productos
- agregarProducto(Producto p)
- listarProductos()
- buscarProductoPorId(String id)
- eliminarProducto(String id)
- actualizarStock(String id, int nuevaCantidad)
- filtrarPorCategoria(CategoriaProducto categoria)
- obtenerTotalStock()
- obtenerProductoConMayorStock()
• filtrarProductosPorPrecio(double min, double max)
• mostrarCategoriasDisponibles()

Funcionalidades demostradas:

1. Creación de 5 productos con diferentes categorías
2. Listado completo de productos
3. Búsqueda por ID
4. Filtrado por categoría
5. Eliminación de productos
6. Actualización de stock
7. Cálculo de stock total
8. Identificación del producto con mayor stock
9. Filtrado por rango de precios
10. Visualización de categorías con descripciones

3.2. Ejercicio 2: Biblioteca y Libros

Descripción:

Sistema para gestionar una biblioteca con libros y autores. Implementa una relación de composición 1 a N donde la Biblioteca contiene múltiples Libros, y cada Libro está asociado obligatoriamente a un Autor.

Relaciones:

- Composición 1 a N: Biblioteca (1) → Libros (N)
- Asociación: Libro → Autor
- Si la Biblioteca se elimina, sus Libros también

Clases implementadas:

Autor:

• id (String) - Identificador único del autor
• nombre (String) - Nombre del autor
• nacionalidad (String) - Nacionalidad del autor
• Método: mostrarInfo()

Libro:

• isbn (String) - Identificador único del libro
• titulo (String) - Título del libro
• anioPublicacion (int) - Año de publicación
• autor (Autor) - Autor del libro
• Método: mostrarInfo()

Biblioteca:

• nombre (String) - Nombre de la biblioteca
• libros (List<Libro>) - Colección de libros
• agregarLibro(String isbn, String titulo, int anioPublicacion, Autor autor)
• listarLibros()
• buscarLibroPorIsbn(String isbn)
• eliminarLibro(String isbn)
• obtenerCantidadLibros()
• filtrarLibrosPorAnio(int anio)
• mostrarAutoresDisponibles()

Funcionalidades demostradas:

1. Creación de una biblioteca
2. Creación de 3 autores
3. Agregado de 5 libros asociados a los autores

4. Listado de todos los libros con información del autor
5. Búsqueda de libro por ISBN
6. Filtrado de libros por año de publicación
7. Eliminación de libros
8. Obtención de cantidad total de libros
9. Listado de autores únicos disponibles

3.3. Ejercicio 3: Universidad, Profesor y Curso

Descripción:

Sistema académico que modela la relación bidireccional entre Profesores y Cursos. Un Profesor puede dictar muchos Cursos, y cada Curso tiene exactamente un Profesor responsable. La Universidad administra el alta, baja y consulta de profesores y cursos.

Relaciones:

- Relación bidireccional 1 a N: Profesor (1) ↔ Cursos (N)
- Desde Curso se accede a su Profesor
- Desde Profesor se accede a su lista de Cursos
- Universidad gestiona ambas entidades

Invariante de asociación: Cada vez que se asigne o cambie el profesor de un curso, debe actualizarse automáticamente en ambos lados de la relación.

Clases implementadas:

Profesor:

- id (String) - Identificador único
- nombre (String) - Nombre completo
- especialidad (String) - Área principal
- cursos (List<Curso>) - Cursos que dicta
- agregarCurso(Curso c) - Sincroniza ambos lados
- eliminarCurso(Curso c) - Sincroniza ambos lados
- listarCursos()
- mostrarInfo()

Curso:

- codigo (String) - Código único

• nombre (String) - Nombre del curso
• profesor (Profesor) - Profesor responsable
• setProfesor(Profesor p) - Asigna/cambia profesor con sincronización
• mostrarInfo()

Universidad:

• nombre (String) - Nombre de la universidad
• profesores (List<Profesor>) - Colección de profesores
• cursos (List<Curso>) - Colección de cursos
• agregarProfesor(Profesor p)
• agregarCurso(Curso c)
• asignarProfesorACurso(String codigoCurso, String idProfesor)
• listarProfesores() / listarCursos()
• buscarProfesorPorId(String id)
• buscarCursoPorCodigo(String codigo)
• eliminarCurso(String codigo) - Rompe relación con profesor
• eliminarProfesor(String id) - Deja cursos sin profesor
• mostrarReporteCursosPorProfesor()

Funcionalidades demostradas:

1. Creación de 3 profesores y 5 cursos
2. Agregado de profesores y cursos a la universidad
3. Asignación de profesores a cursos
4. Listado de cursos con su profesor y profesores con sus cursos
5. Cambio de profesor de un curso con sincronización automática
6. Eliminación de curso con actualización en el profesor
7. Eliminación de profesor dejando cursos sin asignar
8. Generación de reporte de cantidad de cursos por profesor