

PROGRAMACIÓN II

García Cintia.

Trabajo Práctico 6: Colecciones y Sistema de Stock

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..

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.

2. 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()

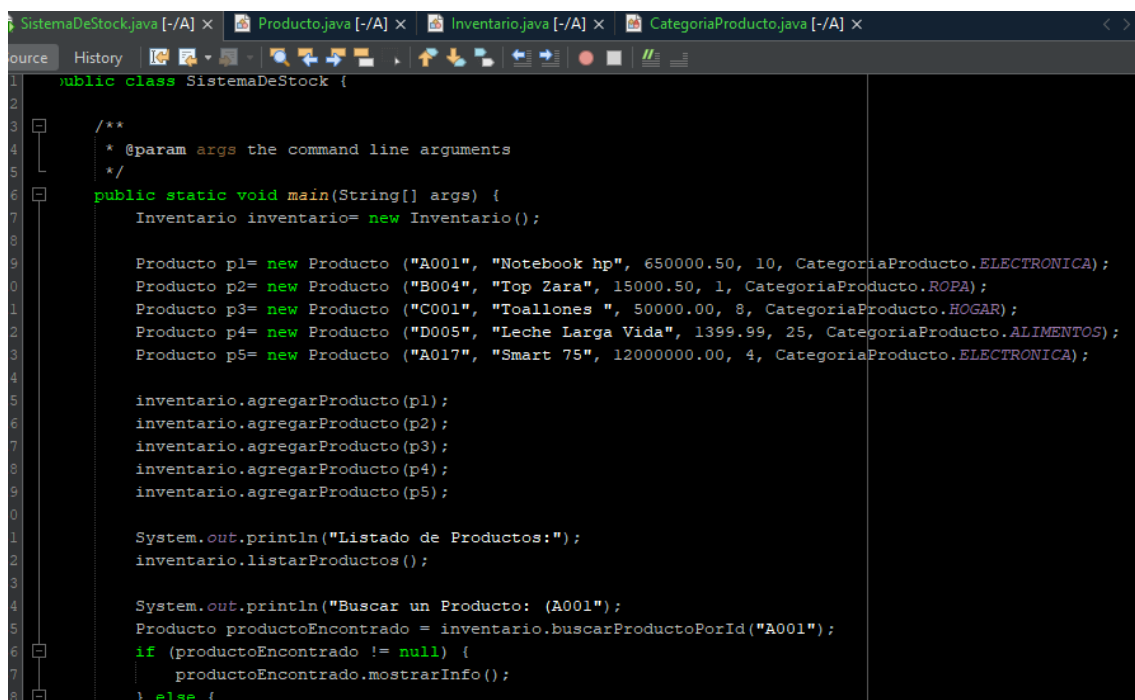
3. Tareas a realizar

1. Crear al menos cinco productos con diferentes categorías y agregarlos al inventario.
2. Listar todos los productos mostrando su información y categoría.

