

**ARRAYLIST****¿QUÉ ES UN ARRAYLIST?**

Un **ArrayList** es una estructura que permite almacenar datos de forma similar a como lo realizaría un Array, con la diferencia que se trata de un **almacenamiento dinámico**; es decir, no tiene un tamaño establecido pudiendo añadir tantos elementos como queramos y borrarlos si no nos hacen falta. El espacio se va adaptando de forma dinámica y automáticamente según se vaya necesitando.

**DECLARACIÓN Y PRINCIPALES MÉTODOS**

De forma general, un ArrayList puede contener objetos de cualquier tipo y se declara como:

```
ArrayList<tipo> nombreArray = new ArrayList<>();  
tipo debe ser una clase (no se pueden usar tipos primitivos).
```

Los siguientes métodos nos permiten realizar operaciones comunes tales como añadir, eliminar, buscar y modificar datos en un ArrayList.

MÉTODO	DESCRIPCIÓN
<b>size()</b>	Devuelve el número de elementos (int)
<b>add(X)</b>	Añade el objeto X al final.
<b>add(posición, X)</b>	Inserta el objeto X en la posición indicada.
<b>get(posición)</b>	Devuelve el elemento que está en la posición indicada.
<b>remove(posición)</b>	Elimina el elemento que se encuentra en la posición indicada.
<b>remove(objeto)</b>	Elimina el objeto referenciado que se le pasa como parametro
<b>clear()</b>	Elimina todos los elementos.
<b>set(posición, X)</b>	Sustituye el elemento que se encuentra en la posición indicada por el objeto X.

Cuando queramos usar ArrayList con tipos simples como char, int, boolean, double usaremos **clases envolventes**. Esto es así porque ArrayList solo puede almacenar clases/objetos.

- **int** le corresponde **Integer**.
- **double** le corresponde **Double**.
- **boolean** le corresponde **Boolean**.
- **char** le corresponde **Character**.

A continuación, se muestra el uso de alguno de los métodos disponibles en ArrayList

```
import java.util.ArrayList;
import java.util.Scanner;

public class EjemploArrayList {

    public static void main(String[] args) {
        Scanner teclado=new Scanner(System.in);
        ArrayList<String> lista_nombres=new ArrayList<>();
        String dato;

        for (int i = 1; i <= 5; i++) {
            System.out.println("Dime un nombre");
            dato=teclado.next();
            lista_nombres.add(dato);
        }

        System.out.println("Cantidad de nombre: "+lista_nombres.size());
        System.out.println("Tercer nombre: "+lista_nombres.get(2));
        //Borrar primera posicion
        lista_nombres.remove(0);
        //Sustituye la posicion 3 por Juan"
        lista_nombres.set(3, "Juan");
    }
}
```

Para recorrerlos completamente podemos usar un bucle for.

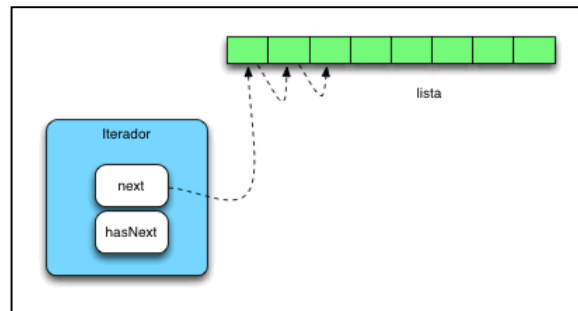
```
for(int i=0;i<lista_nombres.size();i++){
    System.out.println(lista_nombres.get(i));
}
```

El método get si vamos a recorrer el ArrayList completamente no es eficiente y desaconsejado debido a que las posiciones no son fijas ya que el objetivo de estas colecciones es añadir/borrar datos constantemente y por ello no está optimizado para acceder por posición (para ello usar un array normal).

