**PROGRAMACIÓN II**

**Trabajo Práctico 6: Colecciones y Sistema de Stock**

# Alumno:

Franco Sarrú

# Link público de GitHub:

<https://github.com/fsarru/Programacion2.git>

# OBJETIVO GENERAL

Desarrollar estructuras de datos dinámicas en Java mediante el uso de colecciones (**ArrayList**) y enumeraciones (**enum**), implementando un sistema de stock con funcionalidades progresivas que refuerzan conceptos clave de la programación orientada a objetos..

# MARCO TEÓRICO



# Caso Práctico 1

1. Descripción general

Se debe desarrollar un sistema de stock que permita gestionar productos en una tienda, controlando su disponibilidad, precios y categorías. La información se modelará utilizando clases, colecciones dinámicas y enumeraciones en Java.

1. Clases a implementar **Clase Producto**

**Atributos:**

* **id (String)** → Identificador único del producto.
* **nombre (String)** → Nombre del producto.
* **precio (double)** → Precio del producto.
* **cantidad (int)** → Cantidad en stock.
* **categoria (CategoriaProducto)** → Categoría del producto.

**Métodos:**

* **mostrarInfo()** → Muestra en consola la información del producto.

**Enum CategoriaProducto Valores:**

* ALIMENTOS
* ELECTRONICA
* ROPA
* HOGAR

## Método adicional: java public enum

**CategoriaProducto {**

**ALIMENTOS("Productos comestibles"),**

**ELECTRONICA("Dispositivos electrónicos"),**

**ROPA("Prendas de vestir"), HOGAR("Artículos para el hogar"); private final String descripcion; CategoriaProducto(String descripcion) { this.descripcion = descripcion;**

**}**

**public String getDescripcion() { return descripcion;**

**}**

**}**

## Clase Inventario

Atributo:

* **ArrayList<Producto> productos**

Métodos requeridos:

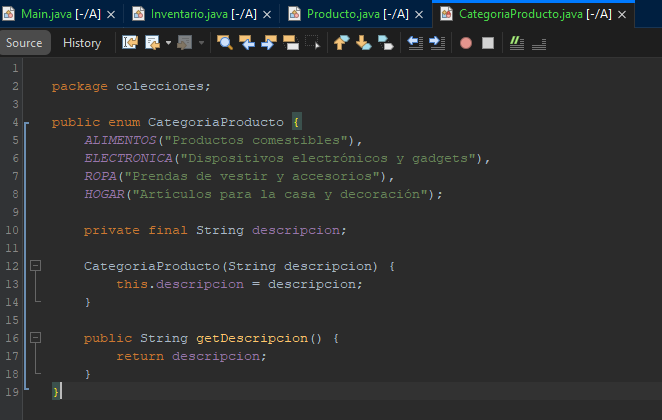
* **agregarProducto(Producto p)**
* **listarProductos()**
* **buscarProductoPorId(String id)**
* **eliminarProducto(String id)**
* **actualizarStock(String id, int nuevaCantidad)**
* **filtrarPorCategoria(CategoriaProducto categoria)**
* **obtenerTotalStock()**
* **obtenerProductoConMayorStock()**
* **filtrarProductosPorPrecio(double min, double max)**
* **mostrarCategoriasDisponibles()**

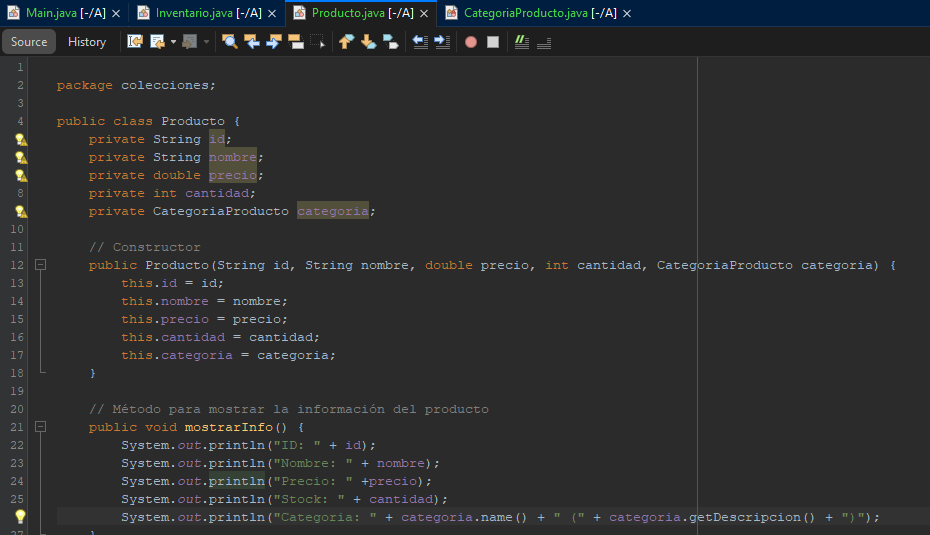
**3. Tareas a realizar**

1. Crear al menos cinco productos con diferentes categorías y agregarlos al inventario.
2. Listar todos los productos mostrando su información y categoría.
3. Buscar un producto por ID y mostrar su información.
4. Filtrar y mostrar productos que pertenezcan a una categoría específica.
5. Eliminar un producto por su ID y listar los productos restantes.
6. Actualizar el stock de un producto existente.
7. Mostrar el total de stock disponible.
8. Obtener y mostrar el producto con mayor stock.
9. Filtrar productos con precios entre $1000 y $3000.
10. Mostrar las categorías disponibles con sus descripciones.

