Trabajo Practico N°6 Colecciones y Sistema de Stock

Materia: Programación II

Nombre: Leandro Lopez

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..

1. D***escripción 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***

Producto

```java
public class Producto {
    private String id;
    private String nombre;
    private double precio;
    private int cantidad; // Stock
    private CategoriaProducto categoria;

    // Constructor principal
    public Producto(String id, String nombre, double precio, int cantidad, CategoriaProducto categoria) {
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
        this.cantidad = cantidad;
        this.categoria = categoria;
    }

    // Constructor sobrecargado (ejemplo: para crear un producto sin stock inicial)
    public Producto(String id, String nombre, double precio, CategoriaProducto categoria) {
        this(id, nombre, precio, 0, categoria); // Llama al constructor principal
    }

    // Getters y Setters
    public String getId() {
        return id;
    }

    public String getNombre() {
        return nombre;
    }
```

```java
    public double getPrecio() {
        return precio;
    }

    public int getCantidad() {
        return cantidad;
    }

    public void setCantidad(int cantidad) {
        this.cantidad = cantidad;
    }

    public CategoriaProducto getCategoria() {
        return categoria;
    }

    public void mostrarInfo() {
        System.out.println("--- Producto ---");
        System.out.println("ID: " + this.id);
        System.out.println("Nombre: " + this.nombre);
        System.out.println("Precio: $" + String.format("%.2f", this.precio));
        System.out.println("Stock: " + this.cantidad + " unidades");
        System.out.println("Categoría: " + this.categoria + " (" + this.categoria.getDescripcion() + ")");
    }

    @Override
    public String toString() {
        return "ID: " + id + " | Nombre: " + nombre + " | Precio: $" + String.format("%.2f", precio) +
               " | Stock: " + cantidad + " | Categoría: " + categoria;
    }
```

Enum CategoriaProducto

```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

```java
public class Inventario {
    private final ArrayList<Producto> productos;

    public Inventario() {
        this.productos = new ArrayList<>();
    }

    // Método para agregar un producto (requerido)
    public void agregarProducto(Producto p) {
        // Validación simple para evitar IDs duplicados
        if (buscarProductoPorId(p.getId()) == null) {
            productos.add(p);
            System.out.println("Producto agregado: " + p.getNombre());
        } else {
            System.out.println("ERROR: Ya existe un producto con el ID: " + p.getId());
        }
    }

    // Método para listar todos los productos (requerido)
    public void listarProductos() {
        System.out.println("\n--- LISTA DE PRODUCTOS EN INVENTARIO (" + productos.size() + " en total) ---");
        if (productos.isEmpty()) {
            System.out.println("El inventario está vacío.");
            return;
        }
        for (Producto p : productos) {
            System.out.println(p); // Usa el método toString()
        }
    }
```

```java
    // Método para buscar un producto por ID
    public Producto buscarProductoPorId(String id) {
        for (Producto p : productos) {
            if (p.getId().equals(id)) {
                return p;
            }
        }
        return null; // Retorna null si no se encuentra
    }

    // Método para eliminar un producto por ID
    public boolean eliminarProducto(String id) {
        Producto p = buscarProductoPorId(id);
        if (p != null) {
            productos.remove(p);
            System.out.println("Producto eliminado: " + p.getNombre() + " (ID: " + id + ")");
            return true;
        }
        System.out.println("ERROR: No se encontró un producto con el ID: " + id + " para eliminar.");
        return false;
    }
```

```java
    public boolean actualizarStock(String id, int nuevaCantidad) {
        Producto p = buscarProductoPorId(id);
        if (p != null) {
            int stockAnterior = p.getCantidad();
            p.setCantidad(nuevaCantidad);
            System.out.println("Stock actualizado para " + p.getNombre() +
                               " (ID: " + id + "). Stock anterior: " + stockAnterior +
                               " | Nuevo stock: " + nuevaCantidad);
            return true;
        }
        System.out.println("ERROR: No se encontró un producto con el ID: " + id + " para actualizar stock.");
        return false;
    }

