

Programación Frontend y Backend

BLOQUE JAVA

Colecciones



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



**Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil**
El FSE invierte en tu futuro



01



MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO

EOI Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



Array

Problemática

- Un Array es la manera más eficiente de almacenar y acceder a un conjunto de elementos.
- Permite almacenar tanto referencias a objetos como tipos de datos primitivos.
- Se puede definir un array de un tipo determinado (String []) lo cual ayuda a evitar “bugs” en el programa.
- Desventaja: Su tamaño tiene que ser fijado y no puede cambiarse en tiempo de ejecución.

Array

```
String palabra;
Scanner sc = new Scanner(System.in);
String[] palabras = new String[2];
palabras[0] = "Jengibre";
palabras[1] = "Desodorante";

do {
    System.out.println("Escribe palabras, (Escribe FIN para terminar):");
    palabra = sc.nextLine();
    //Si la palabra no es FIN "redimensionamos" el array
    if(!palabra.equalsIgnoreCase("FIN")) {
        String[] palabrasAux = new String[palabras.length+1];
        for(int i = 0; i < palabras.length ; i++) {
            palabrasAux[i] = palabras[i];
        }
        palabrasAux[palabras.length] = palabra;
        //Reemplazamos el array de palabras con el auxiliar
        palabras = palabrasAux;
    }
} while (!palabra.equalsIgnoreCase("FIN"));

//Visualizamos el contenido del array Palabras
for(String p : palabras) {
    System.out.println(p);
}
```

02

Colecciones



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



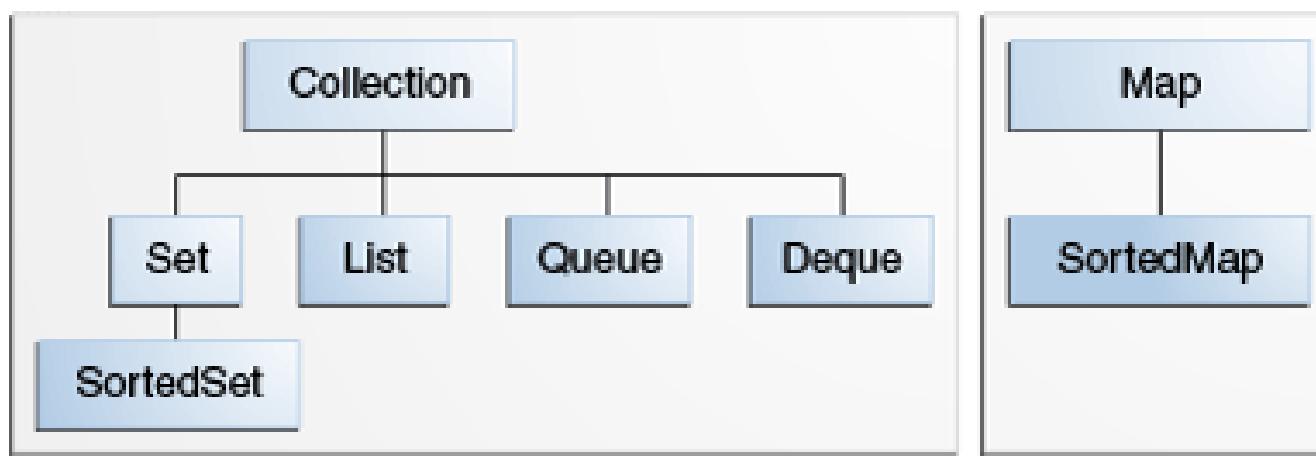
Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



Colecciones

- Una colección representa un grupo de elementos.
- La interfaz **Collection** nos permite almacenar cualquier tipo de objeto y usar una serie de métodos comunes.
- No tenemos que especificar ningún tamaño en nuestra colección.
- Se expanden automáticamente.
- Posee métodos muy útiles para programar con colecciones.
- Posee una interfaz Polimórfica esto facilita la rápida adaptación entre una colección y otra.

