

# Colecciones. ArrayList

Encarnación Sánchez Gallego

# CONTENIDOS

- Colecciones
- Introducción a ArrayList
- Declaración de ArrayList
- Almacenamiento en un ArrayList
- Métodos de acceso y manipulación
- Recorrido de un ArrayList
- ArrayList. Ejemplos de uso con tipos primitivos.
- ArrayList. Ejemplos de uso con objetos

# Colecciones

Una colección es simplemente un objeto que agrupa varios elementos en uno solo.

Se utilizan para guardar y manipular datos así como transmitir información entre métodos

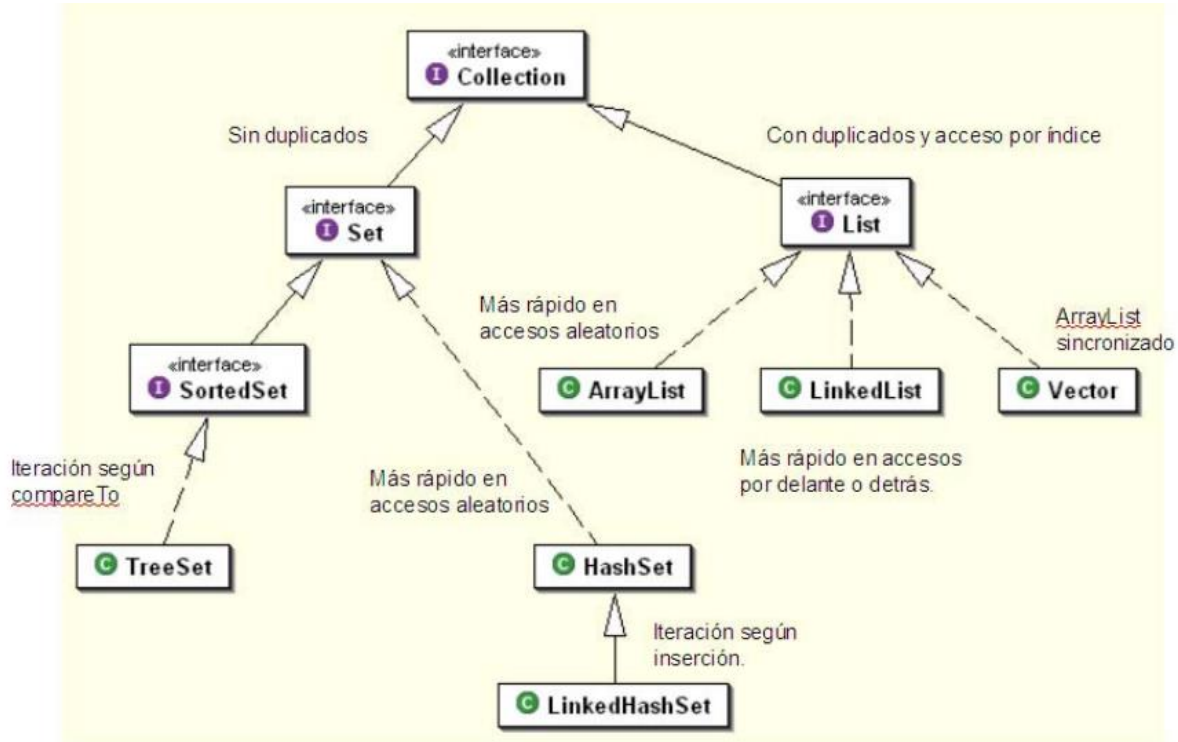
En Java tenemos un framework de colecciones, implementado mediante:

- **Interfaces:** representaciones abstractas de las colecciones que permiten usarlas sin conocer sus detalles.
- **Implementaciones:** colecciones concretas.
- **Algoritmos:** métodos que permiten realizar operaciones como búsquedas, ordenaciones, etc...

# Colecciones

A continuación se muestra la organización jerárquica de este framework y una forma rápida de saber cuál podría ser la elegida a utilizar en cada caso. En el caso de los Collection, hay una clara diferenciación en cuanto si permite duplicados accediendo por índice `java.util.List`, o sin duplicados sin posibilidad de acceso vía índice, `java.util.Set`

# Colecciones



# Colecciones

Todas las colecciones se encuentran en el paquete `java.util.*`; siendo `java.util.Collection` y `java.util.Map` las raíces de la jerarquía de las colecciones.

Existirán especializaciones que permitan elementos duplicados o no, que permitan ordenar los elementos o no, etc.

Estas interfaces contiene la definición de todos los métodos genéricos que deben implementar las colecciones.