    // Método para filtrar por categoría
    public ArrayList<Producto> filtrarPorCategoria(CategoriaProducto categoria) {
        ArrayList<Producto> filtrados = new ArrayList<>();
        System.out.println("\n--- PRODUCTOS FILTRADOS POR CATEGORÍA: " + categoria + " ---");
        for (Producto p : productos) {
            if (p.getCategoria() == categoria) {
                filtrados.add(p);
            }
        }
        if (filtrados.isEmpty()) {
            System.out.println("No se encontraron productos en la categoría " + categoria + ".");
        } else {
            for (Producto p : filtrados) {
                System.out.println(p);
            }
        }
        return filtrados;
```

```java
    // Método para obtener el total de stock
    public int obtenerTotalStock() {
        int total = 0;
        for (Producto p : productos) {
            total += p.getCantidad();
        }
        return total;
    }

    // Método para obtener el producto con mayor stock
    public Producto obtenerProductoConMayorStock() {
        if (productos.isEmpty()) {
            return null;
        }
        // Usamos una expresión Lambda y el Comparator para simplificar la búsqueda del máximo
        return productos.stream()
                .max(Comparator.comparingInt(Producto::getCantidad))
                .orElse(null);
```

```java
    // Método para filtrar productos por rango de precio
    public ArrayList<Producto> filtrarProductosPorPrecio(double min, double max) {
        ArrayList<Producto> filtrados = new ArrayList<>();
        System.out.println("\n--- PRODUCTOS FILTRADOS POR PRECIO ENTRE $" + String.format("%.2f", min) +
                           " y $" + String.format("%.2f", max) + " ---");
        for (Producto p : productos) {
            if (p.getPrecio() >= min && p.getPrecio() <= max) {
                filtrados.add(p);
                System.out.println(p);
            }
        }
        if (filtrados.isEmpty()) {
            System.out.println("No se encontraron productos en ese rango de precio.");
        }
        return filtrados;
    }

    // Método para mostrar las categorías disponibles con sus descripciones
    public void mostrarCategoriasDisponibles() {
        System.out.println("\n--- CATEGORÍAS DE PRODUCTO DISPONIBLES ---");
        for (CategoriaProducto cat : CategoriaProducto.values()) {
            System.out.println(cat + ": " + cat.getDescripcion());
        }
    }
}
```

3. Tareas a realizar

1. Crear al menos cinco productos con diferentes categorías y agregarlos al inventario.

```
//Crear al menos cinco productos y agregarlos al inventario.
System.out.println("\n--- TAREA 1: AGREGAR PRODUCTOS ---");
Producto p1 = new Producto("A001", "Laptop Gamer", 2500.50, 10, CategoriaProducto.ELECTRONICA);
Producto p2 = new Producto("B002", "Camiseta Algodón", 55.99, 50, CategoriaProducto.ROPA);
Producto p3 = new Producto("C003", "Mesa de Centro", 1200.00, 5, CategoriaProducto.HOGAR);
Producto p4 = new Producto("D004", "Leche Entera", 1.25, 200, CategoriaProducto.ALIMENTOS);
Producto p5 = new Producto("E005", "Smartphone X", 1850.75, 25, CategoriaProducto.ELECTRONICA);
Producto p6 = new Producto("F006", "Silla Ergonómica", 3200.90, 8, CategoriaProducto.HOGAR); // Otro producto para la tarea 9

inventario.agregarProducto(p1);
inventario.agregarProducto(p2);
inventario.agregarProducto(p3);
inventario.agregarProducto(p4);
inventario.agregarProducto(p5);
inventario.agregarProducto(p6);
```

2. Listar todos los productos mostrando su información y categoría.

```
// Listar todos los productos mostrando su información y categoría.
// Se usa el método toString() del Producto al listarlos.
inventario.listarProductos();
```

3. Buscar un producto por ID y mostrar su información.

```
System.out.println("\n--- TAREA 3: BUSCAR PRODUCTO POR ID (A001) ---");
Producto buscado = inventario.buscarProductoPorId("A001");
if (buscado != null) {
    buscado.mostrarInfo(); // Usa el método mostrarInfo()
} else {
    System.out.println("Producto no encontrado.");
}
```

4. Filtrar y mostrar productos que pertenezcan a una categoría específica.