Colecciones (Interfaces)



Colecciones (Interfaces)

<https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html>

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

<https://docs.oracle.com/javase/8/docs/api/java/util/SortedSet.html>

<https://docs.oracle.com/javase/8/docs/api/java/util/List.html>

<https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html>

<https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html>

<https://docs.oracle.com/javase/8/docs/api/java/util/Map.html>

<https://docs.oracle.com/javase/8/docs/api/java/util/SortedMap.html>

Colecciones (Interfaces)

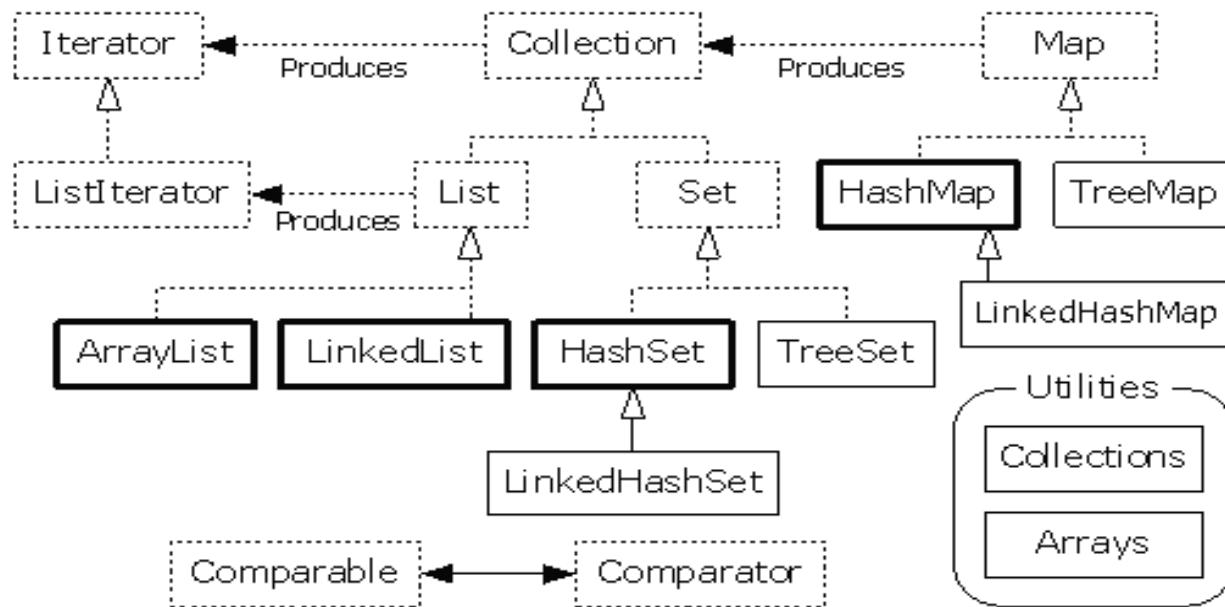
Set → No puede tener elementos duplicados.

List → Almacena los elementos en algún orden secuencial.

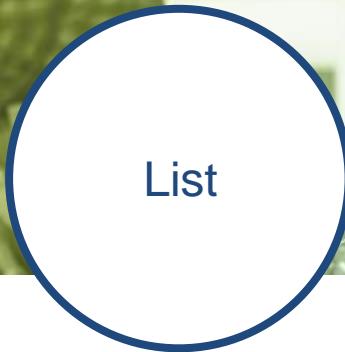
Queue → Cola FIFO (*First In First Out*)

Map → Grupo de pares (*Clave, Valor*).

Colecciones (Implementaciones)



03



MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO

EOI Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



List (Interfaz)

Este tipo de colección es el más utilizado, 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.

Modifier and Type	Method and Description
boolean	add(E e) Appends the specified element to the end of this list (optional operation).
void	add(int index, E element) Inserts the specified element at the specified position in this list (optional operation).
boolean	addAll(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
boolean	addAll(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
int	size() Returns the number of elements in this list.
default void	sort(Comparator<? super E> c) Sorts this list according to the order induced by the specified Comparator .

List (Interfaz)

void

clear()

Removes all of the elements from this list (optional operation).

boolean

contains(Object o)

Returns **true** if this list contains the specified element.

boolean

containsAll(Collection<?> c)

Returns **true** if this list contains all of the elements of the specified collection.

boolean

equals(Object o)

Compares the specified object with this list for equality.

E

get(int index)

Returns the element at the specified position in this list.

int

indexOf(Object o)

Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.

boolean

isEmpty()

Returns **true** if this list contains no elements.

List (Interfaz)

E `remove(int index)`
Removes the element at the specified position in this list (optional operation).

boolean `remove(Object o)`
Removes the first occurrence of the specified element from this list, if it is present (optional operation).

boolean `removeAll(Collection<?> c)`
Removes from this list all of its elements that are contained in the specified collection (optional operation).