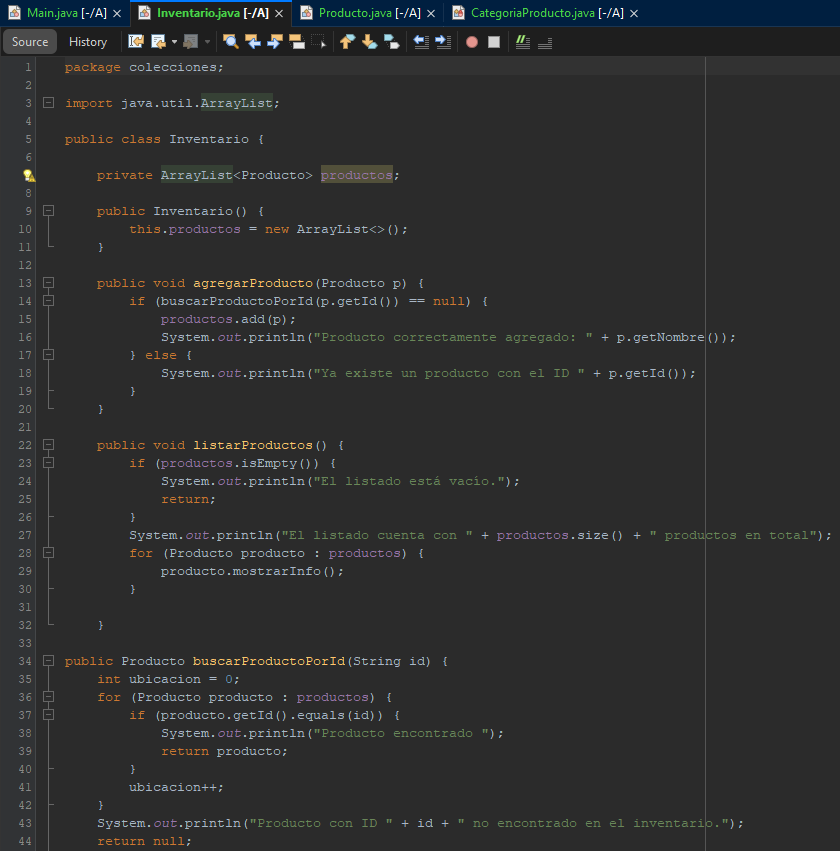
# CONCLUSIONES ESPERADAS

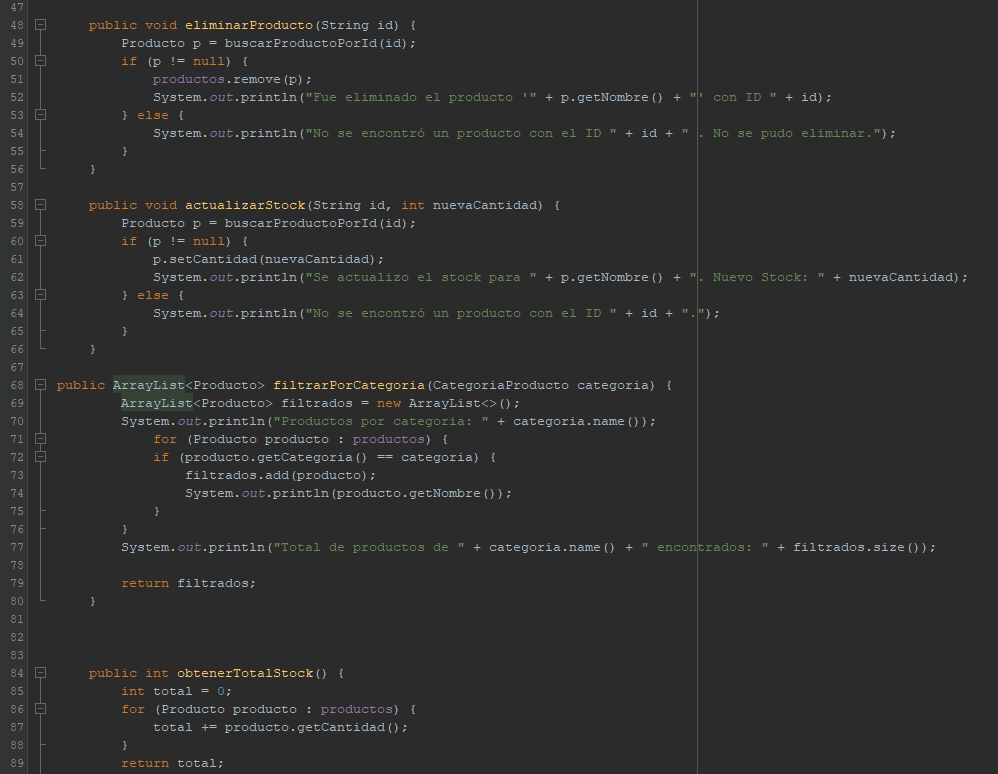
* Comprender el uso de **this** para acceder a atributos de instancia.
* Aplicar constructores sobrecargados para flexibilizar la creación de objetos.
* Implementar métodos con el mismo nombre y distintos parámetros.
* Representar objetos con **toString()** para mejorar la depuración.
* Diferenciar y aplicar atributos y métodos estáticos en Java.
* Reforzar el diseño modular y reutilizable mediante el paradigma orientado a objetos.

