

Operaciones sobre colecciones

DigitalHouse >
Coding School



**Certified Tech
Developer**
The Ultimate Degree

Índice

1. [Crear una colección](#)
2. [Agregar elementos](#)
3. [Eliminar elementos](#)
4. [Obtener o buscar elementos](#)

“

A continuación, estudiaremos las operaciones más importantes que podemos hacer sobre las colecciones.

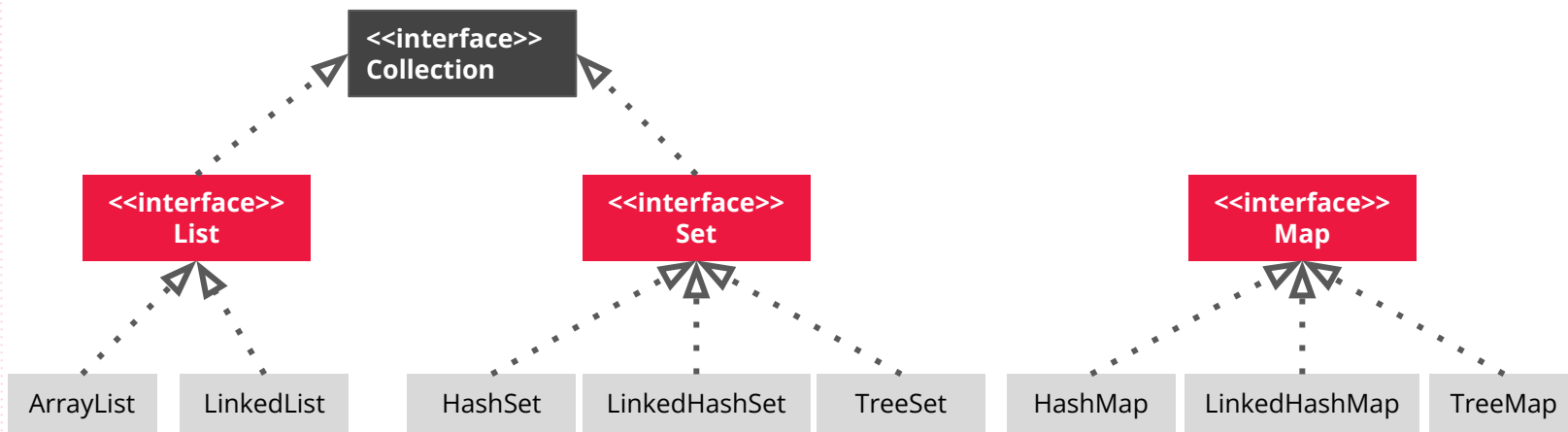


”

1 | Crear una colección

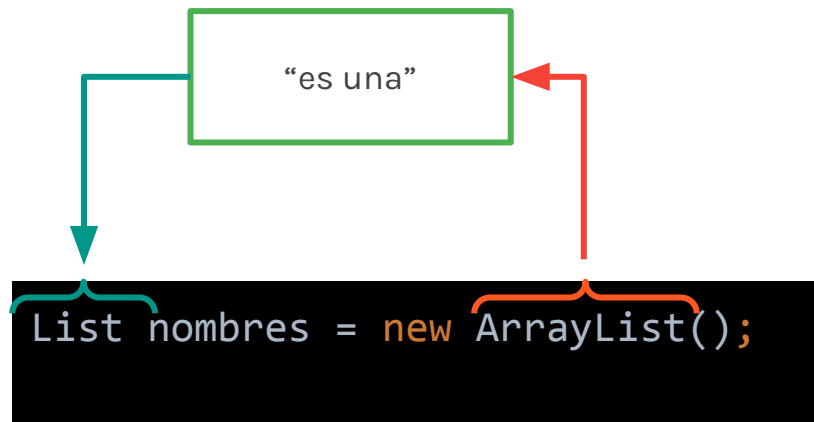
Crear una colección

Las colecciones en Java están implementadas a través de esta familia de clases e interfaces. Conocerla nos permitirá **crear las colecciones de la manera más genérica** posible.