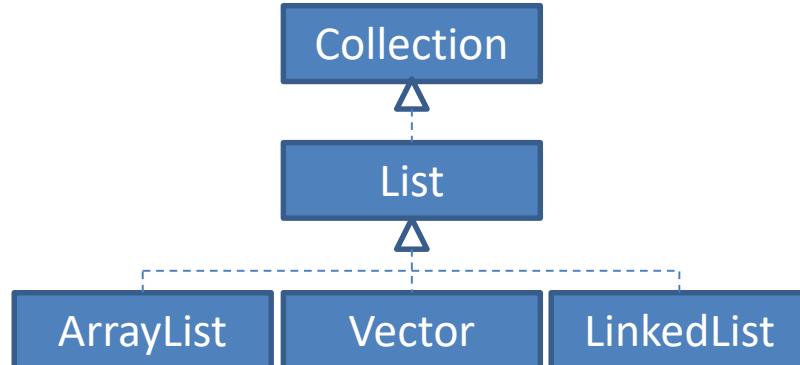
default void `replaceAll(UnaryOperator<E> operator)`
Replaces each element of this list with the result of applying the operator to that element.

List<E> `subList(int fromIndex, int toIndex)`
Returns a view of the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.

Object[] `toArray()`
Returns an array containing all of the elements in this list in proper sequence (from first to last element).

<T> T[] `toArray(T[] a)`
Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.

List (Implementación)



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.

VECTOR: El Vector es una implementación similar al ArrayList. Cuando el tamaño supera el máximo actual, este dobla su tamaño.

LINKEDLIST: En este caso se implementa la lista mediante una lista doblemente enlazada. Adecuada para operaciones de inserción y eliminación de elementos.

ArrayList

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

```
//ArrayList
List<String> palabras = new ArrayList();
palabras.add("Manzana");
palabras.add(0, "Castaña");
palabras.add(1, "Lechuga");

for(String palabra : palabras) {
    System.out.println(palabra);
}
```

Collection



List



ArrayList

Vector

LinkedList

<terminated> EjemploColecciones

Castaña

Lechuga

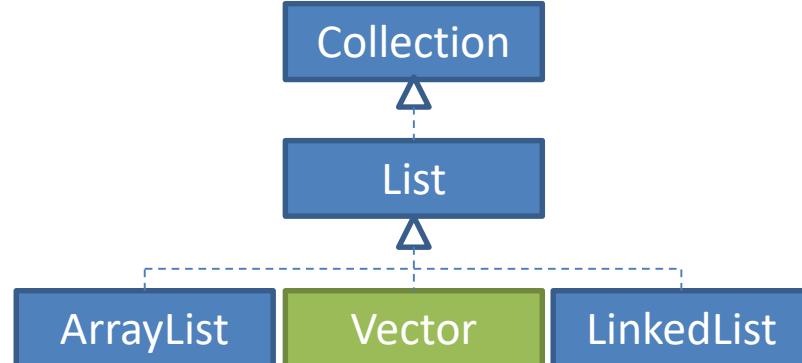
Manzana

Vector

```
import java.util.List;
import java.util.Vector;
```

```
//Vector
List<String> palabras2 = new Vector();
palabras2.add("Manzana2");
palabras2.add(0, "Castaña2");
palabras2.add(1, "Lechuga2");
palabras2.addAll(palabras);

for(String palabra : palabras2) {
    System.out.println(palabra);
}
```