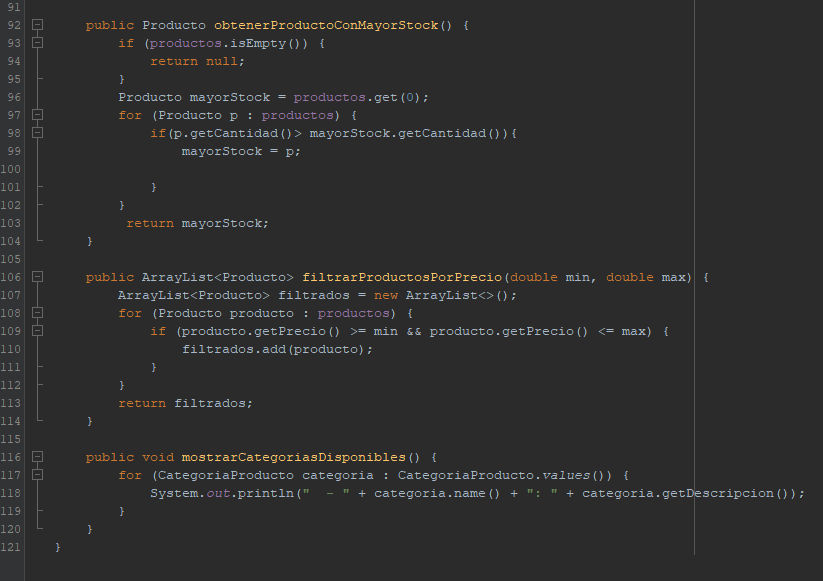


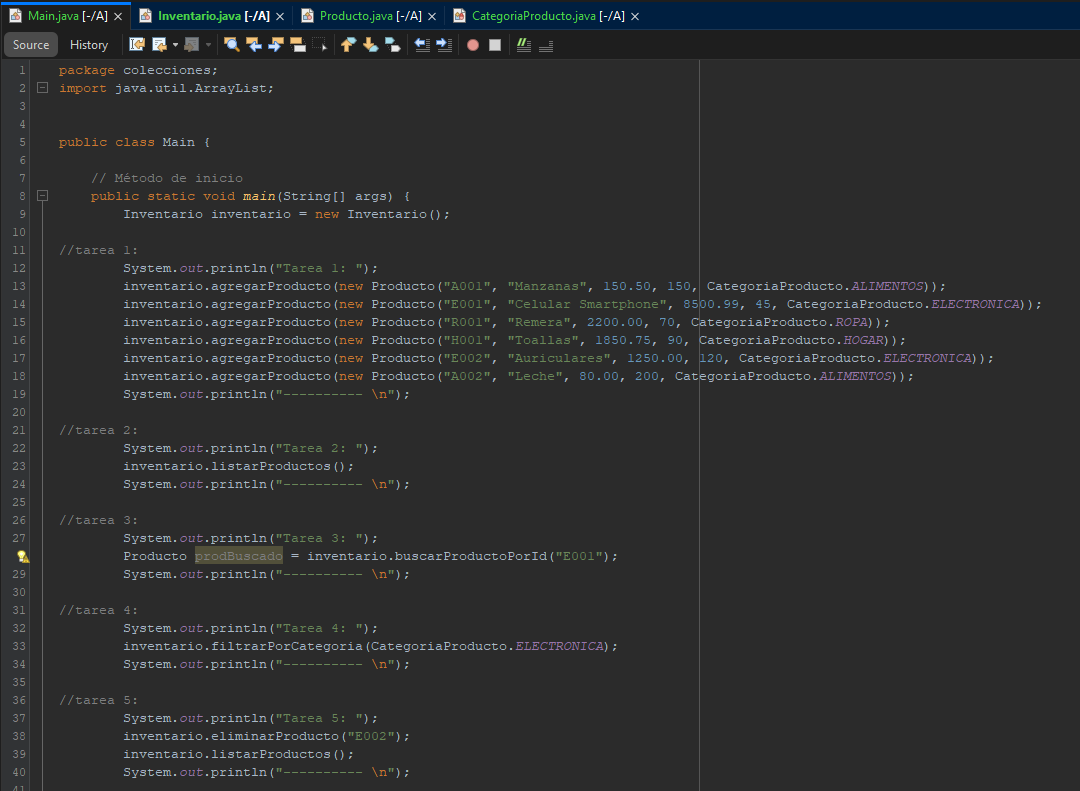


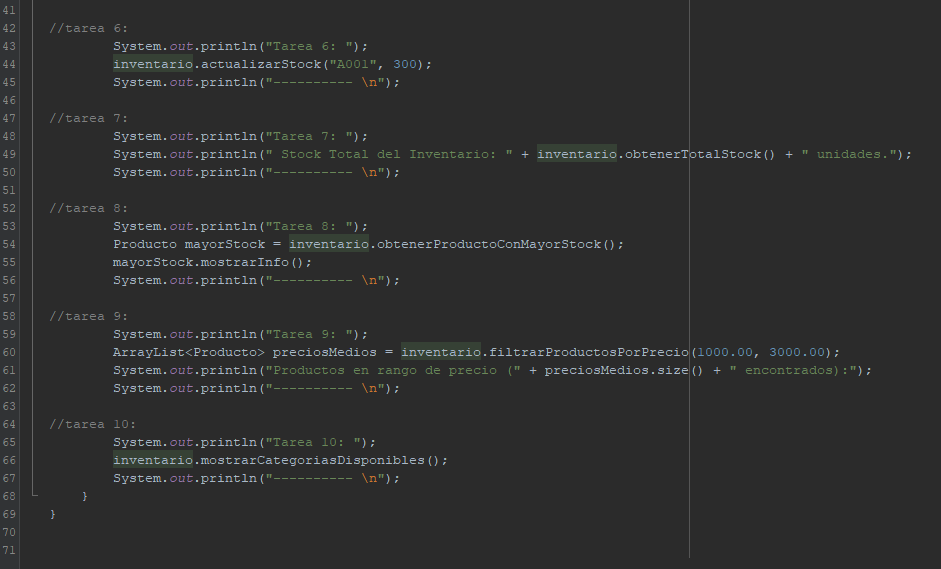


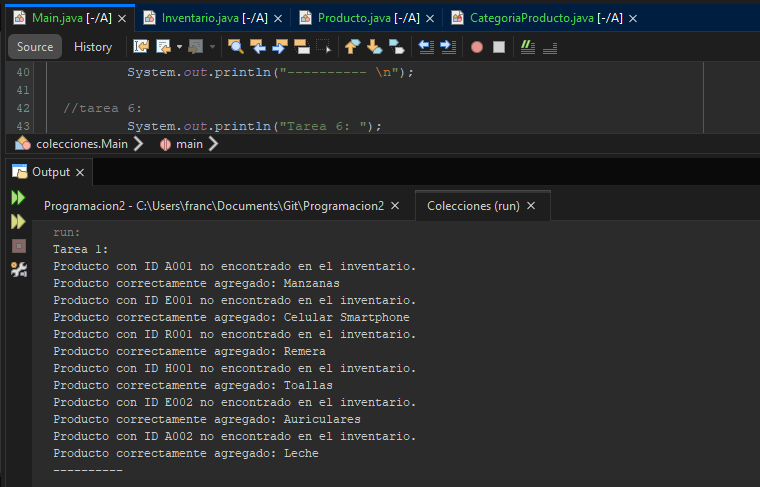


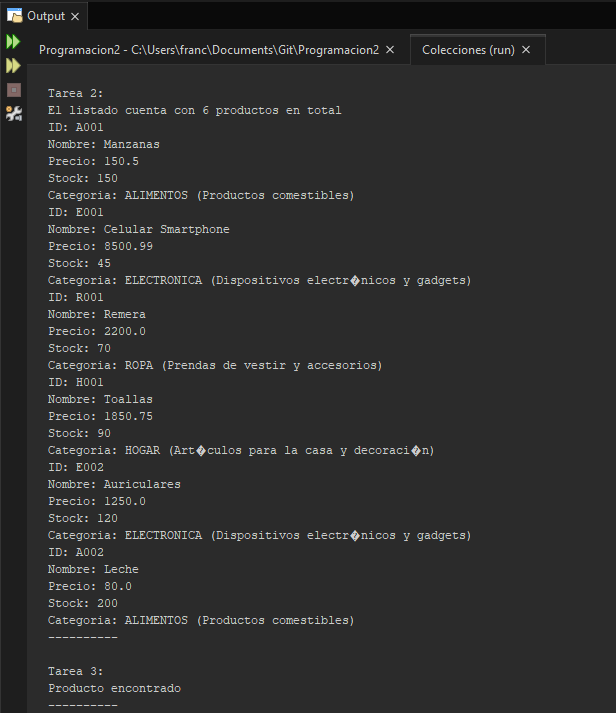


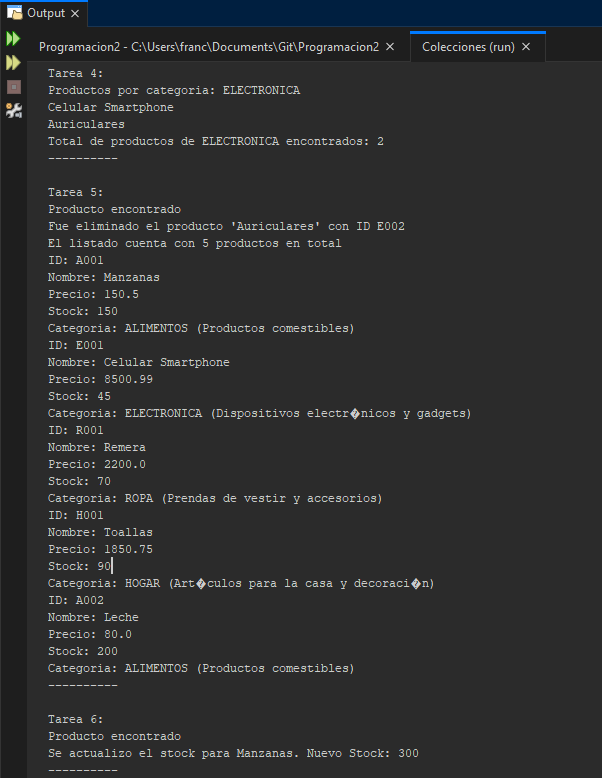


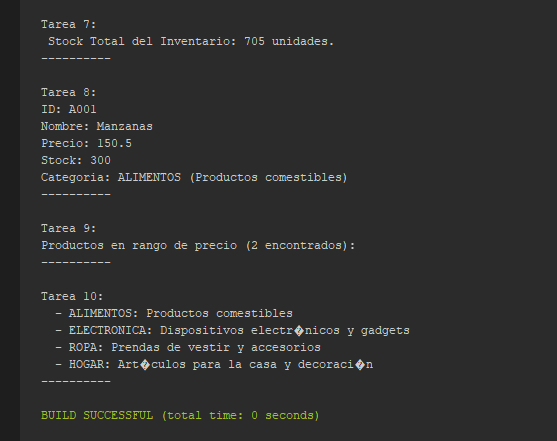












# 📝 Nuevo Ejercicio Propuesto 2: Biblioteca y Libros

1. Descripción general

Se debe desarrollar un sistema para gestionar una **biblioteca**, en la cual se registren los libros disponibles y sus autores. La relación central es de **composición 1 a N**: una Biblioteca contiene múltiples Libros, y cada Libro pertenece obligatoriamente a una Biblioteca. Si la Biblioteca se elimina, también se eliminan sus Libros.

1. Clases a implementar **Clase Autor**

**Atributos:**

* + **id (String) →** Identificador único del autor.
  + **nombre (String)** → Nombre del autor.