```
//Filtrar y mostrar productos que pertenezcan a una categoría específica (ROPA).
inventario.filtrarPorCategoria(CategoriaProducto.ROPA);
```

5. Eliminar un producto por su ID y listar los productos restantes.

```
//Eliminar un producto por su ID (D004) y listar los productos restantes.
System.out.println("\n--- TAREA 5: ELIMINAR PRODUCTO (D004) ---");
inventario.eliminarProducto("D004");
inventario.listarProductos();
```

6. Actualizar el stock de un producto existente.

```
//ctualizar el stock de un producto existente (E005).
System.out.println("\n--- TAREA 6: ACTUALIZAR STOCK (E005) ---");
inventario.actualizarStock("E005", 35);
```

7. Mostrar el total de stock disponible.

```
//Mostrar el total de stock disponible.
System.out.println("\n--- TAREA 7: OBTENER TOTAL STOCK ---");
System.out.println("Total de stock disponible en el inventario: " + inventario.obtenerTotalStock() + " unidades.");
```

8. Obtener y mostrar el producto con mayor stock.

```
//Obtener y mostrar el producto con mayor stock.
System.out.println("\n--- TAREA 8: PRODUCTO CON MAYOR STOCK ---");
Producto mayorStock = inventario.obtenerProductoConMayorStock();
if (mayorStock != null) {
    System.out.println("Producto con mayor stock:");
    System.out.println(mayorStock);
} else {
    System.out.println("El inventario está vacío.");
}
```

9. Filtrar productos con precios entre $1000 y $3000.

```
// Filtrar productos con precios entre $1000 y $3000.
inventario.filtrarProductosPorPrecio(1000.00, 3000.00);
```

10. Mostrar las categorías disponibles con sus descripciones.

```
//Mostrar las categorías disponibles con sus descripciones.
inventario.mostrarCategoriasDisponibles();
```

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

```java
public class Autor {
    private String id;
    private String nombre;
    private String nacionalidad;

    // Constructor
    public Autor(String id, String nombre, String nacionalidad) {
        this.id = id;
        this.nombre = nombre;
        this.nacionalidad = nacionalidad;
    }

    // Getters requeridos para mostrar información en Libro/Biblioteca
    public String getNombre() {
        return nombre;
    }

    public String getId() {
        return id;
    }

    // Método para mostrar información del autor
    public void mostrarInfo() {
        System.out.println("--- Información del Autor ---");
        System.out.println("ID: " + this.id);
        System.out.println("Nombre: " + this.nombre);
        System.out.println("Nacionalidad: " + this.nacionalidad);
    }
```

```java
    // Sobrescritura de toString() para mejor visualización
    @Override
    public String toString() {
        return nombre + " (ID: " + id + ", Nacionalidad: " + nacionalidad + ")";
    }
}
```

Clase Libro

```java
public class Libro {
    private String isbn;
    private String titulo;
    private int anioPublicacion;
    private Autor autor; // Relación de un Libro a un Autor

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

    // Getters
    public String getIsbn() {
        return isbn;
    }

    public int getAnioPublicacion() {
        return anioPublicacion;
    }

    public Autor getAutor() {
        return autor;
    }
```

```java
    // Método para mostrar la información completa del libro y su autor
    public void mostrarInfo() {
        System.out.println("----------------------------------------");
        System.out.println("Título: " + this.titulo);
        System.out.println("ISBN: " + this.isbn);
        System.out.println("Año de Publicación: " + this.anioPublicacion);
        System.out.println("Autor: " + this.autor.toString() + ")");
        System.out.println("----------------------------------------");
    }

    // Sobrescritura de toString() para listados
    @Override
    public String toString() {
        return "ISBN: " + isbn + " | Título: " + titulo + " | Año: " + anioPublicacion +
               " | Autor: " + autor.getNombre();
    }
}
```

Clase Biblioteca

```java
public class Biblioteca {
    private String nombre;
    private List<Libro> libros; // Relación de composición 1 a N

    // Constructor
    public Biblioteca(String nombre) {
        this.nombre = nombre;
        // La composición se asegura inicializando la colección dentro de la clase contenedora.
        this.libros = new ArrayList<>();
        System.out.println("Biblioteca '" + nombre + "' creada.");
    }