<terminated> EjemploColecciones

Castaña2

Lechuga2

Manzana2

Castaña

Lechuga

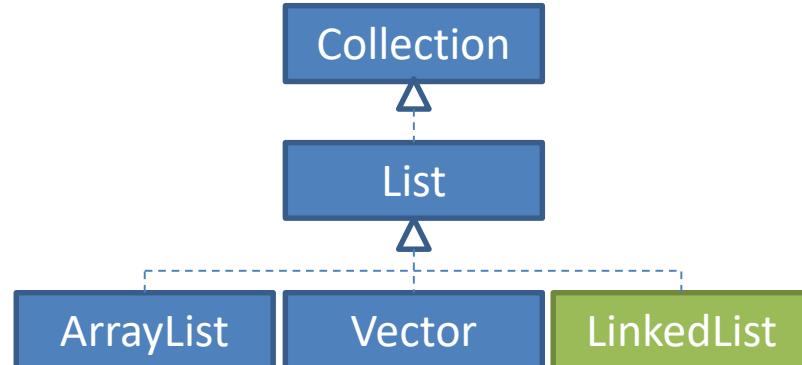
Manzana

LinkedList

```
import java.util.LinkedList;
import java.util.List;
```

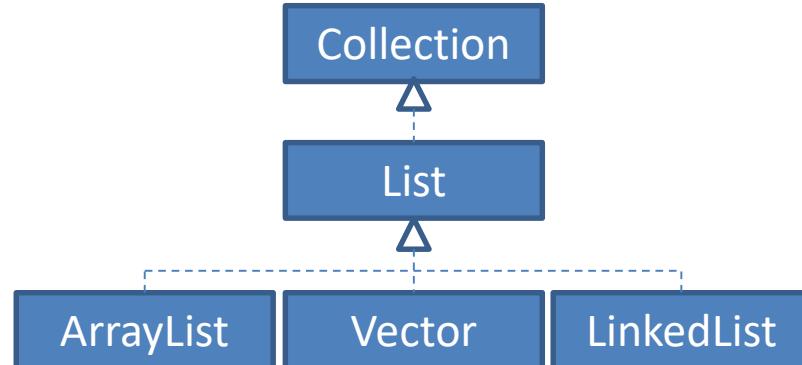
```
//LinkedList
List<String> palabras3 = new LinkedList();
palabras3.add("Manzana3");
palabras3.addAll(0, palabras);

for(String palabra : palabras3) {
    System.out.println(palabra);
}
```



<terminated> EjemploColecciones |

Castaña
Lechuga
Manzana
Manzana3



Las variables a utilizar deben ser siempre referencias a interfaces. Así, si en un momento dado queremos utilizar otra clase concreta, los cambios sólo se realizarían en el momento de creación de las variables.

Si en un momento dado queremos utilizar ArrayList en vez de LinkedList, el cambio sería menos impactante, ya que cambiamos el objeto que se instancia y todos los métodos que aceptan una lista aceptan también la otra (polimorfismo).

```
//ArrayList
List<String> palabras = new ArrayList();
//Vector
List<String> palabras2 = new Vector();
//LinkedList
List<String> palabras3 = new LinkedList();
```

04



MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO

EOI Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



Set (Interfaz)

Los set o conjuntos, son grupos de elementos en los que no encontramos ningún elemento repetido. Se considera que un elemento está repetido si tenemos dos objetos o1 y o2 iguales, comparándolos mediante el operador o1.equals(o2)

Si el objeto a insertar en el conjunto estuviese repetido, no nos dejará insertarlo. El método add devuelve un valor booleano, que servirá para este caso, devolviéndonos true si el elemento a añadir no estaba en el conjunto y ha sido añadido, o false si el elemento ya se encontraba dentro del conjunto.

Un conjunto podrá contener a lo sumo un elemento null.

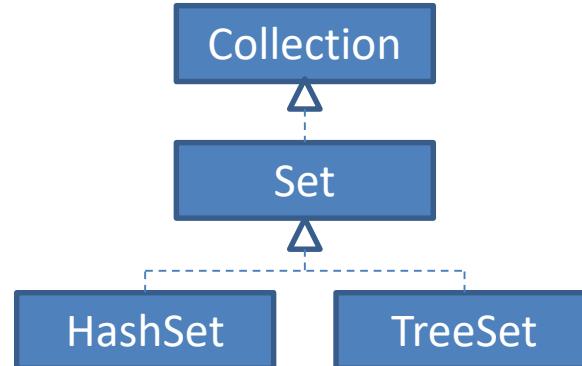
Set (Interfaz)

Modifier and Type	Method and Description
boolean	add(E e) Adds the specified element to this set if it is not already present (optional operation).
boolean	addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	clear() Removes all of the elements from this set (optional operation).
boolean	contains(Object o) Returns true if this set contains the specified element.
boolean	containsAll(Collection<?> c) Returns true if this set contains all of the elements of the specified collection.
boolean	equals(Object o) Compares the specified object with this set for equality.

Set (Interfaz)

boolean	isEmpty()	Returns <code>true</code> if this set contains no elements.
boolean	remove(Object o)	Removes the specified element from this set if it is present (optional operation).
boolean	removeAll(Collection<?> c)	Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	retainAll(Collection<?> c)	Retains only the elements in this set that are contained in the specified collection (optional operation).
int	size()	Returns the number of elements in this set (its cardinality).
default Spliterator<E>	spliterator()	Creates a <code>Spliterator</code> over the elements in this set.
Object[]	toArray()	Returns an array containing all of the elements in this set.
<T> T[]	toArray(T[] a)	Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

Set (Implementación)



HASHSET: Implementación concreta donde el tiempo para búsquedas es importante. Es la clase que implementa Set más utilizada. Los objetos se almacenan en una tabla de dispersión (hash) que almacena un conjunto de datos sin ningún orden entre ellos.

