# ARRAYLISTS

## Contenido


1 ArrayList ..... 1

1.1 Declaración de un objeto ArrayList ..... 1

1.2 Creación de un ArrayList ..... 2

1.3 Añadir elementos al final de la lista ..... 2

1.4 Añadir elementos en cualquier posición de la lista ..... 3

1.5 Suprimir elementos de la lista ..... 3

1.6 Consulta de un determinado elemento de la lista ..... 3

1.7 Modificar un elemento contenido en la lista ..... 4

1.8 Buscar un elemento ..... 4

1.9 Recorrer el contenido de la lista ..... 4

1.10 Otros métodos ..... 5

1.11 Ejemplos de uso de ArrayList ..... 6

1.12 Copiar un ArrayList ..... 7

1.13 ArrayList como parámetro de un método ..... 8


# 1 ArrayList

La clase ArrayList permite el almacenamiento de datos en memoria de forma similar a los arrays convencionales, pero con la gran ventaja de que el número de elementos que puede almacenar es dinámico. La cantidad de elementos que puede almacenar un array está limitada a la dimensión que se declara a la hora de crearlo o inicializarlo. Los ArrayList, en cambio, pueden almacenar un número variable de elementos sin estar limitados por un número prefijado.

## 1.1 Declaración de un objeto ArrayList

La declaración genérica de un ArrayList se hace con el formato siguiente:

**ArrayList nombreDeLista;**

De esta manera no se indica el tipo de datos que va a contener. Sin embargo, suele ser recomendable indicar el tipo (y nos lo va a pedir el compilador), para

que así se empleen las operaciones y métodos adecuados para el tipo de datos manejado.

Para especificar el tipo de datos que va a contener la lista se debe indicar entre los caracteres '<' y '>' la clase de los objetos que se almacenarán:

**ArrayList<nombreClase> nombreDeLista;**

En caso de almacenar datos de un tipo básico como char, int, double, etc, se debe especificar el nombre de la clase asociada: Character, Integer, Double, etc.

**Ej**: ArrayList<String> listaPaises;
ArrayList<Integer> edades;

## 1.2 Creación de un ArrayList

Para crear un ArrayList se puede seguir el siguiente formato:

nombreDeLista = new ArrayList();

Se puede también declarar la lista a la vez que se crea:

ArrayList<nombreClase> nombreDeLista = new ArrayList<nombreClase> ();

**Ej**: ArrayList<String> listaPaises = new ArrayList<String>();

La clase ArrayList forma parte del paquete java.util, por lo que hay que incluir en la parte inicial del código la importación de ese paquete.

## 1.3 Añadir elementos al final de la lista

El método **add** posibilita añadir elementos. Los elementos que se van añadiendo de esta forma, se colocan después del último elemento que hubiera en el ArrayList. El primer elemento que se añada se colocará en la posición 0.

**Ej**:

```
ArrayList<String> listaPaises = new ArrayList();
listaPaises.add("España"); //Ocupa la posición 0
listaPaises.add("Francia"); //Ocupa la posición 1
listaPaises.add("Portugal"); //Ocupa la posición 2
```

**Ej**: Se pueden crear ArrayList para guardar datos numéricos de igual manera

```
ArrayList<Integer> edades = new ArrayList<Integer> ();
edades.add(22);
edades.add(31);
edades.add(18);
```

## 1.4 Añadir elementos en cualquier posición de la lista

También es posible insertar un elemento en una determinada posición desplazando el elemento que se encontraba en esa posición y todos los siguientes, una posición más. Para ello, se emplea también el método **add** indicando como primer parámetro el número de la posición donde se desea colocar el nuevo elemento. Como siempre el primer hueco está en la posición 0.

**Ej**:
```
ArrayList<String> listaPaises = new ArrayList();
listaPaises.add("España");
listaPaises.add("Francia");
listaPaises.add("Portugal"); //El orden es: España, Francia, Portugal
listaPaises.add(1, "Italia"); //Ahora es: España, Italia, Francia, Portugal
```

Si se intenta insertar en una posición que no existe, se produce una excepción (IndexOutOfBoundsException)

## 1.5 Suprimir elementos de la lista

Si se quiere eliminar un determinado elemento de la lista se puede emplear el método **remove** al que se le puede indicar por parámetro un valor int con la posición a suprimir, o bien, se puede especificar directamente el elemento a eliminar si es encontrado en la lista.

**Ej**:
```
ArrayList<String> listaPaises = new ArrayList();
listaPaises.add("España");
listaPaises.add("Francia");
listaPaises.add("Portugal"); //El orden es: España, Francia, Portugal
listaPaises.add(1, "Italia"); //Ahora es: España, Italia, Francia, Portugal
listaPaises.remove(2); //Eliminada Francia, queda: España, Italia, Portugal
listaPaises.remove("Portugal"); //Eliminada Portugal, queda: España, Italia
```

## 1.6 Consulta de un determinado elemento de la lista

El método **get** permite obtener el elemento almacenado en una determinada posición que es indicada con un parámetro de tipo int. Con el elemento obtenido se podrá realizar cualquiera de las operaciones posibles según el tipo de dato del elemento (asignar el elemento a una variable, incluirlo en una expresión, mostrarlo por pantalla, etc).