  + **nacionalidad (String)** → Nacionalidad del autor.

**Métodos:**

* + **mostrarInfo()** → Muestra la información del autor en consola.

## Clase Libro

**Atributos:**

* **isbn (String)** → Identificador único del libro.
* **titulo (String)** → Título del libro.
* **anioPublicacion (int)** → Año de publicación.
* **autor (Autor)** → Autor del libro.

**Métodos:**

* **mostrarInfo()** → Muestra título, ISBN, año y autor.

**Clase Biblioteca Atributo:**

* **String nombre**
* **List<Libro> libros** → Colección de libros de la biblioteca.

**Métodos requeridos:**

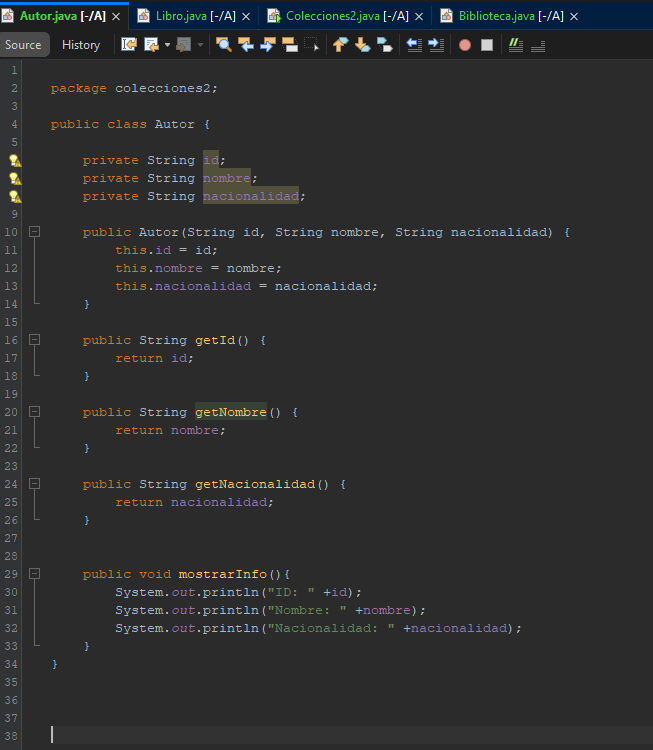
* **agregarLibro(String isbn, String titulo,int anioPublicacion, Autor autor)**
* **listarLibros()**
* **buscarLibroPorIsbn(String isbn)**
* **eliminarLibro(String isbn)**
* **obtenerCantidadLibros()**
* **filtrarLibrosPorAnio(int anio)**
* **mostrarAutoresDisponibles()**

