Trabajo Práctico 6: Colecciones y Sistema de Stock
Materia: Programación 2
Estudiante: Etchegoyen Gabriel
Enlace a repositorio en GitHub: https://github.com/Gabriel071185/UTN-TUPaD-P2

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

Concepto	Aplicación en el proyecto
ArrayList	Estructura principal para almacenar productos en el inventario.
Enumeraciones (enum)	Representan las categorías de productos con valores predefinidos.
Relaciones 1 a N	Relación entre Inventario (1) y múltiples Productos (N).
Métodos en enum	Inclusión de descripciones dentro del enum para mejorar legibilidad.
Ciclo for-each	Recorre colecciones de productos para listado, búsqueda o filtrado.
Búsqueda y filtrado	Por ID y por categoría, aplicando condiciones.
Ordenamientos y reportes	Permiten organizar la información y mostrar estadísticas útiles.
Encapsulamiento	Restringir el acceso directo a los atributos de una clase

Caso Práctico 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.

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.

Caso Práctico 2

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.

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.

Caso Práctico 3

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).

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.

Respuesta:

Ejercitación resuelta en repositorio de GitHub:
https://github.com/Gabriel071185/UTN-TUPaD-P2