# Introducción a ArrayList

Un *ArrayList* es una estructura de datos dinámica del tipo **colección** que implementa una lista de tamaño variable. Es similar a un *array* con las ventajas de que **su tamaño crece dinámicamente conforme se añaden elementos** (no es necesario fijar su tamaño al crearlo) y **permite almacenar datos y objetos de cualquier tipo**.

El API de la clase ArrayList se puede consultar [aquí](#)

# Declaración

Un *ArrayList* se declara como una clase más:

```
ArrayList lista = new ArrayList();
```

Necesitaremos importar la clase:

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



# Almacenamiento de un ArrayList

Para insertar datos o elementos en un ArrayList puede utilizarse el método *add()*.

En el siguiente ejemplo insertamos datos int, double, char y String:

```
lista.add(-25);
```

```
lista.add(3.14);
```

```
lista.add('A');
```

```
lista.add("Luis");
```

```
lista.add(p);
```

# Almacenamiento de un ArrayList

En el siguiente ejemplo insertamos objetos de la clase *Persona*:

```
lista.add(new Persona("28751533Q", "María", "Maida García", 37));
```

```
lista.add(new Persona("65971552A", "Luis", "González Collado", 17));
```

```
lista.add(new Persona("16834954R", "Raquel", "Dobón Pérez", 62));
```

En este código cada posición de la lista apuntará a un objeto *Persona* diferente. También se podría haber hecho lo siguiente:

```
Persona p1 = new Persona("28751533Q", "María", "Maida García", 37);
```

```
Persona p2 = new Persona("65971552A", "Luis", "González Collado", 17);
```

```
Persona p3 = new Persona("16834954R", "Raquel", "Dobón Pérez", 62);
```

```
lista.add(p1);
```

```
lista.add(p2);
```

```
lista.add(p3);
```

# Métodos de acceso y manipulación

Un *ArrayList* da a cada elemento insertado un índice numérico que hace referencia a la posición donde se encuentra, de forma similar a un array. Así, el primer elemento se encuentra almacenado en el índice 0, el segundo en el 1, el tercero en el 2 y así sucesivamente.

# Métodos de acceso y manipulación

- *int* **size()**; devuelve el número de elementos de la lista.
- *E* **get(int index)**; devuelve una referencia al elemento en la posición index.
- *void* **clear()**; elimina todos los elementos de la lista. Establece el tamaño a cero.
- *boolean* **isEmpty()**; retorna true si la lista no contiene elementos.
- *boolean* **add(E element)**; inserta element al final de la lista y devuelve true.
- *void* **add(int index, E element)**; inserta element en la posición index de la lista. Desplaza una posición todos los demás elementos de la lista (no sustituye ni borra otros elementos).

# Métodos de acceso y manipulación

- *void* **set**(*int index*, *E element*); sustituye el elemento en la posición *index* por *element*.
- *boolean* **contains**(*Object o*); busca el objeto *o* en la lista y devuelve *true* si existe. Utiliza el método *equals()* para comparar objetos.
- *int* **indexOf**(*Object o*); busca el objeto *o* en la lista, empezando por el principio, y devuelve el índice dónde se encuentre. Devuelve -1 si no existe. Utiliza *equals()* para comparar objetos.
- *int* **lastIndexOf**(*Object o*); como *indexOf()* pero busca desde el final de la lista.
- *E* **remove**(*int index*); elimina el elemento en la posición *index* y lo devuelve.
- *boolean* **remove**(*Object obj*); elimina la primera ocurrencia de *obj* en la lista. Devuelve *true* si lo ha encontrado y eliminado, *false* en otro caso. Utiliza *equals()* para comparar objetos.
- *void* **remove**(*int index*); Elimina el objeto de la lista que se encuentra en la posición *index*. Es más rápido que el método anterior ya que no necesita recorrer toda la lista.

# Recorrido de un ArrayList

Hay dos maneras diferentes en las que se puede iterar (recorrer) una colección del tipo ArrayList.

1. Utilizando el bucle *for* y el método *get()* con un índice.

```
for(int i = 0; i < lista.size(); i++) {
```

```
// Lo imprimimos por pantalla
```

```
System.out.println(lista.get(i));
```

```
}
```

# Recorrido de un ArrayList

2. Usando un objeto *Iterator* que permite recorrer listas como si fuese un índice. Se necesita importar la clase con *import java.util.Iterator;*

Tiene tres métodos principales:

- *hasNext()*: Verifica si hay más elementos.
- *next()*: devuelve el objeto actual y avanza al siguiente.
- *remove()*: para eliminar el elemento del Iterator.