Crear una colección

Al momento de crear una colección o cualquier tipo de objeto, es una buena práctica que el tipo de la **referencia sea lo más genérico posible**.



Crear una colección

Dado que **ArrayList**, y **LinkedList** implementan la interface **List**, trataremos a estas colecciones siempre como una **List**, ya que las operaciones que necesitamos hacer sobre estas colecciones se encuentran establecidas en esta interface.

```
List nombres = new ArrayList();
```

```
List nombres = new LinkedList();
```

Crear una colección

Por el contrario, **HashSet**, **LinkedHashSet** y **TreeSet** implementan la interface **Set**, por ende, trataremos a estas colecciones siempre como una Set.

```
Set nombres = new HashSet();
```

```
Set nombres = new LinkedHashSet();
```

```
Set nombres = new TreeSet();
```


Crear una colección

HashMap, **LinkedHashMap** y **TreeMap** implementan la interface **Map**, por ende, trataremos a estas colecciones siempre como una Map.

```
Map nombres = new HashMap();
```

```
Map nombres = new LinkedHashMap();
```

```
Map nombres = new TreeMap();
```

2 | Agregar elementos

Agregar elementos

Tanto la interface **List** como **Set** nos proporcionan el método **add** que recibe como parámetro un Object y, como toda clase hereda de Object, podemos almacenar cualquier tipo de objeto en ellas. Comencemos con ArrayList.

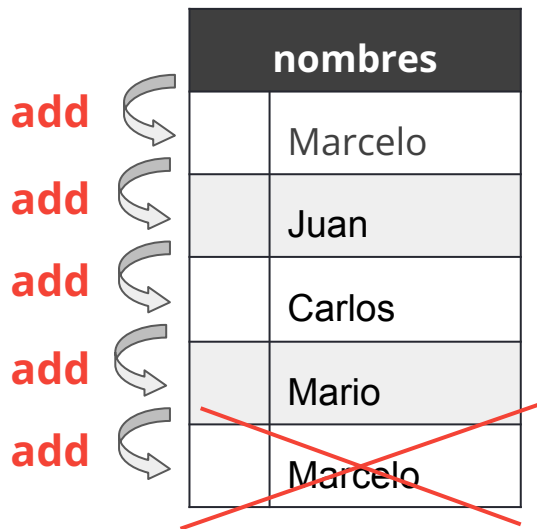


nombres	
0	Juan
1	Mario
2	Carlos
3	Marcelo
4	Marcelo

```
List nombres = new ArrayList();  
nombres.add("Juan");  
nombres.add("Mario");  
nombres.add("Carlos");  
nombres.add("Marcelo");  
nombres.add("Marcelo");
```

Agregar elementos

En el caso de las **Set**, si bien tienen el mismo método **add**, se comportan muy diferente. **No almacenan los valores repetidos ni nulos** y, en el caso de las **HashSet** no respeta el orden de inserción.

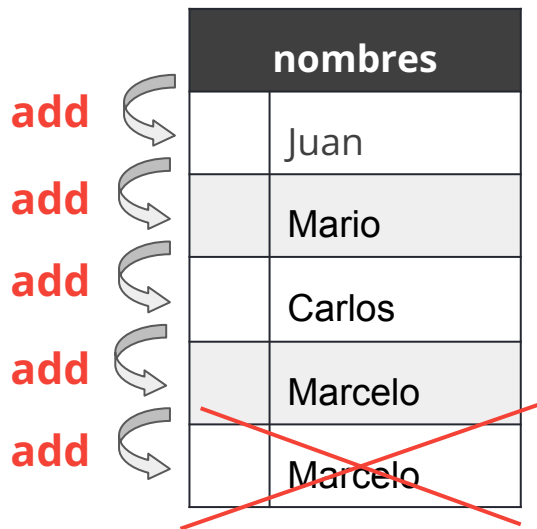


nombres	
add	Marcelo
add	Juan
add	Carlos
add	Mario
add	Marcelo

```
Set nombres = new HashSet();  
nombres.add("Juan");  
nombres.add("Mario");  
nombres.add("Carlos");  
nombres.add("Marcelo");  
nombres.add("Marcelo");
```

