




## **Trabajo Práctico 6: Colecciones**

**Alumno:** Calcatelli Renzo - [rcalcatelli@gmail.com](mailto:rcalcatelli@gmail.com)

**Comisión:** M2025-1

**Materia:** Programación II

**Profesor:** Ariel Enferrel

 [Enlace](#) al repositorio de GitHub

---

## Trabajo Práctico 6: Colecciones

---

### ÍNDICE

1. Resumen
  2. Introducción
  3. Objetivos del Trabajo Práctico
  4. Marco Teórico
  5. Desarrollo de los Ejercicios
    - 5.1 Ejercicio 1: Sistema de Stock
    - 5.2 Ejercicio 2: Biblioteca y Libros
    - 5.3 Ejercicio 3: Universidad (Bidireccional)
  6. Análisis Comparativo
  7. Resultados Obtenidos
  8. Conclusiones
  9. Bibliografía
- 

### RESUMEN

El presente informe documenta el desarrollo completo del Trabajo Práctico 6 de la asignatura Programación II, centrado en la implementación de estructuras de datos dinámicas mediante colecciones (ArrayList) y enumeraciones (enum) en Java.

Se desarrollaron tres ejercicios progresivos que demuestran diferentes tipos de relaciones entre objetos: asociación simple, composición y relación bidireccional. Cada ejercicio implementa un sistema funcional con operaciones CRUD completas y mantiene los principios de programación orientada a objetos.

#### Ejercicios desarrollados:

- **Ejercicio 1:** Sistema de gestión de inventario con 10 funcionalidades.
- **Ejercicio 2:** Sistema de biblioteca con composición 1 a N (9 funcionalidades).
- **Ejercicio 3:** Sistema académico con relación bidireccional (8 funcionalidades).

## INTRODUCCIÓN

En el contexto del aprendizaje de la programación orientada a objetos, el manejo de colecciones de objetos y las relaciones entre clases constituyen conceptos fundamentales. Este trabajo práctico aborda estos temas mediante la implementación de tres sistemas progresivamente más complejos.

La programación orientada a objetos permite modelar problemas del mundo real mediante la abstracción, encapsulamiento, herencia y polimorfismo. En este TP, nos enfocamos especialmente en:

- **Colecciones dinámicas:** Uso de ArrayList para gestionar conjuntos de objetos cuyo tamaño puede variar en tiempo de ejecución.
  - **Enumeraciones:** Tipos de datos que representan un conjunto fijo de constantes.
  - **Relaciones entre objetos:** Asociación, composición y bidireccionalidad.
  - **Encapsulamiento:** Restricción del acceso directo a los atributos de las clases.
- 

## OBJETIVOS DEL TRABAJO PRÁCTICO

### Objetivo General

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

### Objetivos Específicos

1. Implementar colecciones dinámicas utilizando la clase ArrayList de Java.
2. Crear y utilizar enumeraciones con atributos y métodos personalizados.
3. Aplicar correctamente el encapsulamiento mediante modificadores de acceso.
4. Modelar relaciones 1 a N entre clases (asociación, composición y bidireccional).
5. Desarrollar métodos de búsqueda, filtrado y ordenamiento sobre colecciones.
6. Practicar el uso de ciclos for-each para recorrer estructuras de datos.
7. Implementar operaciones CRUD (Create, Read, Update, Delete) completas.
8. Mantener invariantes de asociación en relaciones bidireccionales.
9. Diseñar sistemas modulares y reutilizables.

## MARCO TEÓRICO

### 4.1 Colecciones en Java

Las colecciones son estructuras de datos que permiten almacenar y manipular grupos de objetos. La interfaz **Collection** y sus implementaciones forman el Java Collections Framework.