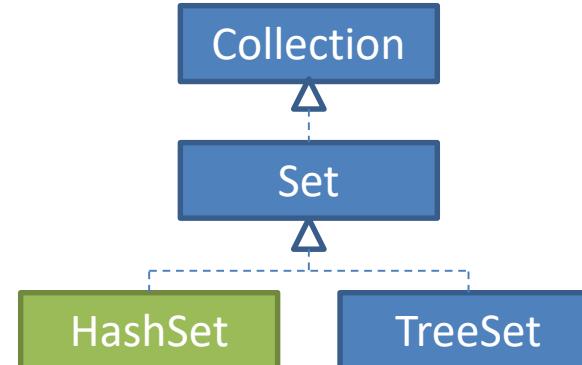
TREESET: Utiliza un árbol para el almacenamiento de los elementos. Mantiene un conjunto de datos ordenado (estructura de árbol).

HashSet

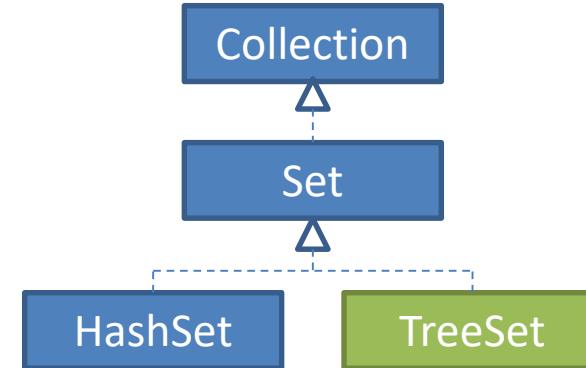
```
import java.util.Set;
import java.util.HashSet;
```

```
/*HashSet
Set<String> nombres = new HashSet();
nombres.add("Javier");
nombres.add("Laura");
nombres.add("Javier");

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



```
<terminated> EjemploColecciones [
Javier
Laura
```



TreeSet

```
import java.util.Set;
import java.util.TreeSet;
```

```
//Treeset
Set<String> nombres2 = new TreeSet();
nombres2.add("Javier");
nombres2.add("Laura");
nombres2.add("Javier");
nombres2.add("Anabel");

for(String nombre : nombres2) {
    System.out.println(nombre);
}
```

```
<terminated> EjemploColecciones |  
Anabel  
Javier  
Laura
```

05



MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO

EOI Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



Map (Interfaz)

Aunque muchas veces se hable de los mapas como una colección, en realidad no lo son, ya que no heredan de la interfaz Collection. Los mapas se definen en la interfaz Map. Un mapa es un objeto que relaciona una clave (key) con un valor (value). Contendrá un conjunto de claves, y a cada clave se le asociará un determinado valor.

Matrícula (key)	Vehículo (Value)
12215335A	Moto(...)
56262626V	Coche(...)
68268882R	Ciclomotor(...)

Map (Interfaz)

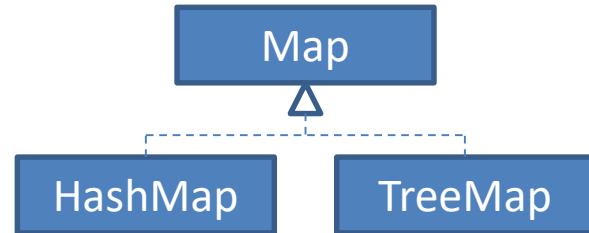
Modifier and Type	Method and Description
V	get(Object key) Returns the value to which the specified key is mapped, or <code>null</code> if this map contains no mapping for the key.
boolean	isEmpty() Returns <code>true</code> if this map contains no key-value mappings.
Set<K>	keySet() Returns a <code>Set</code> view of the keys contained in this map.
V	remove(Object key) Removes the mapping for a key from this map if it is present (optional operation).
default boolean	remove(Object key, Object value) Removes the entry for the specified key only if it is currently mapped to the specified value.
default V	replace(K key, V value) Replaces the entry for the specified key only if it is currently mapped to some value.
default boolean	replace(K key, V oldValue, V newValue) Replaces the entry for the specified key only if currently mapped to the specified value.

Map (Interfaz)

Modifier and Type	Method and Description
void	clear() Removes all of the mappings from this map (optional operation).
boolean	containsKey(Object key) Returns <code>true</code> if this map contains a mapping for the specified key.
boolean	containsValue(Object value) Returns <code>true</code> if this map maps one or more keys to the specified value.
<code>Set<Map.Entry<K,V>></code>	entrySet() Returns a <code>Set</code> view of the mappings contained in this map.
boolean	equals(Object o) Compares the specified object with this map for equality.

Map (Interfaz)

Modifier and Type	Method and Description
int	size() Returns the number of key-value mappings in this map.
Collection<V>	values() Returns a Collection view of the values contained in this map.
V	put(K key, V value) Associates the specified value with the specified key in this map (optional operation).



Map (Implementación)

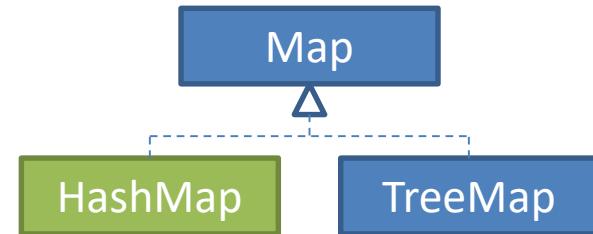
HASHMAP: Utiliza una tabla de dispersión (hash) para almacenar la información del mapa. No se garantiza que se respete el orden de las claves.

TREEMAP: Utiliza un árbol rojo-negro para implementar el mapa. En este caso los elementos se encontrarán ordenados por orden ascendente de clave.

La clase HashMap usa la tabla hash como estructura de datos.

La clase TreeMap usa el árbol rojo-negro como estructura de datos.

La principal diferencia entre HashMap y TreeMap es que el HashMap no conserva el orden de inserción mientras que, el Mapa de árbol hace.



HashMap