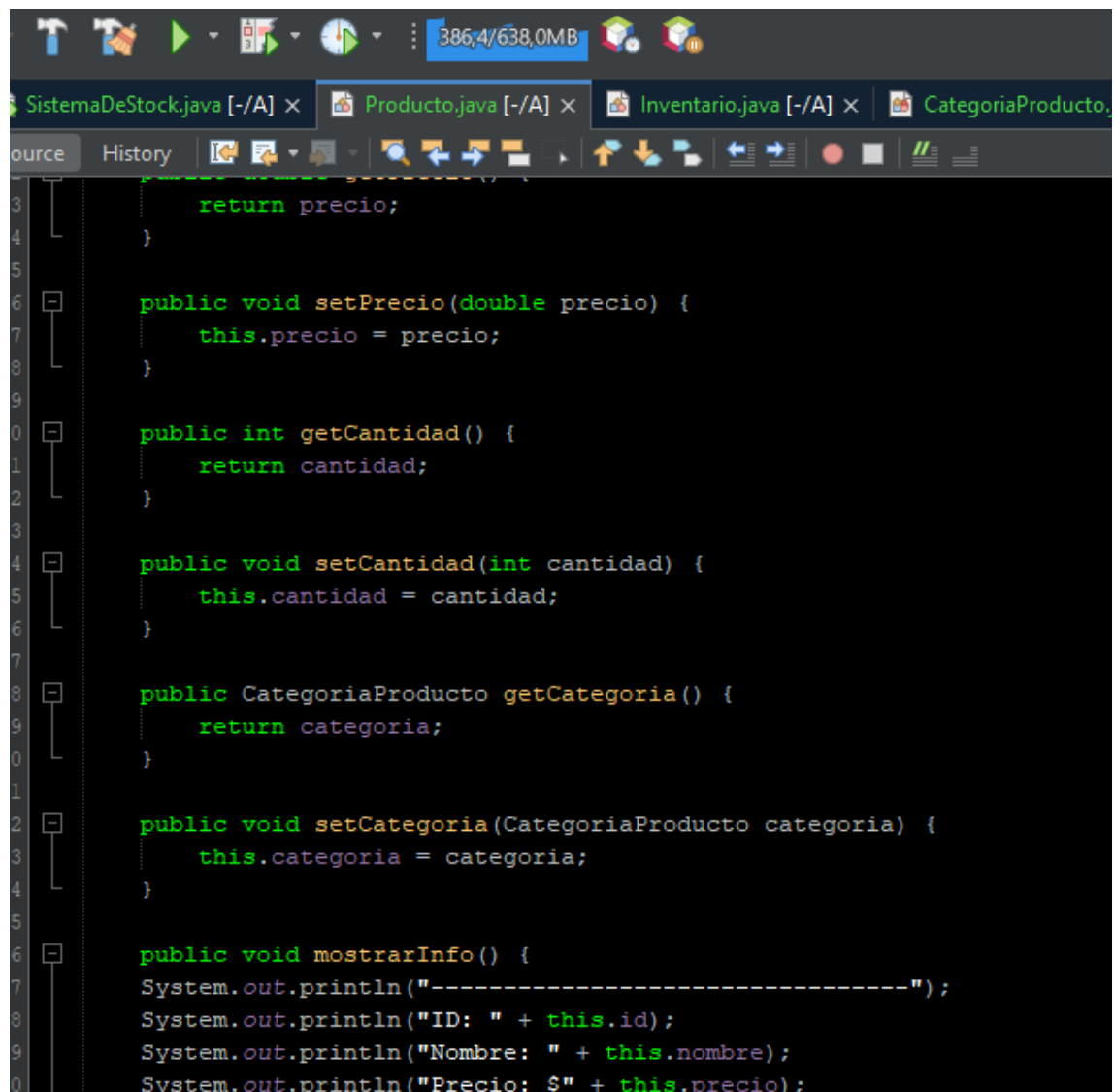
3. Buscar un producto por ID y mostrar su información.
4. Filtrar y mostrar productos que pertenezcan a una categoría específica.
5. Eliminar un producto por su ID y listar los productos restantes.
6. Actualizar el stock de un producto existente.
7. Mostrar el total de stock disponible.
8. Obtener y 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.



```
1 public class SistemaDeStock {
2
3     /**
4      * @param args the command line arguments
5      */
6     public static void main(String[] args) {
7         Inventario inventario= new Inventario();
8
9         Producto p1= new Producto ("A001", "Notebook hp", 650000.50, 10, CategoriaProducto.ELECTRONICA);
10        Producto p2= new Producto ("B004", "Top Zara", 15000.50, 1, CategoriaProducto.ROPA);
11        Producto p3= new Producto ("C001", "Toallones ", 50000.00, 8, CategoriaProducto.HOGAR);
12        Producto p4= new Producto ("D005", "Leche Larga Vida", 1399.99, 25, CategoriaProducto.ALIMENTOS);
13        Producto p5= new Producto ("A017", "Smart 75", 12000000.00, 4, CategoriaProducto.ELECTRONICA);
14
15        inventario.agregarProducto(p1);
16        inventario.agregarProducto(p2);
17        inventario.agregarProducto(p3);
18        inventario.agregarProducto(p4);
19        inventario.agregarProducto(p5);
20
21        System.out.println("Listado de Productos:");
22        inventario.listarProductos();
23
24        System.out.println("Buscar un Producto: (A001)");
25        Producto productoEncontrado = inventario.buscarProductoPorId("A001");
26        if (productoEncontrado != null) {
27            productoEncontrado.mostrarInfo();
28        } else {
```



```
3      return precio;
4    }
5
6    public void setPrecio(double precio) {
7        this.precio = precio;
8    }
9
10   public int getCantidad() {
11       return cantidad;
12   }
13
14   public void setCantidad(int cantidad) {
15       this.cantidad = cantidad;
16   }
17
18   public CategoriaProducto getCategory() {
19       return categoria;
20   }
21
22   public void setCategoria(CategoriaProducto categoria) {
23       this.categoria = categoria;
24   }
25
26   public void mostrarInfo() {