    // ------------------- MÉTODOS REQUERIDOS -------------------

    // Método para agregar un Libro
    public void agregarLibro(String isbn, String titulo, int anioPublicacion, Autor autor) {
        // Validación para evitar duplicados por ISBN
        if (buscarLibroPorIsbn(isbn) == null) {
            Libro nuevoLibro = new Libro(isbn, titulo, anioPublicacion, autor);
            libros.add(nuevoLibro);
            System.out.println("✅ Libro agregado: " + titulo);
        } else {
            System.out.println("❌ ERROR: Ya existe un libro con el ISBN: " + isbn);
        }
    }
```

```java
    public void agregarLibro(String isbn, String titulo, int anioPublicacion, Autor autor) {
        // Validación para evitar duplicados por ISBN
        if (buscarLibroPorIsbn(isbn) == null) {
            Libro nuevoLibro = new Libro(isbn, titulo, anioPublicacion, autor);
            libros.add(nuevoLibro);
            System.out.println("Libro agregado: " + titulo);
        } else {
            System.out.println("ERROR: Ya existe un libro con el ISBN: " + isbn);
        }
    }

    // Método para listar todos los libros
    public void listarLibros() {
        System.out.println("\n--- LISTADO DE LIBROS EN '" + nombre + "' (" + libros.size() + " en total) ---");
        if (libros.isEmpty()) {
            System.out.println("La biblioteca no contiene libros.");
            return;
        }
        for (Libro libro : libros) {
            System.out.println(libro); // Usa el toString()
        }
    }

    // Método para buscar un libro por ISBN
    public Libro buscarLibroPorIsbn(String isbn) {
        for (Libro libro : libros) {
            if (libro.getIsbn().equals(isbn)) {
                return libro;
            }
        }
        return null; // Retorna null si no se encuentra
    }
```

```java
    // Método para eliminar un libro por ISBN
    public boolean eliminarLibro(String isbn) {
        Libro libroAEliminar = buscarLibroPorIsbn(isbn);
        if (libroAEliminar != null) {
            libros.remove(libroAEliminar);
            // La composición se asegura: al eliminarse de la colección, el objeto Libro es candidato a ser recogido por el Garbage Collector.
            System.out.println("Libro eliminado: " + libroAEliminar.toString()+ " (ISBN: " + isbn + ")");
            return true;
        }
        System.out.println("❌ ERROR: No se encontró un libro con el ISBN: " + isbn + " para eliminar.");
        return false;
    }

    // Método para obtener la cantidad total de libros
    public int obtenerCantidadLibros() {
        return libros.size();
    }
}
```

```java
    // Método para filtrar libros por año de publicación
    public List<Libro> filtrarLibrosPorAnio(int anio) {
        List<Libro> filtrados = new ArrayList<>();
        System.out.println("\n--- LIBROS PUBLICADOS EN EL AÑO " + anio + " ---");
        for (Libro libro : libros) {
            if (libro.getAnioPublicacion() == anio) {
                filtrados.add(libro);
                System.out.println(libro);
            }
        }
        if (filtrados.isEmpty()) {
            System.out.println("No se encontraron libros publicados en " + anio + ".");
        }
        return filtrados;
    }
```

```java
    // Método para listar todos los autores ÚNICOS disponibles en la biblioteca
    public void mostrarAutoresDisponibles() {
        // Usamos un HashSet para garantizar que cada autor se muestre solo una vez,
        // incluso si escribió varios libros.
        Set<Autor> autoresUnicos = new HashSet<>();
        for (Libro libro : libros) {
            autoresUnicos.add(libro.getAutor());
        }

        System.out.println("\n--- AUTORES DISPONIBLES EN LA BIBLIOTECA (" + autoresUnicos.size() + " únicos) ---");
        if (autoresUnicos.isEmpty()) {
            System.out.println("No hay autores registrados.");
            return;
        }
        int i = 1;
        for (Autor autor : autoresUnicos) {
            System.out.println(i++ + ". " + autor); // Usa el toString() de Autor
        }
    }
}
```

3. Tareas a realizar

1. Creamos una biblioteca.