```

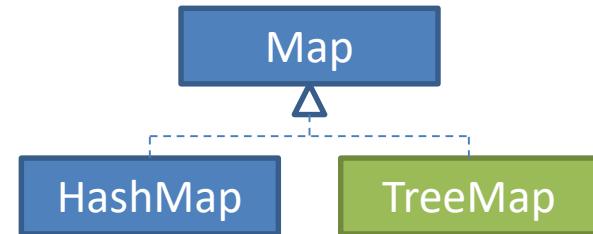
import java.util.Map;
import java.util.HashMap;

//HashMap
Map<Integer, String> diasSemana = new HashMap();
diasSemana.put(6, "Sábado");
diasSemana.put(7, "Domingo");
diasSemana.put(1, "Lunes");
diasSemana.put(2, "Martes");
diasSemana.put(3, "Miércoles");
diasSemana.put(4, "Jueves");
diasSemana.put(5, "Viernes");

for(Integer codigo : diasSemana.keySet()) {
    System.out.println("La clave es:"+codigo+
    " y el valor es:"+diasSemana.get(codigo));
}
  
```

```

Markers Problems Console Search
<terminated> EjemploColecciones [Java Application] C:\ej_icc
La clave es:1 y el valor es:Lunes
La clave es:2 y el valor es:Martes
La clave es:3 y el valor es:Miércoles
La clave es:4 y el valor es:Jueves
La clave es:5 y el valor es:Viernes
La clave es:6 y el valor es:Sábado
La clave es:7 y el valor es:Domingo
  
```



TreeMap

```
import java.util.Map;
import java.util.TreeMap;
```

```
//TreeMap
Map<Integer, String> diasSemana2 = new TreeMap();
diasSemana2.put(6, "Sábado");
diasSemana2.put(7, "Domingo");
diasSemana2.put(1, "Lunes");
diasSemana2.put(2, "Martes");
diasSemana2.put(3, "Miércoles");
diasSemana2.put(4, "Jueves");
diasSemana2.put(5, "Viernes");

for(Integer codigo : diasSemana2.keySet()) {
    System.out.println("La clave es:"+codigo+
        " y el valor es:"+diasSemana2.get(codigo));
}
```

Markers Problems Console Search

<terminated> EjemploColecciones [Java Application] C:\ej_icc

