>>>> ¿Qué es la interfaz List?

Se trata de una interfaz de Java que define el funcionamiento de una lista de objetos. Algo similar a un Array, pero más dinámico. Las principales diferencias con un array son:

- Su tamaño es dinámico, puede crecer y decrecer a medida que voy añadiendo o borrando elementos. Y no hay que darle un tamaño fijo al inicio.
- Puedo borrar elementos (no sólo hacer que apunten a null como en un array, sino borrarlos completamente, de modo que la lista pasará a tener un tamaño menor)
- Tenemos muchos más métodos que nos facilitan el modo de trabajar. Todos estos métodos se pueden consultar en la documentación oficial de Java: https://docs.oracle.com/javase/8/docs/api/java/util/List.html

IMPORTANTE: Recuerda que, en las listas, la primera posición será la 0. Igual que en los arrays.

¿Cómo puedo crearme uno objeto que implemente la interfaz List?

Basta con instanciar un objeto de una clase que implemente dicha interfaz. Las clases que implementan dicha interfaz más conocidas son:

- ArrayList
- LinkedList
- Vector

Al crear el objeto, tendremos que indicar qué tipo de elementos son los que van a estar dentro de la lista. Esto lo hacemos con los caracteres "< >". Veamos un par de ejemplos:

```
// Esto es una lista de String. La hemos creado usando la clase "ArrayList"
List<String> listaCadenas = new ArrayList<>();
```

```
// Esto es una lista de Alumnos. La hemos creado usando la clase "LinkedList"
List<Alumno> listaAlumnos = new LinkedList<>();
```

¿Cómo puedo añadir objetos a la lista?

Basta con llamar al método add() de la lista. Tenemos cuatro opciones:

1. Añadir un objeto al final de la lista:

```
listaCadenas.add("Esto irá al final");
```

2. Añadir un objeto en una posición determinada. Todos los demás elementos, se desplazarán una posición:

```
// Esto añadirá la cadena "hola" en cuarta posición.
// El elemento que antes estaba ahí, pasará a estar en quinto lugar.
// El quinto pasará al sexto. Y así sucesivamente.
listaCadenas.add(3, "hola");
```

3. Sustituir un elemento por otro. Para ellos usamos el método set():

```
List<String> listaCadenas1 = new ArrayList<>();
listaCadenas1.add("hola");
listaCadenas1.add("adios");
listaCadenas1.set(1, "prueba");
```

La lista quedará así: [hola, prueba]

4. Añadir todos los elementos de una lista en otra. Usamos el método addAll():

```
List<String> listaCadenas1 = new ArrayList<>();
listaCadenas1.add("hola");
listaCadenas1.add("adios");

List<String> listaCadenas2 = new ArrayList<>();
listaCadenas2.add("primero");
listaCadenas2.addAll(listaCadenas1);
```

La lista 2 quedará así: [primero, hola, adios]
La lista 1 quedará sin modificar: [hola, adios]

¿Cómo puedo obtener un objeto de la lista?

Basta con llamar al método get() de la lista indicando el índique que queremos obtener:

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("hola");
listaCadenas.add("adios");
String cadena = listaCadenas.get(0); // esto será "hola"
```

¿Cómo puedo borrar un objeto de la lista?

Depende de lo que queramos hacer tenemos varias opciones:

1. Si queremos que una posición de la lista apunte a null, tendremos que usar el método set(). En este caso, la lista seguirá teniendo el mismo tamaño, pero uno de sus elementos será null. No es algo habitual.

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("primero");
listaCadenas.add("segundo");
listaCadenas.add("tercero");
listaCadenas.set(1, null);
```

La lista queda así: [primero, null, tercero]

2. Si queremos eliminar por completo el elemento de la lista, usamos el método remove(). La lista pasará a tener un tamaño menor.

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("primero");
listaCadenas.add("segundo");
listaCadenas.add("tercero");
listaCadenas.remove(1);
```

La lista queda así: [primero, tercero]

En lugar de indicar la posición que queremos borrar, también podemos indicar el objeto:

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("primero");
listaCadenas.add("segundo");
listaCadenas.add("tercero");
listaCadenas.remove("segundo");
```

En este caso hay que tener en cuenta que sólo se borra el primer objeto que sea igual al indicado. Si hubiera más de uno, no se borrarían los demás.

