Estructuras Internas

https://bronze-infinity-741.notion.site/Estructuras-Internas-50fc9257fa654b84991566cfc1986030

▼ Colecciones e Iteradores

▼ Code

```
package Internas;

import ejercicio04.Coche;

import java.util.HashSet;
import java.util.Set;

import java.util.ArrayList;
import java.util.List;

import  java.util.HashMap;
import java.util.Map;

import java.util.Iterator;

public class Interna {
    public static void main(String[] args) {

        // Creación de los objetos
        Coche coche1 = new Coche("Peugeot");
        Coche coche2 = new Coche("Seat");
        Coche coche3 = new Coche("Renault");

        // Creación de las colecciones de objetos, aunque map no hereda de collection.
        Set<Coche> conjunto=new HashSet<>();
        List<Coche> lista=new ArrayList<>();
        Map<Integer, Coche> mapa = new HashMap<>();

        // Añadimos los objetos a las colecciones
        conjunto.add(coche1);
        conjunto.add(coche2);
        conjunto.add(coche3);

        lista.add(coche1);
        lista.add(coche2);
        lista.add(coche3);

        mapa.put(1, coche1);
        mapa.put(2, coche2);
        mapa.put(3, coche3);

        // Recorremos las colecciones con los iteradores
        Iterator<Coche> itConjunto = conjunto.iterator();
        Iterator<Coche> itLista = lista.iterator();
        Iterator<Integer> itMapa = mapa.keySet().iterator(); // Devuelve un conjunto de claves

        while (itConjunto.hasNext()) {
            System.out.println(itConjunto.next());
        }
        while (itLista.hasNext()) {
            Coche cocheLista = itLista.next(); // Hacemos una copia del objeto para que no mute el original
            System.out.println(cocheLista.toString());
        }
```

```
        while (itMapa.hasNext()) {
            System.out.println(mapa.get(itMapa.next()).getNombre());
        }
        for (Integer llave:mapa.keySet()) {
            System.out.println(mapa.get(llave));
        }
    }
}
```

▼ Operaciones

- Método `int size()`: devuelve el número de elementos de la colección.
- Método `boolean isEmpty()`: devuelve `true` si la colección está vacía.
- Método `boolean contains (Object element)`: retornará `true` si la colección tiene el elemento pasado como parámetro.
- Método `boolean add(E element)`: permitirá añadir elementos a la colección.
- Método `boolean remove (Object element)`: permitirá eliminar elementos de la colección.
- Método `Iterator<E> iterator()`: permitirá crear un iterador para recorrer los elementos de la colección. Esto se ve más adelante, no te preocupes.
- Método `Object[] toArray()`: permite pasar la colección a un array de objetos tipo `Object`.
- Método `containsAll(Collection<?> c)`: permite comprobar si una colección contiene los elementos existentes en otra colección, si es así, retorna verdadero.
- Método `addAll (Collection<? extends E> c)`: permite añadir todos los elementos de una colección a otra colección, siempre que sean del mismo tipo (o deriven del mismo tipo base).
- Método `boolean removeAll(Collection<?> c)`: si los elementos de la colección pasada como parámetro están en nuestra colección, se eliminan, el resto se quedan.
- Método `boolean retainAll(Collection<?> c)`: si los elementos de la colección pasada como parámetro están en nuestra colección, se dejan, el resto se eliminan.
- Método `void clear()`: vacíar la colección.

▼ Conjuntos

- `HashSet`: Es útil cuando quieres una colección sin duplicados y no te importa el orden de los elementos.
- `LinkedHashSet`: Es útil cuando quieres una colección sin duplicados, y también quieres mantener el orden en que los elementos fueron insertados.
- `TreeSet`: Es útil cuando quieres una colección sin duplicados y necesitas que los elementos estén en orden ascendente. También, TreeSet es una buena opción cuando necesitas realizar operaciones que requieren un orden natural de los elementos.

▼ Listas

- `LinkedList`: Es útil cuando necesitas frecuentemente agregar y eliminar elementos desde el principio o el final de la lista, como en las implementaciones de colas y pilas. También es útil cuando el tamaño de la lista varía frecuentemente.
- `ArrayList`: Es útil cuando necesitas acceso aleatorio frecuente a los elementos. Es más eficiente en términos de memoria en comparación con LinkedList. También es una mejor opción cuando el tamaño de la lista es relativamente estable, y las operaciones de adición y eliminación son menos frecuentes.

Además de los anteriores también implementa los siguienetes:

- `E get(int index)`. El método `get()` permite obtener un elemento partiendo de su posición (`index`).
- `E set(int index, E element)`. El método `set()` permite cambiar el elemento almacenado en una posición de la lista (`index`), por otro (`element`).
- `void add(int index, E element)`. Se añade otra versión del método `add()`, en la cual se puede insertar un elemento (`element`) en la lista en una posición concreta (`index`), desplazando los existentes.
- `E remove(int index)`. Se añade otra versión del método `remove()`, esta versión permite eliminar un elemento indicando su posición en la lista.
- `boolean addAll(int index, Collection<? extends E> c)`. Se añade otra versión del método `addAll()`, que permite insertar una colección pasada por parámetro en una posición de la lista, desplazando el resto de elementos.
- `int indexOf(Object objeto)`. El método `indexOf()` permite conocer la posición (**índice**) de un elemento (**objeto**), si dicho elemento no está en la lista retornará **-1**.
- `int lastIndexOf(Object objeto)`. El método `lastIndexOf()` nos permite obtener la última ocurrencia del objeto en la lista (dado que la lista sí puede almacenar duplicados).
- `List<E> subList(int from, int to)`. El método `subList()` genera una sublista (una vista parcial de la lista) con los elementos comprendidos entre la posición inicial (incluida) y la posición final (no incluida).

▼ Mapas

- `java.util.HashMap`
- `java.util.TreeMap`
- `java.util.LinkedHashMap`.

Operaciones:

`V put(K key, V value)`	Inserta un par de objetos llave (`key`) y valor (`value`) en el mapa. Si la llave ya existe en el mapa, entonces devolverá el valor asociado que tenía antes, si la llave no existía, entonces devolverá `null`.
`V get(Object key)`	Obtiene el valor asociado a una llave ya almacenada en el mapa. Si no existe la llave, devolverá `null`.
`V remove(Object key)`	Elimina la llave y el valor asociado. Devuelve el valor asociado a la llave, por si lo queremos

	utilizar para algo, o `null`, si la llave no existe.
`boolean containsKey(Object key)`	Devolverá `true` si el mapa tiene almacenada la llave pasada por parámetro, `false` en cualquier otro caso.
`boolean containsValue(Object value)`	Devolverá `true` si el mapa tiene almacenado el valor pasado por parámetro, `false` en cualquier otro caso.
`int size()`	Devolverá el número de pares llave y valor almacenado en el mapa.
`boolean isEmpty()`	Devolverá `true` si el mapa está vacío, `false` en cualquier otro caso.
`void clear()`	Vacía el mapa.