**Ej**: System.out.println(listaPaises.get(3)); // Mostraría: Portugal

## 1.7 Modificar un elemento contenido en la lista

Es posible modificar un elemento que previamente ha sido almacenado en la lista utilizando el método **set**. Como primer parámetro se indica, con un valor int, la posición que ocupa el elemento a modificar, y en el segundo parámetro se especifica el nuevo elemento que ocupará dicha posición sustituyendo al elemento anterior. La posición debe estar ocupada ya por otro elemento.

**Ej**: En la lista de países se desea modificar el que ocupa la posición 1 (segundo en la lista) por "Alemania".

```
listaPaises.set(1, "Alemania");
```

## 1.8 Buscar un elemento

La clase ArrayList facilita mucho las búsquedas de elementos gracias al método **indexOf** que retorna, con un valor int, la posición que ocupa el elemento que se indique por parámetro. Si el elemento se encontrara en más de una posición, este método retorna la posición del primero que se encuentre. El método lastIndexOf obtiene la posición del último encontrado.

En caso de que no se encuentre en la lista el elemento buscado, se obtiene el valor -1.

**Ej**: Comprobar si Francia está en la lista, y mostrar su posición.

```
String paisBuscado = "Francia";
int pos = listaPaises.indexOf(paisBuscado);
if (pos != -1)
        System.out.println(paisBuscado + " encontrado en la posición: "+pos);
else
        System.out.println(paisBuscado + " no se ha encontrado");
```

## 1.9 Recorrer el contenido de la lista

Es posible obtener cada uno de los elementos de la lista utilizando un bucle con tantas iteraciones como elementos contenga, de forma similar a la usada con los arrays convencionales. Para obtener el número de elementos de forma automática se puede emplear el método **size**() que devuelve un valor int con el número de elementos que contiene la lista.

**Ej**:
```
for (int i=0;  i<listaPaises.size();  i++)
        System.out.println(listaPaises.get(i));
```

También se puede emplear otro formato del bucle for (bucle **foreach**) en el que se va asignando cada elemento de la lista a una variable declarada del mismo tipo que los elementos del ArrayList.

**Ej**: for (String pais:listaPaises)
System.out.println(pais);

Si el ArrayList contiene objetos de tipos distintos o se desconoce el tipo se pondría.

for(Object o: nombreArrayList)

También es posible hacerlo utilizando un objeto **Iterator**. La ventaja de usar un Iterador es que se puede usar con cualquier tipo de colección y además, en el transcurso de un bucle podemos eliminar elementos de la colección. Iterator tiene como métodos:

**hasNext**: devuelve true si hay más elementos en el array.
**next**: devuelve el siguiente objeto contenido en el array.
**remove**: borra el elemento que se está recorriendo dentro del iterador.

**Ej**:
```
ArrayList<Integer> nros = new ArrayList<Integer>();
//se insertan elementos
Iterator<Integer> it = nros.iterator();
//se crea el iterador it para el ArrayList nros
while (it.hasNext()) //mientras queden elementos
        System.out.println(it.next()); //se obtienen y se muestran
```

## 1.10 Otros métodos

void **clear**(): Borra todo el contenido de la lista.
Object **clone**(): Retorna una copia de la lista.
boolean **contains**(Object elemento): Retorna true si se encuentra el elemento indicado en la lista, y false en caso contrario.
boolean **isEmpty**(): Retorna true si la lista está vacía.

## 1.11 Ejemplos de uso de ArrayList

**Ej**:

```
ArrayList<String> nombres = new ArrayList<String>();
nombres.add("Ana");
nombres.add("Luisa");
nombres.add("Felipe");
System.out.println(nombres); // [Ana, Luisa, Felipe]
nombres.add(1, "Pablo");
System.out.println(nombres); // [Ana, Pablo, Luisa, Felipe]
nombres.remove(0);
System.out.println(nombres); // [Pablo, Luisa, Felipe]
nombres.set(0,"Alfonso");
System.out.println(nombres); // [Alfonso, Luisa, Felipe]
String s = nombres.get(1);
String ultimo = nombres.get(nombres.size() – 1);
System.out.println(s + " " + ultimo); // Luisa Felipe
```

**Ej**: Escribe un programa que lea números enteros y los guarde en un ArrayList hasta que se lea un 0 y muestra los números leídos, su suma y su media.