3. Si queremos borrar todos los elementos de la lista, podemos utilizar el método clear():

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("primero");
listaCadenas.add("segundo");
listaCadenas.add("tercero");
listaCadenas.clear();
```

La lista quedaría vacía

¿Cómo puedo saber el tamaño de la lista?

Tenemos dos métodos para preguntar por el tamaño:

1. Si queremos saber cuál es el tamaño exacto, usamos el método size():

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("primero");
listaCadenas.add("segundo");
listaCadenas.add("tercero");
Integer tamaño = listaCadenas.size(); // será 3
```

2. Si queremos simplemente saber si la lista tiene tamaño 0, podemos usar el método isEmpty():

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("primero");
listaCadenas.add("segundo");
listaCadenas.add("tercero");
Boolean estaVacia = listaCadenas.isEmpty(); // Devolverá false en este caso
```

¿Cómo puedo localizar objetos dentro de la lista?

Tenemos métodos similares a los que usábamos con la clase String. Sería estos:

1. Si quiero saber si existe en la lista un determinado objeto, utilizo el método contains():

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("primero");
listaCadenas.add("segundo");
listaCadenas.add("tercero");
Boolean existe = listaCadenas.contains("segundo"); // Devolverá true en este caso
```

IMPORTANTE: Se usará el método equals() de los objetos para saber si son iguales.

2. Si quiero saber en qué posición está un objeto, utilizo el método indexOf() o lastIndexOf():

Ejemplo 1:

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("primero");
listaCadenas.add("segundo");
listaCadenas.add("segundo");
listaCadenas.add("tercero");
Integer primerLugar = listaCadenas.indexOf("segundo"); // será 1
```

Ejemplo 2:

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("primero");
listaCadenas.add("segundo");
listaCadenas.add("segundo");
listaCadenas.add("tercero");
Integer primerLugar = listaCadenas.lastIndexOf("segundo"); // será 2
```

Ejemplo 3:

```
List<String> listaCadenas = new ArrayList<>();
listaCadenas.add("primero");
listaCadenas.add("segundo");
listaCadenas.add("segundo");
listaCadenas.add("tercero");
Integer primerLugar = listaCadenas.indexOf("cuarto"); // será -1
```

¿Cómo puedo recorrer una lista para hacer algo con todos los elementos?

Podemos utilizar los bucles for, while y foreach que ya utilizábamos anteriormente. Veamos algunos ejemplos:

```
// Con un for
for (int i = 0; i < listaCadenas.size(); i++) {
    System.out.println(listaCadenas.get(i));
}

// Con un while
int j = 0;
while(j < listaCadenas.size()) {
    System.out.println(listaCadenas.get(j));
    j++;
}

// Con un foreach
for (String cadena : listaCadenas) {
    System.out.println(cadena);
}
```

Además de los modos tradicionales, las listas ofrecen otro modo para recorrerlas: el iterador. Se utiliza así:

```
Iterator<String> iterador = listaCadenas.iterator();
while (iterador.hasNext()) {
    String cadena = iterador.next();
    System.out.println(cadena);
}
```

IMPORTANTE: Si queremos borrar elementos de una lista a la vez que la estamos recorriendo, es obligatorio utilizar un iterador. Si no, tendremos errores.

>>>> ¿Qué es la interfaz Set?

Se trata de una interfaz de Java que define el funcionamiento de un conjunto de objetos. Algo similar a una Lista, pero con dos diferencias:

- En un conjunto no puede haber elementos repetidos. Si yo intento añadir dos veces dos objetos iguales, la segunda vez será ignorado. Para saber si dos objetos son iguales, se usará el método equals() y el método hashCode() de la clase de los objetos.
- En los conjuntos no hay índices ni posiciones. No hay un orden. Todo está como en un saco mezclado. Por tanto, no puedo obtener un objeto de la posición i-ésima, ni borrar algo de la posición i-ésima, ni añadir en la posición i-ésima. Tampoco existen método set() para cambiar algo de una posición.