```
La clave es:1 y el valor es:Lunes
La clave es:2 y el valor es:Martes
La clave es:3 y el valor es:Miércoles
La clave es:4 y el valor es:Jueves
La clave es:5 y el valor es:Viernes
La clave es:6 y el valor es:Sábado
La clave es:7 y el valor es:Domingo
```

06

Iterator



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



Iterator

Es un objeto cuya función es moverse a través de una secuencia de objetos, y obtener cada objeto en la secuencia sin que el programador conozca la estructura de almacenamiento utilizada.

Operativa con *Iterators*:

Pedir al contenedor (*objeto collection*) un *iterator*:

Iterator it = coll.iterator();

Obtener el primero objeto invocando *next()*:

it.next();

Comprobar si existen más objetos en la secuencia:

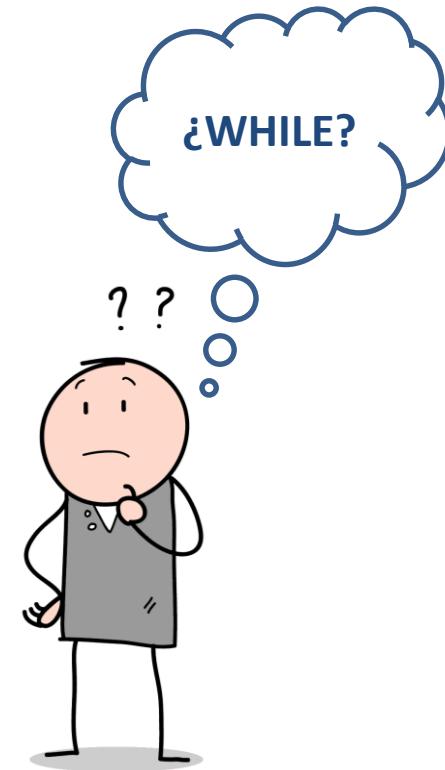
If (it.hasNext())

Opcionalmente eliminar el objeto a partir del *iterator*:

It.remove();

Iterator

```
import java.util.Iterator;  
  
//Iterator ArrayList  
Iterator iteradorPalabras = palabras.iterator();  
while(iteradorPalabras.hasNext()) {  
    System.out.println(iteradorPalabras.next());  
}  
  
//Iterator HashSet  
Iterator iteratorNombres = nombres.iterator();  
while(iteratorNombres.hasNext()) {  
    System.out.println(iteratorNombres.next());  
}
```



07

Comparable



GOBIERNO
DE ESPAÑA

MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO

EOI Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro



Comparable

Comparable es un interfaz de java, que esta incluida dentro de Java. La interfaz comparable contiene un método abstracto **compareTo** el cual permite ordenar un objeto según un atributo especificado en un orden ascendente o descendente.

El método **CompareTo** es un método abstracto que retorna un int (entero) para todos los casos:

1 → si el objeto A es mayor que el objeto B

0 → si A y B son iguales.

-1 → si el objeto A es menor que el objeto B

Comparable

Una clase puede implementar esta interfaz para proporcionar un ordenamiento a sus instancias. Este ordenamiento se llama ordenamiento natural. La clase que implemente esta interfaz debe implementar el método `compareTo`.

```
public class Libro implements Comparable<Libro>{  
     The type Libro must implement the inherited abstract method Comparable<Libro>.compareTo(Libro).  
    private S  
    private i  
    private d  
    private b  
    private S  
         2 quick fixes available:  
        ⚡ Add unimplemented methods  
        ⚡ Make type 'Libro' abstract
```

Objeto 1



Objeto 2

Comparable

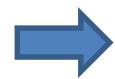
```
@Override  
public int compareTo(Libro otroLibro) {  
    return this.getTitulo().compareTo(otroLibro.getTitulo());  
}
```

```
List<Libro> libros = new ArrayList();  
libros.add(new Libro("Los Juegos del Hambre", 11, 300));  
libros.add(new Libro("El Hobbit", 12, 432));  
libros.add(new Libro("El Juego de Ender", 13, 210));  
libros.add(new Libro("Crónicas Marcianas", 20, 982));
```

```
System.out.println("Antes de ordenar:");  
for(Libro libro : libros) {  
    System.out.println(libro);  
}  
  
Collections.sort(libros);  
  
System.out.println("Después de ordenar:");  
for(Libro libro : libros) {  
    System.out.println(libro);  
}
```

```
Antes de ordenar:  
Titulo:Los Juegos del Hambre, Autor:Desconocido, Nº Paginas:300, Precio:11.0  
Titulo:El Hobbit, Autor:Desconocido, Nº Paginas:432, Precio:12.0  
Titulo:El Juego de Ender, Autor:Desconocido, Nº Paginas:210, Precio:13.0  
Titulo:Crónicas Marcianas, Autor:Desconocido, Nº Paginas:982, Precio:20.0  
Después de ordenar:  
Titulo:Los Juegos del Hambre, Autor:Desconocido, Nº Paginas:300, Precio:11.0  
Titulo:El Juego de Ender, Autor:Desconocido, Nº Paginas:210, Precio:13.0  
Titulo:El Hobbit, Autor:Desconocido, Nº Paginas:432, Precio:12.0  
Titulo:Crónicas Marcianas, Autor:Desconocido, Nº Paginas:982, Precio:20.0
```

Objeto 2



Objeto 1

Comparable

```
@Override
public int compareTo(Libro otroLibro) {
    return otroLibro.getTitulo().compareTo(this.getTitulo());
}
```