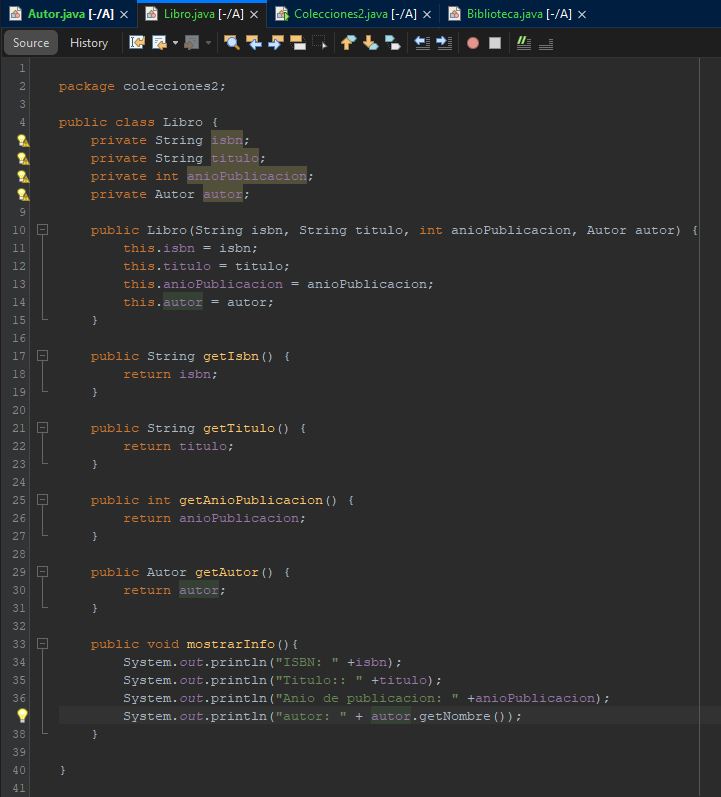
**3. Tareas a realizar**

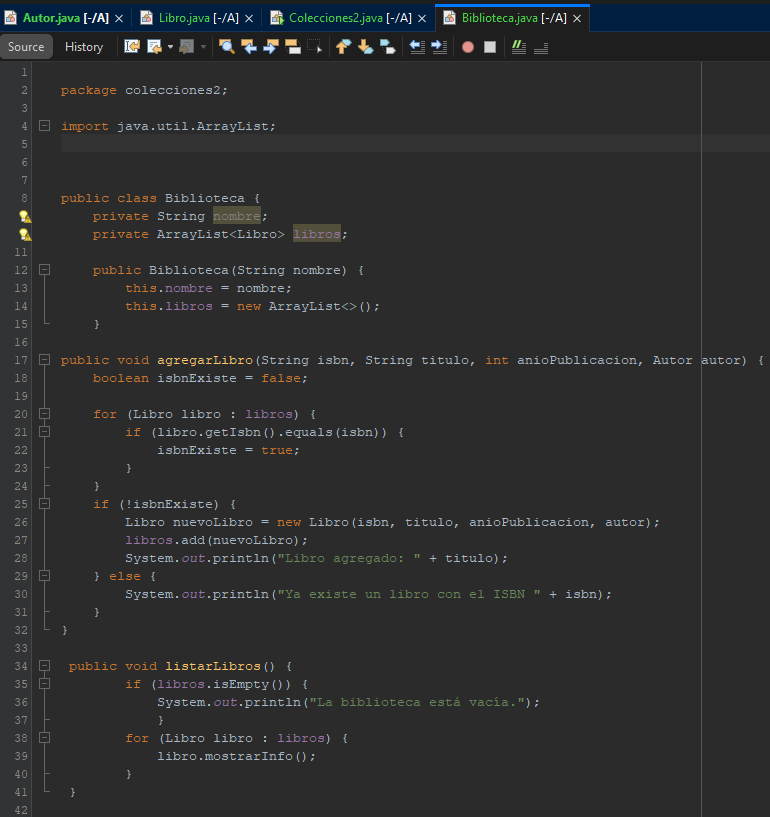
1. Creamos una biblioteca.
2. Crear al menos tres autores
3. Agregar 5 libros asociados a alguno de los Autores a la biblioteca.
4. Listar todos los libros con su información y la del autor.
5. Buscar un libro por su ISBN y mostrar su información.
6. Filtrar y mostrar los libros publicados en un año específico.
7. Eliminar un libro por su ISBN y listar los libros restantes.
8. Mostrar la cantidad total de libros en la biblioteca.
9. Listar todos los autores de los libros disponibles en la biblioteca.

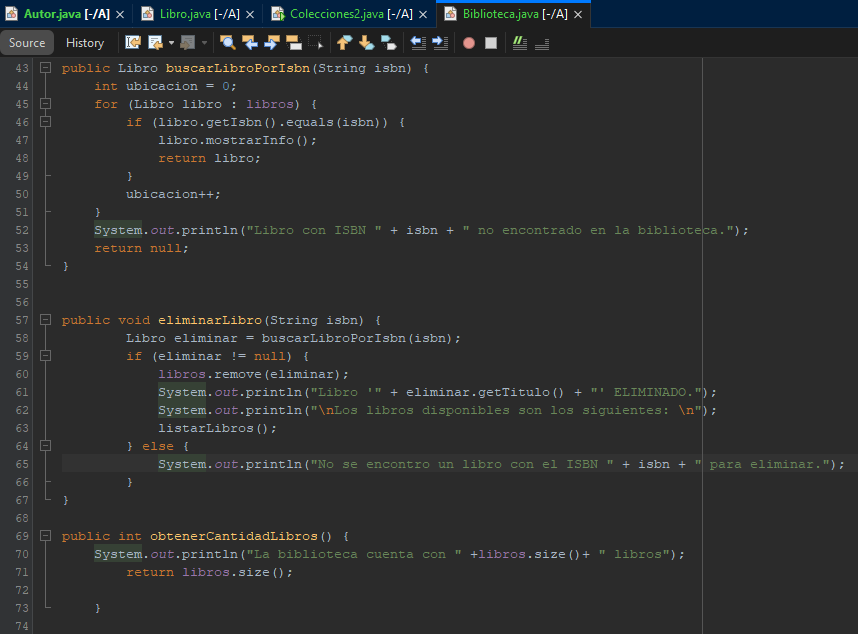
# Conclusiones esperadas

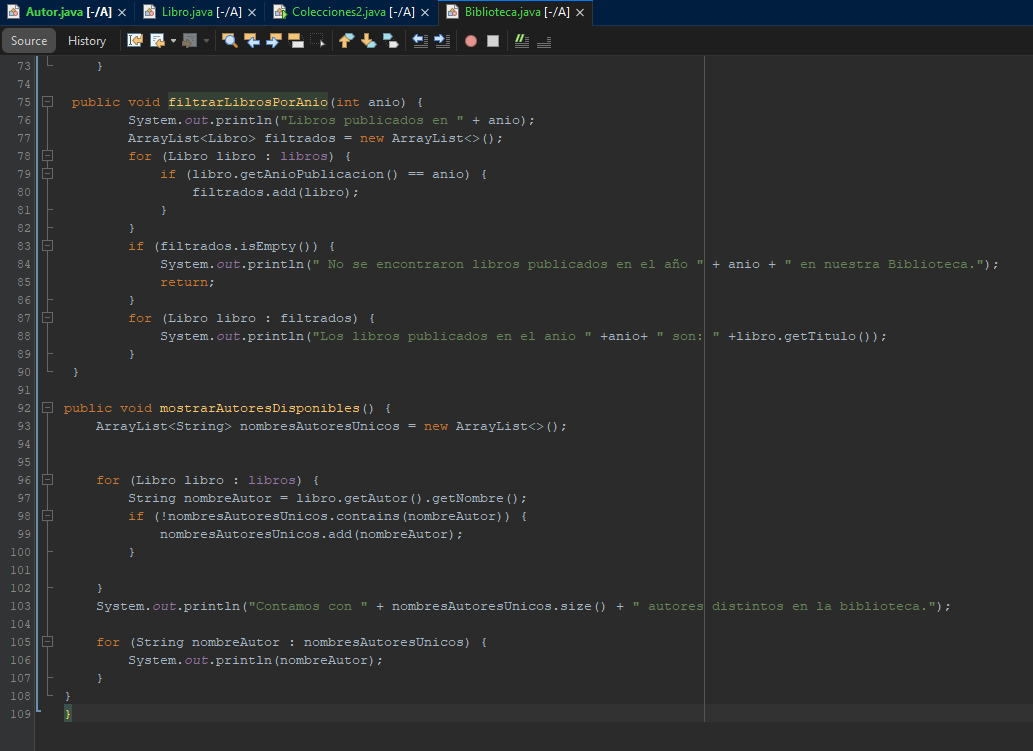
* Comprender la **composición 1 a N** entre Biblioteca y Libro.
* Reforzar el manejo de **colecciones dinámicas** (ArrayList).
* Practicar el uso de **métodos de búsqueda, filtrado y eliminación**.
* Mejorar la modularidad aplicando el paradigma de **programación orientada a objetos.**