```java
    // 1. Crear una biblioteca.
    Biblioteca bibliotecaCentral = new Biblioteca("Biblioteca Central UNQ");
```

2. Crear al menos tres autores

```java
    // 2. Crear al menos tres autores
    System.out.println("\n--- TAREA 2: CREACIÓN DE AUTORES ---");
    Autor autor1 = new Autor("A001", "Gabriel García Márquez", "Colombiana");
    Autor autor2 = new Autor("A002", "Jane Austen", "Británica");
    Autor autor3 = new Autor("A003", "Jorge Luis Borges", "Argentina");
```

3. Agregar 5 libros asociados a alguno de los Autores a la biblioteca.

```
// 3. Agregar 5 libros asociados a alguno de los Autores a la biblioteca.
System.out.println("\n--- TAREA 3: AGREGAR LIBROS ---");
bibliotecaCentral.agregarLibro("978-001", "Cien años de soledad", 1967, autor1);
bibliotecaCentral.agregarLibro("978-002", "Orgullo y prejuicio", 1813, autor2);
bibliotecaCentral.agregarLibro("978-003", "Ficciones", 1944, autor3);
bibliotecaCentral.agregarLibro("978-004", "El amor en los tiempos del cólera", 1985, autor1); // Mismo autor1
bibliotecaCentral.agregarLibro("978-005", "Emma", 1815, autor2); // Mismo autor2
```

4. Listar todos los libros con su información y la del autor.

```
// 4. Listar todos los libros con su información y la del autor.
bibliotecaCentral.listarLibros();
```

5. Buscar un libro por su ISBN y mostrar su información.

```
// 5. Buscar un libro por su ISBN y mostrar su información.
System.out.println("\n--- TAREA 5: BUSCAR LIBRO POR ISBN (978-002) ---");
Libro libroBuscado = bibliotecaCentral.buscarLibroPorIsbn("978-002");
if (libroBuscado != null) {
    libroBuscado.mostrarInfo();
} else {
    System.out.println("Libro no encontrado.");
}
```

6. Filtrar y mostrar los libros publicados en un año específico.

```
// 6. Filtrar y mostrar los libros publicados en un año específico (1815).
bibliotecaCentral.filtrarLibrosPorAnio(1815);
```

7. Eliminar un libro por su ISBN y listar los libros restantes.

```
// 7. Eliminar un libro por su ISBN (978-001) y listar los libros restantes.
System.out.println("\n--- TAREA 7: ELIMINAR LIBRO (978-001) ---");
bibliotecaCentral.eliminarLibro("978-001");
bibliotecaCentral.listarLibros();
```

8. Mostrar la cantidad total de libros en la biblioteca.

```
// 8. Mostrar la cantidad total de libros en la biblioteca.
System.out.println("\n--- TAREA 8: CANTIDAD TOTAL DE LIBROS ---");
System.out.println("La biblioteca tiene un total de " + bibliotecaCentral.obtenerCantidadLibros() + " libros disponibles.");
```

9. Listar todos los autores de los libros disponibles en la biblioteca.

```
// 9. Listar todos los autores de los libros disponibles en la biblioteca.
// Después de eliminar el libro de García Márquez, este autor aún debe aparecer porque tiene otro libro (978-004).
bibliotecaCentral.mostrarAutoresDisponibles();
```

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).

Clase Profesor

```java
public class Profesor {
    private String id;
    private String nombre;
    private String especialidad;
    private List<Curso> cursos; // Lado 'N' de la relación (bidireccional)

    public Profesor(String id, String nombre, String especialidad) {
        this.id = id;
        this.nombre = nombre;
        this.especialidad = especialidad;
        this.cursos = new ArrayList<>(); // Inicialización para la composición
    }

    // Getters
    public String getId() {
        return id;
    }

    public String getNombre() {
        return nombre;
    }

    public List<Curso> getCursos() {
        return cursos;
    }
```

```java
    // Método sugerido: Agrega un curso y Sincroniza el lado del Curso
    public void agregarCurso(Curso c) {
        // 1. Agrega a la lista del profesor (si no está)
        if (!this.cursos.contains(c)) {
            this.cursos.add(c);
        }
        // 2. Sincroniza el lado del Curso: llama a setProfesor para que el Curso apunte a 'this'
        // y maneje la eliminación del profesor anterior si existiera.
        if (c.getProfesor() != this) {
            c.setProfesor(this);
        }
    }