Ejemplo:

```
Iterator iter = lista.iterator(); // Creamos el Iterator a partir de la lista  
while(iter.hasNext()) { // Mientras haya siguiente en la lista  
    System.out.println(iter.next()); // Lo imprimimos por pantalla  
}
```

# ArrayList. Ejemplos de uso con tipos primitivos.

## Ejemplo ArrayList de Strings

En el siguiente fragmento de código, declaramos un ArrayList de Strings y lo rellenamos con 10 Strings (Elemento i).

Esto lo hacemos con el método *"add()"*.

Después añadimos un nuevo elemento al ArrayList en la posición '2' (con el metodo *"add(posición,elemento)"*) que le llamaremos "Elemento 3" y posteriormente imprimiremos el contenido del ArrayList, recorriendolo con un Iterator.



El fragmento de este código es el siguiente:

```
// Declaración el ArrayList
ArrayList<String> nombreArrayList = new ArrayList<String>();

// Añadimos 10 Elementos en el ArrayList
for (int i=1; i<=10; i++){
    nombreArrayList.add("Elemento "+i);
}

// Añadimos un nuevo elemento al ArrayList en la posición 2
nombreArrayList.add(2, "Elemento 3");

// Declaramos el Iterador e imprimimos los Elementos del ArrayList
Iterator<String> nombreIterator = nombreArrayList.iterator();
while(nombreIterator.hasNext()){
    String elemento = nombreIterator.next();
    System.out.print(elemento+" / ");
}
```

Ejecutando esta código obtenemos por pantalla lo siguiente:

```
Elemento 1 / Elemento 2 / Elemento 3 / Elemento 3 / Elemento 4 / Elemento 5 /  
Elemento 6 / Elemento 7 / Elemento 8 / Elemento 9 / Elemento 10 /
```

Como se observa en el resultado tenemos repetido el elemento "*Elemento 3*" dos veces y esto lo hemos puesto a propósito para mostrar el siguiente ejemplo. Ahora para seguir trabajando con los ArrayList, lo que vamos a hacer es mostrar el número de elementos que tiene el ArrayList y después eliminaremos el primer elemento del ArrayList y los elementos del ArrayList que sean iguales a "*Elemento 3*", que por eso lo hemos puesto repetido.

El "*Elemento 3*" lo eliminaremos con el método "remove()" del iterador. A continuación mostramos el código que realiza lo descrito:

```

// Recordar que previamente ya hemos declarado el ArrayList y el Iterator de la siguiente forma:
// ArrayList<String> nombreArrayList = new ArrayList<String>();
// Iterator<String> nombreIterator = nombreArrayList.iterator();

// Obtenemos el numero de elementos del ArrayList
int numElementos = nombreArrayList.size();
System.out.println("\nEl ArrayList tiene "+numElementos+" elementos");

// Eliminamos el primer elemento del ArrayList, es decir el que ocupa la posición '0'
System.out.println("n... Eliminamos el primer elemento del ArrayList (" +nombreArrayList.get(0)+")...");
nombreArrayList.remove(0);

// Eliminamos los elementos de ArrayList que sean iguales a "Elemento 3"
System.out.println("n... Eliminamos los elementos de ArrayList que sean iguales a "Elemento 3" ...");
nombreIterator = nombreArrayList.iterator();
while(nombreIterator.hasNext()){
    String elemento = nombreIterator.next();
    if(elemento.equals("Elemento 3"))
        nombreIterator.remove();          // Eliminamos el Elemento que hemos obtenido del Iterator
}

// Imprimimos el ArrayList despues de eliminar los elementos iguales a "Elemento 3"
System.out.println("nImprimimos los elementos del ArrayList tras realizar las eliminaciones: ");
nombreIterator = nombreArrayList.iterator();
while(nombreIterator.hasNext()){
    String elemento = nombreIterator.next();
    System.out.print(elemento+" / ");
}

// Mostramos el numero de elementos que tiene el ArrayList tras las eliminaciones:
numElementos = nombreArrayList.size();
System.out.println("nNumero de elementos del ArrayList tras las eliminaciones = "+numElementos);

```

Como salida a este código tenemos lo siguiente:

```
El ArrayList tiene 11 elementos

... Eliminamos el primer elemento del ArrayList (Elemento 1)...

... Eliminamos los elementos de ArrayList que sean iguales a "Elemento 3" ...

Imprimimos los elementos del ArrayList tras realizar las eliminaciones:
Elemento 2 / Elemento 4 / Elemento 5 / Elemento 6 / Elemento 7 / Elemento 8 / Elemento 9 / Elemento 10 /

Numero de elementos del ArrayList tras las eliminaciones = 8
```

Si os fijáis, hemos eliminado 3 elementos del ArrayList de dos formas distintas, preguntando por la posición que ocupa un elemento en el ArrayList y preguntando por el contenido de algún elemento del ArrayList

# ArrayList. Ejemplos de uso con objetos

Por lo general los ArrayList se suelen utilizar con objetos más que con estructuras atómicas de datos, ya que los ArrayList en Java son estructuras muy potentes y sencillas de manejar.