Por ello se proporciona una variante de bucle for (**tipo for each**) que no es necesario usar los métodos y directamente nos coloca uno a uno cada elemento del ArrayList. Por cada elemento de la **lista\_nombre** el bucle da una vuelta y coloca el elemento en orden en la variable **nombre**. Si no necesitamos saber la posición, de esta forma es más compacto.

```
for (String nombre : lista_nombres) {
    System.out.println(nombre);
}
```

El bucle anterior recorre siempre completamente los datos del ArrayList y además si queremos eliminar un dato en concreto que estemos buscando obtenemos lo siguiente:



Un iterador es una estructura que nos controla cuando se llega al final del array sin necesidad de controlar una variable índice (la *i* del `for` de toda la vida) y nos proporciona uno a uno cada uno de los objetos que forman parte de la colección. Es una forma de recorrer un `ArrayList` más eficiente que **get(posicion)** y la usaremos en su lugar.

En un principio el `ArrayList` tiene un método **iterator()** que devuelve objeto iterador que en esencia solo tiene el comportamiento propio del iterador que actúa sobre ese `ArrayList`. (Mirar la figura anterior). Para ello se usa una definición que es la clase `Iterator` que al igual que `ArrayList` es genérica usando `<TipoDedatos>`. Por ejemplo, para recorrer una `ArrayList` de cervezas podemos usar este código:

```
ArrayList<Cerveza> lista=new ArrayList<>();  
//...Rellenar datos  
  
Iterator<Cerveza> it=lista.iterator();  
Cerveza objeto;  
while(it.hasNext()){//True si quedan objetos  
    objeto=it.next();//nos da el objeto y avanza al siguiente  
    System.out.println("Nombre:"+objeto.getNombre());  
}
```

- El método **hasNext** devuelve un booleano que es true mientras haya un objeto por recorrer, o sea cuando llegue al último e invoquemos a **hasNext** devolverá false.
- El método **next** nos devuelve el siguiente objeto de la lista que toca y avanza al siguiente (Mirar la figura anterior). Si queremos buscar un nombre de cerveza en concreto y poder usar un `while` que es la forma más eficiente:

```
Cerveza res = null, cerveza;  
Iterator<Cerveza> it = this.almacen.iterator();  
  
while (it.hasNext() && res == null) {  
    cerveza = it.next();  
    if (cerveza.getNombre().equalsIgnoreCase(nombre)) {  
        res = cerveza;  
    }  
}
```

En este caso no usamos la posición sino una referencia al objeto que queremos buscar, siendo **null** al principio porque indica que no lo hemos encontrado.

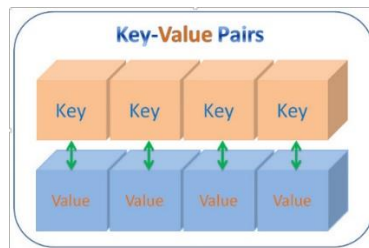
El concepto de iterador es importante porque muchas estructuras son colecciones, pero

no tienen por qué estar organizadas posicionalmente como veremos los ficheros o las bases de datos, etc. El iterador proporciona independencia de planteamiento independientemente del lenguaje y de la colección.

El ArrayList se puede ordenar con el método sort. Este método necesita que le indiquemos bajo qué criterio ordenarlo en base a los objetos que contiene. Por ejemplo, si contiene String podemos usar el método compareTo de la clase String

### HASHMAP

Dentro de las colecciones existe un tipo especial llamadas HashMap o también llamados diccionarios. Son colecciones que no se organizan posicionalmente sino en forma clave->valor (key->value). Se les llama diccionarios justamente por parecerse a un diccionario cuyo contenido está asociado a cómo encontrar el contenido, o sea, si quiero buscar la definición de mesa busco la palabra mesa, no voy buscando la posición en la que está.



Como ejemplo podríamos pensar en que una clave puede ser un número de pasaporte de una persona y como valor el objeto Persona que contiene toda la información y métodos de una persona en concreto. Aunque la clave lo más frecuente es que sea algo conciso (como un número) también puede ser otra cosa. Por ejemplo, el objeto clave podría ser una imagen, al que le correspondiera por ejemplo otra imagen valor. Por ejemplo, podríamos tener un mapa de conversión de imágenes, que, dada una imagen clave, le correspondiera una imagen valor que fuera la misma imagen escalada a un determinado tamaño. Esto no es lo más usual, pero sería posible.