Los conjuntos se usan principalmente cuando queremos tener una lista evitando repeticiones. Los métodos de la interfaz Set son similares a los de List, se pueden ver aquí:

https://docs.oracle.com/javase/8/docs/api/java/util/Set.html

¿Cómo puedo crearme uno objeto que implemente la interfaz Set?

Basta con instanciar un objeto de una clase que implemente dicha interfaz. Las clases más conocidas que implementan dicha interfaz son:

- HashSet
- LinkedHashSet
- TreeSet

Al crear el objeto, tendremos que indicar qué tipo de elementos son los que van a estar dentro, igual que con las listas. Esto lo hacemos con los caracteres "< >". Veamos un par de ejemplos:

```
// Creamos un conjunto usando la clase HashSet
Set<String> conjunto1 = new HashSet<>();

// Creamos un conjunto usando la clase LinkedHashSet
Set<String> conjunto2 = new LinkedHashSet<>();
```

¿Cómo puedo añadir y borrar elementos en un Set?

Los hacemos del mismo que en una lista. La diferencia es que no hay posiciones. Sólo podré borrar pasando el objeto que quiero borrar.

¿Cómo puedo obtener elementos en un Set?

Como no hay posiciones, no puedo pedirle que me dé el objeto que está en la posición i-ésima. Sólo podré obtener los objetos de un conjunto recorriéndolo.

¿Cómo recorrer un Set?

Como no tengo posiciones, no podré usar ni un for clásico ni un while. Tendré que utilizar o bien el foreach o bien el iterator. Estos se usan igual que en las listas.

>>>> ¿Qué es la interfaz Map?

Se trata de una interfaz de Java que define el funcionamiento de un mapa. Un mapa, o un "mapeo", es una lista de asociaciones entre pares. De esos pares, siempre tenemos una clave y un valor que le corresponde. Nosotros solicitamos al mapa el valor que corresponde a una determinada clave. Para entenderlo, veamos un par de ejemplos:

Mapa de DNI → Nombre. En este caso, la clave del mapa será un String con el DNI de cada persona y el valor será otro String con el nombre de esa persona:

key	value
37637322D	Blas de los Montes
98322381F	Abel Morillo
20998129L	Laura de Blas

Mapa de Palabra → Definición. Probablemente sea el ejemplo más sencillo de imaginar de un mapa: un diccionario. Las claves del mapa son palabras, y sus valores son la definición. En el mapa solicitamos la definición a partir de una palabra:

key	value
cuento	Narración breve de ficción.
árbol	Planta de tallo leñoso y elevado, que se ramifica a cierta altura del suelo.
pájaro	Ave, especialmente si es pequeña.

¿Qué restricciones tiene un mapa?

El conjunto de todas las claves (keys) de un mapa es como si fuera un Set (un conjunto). Por tanto, en un mapa, no puede haber dos claves repetidas. Si en el mapa del primer ejemplo intentamos añadir un nuevo par que sea (20998129L → Lucas Aguilar), como la clave 20998129L ya existe en el mapa, lo que se hace es que se sobrescribe la anterior. Es decir, "Laura de Blas" desaparece, y 20998129L ya sólo apuntará a Lucas Aguilar.

¿Qué puedo meter en un mapa?

Se puede meter cualquier tipo de objeto tanto en las claves como en los valores. Por tanto, podría hacer un mapa que relacionara Integer con Persona, o Persona con Coche, o Coche con String, o String con Object. Lo que quiera.

Ten en cuenta que para que se cumpla la restricción anterior (no se repitan claves), se usará lo mismo que en cualquier conjunto (Set), es decir, se usará el método equals() y hashCode() de la clase que usamos como Key para saber si ya existe esa clave o no.

Más info: https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

IMPORTANTE: Al igual que en un Set, los valores que voy metiendo en un Map no están ordenados. Si necesitara que tuvieran un orden, tendría que usar un SortedMap. Nosotros no vamos a ver esta interfaz en el presente curso.