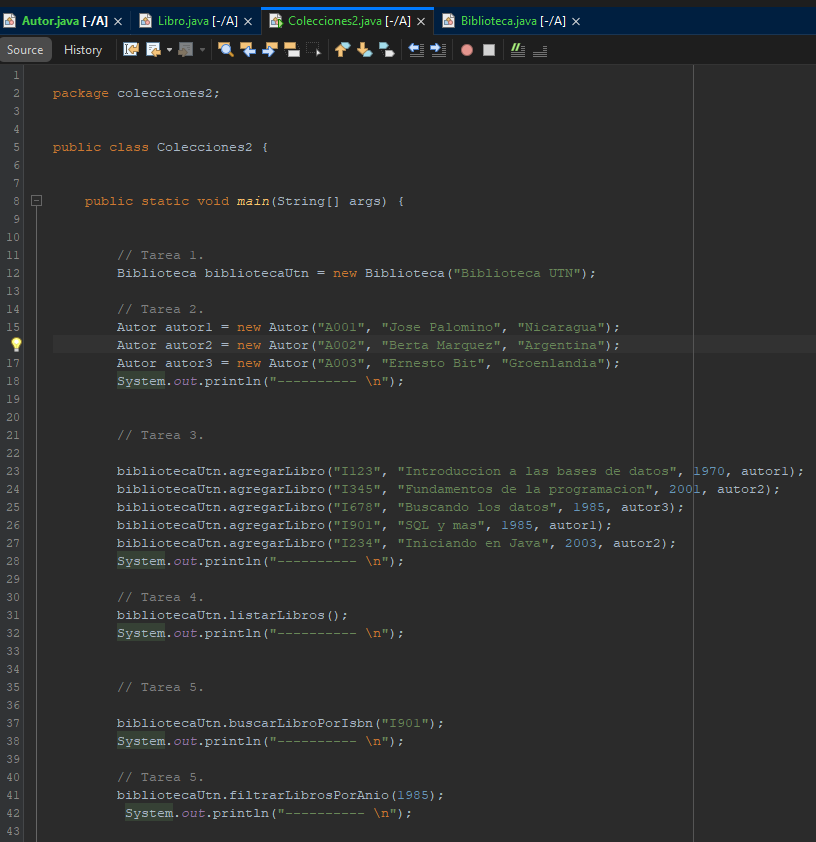


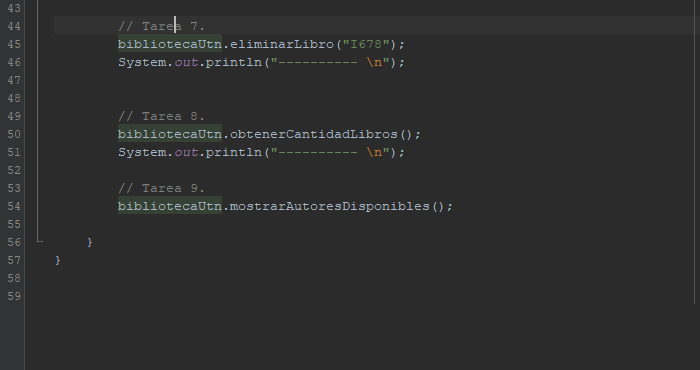


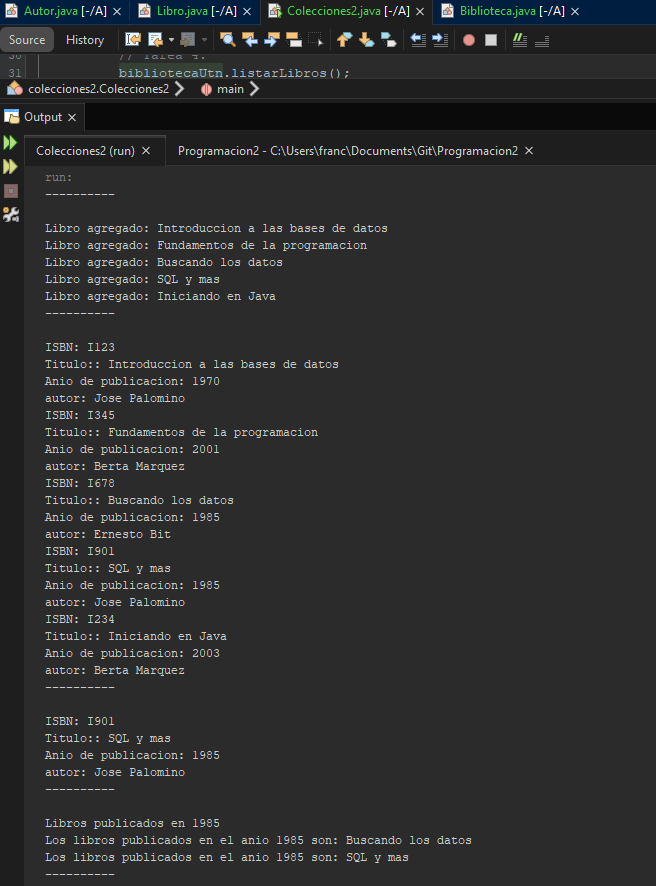


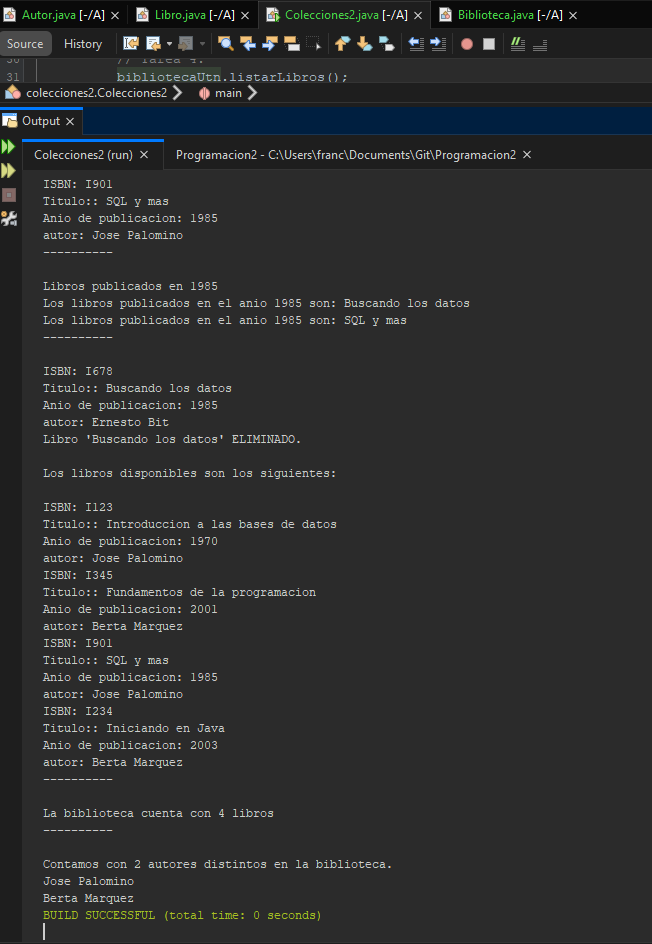












# 📝 Ejercicio: Universidad, Profesor y Curso (bidireccional 1 a N)

1. Descripción general

Se debe modelar un sistema académico donde **un Profesor dicta muchos Cursos** y cada **Curso** tiene exactamente **un Profesor responsable**. La relación **Profesor– Curso** es **bidireccional**:

* Desde **Curso** se accede a su **Profesor**.
* Desde **Profesor** se accede a la **lista de Cursos** que dicta.

Además, existe la clase **Universidad** que administra el alta/baja y consulta de profesores y cursos.

**Invariante de asociación:** cada vez que se asigne o cambie el profesor de un curso, **debe actualizarse en los dos lados** (agregar/quitar en la lista del profesor correspondiente).

2. Clases a implementar

## Clase Profesor

**Atributos:**

* **id (String)** → Identificador único.
* **nombre (String)** → Nombre completo.
* **especialidad (String)** → Área principal.
* **List<Curso> cursos** → Cursos que dicta.