Ejemplo de uso como un diccionario de español a inglés:

```
HashMap<String,String> diccionario=new HashMap<>();  
diccionario.put("gato", "cat");  
diccionario.put("perro", "dog");  
diccionario.put("España", "Spain");  
diccionario.put("pelo","hair");
```

Este tipo de dato hoy día se encuentra de una manera u otro en la mayoría de las aplicaciones en todo el mundo ya que tiene posibilidades infinitas. En PHP se llaman arrays asociativos, también se usan para intercambio de información como los objetos JSON usados en JavaScript y otros lenguajes. Por tanto, es de gran utilidad comprender su funcionamiento.

```
System.out.print("Que palabra quieres traducir al inglés:");
español=teclado.next();
System.out.println(diccionario.get(español));
```

Como acabamos de ver podemos encontrar fácilmente un objeto a través de clave, pero además un HashMap aporta las siguientes funcionalidades respecto a su contenido: Puede que una clave no exista y en ese caso lo anterior devolvería null. Para comprobar si una clave existe tenemos el método containsKey

```
System.out.print("Que palabra quieres traducir al inglés:");
español=teclado.next();

if(diccionario.containsKey(español)){
    System.out.println(diccionario.get(español));
}else{
    System.out.println("No existe traducción");
}
```

También podemos comprobar si un valor concreto está asociado a una clave, para ello usamos el método containsValue:

```
System.out.print("Palabra en ingles:");
ingles=teclado.next();
if(diccionario.containsValue(ingles)){
    System.out.println("Palabra correcta en inglés");
}else{
    System.out.println("No existe esa palabra en inglés");
}
```

Pero no podemos saber que clave le correspondería (correspondencia en sentido contrario), en este caso si es muy necesario tener una correspondencia bidireccional podríamos un HashMap en sentido inverso, depende de las necesidades de nuestra aplicación.

Los HashMap como colecciones que son también pueden ser iterados de la misma manera que un ArrayList, solo que en este caso existen dos colecciones, las claves y los valores. Los podemos tratar como tal individualmente de las siguientes maneras:

- Recorrer las claves:

```
for (String pal_esp : diccionario.keySet()) {
    System.out.println(pal_esp);
}
```

- Recorrer los valores:

```
for (String pal_ing : diccionario.values()) {
    System.out.println(pal_ing);
}
```

- Recorrer los dos al mismo tiempo (EntrySet) a los pares claves-valor se les llama entradas:

```
for (Entry<String,String> traduccion: diccionario.entrySet()) {  
    System.out.println(traduccion.getKey()+" es "+traduccion.getValue());  
}
```

Este código se utiliza para llevar un registro del número total de goles marcados por varios equipos, y luego imprime estos datos al final del proceso. Utilizando un HashMap llamado goles para asociar cadenas de texto (strings) con valores enteros (números de goles)

```
HashMap<String,Integer> goles=new HashMap<>();  
//Asociar string a valores  
goles.put("FC Barcelona",15);  
goles.put("PSG",75);  
goles.put("Real Madrid",7);  
goles.put("Manchester United",41);
```

```
for(Map.Entry<String,Integer> equipo:goles.entrySet()){  
    System.out.println(equipo.getKey()+":"+equipo.getValue());  
}
```

Se itera a través de todas las entradas en el mapa goles usando un bucle for-each. Para cada entrada, se imprime el nombre del equipo (equipo.getKey()) seguido por el número total de goles que ha marcado (equipo.getValue()).

Lo normal es que los datos provengan de otras fuentes y el HashMap sea el contenedor para resumir toda esa información

```
//datos recolectados de una serie de objetos  
for(Club c:this.liga){  
    int actual=goles.getDefault(c.getNombre(),0);  
    goles.put(c.getNombre(),actual+c.getGoles());  
}
```

En este bucle for, se está iterando a través de una lista de objetos Club. Para cada objeto Club, se obtiene el nombre del equipo (c.getNombre()) y se verifica si ya hay una entrada correspondiente en el mapa goles. Si existe, se obtiene el valor actual (el número actual de goles para ese equipo). Si no existe (usando getDefault(), que devuelve 0 si no hay una entrada para la clave especificada), se inicializa el valor como 0. Luego, se suma el número de goles del equipo actual (c.getGoles()) al valor actual y se vuelve a poner en el mapa goles.