27       System.out.println("-----");
28       System.out.println("ID: " + this.id);
29       System.out.println("Nombre: " + this.nombre);
30       System.out.println("Precio: $" + this.precio);
31   }
```

```
emaDeStock.java [-/A] × Producto.java [-/A] × Inventario.java [-/A] × CategoriaProducto.java [-/A] ×
History
    total += p.getCantidad();
}
return total;
}
public Producto obtenerProductoConMayorStock() {
    if (this.productos.isEmpty()) {
        return null;
    }
    Producto productoMayorStock = this.productos.get(0);
    for (int i = 1; i < this.productos.size(); i++) {
        if (this.productos.get(i).getCantidad() > productoMayorStock.getCantidad())
            productoMayorStock = this.productos.get(i);
    }
    return productoMayorStock;
}
public void filtrarProductosPorPrecio(double min, double max) {
    System.out.println("Rango de Precios $" + min + " y $: " + max);
    boolean encontrado = false;
    for (Producto p : this.productos) {
        if (p.getPrecio() >= min && p.getPrecio() <= max) {
            p.mostrarInfo();
            encontrado = true;
        }
    }
    if (encontrado) {
```

```
 */
public enum CategoriaProducto {
    ALIMENTOS ("Productos Comestibles"),
    ROPA("Prendas de Vestir"),
    ELECTRONICA("Dispositivos Electronicos"),
    HOGAR("Articulos para el Hogar"),;
    private final String descripcion;

    private CategoriaProducto(String descripcion) {
        this.descripcion = descripcion;
    }

    public String getDescripcion() {
        return descripcion;
    }
}

run:
Producto: Notebook hpagregado
Producto: Top Zaraagregado
Producto: Toallones agregado
Producto: Leche Larga Vidaagregado
Producto: Smart 75agregado
Listado de Productos:
LISTA DE PRODUCTOS:
-----
```

```
HP - C:\Users\HP ×  SistemaDeStock (run) ×
ID: A001
Nombre: Notebook hp
Precio: $650000.5
Stock: 10
Categoria: Dispositivos Electronicos
-----
ID: B004
Nombre: Top Zara
Precio: $15000.5
Stock: 1
Categoria: Prendas de Vestir
-----
ID: C001
Nombre: Toallones
Precio: $50000.0
Stock: 8
Categoria: Articulos para el Hogar
-----
ID: D005
Nombre: Leche Larga Vida
Precio: $1399.99
Stock: 25
Categoria: Productos Comestibles
-----
ID: A017
```