#### ArrayList:

- Implementación de lista dinámica basada en arrays.
- Permite elementos duplicados.
- Mantiene el orden de inserción.
- Acceso indexado en  $O(1)$ .
- Inserción/eliminación en  $O(n)$  en el peor caso.

```
ArrayList<Producto> productos = new ArrayList<>();  
productos.add(new Producto(...));
```

### 4.2 Enumeraciones (Enum)

Los enums son tipos de datos especiales que representan un conjunto fijo de constantes. En Java, los enums pueden tener atributos, constructores y métodos.

```
public enum CategoriaProducto {  
    ALIMENTOS("Productos comestibles"),  
    ELECTRONICA("Dispositivos electrónicos");  
  
    private final String descripcion;  
  
    CategoriaProducto(String descripcion) {  
        this.descripcion = descripcion;  
    }  
  
    public String getDescripcion() {  
        return descripcion;  
    }  
}
```

### 4.3 Tipos de Relaciones entre Objetos

#### Asociación Simple

Relación donde un objeto utiliza o interactúa con otro, pero ambos tienen ciclos de vida independientes.

```
Inventario —> Producto  
(El inventario referencia productos que existen independientemente)
```

#### Composición

Relación fuerte donde el objeto "parte" no puede existir sin el objeto "todo". Si el todo se destruye, las partes también.

```
Biblioteca ◆——> Libro  
(Los libros se crean dentro de la biblioteca y mueren con ella)
```

#### Relación Bidireccional

Navegación posible desde ambos extremos de la relación. Requiere sincronización para mantener la coherencia.

```
Profesor ◆↔◆ Curso  
(Desde profesor puedo acceder a sus cursos, desde curso puedo acceder a su profesor)
```

### 4.4 Encapsulamiento

Principio que establece que los atributos de una clase deben ser privados y accesibles solo mediante métodos públicos (getters/setters).

#### Ventajas:

- Control sobre cómo se modifican los datos
- Validación de valores antes de asignarlos
- Facilita el mantenimiento y evolución del código
- Oculta la implementación interna

### 4.5 Invariantes de Asociación

En relaciones bidireccionales, un invariante de asociación es una regla que debe cumplirse siempre para mantener la consistencia:

**Regla:** Si A referencia a B, entonces B debe referenciar a A.

```
Si curso.profesor == profesorX
Entonces profesorX.cursos debe contener curso
```

---

## DESARROLLO DE LOS EJERCICIOS

### Ejercicio 1: Sistema de Stock

#### Descripción

Sistema de gestión de inventario que permite controlar productos, sus precios, stock y categorías en una tienda.

#### Componentes Implementados

##### 1. Enum CategoriaProducto

- **4 categorías predefinidas:** ALIMENTOS, ELECTRONICA, ROPA, HOGAR
- Atributo **descripcion** con getter.
- Constructor privado.

##### 2. Clase Producto

- **Atributos:** id, nombre, precio, cantidad, categoria.
- Encapsulamiento completo.
- Método **mostrarInfo()** para visualización.

##### 3. Clase Inventario

- **Atributo:** ArrayList<Producto>
- **10 métodos implementados:**
  - agregarProducto()
  - listarProductos()
  - buscarProductoPorId()
  - eliminarProducto()
  - actualizarStock()
  - filtrarPorCategoria()
  - obtenerTotalStock()
  - obtenerProductoConMayorStock()
  - filtrarProductosPorPrecio()
  - mostrarCategoriasDisponibles()

### Relación Modelada

**Tipo:** Asociación simple 1 a N

Inventario (1) —> Producto (N)

El inventario contiene productos, pero los productos podrían existir independientemente del inventario.