Ahora vamos a poner un ejemplo de la utilización de ArrayList con Objetos.

Tenemos la clase Producto con:

- Dos atributos: nombre (String) y cantidad (int).
- Un constructor con parámetros.
- Un constructor sin parámetros.
- Métodos get y set asociados a los atributos.

```
9 public class Producto {
10     // Atributos
11     private String nombre;
12     private int cantidad;
13
14     // Métodos
15
16     // Constructor con parámetros donde asignamos el valor dado a los atributos
17     public Producto(String nom, int cant){
18         this.nombre = nom;
19         this.cantidad = cant;
20     }
21
22     // Constructor sin parámetros donde inicializamos los atributos
23     public Producto(){
24         // La palabra reservada null se utiliza para inicializar los objetos,
25         // indicando que el puntero del objeto no apunta a ninguna dirección
26         // de memoria. No hay que olvidar que String es una clase.
27         this.nombre = null;
28         this.cantidad = 0;
29     }
30
```

```
31 // Métodos get y set
32 [-] public String getNombre() {
33     return nombre;
34 }
35
36 [-] public void setNombre(String nombre) {
37     this.nombre = nombre;
38 }
39
40 [-] public int getCantidad() {
41     return cantidad;
42 }
43
44 [-] public void setCantidad(int cantidad) {
45     this.cantidad = cantidad;
46 }
47 }
```

En el programa principal creamos una lista de productos y realizamos operaciones sobre ella:


```
8  import java.util.ArrayList;
9  import java.util.Iterator;
10
11  public class Ejemplos {
12
13      public static void main(String[] args) {
14
15          // Definimos 5 instancias de la Clase Producto
16          Producto p1 = new Producto("Pan", 6);
17          Producto p2 = new Producto("Leche", 2);
18          Producto p3 = new Producto("Manzanas", 5);
19          Producto p4 = new Producto("Brocoli", 2);
20          Producto p5 = new Producto("Carne", 2);
21
22          // Definir un ArrayList
23          ArrayList lista = new ArrayList();
24
25          // Colocar Instancias de Producto en ArrayList
26          lista.add(p1);
27          lista.add(p2);
28          lista.add(p3);
29          lista.add(p4);
30      }
```



```
30
31 // Añadimos "Carne" en la posición 1 de la lista
32 lista.add(1, p5);
33
34 // Añadimos "Carne" en la última posición
35 lista.add(p5);
36
37 // Imprimir contenido de ArrayLists
38 System.out.println(" - Lista con " + lista.size() + " elementos");
39
40 // Definir Iterator para extraer/imprimir valores
41 // si queremos utilizar un for con el iterador no hace falta poner el incremento
42 for( Iterator it = lista.iterator(); it.hasNext(); )
43 {
44     // Hacemos un casting para poder guardarlo en una variable Producto
45     Producto p = (Producto)it.next();
46     System.out.println(p.getNombre() + " : " + p.getCantidad());
47 }
48
```

```
49 // Eliminar elemento de ArrayList
50 lista.remove(2);
51 System.out.println(" - Lista con " + lista.size() + " elementos");
52
53 // Definir Iterator para extraer/imprimir valores
54 for( Iterator it2 = lista.iterator(); it2.hasNext();) {
55     Producto p = (Producto)it2.next();
56     System.out.println(p.getNombre() + " : " + p.getCantidad());
57 }
58
59 // Eliminar todos los valores del ArrayList
60 lista.clear();
61 System.out.println(" - Lista final con " + lista.size() + " elementos");
62 }
63 }
```

## Salida del código:



```
run:
- Lista con 6 elementos
Pan : 6
Carne : 2
Leche : 2
Manzanas : 5
Brocoli : 2
Carne : 2
- Lista con 5 elementos
Pan : 6
Carne : 2
Manzanas : 5
Brocoli : 2
Carne : 2
- Lista final con 0 elementos
BUILD SUCCESSFUL (total time: 0 seconds)
```