    // Método sugerido: Quita un curso y Sincroniza el lado del Curso
    public void eliminarCurso(Curso c) {
        // 1. Quita de la lista del profesor
        if (this.cursos.remove(c)) {
            // 2. Sincroniza el lado del Curso: solo si el curso realmente apuntaba a 'this'
            if (c.getProfesor() == this) {
                // Esto llamará a setProfesor(null) sin volver a llamarnos a nosotros
                c.removerProfesor();
            }
        }
    }
```

```java
    // Muestra códigos y nombres de los cursos
    public void listarCursos() {
        System.out.println("\n--- Cursos dictados por " + nombre + " (" + cursos.size() + " en total) ---");
        if (cursos.isEmpty()) {
            System.out.println("Este profesor no dicta cursos actualmente.");
            return;
        }
        for (Curso c : cursos) {
            System.out.println("  [" + c.getCodigo() + "] " + c.getNombre());
        }
    }

    // Imprime datos del profesor y cantidad de cursos
    public void mostrarInfo() {
        System.out.println("\n=== Profesor ===");
        System.out.println("ID: " + id);
        System.out.println("Nombre: " + nombre);
        System.out.println("Especialidad: " + especialidad);
        System.out.println("Cursos a cargo: " + cursos.size());
    }

    // Sobrescritura para facilitar listados
    @Override
    public String toString() {
        return "ID: " + id + " | Nombre: " + nombre + " | Especialidad: " + especialidad;
    }
}
```

Clase Curso

```java
public class Curso {
    private String codigo;
    private String nombre;
    private Profesor profesor; // Lado '1' de la relación (bidireccional)

    public Curso(String codigo, String nombre) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.profesor = null; // Inicialmente sin profesor asignado
    }

    // Getters
    public String getCodigo() {
        return codigo;
    }

    public String getNombre() {
        return nombre;
    }

    public Profesor getProfesor() {
        return profesor;
    }
```

```java
    // Método sugerido: Asigna/cambia el profesor sincronizando ambos lados.
    public void setProfesor(Profesor nuevoProfesor) {
        // Caso 1: Quitarse del profesor anterior (si existe y es diferente al nuevo)
        if (this.profesor != null && this.profesor != nuevoProfesor) {
            // Quitarse de la lista del profesor anterior sin volver a llamar a setProfesor
            this.profesor.getCursos().remove(this);
        }

        // 2. Asignar el nuevo profesor (puede ser null)
        this.profesor = nuevoProfesor;

        // 3. Sincronizar el lado del nuevo profesor (si no es null)
        if (nuevoProfesor != null) {
            // Llamar a agregarCurso para que el nuevo profesor nos añada a su lista
            // (Esta llamada ya tiene lógica para evitar bucles infinitos)
            nuevoProfesor.agregarCurso(this);
        }
    }
```

```java
    // Método auxiliar para romper la relación sin llamar a setProfesor(null) directamente
    // y evitar recursión infinita o bugs de doble eliminación.
    public void removerProfesor() {
        // 1. Quitarse del profesor anterior (si existe)
        if (this.profesor != null) {
            // Remover sin llamar al método setProfesor(null) ni eliminarCurso,
            // ya que estas llamadas provienen de setProfesor o eliminarCurso
            this.profesor.getCursos().remove(this);
        }
        // 2. Asignar null
        this.profesor = null;
    }

    // Muestra código, nombre y nombre del profesor
    public void mostrarInfo() {
        String nombreProfesor = (profesor != null) ? profesor.getNombre() : "SIN ASIGNAR";
        System.out.println("Curso: [" + codigo + "] " + nombre + " | Profesor: " + nombreProfesor);
    }

    @Override
    public String toString() {
        return "Código: " + codigo + " | Nombre: " + nombre;
    }
}
```

Clase Universidad

```java
public class Universidad {
    private String nombre;
    private List<Profesor> profesores;
    private List<Curso> cursos;

    public Universidad(String nombre) {
        this.nombre = nombre;
        this.profesores = new ArrayList<>();
        this.cursos = new ArrayList<>();
        System.out.println("Sistema de la " + nombre + " iniciado.");
    }