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.

2. 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()

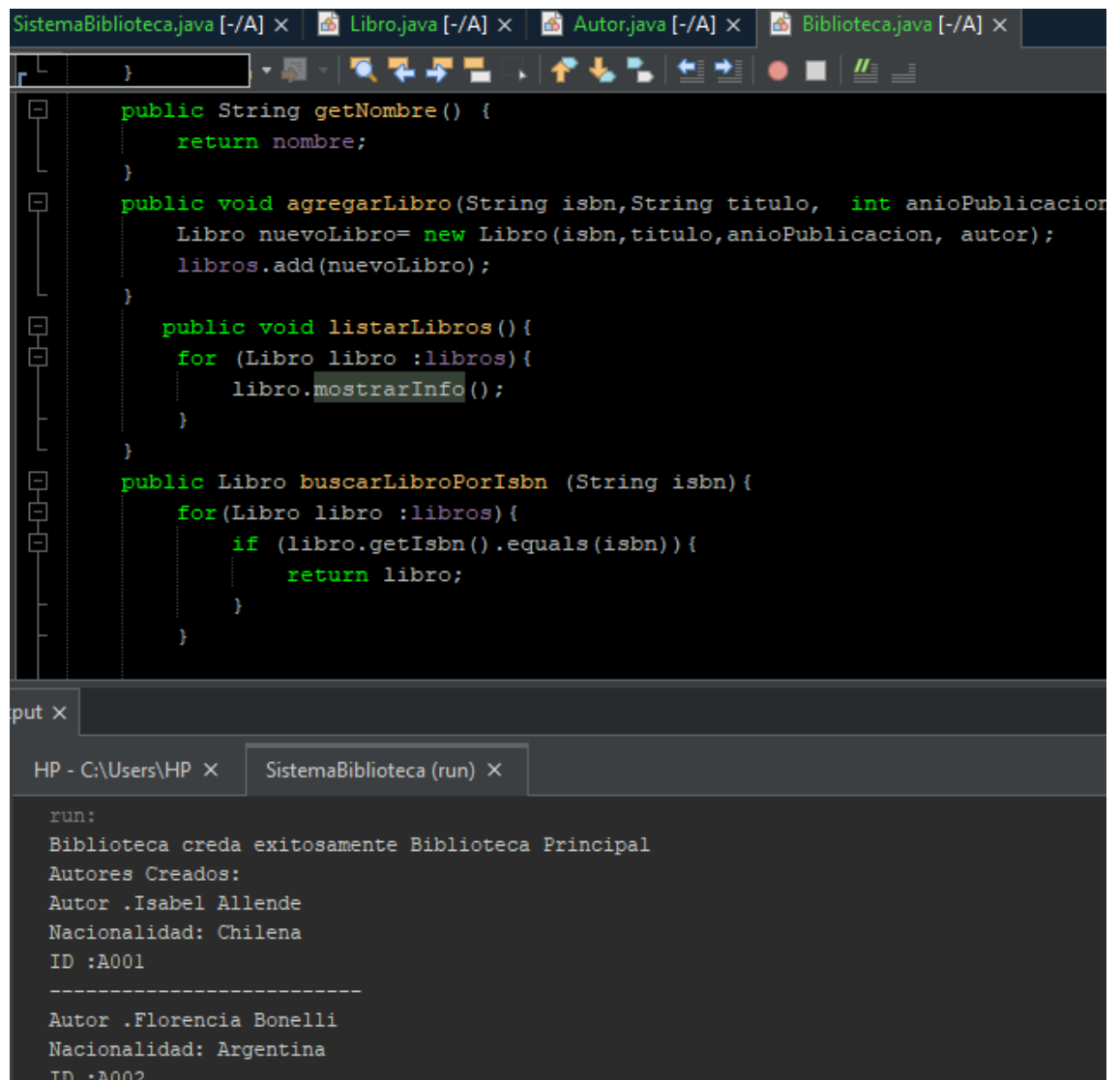
3. Tareas a realizar

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.



The screenshot shows an IDE with four tabs: SistemaBiblioteca.java, Libro.java, Autor.java, and Biblioteca.java. The SistemaBiblioteca.java tab is active, displaying the following Java code:

```
public String getNombre() {  
    return nombre;  
}  
  
public void agregarLibro(String isbn,String titulo, int anioPublicacion,  
    Libro nuevoLibro= new Libro(isbn,titulo,anioPublicacion, autor);  
    libros.add(nuevoLibro);  
}  
  
public void listarLibros(){  
    for (Libro libro :libros){  
        libro.mostrarInfo();  
    }  
}  
  
public Libro buscarLibroPorIsbn (String isbn){  
    for(Libro libro :libros){  
        if (libro.getIsbn().equals(isbn)){  
            return libro;  
        }  
    }  
}
```