```
List<Libro> libros = new ArrayList();
libros.add(new Libro("Los Juegos del Hambre", 11, 300));
libros.add(new Libro("El Hobbit", 12, 432));
libros.add(new Libro("El Juego de Ender", 13, 210));
libros.add(new Libro("Crónicas Marcianas", 20, 982));
```

```
System.out.println("Antes de ordenar:");
for(Libro libro : libros) {
    System.out.println(libro);
}

Collections.sort(libros);

System.out.println("Después de ordenar:");
for(Libro libro : libros) {
    System.out.println(libro);
}
```

Antes de ordenar:

Titulo:Los Juegos del Hambre, Autor:Desconocido, Nº Paginas:300, Precio:11.0
 Titulo:El Hobbit, Autor:Desconocido, Nº Paginas:432, Precio:12.0
 Titulo:El Juego de Ender, Autor:Desconocido, Nº Paginas:210, Precio:13.0
 Titulo:Crónicas Marcianas, Autor:Desconocido, Nº Paginas:982, Precio:20.0

Después de ordenar:

Titulo:Crónicas Marcianas, Autor:Desconocido, Nº Paginas:982, Precio:20.0
 Titulo:El Hobbit, Autor:Desconocido, Nº Paginas:432, Precio:12.0
 Titulo:El Juego de Ender, Autor:Desconocido, Nº Paginas:210, Precio:13.0
 Titulo:Los Juegos del Hambre, Autor:Desconocido, Nº Paginas:300, Precio:11.0

Comparable

```
@Override  
public int compareTo(Libro otroLibro) {  
    if(this.numeroPaginas == otroLibro.getNumeroPaginas()) {  
        return 0;  
    } else if (this.numeroPaginas > otroLibro.getNumeroPaginas()) {  
        return 1;  
    } else {  
        return -1;  
    }  
}
```

```
List<Libro> libros = new ArrayList();  
libros.add(new Libro("Los Juegos del Hambre", 11, 300));  
libros.add(new Libro("El Hobbit", 12, 432));  
libros.add(new Libro("El Juego de Ender", 13, 210));  
libros.add(new Libro("Crónicas Marcianas", 20, 982));
```

```
System.out.println("Antes de ordenar:");  
for(Libro libro : libros) {  
    System.out.println(libro);  
}  
  
Collections.sort(libros);  
  
System.out.println("Después de ordenar:");  
for(Libro libro : libros) {  
    System.out.println(libro);  
}
```

Antes de ordenar:

Titulo:Los Juegos del Hambre, Autor:Desconocido, Nº Paginas:300, Precio:11.0
Titulo:El Hobbit, Autor:Desconocido, Nº Paginas:432, Precio:12.0
Titulo:El Juego de Ender, Autor:Desconocido, Nº Paginas:210, Precio:13.0
Titulo:Crónicas Marcianas, Autor:Desconocido, Nº Paginas:982, Precio:20.0

Después de ordenar:

Titulo:El Juego de Ender, Autor:Desconocido, Nº Paginas:210, Precio:13.0
Titulo:Los Juegos del Hambre, Autor:Desconocido, Nº Paginas:300, Precio:11.0
Titulo:El Hobbit, Autor:Desconocido, Nº Paginas:432, Precio:12.0
Titulo:Crónicas Marcianas, Autor:Desconocido, Nº Paginas:982, Precio:20.0

08

Ejercicios



MINISTERIO
DE INDUSTRIA, COMERCIO
Y TURISMO

EOI Escuela de
organización
industrial



Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro

SISTEMA NACIONAL DE
GARANTÍA JUVENIL

Ejercicio 1

Vamos a modificar nuestra clase Persona para que podamos ordenar una lista de personas, a la hora de ordenar lo haremos por el nombre de la persona (alfabéticamente) en caso de tener el mismo nombre se ordenará por la edad de mayor a menor y en caso de coincidir también en edad ordenaremos por año de nacimiento de más antiguo a más reciente. La lista de personas es la siguiente:

ANA	26	2010	RESTO DE CAMPOS
ALBA	21	1988	RESTO DE CAMPOS
ANA	26	2009	RESTO DE CAMPOS
ALBA	26	2007	RESTO DE CAMPOS

Ejercicio 2

Vamos a modificar la clase Persona.java y vamos a añadirle el atributo DNI,

Vamos a crear un HashMap de personas donde el DNI será la clave y el objeto persona será el valor.

Crearemos el siguiente menú:

- 1 – Nueva Persona**
- 2 – Listado Personas (Orden Alfabético Nombre y Apellidos)**
- 3 – Buscar por NIF**
- 4 - SALIR**