    // ------------------- MÉTODOS REQUERIDOS -------------------

    public void agregarProfesor(Profesor p) {
        if (buscarProfesorPorId(p.getId()) == null) {
            profesores.add(p);
            System.out.println("Profesor agregado: " + p.getNombre());
        } else {
            System.out.println("ERROR: Profesor con ID " + p.getId() + " ya existe.");
        }
    }

    public void agregarCurso(Curso c) {
        if (buscarCursoPorCodigo(c.getCodigo()) == null) {
            cursos.add(c);
            System.out.println("Curso agregado: " + c.getNombre());
        } else {
            System.out.println("ERROR: Curso con código " + c.getCodigo() + " ya existe.");
        }
    }
```

```java
    // Usa setProfesor del curso (clave para la sincronización)
    public boolean asignarProfesorACurso(String codigoCurso, String idProfesor) {
        Curso c = buscarCursoPorCodigo(codigoCurso);
        Profesor p = buscarProfesorPorId(idProfesor);

        if (c == null) {
            System.out.println("ERROR: Curso con código " + codigoCurso + " no encontrado.");
            return false;
        }
        if (p == null) {
            System.out.println("ERROR: Profesor con ID " + idProfesor + " no encontrado.");
            return false;
        }

        // La llamada a setProfesor es la que se encarga de la lógica bidireccional
        c.setProfesor(p);
        System.out.println("Profesor " + p.getNombre() + " asignado a " + c.getNombre());
        return true;
    }

    public void listarProfesores() {
        System.out.println("\n--- LISTADO DE PROFESORES (" + profesores.size() + " en total) ---");
        for (Profesor p : profesores) {
            System.out.println(p);
        }
    }
```

```java
public void listarCursos() {
    System.out.println("\n--- LISTADO DE CURSOS (" + cursos.size() + " en total) ---");
    for (Curso c : cursos) {
        c.mostrarInfo();
    }
}

public Profesor buscarProfesorPorId(String id) {
    for (Profesor p : profesores) {
        if (p.getId().equals(id)) {
            return p;
        }
    }
    return null;
}

public Curso buscarCursoPorCodigo(String codigo) {
    for (Curso c : cursos) {
        if (c.getCodigo().equals(codigo)) {
            return c;
        }
    }
    return null;
}
```

```java
// Rompe la relación con el profesor
public boolean eliminarCurso(String codigo) {
    Curso c = buscarCursoPorCodigo(codigo);
    if (c != null) {
        // 1. Romper la relación bidireccional (lado del curso)
        if (c.getProfesor() != null) {
            // Al llamar a removerProfesor, el curso se elimina de la lista del profesor.
            c.removerProfesor();
        }

        // 2. Eliminar de la lista de la Universidad
        cursos.remove(c);
        System.out.println("Curso eliminado: " + c.getNombre());
        return true;
    }
    System.out.println("ERROR: Curso con código " + codigo + " no encontrado para eliminar.");
    return false;
}
```

```java
// Antes de remover, dejar null los cursos que dictaba.
public boolean eliminarProfesor(String id) {
    Profesor p = buscarProfesorPorId(id);
    if (p != null) {
        // 1. Romper la relación bidireccional con TODOS sus cursos
        // (Usamos un bucle while para evitar ConcurrentModificationException al modificar la lista mientras iteramos)
        while (!p.getCursos().isEmpty()) {
            Curso c = p.getCursos().get(0);
            // Esto rompe la relación en ambos lados (quita de la lista del profesor y pone profesor en null en el curso)
            c.removerProfesor();
            System.out.println("  > Relación rota con el curso: " + c.getNombre());
        }

        // 2. Eliminar de la lista de la Universidad
        profesores.remove(p);
        System.out.println("Profesor eliminado: " + p.getNombre());
        return true;
    }
    System.out.println("ERROR: Profesor con ID " + id + " no encontrado para eliminar.");
    return false;
}
```

```java
// Muestra un reporte: cantidad de cursos por profesor.
public void mostrarReporteCursosPorProfesor() {
    System.out.println("\n--- REPORTE: Cantidad de Cursos por Profesor ---");
    if (profesores.isEmpty()) {
        System.out.println("No hay profesores registrados.");
        return;
    }
    for (Profesor p : profesores) {
        System.out.println(p.getNombre() + ": " + p.getCursos().size() + " curso(s) a cargo.");
    }
}
```

Tareas a realizar

1. Crear al menos 3 profesores y 5 cursos.
2. Agregar profesores y cursos a la universidad.

