# 20. Colecciones (I)

## 20.1 Interfaz Collection.

Las colecciones representan grupos de objetos, denominados elementos. Podemos encontrar diversos tipos de colecciones, según si sus elementos están ordenados, o si permitimos repetición de elementos o no.

Es el tipo más genérico en cuanto a que se refiere a cualquier tipo que contenga un grupo de elementos. Viene definido por la interfaz Collection, de la cual heredará cada subtipo específico. En esta interfaz encontramos una serie de métodos que nos servirán para acceder a los elementos de cualquier colección de datos, sea del tipo que sea. Estos métodos generales son:

***boolean add(Object o)*** Añade un elemento (objeto) a la colección. Nos devuelve true si tras añadir el elemento la colección ha cambiado, es decir, el elemento se ha añadido correctamente, o false en caso contrario.

***void clear()*** Elimina todos los elementos de la colección.

***boolean contains(Object o)*** Indica si la colección contiene el elemento (objeto) indicado.

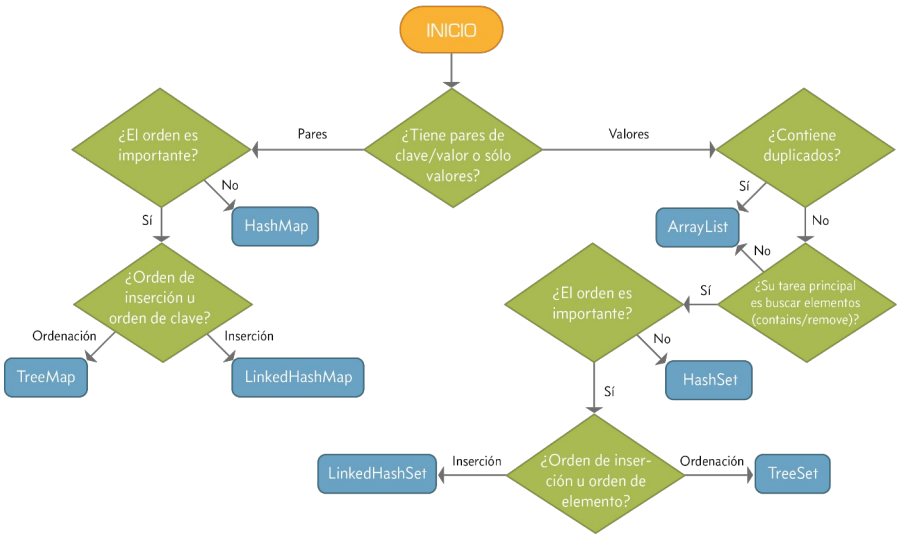
***boolean isEmpty()*** Indica si la colección está vacía (no tiene ningún elemento).

***Iterator iterator()*** Proporciona un iterador para acceder a los elementos de la colección.

***boolean remove(Object o)*** Elimina un determinado elemento (objeto) de la colección, devolviendo true si dicho elemento estaba contenido en la colección, y false en caso contrario.

***int size()*** Nos devuelve el número de elementos que contiene la colección.

Diagrama de decisión para uso de colecciones Java



## 20.2 Listas

Este tipo de colección se refiere a listas en las que los elementos de la colección tienen un orden, existe una secuencia de elementos. En ellas cada elemento estará en una determinada posición (índice) de la lista.

Las listas vienen definidas en la interfaz List, que además de los métodos generales de las colecciones, nos ofrece los siguientes para trabajar con los índices:

***void add(int indice, Object obj)***
Inserta un elemento (objeto) en la posición de la lista dada por el índice indicado.

***Object get(int indice)***
Obtiene el elemento (objeto) de la posición de la lista dada por el índice indicado.

***int indexOf(Object obj)***
Nos dice cual es el índice de dicho elemento (objeto) dentro de la lista. Nos devuelve -1 si el objeto no se encuentra en la lista.

***Object remove(int indice)***
Elimina el elemento que se encuentre en la posición de la lista indicada mediante dicho índice, devolviéndonos el objeto eliminado.

***Object set(int indice, Object obj)***
Establece el elemento de la lista en la posición dada por el índice al objeto indicado, sobrescribiendo el objeto que hubiera anteriormente en dicha posición. Nos devolverá el elemento que había previamente en dicha posición.