### Conceptos Aplicados

- Colecciones dinámicas (ArrayList)
- Enumeraciones con métodos
- Encapsulamiento
- Ciclo for-each
- Búsqueda lineal
- Filtrado con condicionales
- Operaciones de agregación (suma, máximo)

### Tareas Realizadas

1. Creación de 5 productos con diferentes categorías
2. Listado completo con información y categoría
3. Búsqueda por ID
4. Filtrado por categoría específica
5. Eliminación y listado de restantes
6. Actualización de stock
7. Cálculo de stock total
8. Identificación del producto con mayor stock
9. Filtrado por rango de precio (\$1000-\$3000)
10. Visualización de categorías disponibles

#### Complejidad de Operaciones

Operación	Complejidad Temporal	Complejidad Espacial
Agregar	$O(1)$ amortizado	$O(1)$
Listar	$O(n)$	$O(1)$
Buscar	$O(n)$	$O(1)$
Eliminar	$O(n)$	$O(1)$
Filtrar	$O(n)$	$O(k)$ donde $k$ = resultados

## Ejercicio 2: Biblioteca y Libros

### Descripción

Sistema de gestión bibliotecaria que registra libros y sus autores, implementando una relación de composición donde los libros pertenecen obligatoriamente a la biblioteca.

### Componentes Implementados

#### 1. Clase Autor

- **Atributos:** id, nombre, nacionalidad
- Clase independiente reutilizable
- Método mostrarInfo()

#### 2. Clase Libro



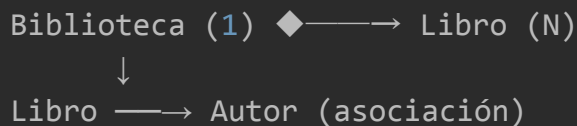
- **Atributos:** isbn, titulo, anioPublicacion, autor
- Asociación con Autor
- Método mostrarInfo()

### 3. Clase Biblioteca

- **Atributos:** nombre, List<Libro>
- Los libros se crean internamente (composición)
- **7 métodos implementados:**
  - agregarLibro() - Crea el libro internamente
  - listarLibros()
  - buscarLibroPorIsbn()
  - eliminarLibro()
  - obtenerCantidadLibros()
  - filtrarLibrosPorAnio()
  - mostrarAutoresDisponibles() - Usa HashSet para evitar duplicados

### Relación Modelada

**Tipo:** Composición 1 a N



La biblioteca POSEE los libros (si se elimina la biblioteca, se eliminan los libros). Los libros REFERENCIAN autores (los autores existen independientemente).

### Diferencias con Ejercicio 1

Aspecto	Ejercicio 1	Ejercicio 2
Creación	Producto se crea fuera	Libro se crea dentro
Método add	Recibe objeto	Recibe parámetros
Ciclo de vida	Independiente	Dependiente
Responsabilidad	Inventario solo referencia	Biblioteca es dueña

### Conceptos Aplicados

- Composición fuerte
- List<T> (programación contra interfaces)
- HashSet para eliminar duplicados
- Relaciones múltiples (Biblioteca-Libro-Autor)
- Filtrado por atributos

#### Tareas Realizadas

1. Creación de biblioteca.
  2. Creación de 3 autores.
  3. Agregado de 5 libros asociados a autores.
  4. Listado completo con información del autor.
  5. Búsqueda por ISBN.
  6. Filtrado por año de publicación.
  7. Eliminación de libro y listado de restantes.
  8. Obtención de cantidad total.
  9. Listado de autores únicos disponibles.
- 

### Ejercicio 3: Universidad (Bidireccional)

#### Descripción

Sistema académico que modela la relación entre profesores y cursos mediante navegación bidireccional, donde un profesor dicta múltiples cursos y cada curso tiene un profesor responsable.

#### Componentes Implementados

##### 1. Clase Profesor

- **Atributos:** id, nombre, especialidad, List<Curso>
- **Métodos auxiliares:**
  - agregarCurso() - Solo modifica lista interna
  - eliminarCurso() - Solo modifica lista interna
  - listarCursos()
  - mostrarInfo()

##### 2. Clase Curso

- **Atributos:** codigo, nombre, Profesor
- **Método coordinador:**
  - setProfesor() - Sincroniza ambos lados automáticamente
- Método mostrarInfo()

### 3. Clase Universidad

- **Atributos:** nombre, List<Profesor>, List<Curso>
- **Métodos de gestión:**
  - agregarProfesor()
  - agregarCurso()
  - asignarProfesorACurso() - Usa setProfesor()
  - eliminarCurso() - Rompe relación antes
  - eliminarProfesor() - Desasigna todos los cursos
  - mostrarReporteCursosPorProfesor()