```
// Tarea 1 & 2: Crear Universidad, Profesores y Cursos, y agregarlos.
Universidad uni = new Universidad("Universidad Nacional de Programación");

Profesor p1 = new Profesor("P101", "Dr. Alan Turing", "Computación");
Profesor p2 = new Profesor("P102", "Ing. Ada Lovelace", "Matemáticas");
Profesor p3 = new Profesor("P103", "Lic. Grace Hopper", "Sistemas");

uni.agregarProfesor(p1);
uni.agregarProfesor(p2);
uni.agregarProfesor(p3);

Curso c1 = new Curso("C201", "Algoritmos I");
Curso c2 = new Curso("C202", "Bases de Datos");
Curso c3 = new Curso("C203", "Programación Orientada a Objetos");
Curso c4 = new Curso("C204", "Cálculo Avanzado");
Curso c5 = new Curso("C205", "Redes de Computadoras");

uni.agregarCurso(c1);
uni.agregarCurso(c2);
uni.agregarCurso(c3);
uni.agregarCurso(c4);
uni.agregarCurso(c5);
```

3. Asignar profesores a cursos usando asignarProfesorACurso(...).

```
// Tarea 3: Asignar profesores a cursos.
System.out.println("\n--- TAREA 3: ASIGNACIÓN INICIAL DE PROFESORES ---");
uni.asignarProfesorACurso("C201", "P101"); // Algoritmos -> Turing
uni.asignarProfesorACurso("C202", "P103"); // Bases de Datos -> Hopper
uni.asignarProfesorACurso("C203", "P101"); // POO -> Turing (Turing tiene 2)
uni.asignarProfesorACurso("C204", "P102"); // Cálculo -> Lovelace
```

4. Listar cursos con su profesor y profesores con sus cursos.

```
// Tarea 4: Listar cursos con su profesor y profesores con sus cursos.
System.out.println("\n--- TAREA 4: LISTADO Y VERIFICACIÓN BIDIRECCIONAL ---");
uni.listarCursos();
p1.listarCursos(); // Debería tener Algoritmos I y POO.
```

5. Cambiar el profesor de un curso y verificar que ambos lados quedan sincronizados.

```
// Tarea 5: Cambiar el profesor de un curso y verificar sincronización.
System.out.println("\n--- TAREA 5: CAMBIO DE PROFESOR (C203: Turing -> Lovelace) ---");
uni.asignarProfesorACurso("C203", "P102"); // POO: Asignar a Lovelace
```

6. Remover un curso y confirmar que ya no aparece en la lista del profesor.

```
// Tarea 6: Remover un curso y confirmar que ya no aparece en la lista del profesor.
System.out.println("\n--- TAREA 6: REMOVER CURSO (C204: Lovelace) ---");
uni.eliminarCurso("C204");

System.out.println("\n> Verificación después de eliminar C204 (Cálculo Avanzado):");
uni.listarCursos(); // Ya no debe aparecer C204
p2.listarCursos(); // Lovelace debe haber perdido C204 (quedar con 1: C203)
```

7. Remover un profesor y dejar profesor = null,

```
// Tarea 7: Remover un profesor y dejar profesor = null en sus cursos.
System.out.println("\n--- TAREA 7: REMOVER PROFESOR (P103: Hopper) ---");
uni.eliminarProfesor("P103");

System.out.println("\n> Verificación de C202 (Bases de Datos) después de eliminar a Hopper:");
Curso c2b = uni.buscarCursoPorCodigo("C202");
if (c2b != null) {
    c2b.mostrarInfo(); // Bases de Datos debe decir "SIN ASIGNAR"
}
```

8. Mostrar un reporte: cantidad de cursos por profesor.

```
// Tarea 8: Mostrar un reporte: cantidad de cursos por profesor.
uni.mostrarReporteCursosPorProfesor();
```