UT7- Clases de Recuperación - Colecciones, Iteradores y Comparadores en Java

¿Para qué sirve todo esto?

Colecciones: para guardar y organizar datos dinámicos
Iteradores: para recorrer colecciones sin errores
Comparadores: para ordenar objetos personalizados

¿Por qué es importante aprender esto?

Porque nos permite resolver problemas del mundo real como:

- Gestionar una lista de tareas pendientes (List)
- Evitar duplicados (Set)
- Crear rankings u ordenaciones (Comparator)
- Asociar datos clave-valor (Map)
- Automatizar procesos sobre colecciones de datos (Iterator)

Aprender nos da herramientas para escribir código más limpio, ordenado, eficiente y escalable.

Estas herramientas son esenciales para resolver problemas reales de programación, ya que permiten trabajar con grandes cantidades de datos de forma eficiente, segura y ordenada.

1. Colecciones en Java

Una colección es un objeto que agrupa múltiples elementos. Tipos comunes:
- List: Permite duplicados y mantiene el orden (ArrayList, LinkedList).
- Set: No permite duplicados (HashSet, TreeSet).
- Map: Almacena pares clave-valor (HashMap, TreeMap).

En Java, las interfaces **List**, **Set** y **Map** forman parte del **Java Collections Framework**, y tienen varias **implementaciones concretas** que puedes usar dependiendo de tus necesidades (rendimiento, orden, duplicados, sincronización, etc.).

Clases que implementan `List`:

Clase	Características principales
`ArrayList`	Lista redimensionable, acceso rápido por índice, **no sincronizada**.
`LinkedList`	Lista doblemente enlazada, buena para inserciones/eliminaciones.
`Vector`	Similar a `ArrayList`, pero **sincronizada** (más lenta, legada).
`Stack`	Subclase de `Vector`, implementa pila (LIFO).

Clases que implementan `Set`:

Clase	Características principales
`HashSet`	Basado en `HashMap`, **no garantiza orden**, rápido.
`LinkedHashSet`	Como `HashSet`, pero **mantiene el orden de inserción**.
`TreeSet`	Basado en un árbol rojo-negro, **ordenado** (natural o con `Comparator`).
`EnumSet`	Set optimizado para `enum`.

Clases que implementan `Map`:

Clase	Características principales
`HashMap`	Mapa más común. No garantiza orden. Permite `null` en clave/valor.
`LinkedHashMap`	Mantiene el **orden de inserción**.
`TreeMap`	Ordenado por claves. Basado en árbol rojo-negro.
`Hashtable`	Antigua, sincronizada, **no permite claves ni valores** `null`.
`ConcurrentHashMap`	Segura para múltiples hilos, **muy eficiente** en concurrencia.
`WeakHashMap`	Las claves pueden ser recolectadas por el GC si no se referencian.

2. Iteradores

Permiten recorrer colecciones.

Ejemplo:

```
Iterator<String> it = lista.iterator();
while (it.hasNext()) {
    String elem = it.next();
    System.out.println(elem);
}
```

3. Comparadores

Sirven para ordenar objetos
:
- Comparable<T>: Se implementa en la clase (compareTo).
- Comparator<T>: Se define externamente.

```
Comparator<Persona> porNombre = (a, b) -> a.nombre.compareTo(b.nombre);
```

Ejercicios

Nivel 1: Ejercicios de introducción

1. Crear una lista de enteros del 1 al 5 y mostrarla por consola.
2. Agregar elementos duplicados a un Set y mostrar cuántos elementos contiene.
3. Crear una lista de nombres y recorrerla con for y foreach.

Nivel 2: Iteradores

4. Crear una lista de 5 palabras. Usar un Iterator para imprimirlas en mayúscula.
5. Eliminar palabras que empiecen por vocal usando Iterator.
6. Eliminar los impares de una lista de enteros usando Iterator.

Nivel 3: Comparable

7. Clase Alumno con nombre y nota. Implementar Comparable para ordenar por nota.
8. Lista de 5 alumnos. Ordenarlos con Collections.sort().
9. Cambiar orden para que sea por nombre en vez de nota.

Nivel 4: Comparator

10. Usar Comparator para ordenar alumnos por longitud de nombre.
11. Ordenar por nota descendente y luego nombre alfabéticamente.
12. Crear TreeSet con Comparator personalizado.

Nivel 5: Mapas

13. Crear Map de productos y precios. Mostrar claves y valores.
14. Ordenar Map por valor con Comparator.

15. Dado Map<String, List<String>>, imprimir nombres que estén en varias asignaturas.