#### Relación Modelada

**Tipo:** Bidireccional 1 a N

Profesor (1) ◆ ↔ ◆ Curso (N)

**Navegación en ambas direcciones con sincronización automática.**

#### Patrón de Sincronización

**Patrón Coordinador-Auxiliar:**

```
// COORDINADOR (en Curso) - Hace toda la sincronización
public void setProfesor(Profesor nuevo) {
    // 1. Quitar del profesor anterior
    if (this.profesor != null && this.profesor != nuevo) {
        this.profesor.eliminarCurso(this);
    }

    // 2. Asignar nuevo
    this.profesor = nuevo;

    // 3. Agregar al nuevo profesor
    if (nuevo != null) {
```

```
        nuevo.agregarCurso(this);
    }
}

// AUXILIARES (en Profesor) - Solo modifican listas
public void agregarCurso(Curso c) {
    if (!cursos.contains(c)) {
        cursos.add(c);
        // NO llama a setProfesor() → Rompe el ciclo
    }
}
```

### Invariante de Asociación

Regla que se mantiene siempre:

```
Si curso.profesor == profesorX
Entonces profesorX.cursos contiene curso
```

Y viceversa:

```
Si profesorX.cursos contiene cursoY
Entonces cursoY.profesor == profesorX
```

### Conceptos Aplicados

- Relación bidireccional
- Sincronización automática
- Invariantes de asociación
- Patrón coordinador-auxiliar
- Prevención de recursión infinita
- Prevención de ConcurrentModificationException
- Manejo seguro de referencias

### Tareas Realizadas

1. Creación de 3 profesores y 5 cursos
2. Registro en universidad
3. Asignación de profesores a cursos
4. Listado bidireccional (cursos con profesor, profesores con cursos)
5. Cambio de profesor con verificación de sincronización
6. Eliminación de curso con ruptura de relación
7. Eliminación de profesor con desasignación masiva
8. Reporte de carga académica por profesor

### Problemas Resueltos

#### 1. Recursión infinita:

- **Problema:** setProfesor() llamaba a agregarCurso() que llamaba a setProfesor().
- **Solución:** Patrón coordinador-auxiliar (solo el coordinador sincroniza).

#### 2. ConcurrentModificationException:

- **Problema:** Modificar lista mientras se recorre
- **Solución:** Crear copia con **new ArrayList<>(original)** antes de recorrer

#### 3. Referencias huérfanas:

- **Problema:** Eliminar profesor sin desasignar cursos
  - **Solución:** Desasignar todos los cursos antes de eliminar
- 

## ANÁLISIS COMPARATIVO

## 6.1 Comparación de Relaciones

Característica	Ejercicio 1 (Asociación)	Ejercicio 2 (Composición)	Ejercicio 3 (Bidireccional)
Creación	Fuera del contenedor	Dentro del contenedor	Independiente
Navegación	Una dirección	Una dirección	Ambas direcciones
Ciclo de vida	Independiente	Dependiente	Independiente
Sincronización	No requerida	No requerida	Crítica
Complejidad	Baja	Media	Alta
Método add	Recibe objeto	Recibe parámetros	Usa setProfesor()
Eliminación	Quita referencia	Destruye objeto	Sincroniza ambos lados

## 6.2 Evolución de Complejidad

### Ejercicio 1: ASOCIACIÓN SIMPLE

- ─ Concepto: Referencias básicas
- ─ Complejidad: ★★ (2/5)
- ─ Foco: Colecciones y enums

### Ejercicio 2: COMPOSICIÓN

- ─ Concepto: Pertenencia fuerte
- ─ Complejidad: ★★★ (3/5)
- ─ Foco: Ciclo de vida dependiente

### Ejercicio 3: BIDIRECCIONAL

- ─ Concepto: Navegación dual
- ─ Complejidad: ★★★★★ (5/5)
- ─ Foco: Sincronización y coherencia