**Métodos sugeridos:**

* **agregarCurso(Curso c)** → Agrega el curso a su lista si no está y sincroniza el lado del curso.
* **eliminarCurso(Curso c)** → Quita el curso y sincroniza el lado del curso (dejar profesor en null si corresponde).
* **listarCursos()** → Muestra códigos y nombres.
* **mostrarInfo()** → Imprime datos del profesor y cantidad de cursos.

## Clase Curso

**Atributos:**

* **codigo (String)** → Código único.
* **nombre (String)** → Nombre del curso.
* **profesor (Profesor)** → Profesor responsable.

**Métodos sugeridos:**

* **setProfesor(Profesor p)** → Asigna/cambia el profesor **sincronizando ambos lados**: o Si tenía profesor previo, quitarse de su lista.
* **mostrarInfo()** → Muestra código, nombre y nombre del profesor (si tiene).

**Clase Universidad Atributos:**

* **String nombre**
* **List<Profesor> profesores**
* **List<Curso> cursos**

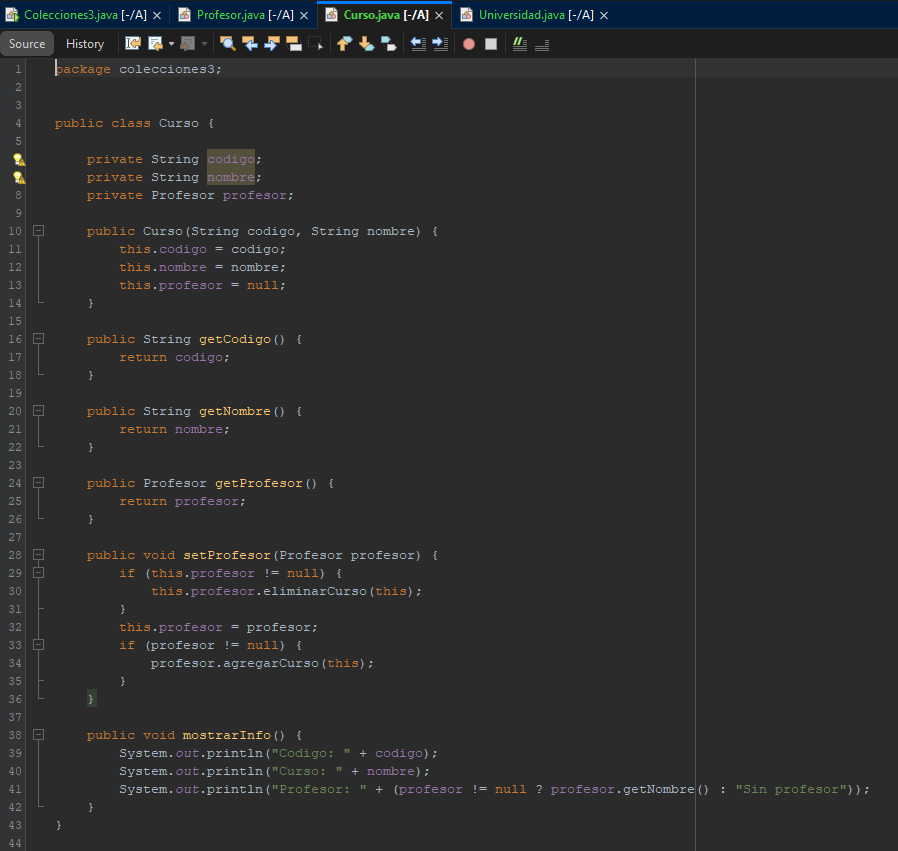
**Métodos requeridos:**

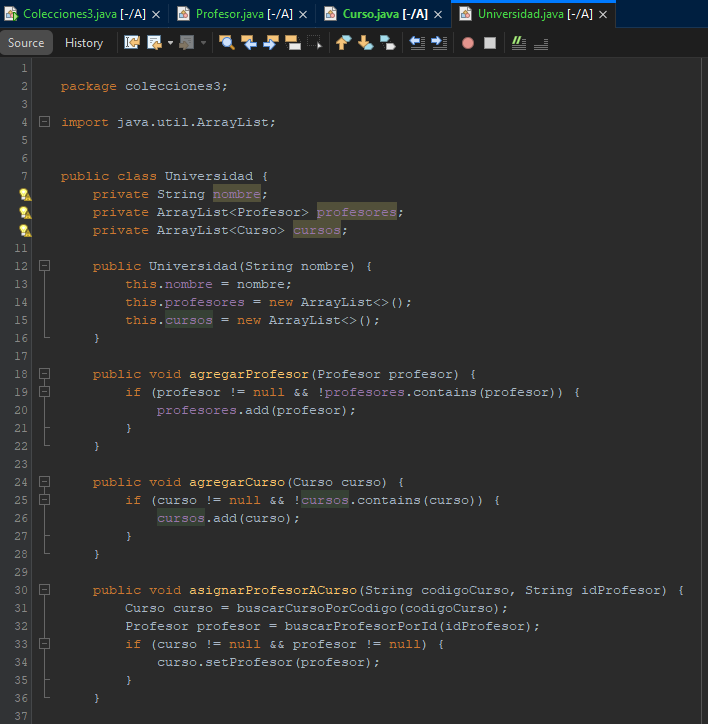
* **agregarProfesor(Profesor p)**
* **agregarCurso(Curso c)**
* **asignarProfesorACurso(String codigoCurso, String idProfesor)** → Usa setProfesor del curso.
* **listarProfesores() / listarCursos()**
* **buscarProfesorPorId(String id)**
* **buscarCursoPorCodigo(String codigo)**
* **eliminarCurso(String codigo)** → Debe **romper la relación** con su profesor si la hubiera.
* **eliminarProfesor(String id)** → Antes de remover, dejar null los cursos que dictaba.

## Tareas a realizar

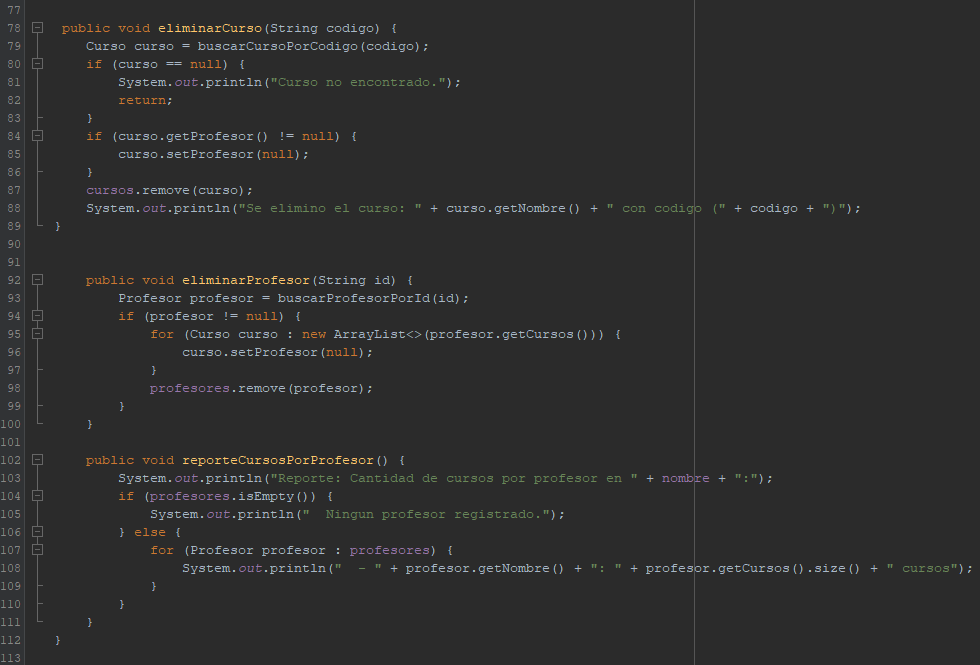
1. Crear **al menos 3 profesores** y **5 cursos**.
2. Agregar profesores y cursos a la universidad.
3. Asignar profesores a cursos usando asignarProfesorACurso(...).
4. Listar cursos con su profesor y profesores con sus cursos.
5. Cambiar el profesor de un curso y verificar que ambos lados quedan sincronizados.
6. Remover un curso y confirmar que ya **no** aparece en la lista del profesor.
7. Remover un profesor y dejar profesor = null,
8. 8. Mostrar un reporte: cantidad de cursos por profesor.