## 20.3 ArrayList

Implementa una lista de elementos mediante un array de tamaño variable. Conforme se añaden elementos el tamaño del array irá creciendo si es necesario. El array tendrá una capacidad inicial, y en el momento en el que se rebase dicha capacidad, se aumentará el tamaño del array.

```java
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class EjemploArrayList {



    static public void  main(String[] args) {


        List<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);
        list.add(50);
        list.add(60);
        list.add(70);
        list.add(80);

        System.out.println(list);

        // Recorrido old-style
        for (int i = 0; i < list.size(); i++) {
            System.out.println("El elemento es: " + list.get(i));
        }
        //Recorrido foreach
        for (Integer element: list) {
            System.out.println("foreach El elemento es " + element);
        }

        //Recorrido patrón Iterator
        Iterator<Integer> it = list.iterator();

        while (it.hasNext()) {
            System.out.println("iterator " + it.next());
        }

        //Inserta el valor 1000 en la posición 2
        list.set(2, 1000);

        System.out.println(list);

        // Comprueba si la lista contiene un objeto.
        System.out.println(list.contains(500));

        //Elimina el elemento en la posición 1, es decir, el valor 20
        list.remove(1);
        System.out.println(list);
```

```
        //Elimina el valor 30 de la lista
        list.remove(Integer.valueOf(30));
        System.out.println(list);

        //Elimina todos los elementos de la lista
        list.clear(); //This will remove all the elements from the list.
        System.out.println(list);



    }
}
```

## 20.4 Sobrecarga de equals

Todos los Object y clases derivadas tienen un método equals(Object o) que compara un objeto con otro devolviendo un booleano verdadero en caso de igualdad. El criterio de igualdad puede ser personalizado, según la clase. Para personalizarlo se puede sobrecargar el método de comparación:

```
public class TrianguloEquilatero extends Poligono{


    private double base;
    private double altura;

    public boolean equals(TrianguloEquilatero te) {
        if (this.getBase()==te.getBase() && this.getAltura()==te.getAltura()) {
            return true;
        }
        return false;
    }
```

### *Ejercicio 20.1*

Crea un ArrayList con los nombres de 6 compañeros de clase. A continuación, muestra esos nombres por pantalla. Utiliza para ello un bucle for que recorra todo el ArrayList sin usar ningún índice.

### *Ejercicio 20.2*

Realiza un programa que introduzca valores aleatorios (entre 0 y 100) en un ArrayList y que luego calcule la suma, la media, el máximo y el mínimo de esos números. El tamaño de la lista también será aleatorio y podrá oscilar entre 10 y 20 elementos ambos inclusive.

### *Ejercicio 20.3*

Escribe un programa que ordene 10 números enteros introducidos por teclado y almacenados en un objeto de la clase ArrayList.

### *Ejercicio 20.4*

Realiza un programa equivalente al anterior pero en esta ocasión, el programa debe ordenar palabras en lugar de números.

### *Ejercicio 20.5*

Haz un programa que una serie de valores numéricos enteros desde el teclado y los guarde en un ArrayList de tipo Integer. La lectura de números enteros termina cuando se introduzca el valor -99. Este valor no se guarda en el ArrayList. A continuación el programa mostrará por pantalla el número de valores que se han leído, su suma y su media. Por último se mostrarán todos los valores leídos, indicando cuántos de ellos son mayores que la media.

### *Ejercicio 20.6*

La aplicación consta de 2 clases: Grupo y Alumno.

La clase Alumno tendrá tres atributos: numeroExpediente(int), nombre (String) y edad (int).

La clase Grupo tendrá un atributo nombre, y con el método getAlumnos(), devolverá una colección con sus alumnos, implementará además métodos para las siguientes operaciones:

- Insertar un nuevo alumno.

- Eliminar un alumno por nombre. Elimina a un alumno a partir del nombre.

- Eliminar un alumno. Elimina a un alumno a partir del número de expediente.

- Modificar los datos de un alumno. Si el número de expediente no existe devolverá una excepción.

Realiza un sencillo menú para interactuar con la clase Grupo.