Below the code editor, the output window shows the results of running the program:

```
run:  
Biblioteca creada exitosamente Biblioteca Principal  
Autores Creados:  
Autor .Isabel Allende  
Nacionalidad: Chilena  
ID :A001  
-----  
Autor .Florencia Bonelli  
Nacionalidad: Argentina  
ID :A002
```

```
SistemaBiblioteca.java [-/A] x Libro.java [-/A] x Autor.java [-/A] x Biblioteca.java [-/A] x
Source History
10 */
11 public class Autor {
12     private String id;
13     private String nombre;
14     private String nacionalidad;
15
16     public void mostrarInfo(){
17         System.out.println("Autor ." + nombre );
18         System.out.println("Nacionalidad: " + nacionalidad);
19         System.out.println("ID : " + id);
20         System.out.println("-----");
21     }
22
23     public Autor(String id, String nombre, String nacionalidad) {
24         this.id = id;
25         this.nombre = nombre;
26         this.nacionalidad = nacionalidad;
27     }
}
```

```
SistemaBiblioteca.java [-/A] x Libro.java [-/A] x Autor.java [-/A] x Biblioteca.java [-/A] x
Source History
    private String titulo;
    private int anioPublicacion;
    private Autor autor;

    public void mostrarInfo(){
        System.out.println("Titulo" + titulo );
        System.out.println("ISBN" + isbn);
        System.out.println("Año publicacion:" + anioPublicacion);
        autor.mostrarInfo ();
        System.out.println("-----");
    }

    public Libro(String isbn, String titulo, int anioPublicacion, Autor autor) {
        this.isbn = isbn;
        this.titulo = titulo;
        this.anioPublicacion = anioPublicacion;
        this.autor = autor;
    }
}
```

```
SistemaBiblioteca.java [-/A] x Libro.java [-/A] x Autor.java [-/A] x Biblioteca.java [-/A] x
source History
5 Libro libro2=new Libro ("ISBN02" , "Caballos de fuego", 2011, autor2);
6 Libro libro3=new Libro ("ISBN03" , "Indias Blancas", 2005, autor2);
7 Libro libro4=new Libro ("ISBN04" , "Los padecientes", 2011, autor3);
8 Libro libro5=new Libro ("ISBN05" , "La Felicidad", 2023, autor2);
9
10 System.out.println("Biblioteca creada exitosamente " + miBiblioteca.getNombre());
11 System.out.println("Autores Creados:");
12 autor1.mostrarInfo();
13 autor2.mostrarInfo();
14 autor3.mostrarInfo();
15
16 System.out.println("Libros creados:");
17 libro1.mostrarInfo();
18 libro2.mostrarInfo();
19 libro3.mostrarInfo();
20 libro4.mostrarInfo();
21 libro5.mostrarInfo();
22
```

```
HP - C:\Users\HP x SistemaBiblioteca (run) x
run:
Biblioteca creada exitosamente Bibliote
Autores Creados:
Autor .Isabel Allende
Nacionalidad: Chilena
ID :A001
-----
Autor .Florencia Bonelli
Nacionalidad: Argentina
ID :A002
-----
Autor .Gabriel Rolon
Nacionalidad: Argentino
ID :A003
-----
Libros creados:
TituloLa casa de los espíritus
ISBNISBN01
Fecha publicación:1982
```

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:
- 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. 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.

The screenshot shows an IDE with four tabs: `SistemaAcademicoBidireccional.java`, `Profesor.java`, `Curso.java`, and `Universidad.java`. The `Source` tab is active, displaying the `main` method of `SistemaAcademicoBidireccional.java`. The code initializes a `Universidad` object and creates three `Profesor` and five `Curso` objects. It then uses `uni.agregarProfesor` and `uni.agregarCurso` to associate them. The `Output` tab shows the execution results, including professor assignments and course listings.