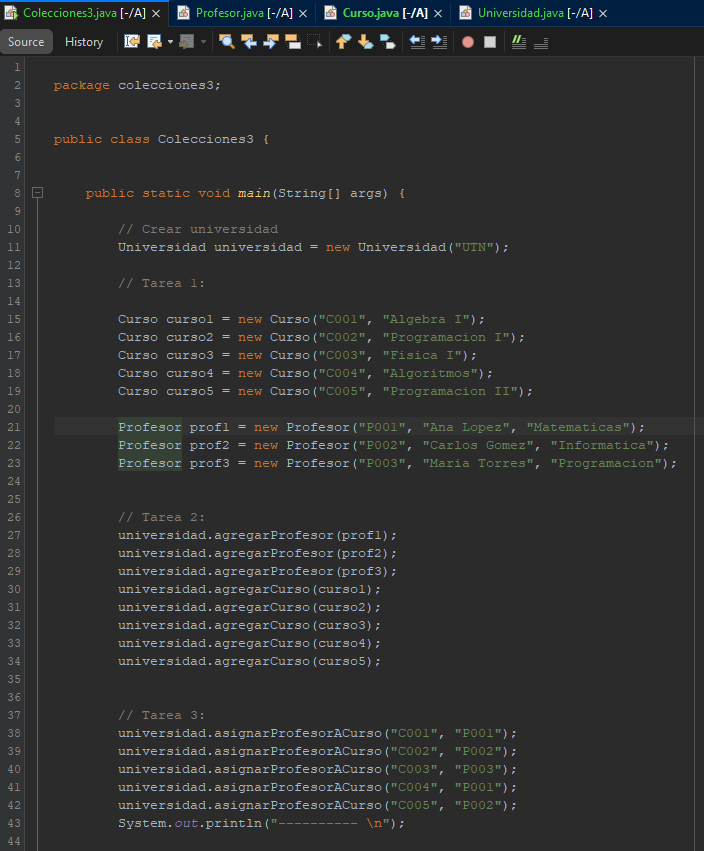




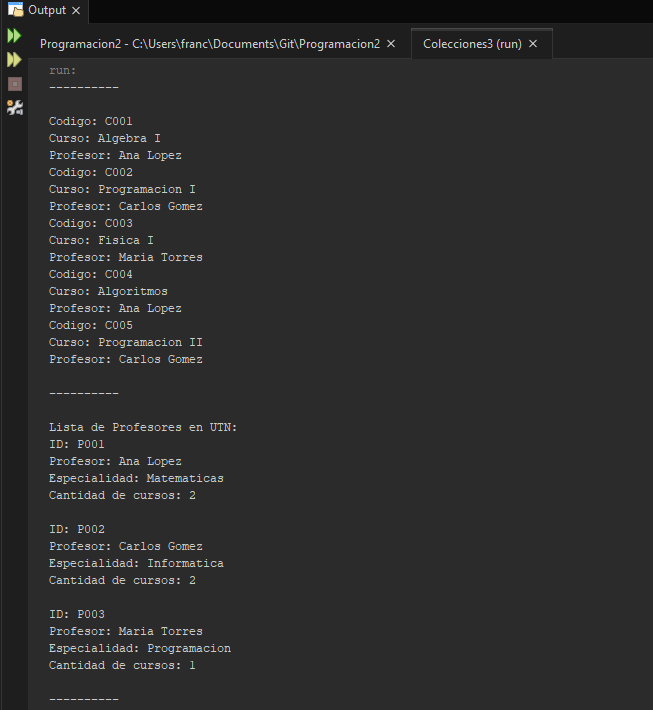


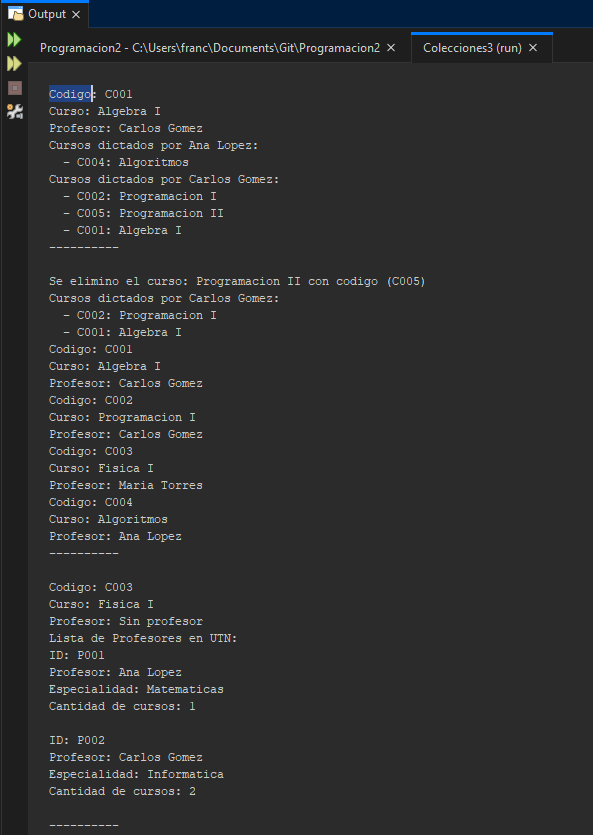


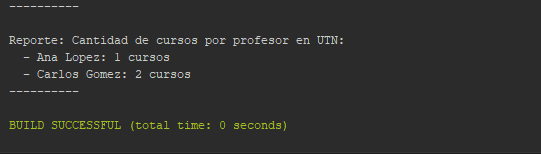












## Conclusiones esperadas

* Diferenciar **bidireccionalidad** de una relación unidireccional (navegación desde ambos extremos).
* Mantener **invariantes de asociación** (coherencia de referencias) al **agregar, quitar o reasignar**.
* Practicar colecciones (ArrayList), búsquedas y operaciones de alta/baja.
* Diseñar métodos “seguros” que **sincronicen** los dos lados siempre.