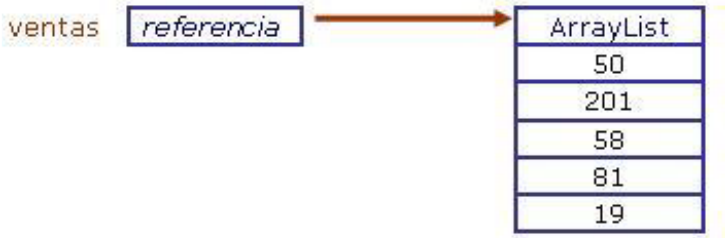
```
import java.util.*;
public class ArrayList2 {
	public static void main(String[] args) {
		Scanner teclado = new Scanner(System.in);
		ArrayList<Integer> numeros = new ArrayList<Integer>();
		int n;
		do {
			System.out.println("Introduce números enteros. 0 para
		acabar: ");
			System.out.println("Numero: ");
			n = teclado.nextInt();
			if (n != 0)
				numeros.add(n);
		}while (n != 0);
		System.out.println("Ha introducido: " + numeros.size() + "
		números:");
		System.out.println(numeros); //Muestra el arrayList completo
		Iterator<Integer> it = numeros.iterator();
```

```
            while(it.hasNext())
                System.out.println(it.next());
            double suma = 0;
            for(Integer i: numeros)
                suma = suma + i;
            System.out.println("Suma: " + suma);
            System.out.println("Media: " + suma/ numeros.size());
        }
}
```

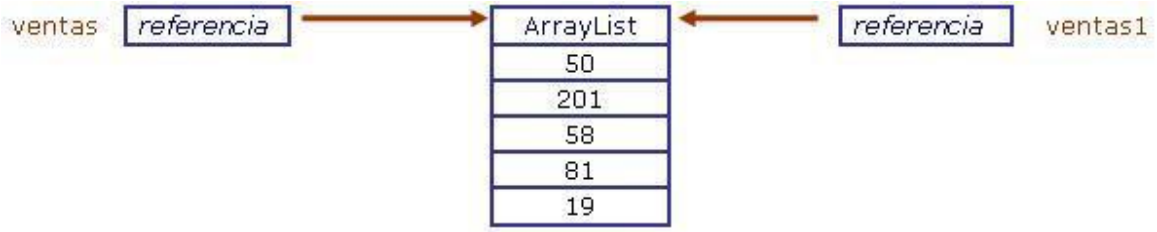
## 1.12 Copiar un ArrayList

El nombre de un ArrayList contiene la referencia al ArrayList, es decir, la dirección de memoria donde se encuentra el ArrayList, igual que sucede con los arrays estáticos.

**Ej**: Se tiene un ArrayList de enteros llamado ventas.



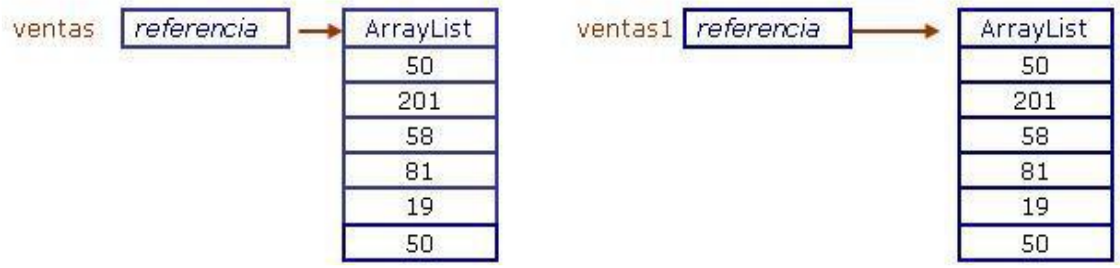

La instrucción ArrayList<Integer> ventas1 = ventas; no copia el array ventas en un nuevo ArrayList de nombre ventas1 sino que crea una referencia ventas1 al ArrayList.




De esta forma se tiene dos formas de acceder al mismo ArrayList: mediante la referencia ventas y mediante la referencia ventas1.

Para hacer una copia se puede hacer de forma manual elemento a elemento o se puede pasar la referencia del ArrayList original al constructor del nuevo.

```
ArrayList<Integer> ventas1 = new ArrayList<Integer>(ventas);
```




## 1.13 ArrayList como parámetro de un método

Un ArrayList puede ser usado como parámetro de un método. Además, un método también puede devolver un ArrayList mediante la sentencia return.

**Ej**: Método que recibe un ArrayList de String y lo modifica invirtiendo su contenido.

```
import java.util.*;
public class ArrayList1 {
public static void main(String[] args) {
        ArrayList<String> nombres = new ArrayList<String>();
        nombres.add("Ana");
        nombres.add("Luisa");
        nombres.add("Felipe");
        nombres.add("Pablo");
        System.out.println(nombres);
        nombres = invertir(nombres);
        System.out.println(nombres);
}
public static ArrayList<String> invertir(ArrayList<String> nombres) {
        ArrayList<String> resultado = new ArrayList<String>();
        for (int i = nombres.size() - 1; i >= 0; i--)
                resultado.add(nombres.get(i));
        return resultado;
}
}
```

Como **ejercicio**, hacer un nuevo método invertir, que modifique el ArrayList que se recibe en lugar de devolver una copia:

```
        public static void invertir(ArrayList<String> nombres)
```