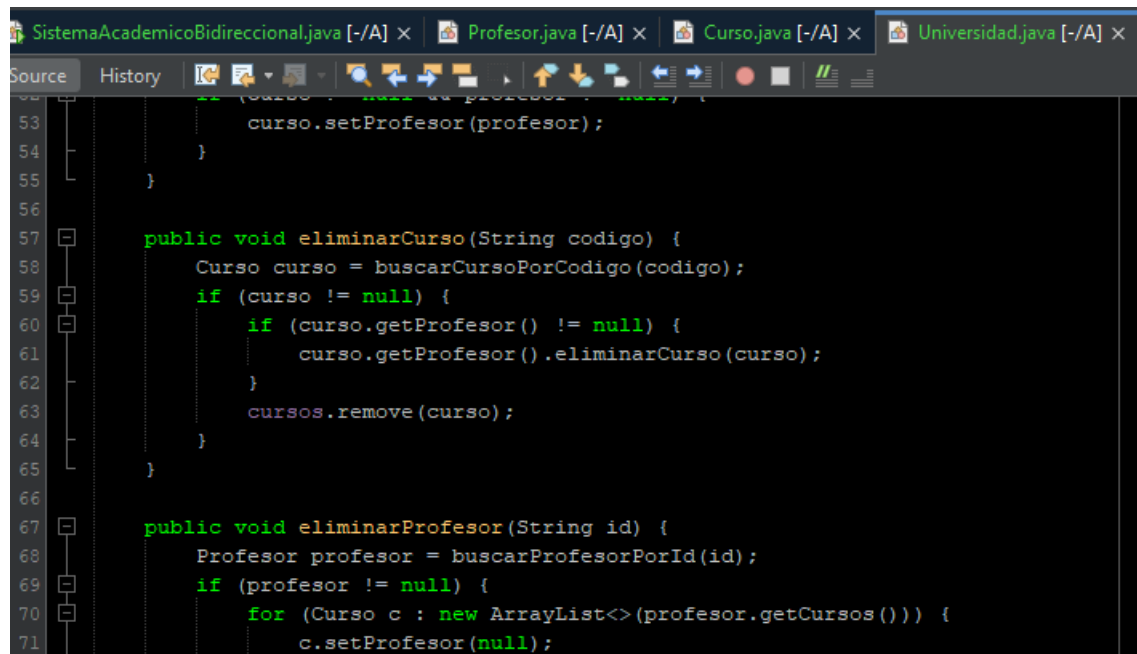
```
7 public static void main(String[] args) {
8     Universidad uni = new Universidad("Universidad Nacional");
9
10    Profesor p1 = new Profesor("P001", "Ana Torres", "Matemática");
11    Profesor p2 = new Profesor("P002", "Luis Gómez", "Historia");
12    Profesor p3 = new Profesor("P003", "María López", "Programación");
13
14    Curso c1 = new Curso("C101", "Álgebra");
15    Curso c2 = new Curso("C102", "Historia Antigua");
16    Curso c3 = new Curso("C103", "Java Básico");
17    Curso c4 = new Curso("C104", "Estadística");
18    Curso c5 = new Curso("C105", "Historia Moderna");
19
20    uni.agregarProfesor(p1);
21    uni.agregarProfesor(p2);
22    uni.agregarProfesor(p3);
23
24    uni.agregarCurso(c1);
25    uni.agregarCurso(c2);
26    uni.agregarCurso(c3);
```

Output:

```
Profesor a cargo:María López
-----
CursoEstadísticacodigoC104
Profesor a cargo:Ana Torres
-----
CursoHistoria ModernacodigoC105
Profesor a cargo:Luis Gómez
-----
```

```
SistemaAcademicoBidireccional.java [-/A] x Profesor.java [-/A] x Curso.java [-/A] x Universidad.jav
Source History
7 import java.util.ArrayList;
8 import java.util.List;
9
10 /**
11  *
12  * @author HP
13  */
14 public class Profesor {
15     private final String id;
16     private final String nombre;
17     private final String especialidad;
18     private final List<Curso> cursos;
19
20     public Profesor(String id, String nombre, String especialidad) {
21         this.id = id;
22         this.nombre = nombre;
23         this.especialidad = especialidad;
24         this.cursos = new ArrayList<>();
25     }
26 }
```

```
SistemaAcademicoBidireccional.java [-/A] x Profesor.java [-/A] x Curso.java [-/A] x Universidad.java [-/A] x
Source History
21 public void setProfesor(Profesor nuevoProfesor) {
22     if ( this.profesor != null && profesor != nuevoProfesor) {
23         this.profesor.eliminarCurso(this);
24     }
25     this.profesor = nuevoProfesor;
26
27     if (nuevoProfesor !=null && !nuevoProfesor.getCursos().contains(this)) {
28         nuevoProfesor.agregarCurso(this);
29     }
30 }
31
32 public void mostrarInfo() {
33     System.out.println("Curso"+ nombre + "codigo" + codigo);
34     if (profesor!=null) {
35         System.out.println("Profesor a cargo:"+ profesor.getNombre());
36     }
37     else {
38         System.out.println("Sin profesor asignado");
39     }
40 }
```



```
SistemaAcademicoBidireccional.java [-/A] x  Profesor.java [-/A] x  Curso.java [-/A] x  Universidad.java [-/A] x
Source  History  [Icons]
53      curso.setProfesor(profesor);
54  }
55  }
56
57  public void eliminarCurso(String codigo) {
58      Curso curso = buscarCursoPorCodigo(codigo);
59      if (curso != null) {
60          if (curso.getProfesor() != null) {
61              curso.getProfesor().eliminarCurso(curso);
62          }
63          cursos.remove(curso);
64      }
65  }
66
67  public void eliminarProfesor(String id) {
68      Profesor profesor = buscarProfesorPorId(id);
69      if (profesor != null) {
70          for (Curso c : new ArrayList<>(profesor.getCursos())) {
71              c.setProfesor(null);
```