## 6.3 Líneas de Código por Ejercicio

Ejercicio	Clases	LOC (aprox.)	Métodos	Complejidad Ciclomática
-----------	--------	--------------	---------	-------------------------

---

Ejercicio 1	3	450	13	Baja
Ejercicio 2	3	480	10	Media
Ejercicio 3	3	520	15	Alta
Total	9	1,450	38	-

---

## RESULTADOS OBTENIDOS

### 7.1 Funcionalidades Implementadas

**Total de tareas completadas:** 27 (10 + 9 + 8)

**Operaciones CRUD:**

- **Create:** 9 métodos de agregado.
- **Read:** 12 métodos de listado/búsqueda.
- **Update:** 3 métodos de actualización.
- **Delete:** 6 métodos de eliminación.

**Operaciones adicionales:**

- **Filtrado:** 5 métodos.
- **Agregación:** 3 métodos (suma, máximo).
- **Reportes:** 3 métodos.

### 7.2 Pruebas Realizadas

**Todos los ejercicios fueron probados con:**

- Casos normales de uso.
- Casos límite (listas vacías, elementos no encontrados).
- Cambios de estado (actualización, eliminación).
- Sincronización bidireccional (Ejercicio 3).

## CONCLUSIONES

## 8.1 Cumplimiento de Objetivos

Se han cumplido satisfactoriamente todos los objetivos planteados en el trabajo práctico:

### Objetivos técnicos alcanzados:

1. **ArrayList y colecciones:** Se implementaron correctamente en los 3 ejercicios, demostrando dominio de operaciones CRUD, iteración con for-each y métodos de búsqueda/filtrado.
2. **Enumeraciones con métodos:** El enum CategoriaProducto incluye atributo privado, constructor y método getter, demostrando que los enums pueden ser más que simples constantes.
3. **Encapsulamiento:** Todos los atributos de todas las clases son privados con acceso controlado mediante getters/setters, cumpliendo con el principio de ocultamiento de información.
4. **Relaciones 1 a N:** Se modelaron exitosamente tres tipos diferentes:
  - Asociación simple (Inventario-Producto)
  - Composición (Biblioteca-Libro)
  - Bidireccional (Profesor-Curso)
5. **Invariantes de asociación:** En el Ejercicio 3 se mantiene automáticamente la coherencia bidireccional mediante el patrón coordinador-auxiliar.
6. **Modularidad:** Cada clase tiene responsabilidades bien definidas, facilitando el mantenimiento y la extensibilidad.

## 8.2 Conceptos Clave Aprendidos

### 1. Diferenciación entre tipos de relaciones:

- **Asociación:** Uso o referencia de un objeto por otro
- **Composición:** Pertenencia fuerte donde la parte no existe sin el todo
- **Bidireccional:** Navegación en ambas direcciones con sincronización

### 2. Importancia de la sincronización:



En relaciones bidireccionales, mantener la coherencia es crítico. El patrón coordinador-auxiliar permite:

- Un único punto de sincronización (setProfesor)
- Métodos auxiliares que solo modifican listas
- Prevención de recursión infinita
- Mantenimiento automático del invariante

### 3. Uso efectivo de colecciones:

- ArrayList para listas dinámicas
- List<T> para programar contra interfaces
- HashSet para eliminar duplicados
- Iteración segura evitando modificaciones concurrentes

### 4. Diseño orientado a objetos:

- Encapsulamiento protege la integridad de los datos
- Métodos públicos proporcionan interfaces claras
- Responsabilidad única por clase
- Bajo acoplamiento, alta cohesión

## 8.3 Desafíos Superados

### 1. Recursión infinita en Ejercicio 3:

Inicialmente, los métodos agregarCurso() y setProfesor() se llamaban mutuamente, generando StackOverflowError. Se resolvió mediante el patrón coordinador-auxiliar.

### 2. ConcurrentModificationException:

Al eliminar profesores, modificar la lista de cursos durante la iteración causaba excepciones. Se resolvió creando copias de las listas antes de iterar.