¿Cómo puedo crearme un objeto que implemente la interfaz Map?

Basta con instanciar un objeto de una clase que implemente dicha interfaz. La clase más utilizada es HashMap

Al crear el objeto, tendremos que indicar qué tipo de elementos son los que van a estar dentro del mapa, igual que con las listas. Pero en este caso tendremos que indicar dos tipos de elementos: La clave (Key) y el valor (Value). Veamos unos ejemplos:

```
Map<String, Integer> mapa1 = new HashMap<>();

Map<Integer, Alumno> mapa2 = new HashMap<>();

Map<Alumno, Gato> mapa3 = new HashMap<>();
```

¿Cómo puedo añadir y borrar elementos en un Map?

Para añadir pares a un mapa tenemos que usar el método put(). A este método tenemos que pasarle dos parámetros: la clave y el valor asociado. Ejemplo:

```
Map<String, Alumno> mapa = new HashMap<>();

Alumno alumno1 = new Alumno();
mapa.put("111AA", alumno1);

Alumno alumno2 = new Alumno();
mapa.put("222BB", alumno2);
```

Para borrar una par, utilizamos el método remove() indicando únicamente la clave que queremos borrar. Eso borrará tanto la clave como el valor

```
mapa.remove("111AA");
```

Una forma alternativa de borrar es hacer un put() de una clave con valor null. En este caso, la clave seguirá existiendo, pero no estará asociada a ningún valor.

```
mapa.put("111AA", null);
```

¿Cómo puedo obtener elementos en un Mapa?

La forma de obtener un valor de un mapa a partir de su clave es usando el método get(). Si la clave no existe en el mapa o no tiene ningún valor asociado, el método nos devolverá null.

```
Alumno a = mapa.get("111AA");
```

También podemos usar el método getOrDefault() al que le pasamos dos parámetros: por un lado la clave del valor que queremos obtener; y, por otro, el valor por defecto que queremos que nos devuelva si la clave no está en el mapa.

¿Otros métodos para conocer el estado o modificar el Mapa?

Tenemos una serie de métodos similares a los de List o Set:

- Saber si el mapa está vacío: isEmpty()
- Vaciar el mapa: clear()
- Saber si el mapa contiene una clave: containsKey()
- Saber si el mapa contiene un valor determinado: containsValue()

¿Cómo recorrer un Mapa?

Para recorrer un mapa tenemos varias opciones depende de lo que nos interese:

a) Podemos recorrer la lista de todas las claves del mapa (realmente sería un conjunto Set de claves). Para ello, usamos el método keySet()

```
Set<String> keys = mapa.keySet();
for (String key : keys) {
    System.out.println("Clave : "  + key);
    System.out.println("Valor: " + mapa.get(key));
}
```

b) Podemos recorrer la lista de todos los valores del mapa. No sería una lista ni un conjunto, sería un Collection, que es como una interfaz padre de ambas con cosas comunes, pero se recorre igual. Para obtener todos los valores de un mapa usamos el método values()

```
Collection<Alumno> values = mapa.values();
for (Alumno alumno : values) {
    System.out.println(alumno);
}
```

Si nos extraña trabajar con Collection, podemos convertirla a una lista así:

```
Collection<Alumno> values = mapa.values();
List<Alumno> lista = new ArrayList<>(values);
for (Alumno alumno : lista) {
    System.out.println(alumno);
}
```

c) La otra opción es recorrer directamente el conjunto de los pares. Cada pareja de (key → value) se guarda en un objeto de tipo Entry. Para obtener el conjunto de todos los pares (de todas las Entry) podemos usar el método entrySet()

```
Set<Entry<String, Alumno>> pares = mapa.entrySet();
for (Entry<String, Alumno> par : pares) {
    System.out.println("Clave: " + par.getKey());
    System.out.println("Valor: " + par.getValue());
}
```

Si necesitamos borrar elementos mientras los recorremos, tendremos que usar, como siempre, un Iterator. El iterador lo podemos obtener del keySet (el conjunto de claves) o del entrySet (conjunto de pares).