### 3. Gestión de referencias huérfanas:

Al eliminar profesores sin desasignar cursos, quedaban referencias inconsistentes. Se solucionó implementando limpieza antes de la eliminación.

## 8.4 Aplicabilidad Práctica

Los conceptos implementados son fundamentales para el desarrollo de software profesional:

**Sistemas de información:**

- Gestión de inventarios (retail, logística)
- Sistemas bibliotecarios (educación)
- Plataformas académicas (universidades)

**Patrones aplicables:**

- Repository pattern (clases gestoras)
  - Domain model (modelado de entidades)
  - Observer pattern (extensión de bidireccionalidad)
- 

## BIBLIOGRAFÍA

### Material de Cátedra - Universidad Tecnológica Nacional

#### Programación II - Tecnicatura Universitaria en Programación a Distancia

##### Videos Educativos

- Universidad Tecnológica Nacional. (2024). *ARRAYLIST en Java - Lo básico para arrancar.*
- Universidad Tecnológica Nacional. (2024). *ATRIBUTOS tipo COLECCION de OBJETOS en Java.*
- Universidad Tecnológica Nacional. (2024). *Ciclo FOR EACH en Java.*
- Universidad Tecnológica Nacional. (2024). *EJERCICIO de ARRAYLIST en Java - 03: RESUELTO.*
- Universidad Tecnológica Nacional. (2024). *ENUMS en Java - Código y UML.*
- Universidad Tecnológica Nacional. (2024). *ENUMS en Java - Métodos útiles.*
- Universidad Tecnológica Nacional. (2024). *ENUMS en Java - ¡Con atributos y métodos!.*
- Universidad Tecnológica Nacional. (2024). *MÁXIMOS y MÍNIMOS en una COLECCIÓN de OBJETOS.*
- Universidad Tecnológica Nacional. (2024). *PROMEDIO y SUMATORIA en una COLECCIÓN de OBJETOS.*
- Universidad Tecnológica Nacional. (2024). *Relaciones 1 a N (muchos) en CLASES UML.*
- 

##### Documentos PDF

- Universidad Tecnológica Nacional. (2024). *Apunte\_Enums\_Java.pdf.*

- Universidad Tecnológica Nacional. (2024). *Apunte\_Relaciones\_1aN\_en\_Java.pdf*.
- Universidad Tecnológica Nacional. (2024). *Diagramas de Clase en UML 1.1 - Repositorio Grial*.
- Universidad Tecnológica Nacional. (2024). *Colecciones-Dinámicas-Algoritmos-y-Enumeraciones*.
- Universidad Tecnológica Nacional. (2024). *Relaciones-Uno-a-Muchos-1N-en-Java-Guía-Completa*.

### Material Teórico Complementario

- Universidad Tecnológica Nacional. (2024). *17. Otras relaciones entre objetos*.
- Universidad Tecnológica Nacional. (2024). *Agregación Vs Composición en diagramas de clases UML*.
- Universidad Tecnológica Nacional. (2024). *ArrayList de Java: cómo crear y utilizar una de estas listas*.
- Universidad Tecnológica Nacional. (2024). *Relaciones de diagramas de clases en UML explicadas*.
- Universidad Tecnológica Nacional. (2024). *Course Módulo 1. Java Syntax - Lecture: Consejos para escribir código limpio*.
- Universidad Tecnológica Nacional. (2024). *Problemas comunes en Java y cómo Solucionarlos - Importancia de la Legibilidad*.

### Referencias Externas

- Oracle. (2023). [Java ArrayList Documentation. Java Platform SE 8.](#)
  - Oracle. (2023). [Enum Types - The Java Tutorials.](#)
  - Oracle. (2023). [Encapsulation in Java. The Java Tutorials.](#)
  - Baeldung. (2023). [Composition, Aggregation, and Association in Java.](#)
  - Baeldung. (2023). [Java Collections Framework.](#)
-