Agregar elementos

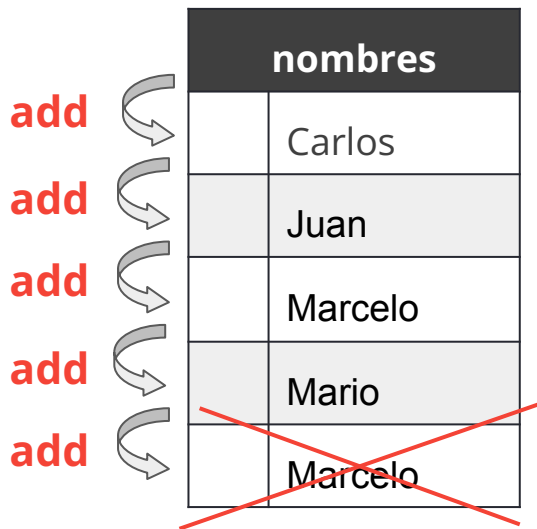
Las **LinkedHashSet**, como toda Set, no almacenan valores repetidos ni nulos, pero, a diferencia de la HashSet, sí **respetan el orden de inserción**.



```
Set nombres = new  
LinkedHashSet();  
nombres.add("Juan");  
nombres.add("Mario");  
nombres.add("Carlos");  
nombres.add("Marcelo");  
nombres.add("Marcelo");
```

Agregar elementos

Las **TreeSet** como toda Set no almacenan valores repetidos ni nulos y **los inserta ordenadamente**. En el siguiente ejemplo, al ser elementos String los inserta alfabéticamente.



nombres	
add	Carlos
add	Juan
add	Marcelo
add	Mario
add	Marcelo

```
Set nombres = new TreeSet();  
nombres.add("Juan");  
nombres.add("Mario");  
nombres.add("Carlos");  
nombres.add("Marcelo");  
nombres.add("Marcelo");
```

Agregar elementos

Las **Map** no poseen un método `add`, en su lugar, poseen un método llamado **`put`** que recibe dos parámetros: **una key y un valor**. Permiten valores duplicados, pero no keys duplicadas. Las **HashMap**, además, **no respetan el orden de inserción**.

put



put



put



put



put



nombres	
30888999	Mario
40888999	Marcelo
27888999	Carlos
29888999	Juan
50888999	Marcelo

```
Map nombres = new HashMap();  
nombres.put(29888999, "Juan");  
nombres.put(30888999, "Mario");  
nombres.put(27888999, "Carlos");  
nombres.put(40888999, "Marcelo");  
nombres.put(50888999, "Marcelo");
```

Agregar elementos

Las **LinkedHashMap** tienen el mismo comportamiento que una Map, pero a diferencia de las HashMap **respetan el orden de inserción**.

	nombres	
put	29888999	Juan
put	30888999	Mario
put	27888999	Carlos
put	40888999	Marcelo
put	50888999	Marcelo

```
Map nombres = new LinkedHashMap();  
nombres.put(29888999, "Juan");  
nombres.put(30888999, "Mario");  
nombres.put(27888999, "Carlos");  
nombres.put(40888999, "Marcelo");  
nombres.put(50888999, "Marcelo");
```


Agregar elementos

Las **TreeMap** tienen el mismo comportamiento que una Map, pero a diferencia del resto **los inserta ordenadamente según la key**. En este caso, la Key es un entero, por lo tanto, los ordena de menor a mayor.

	nombres	
put	27888999	Carlos
put	29888999	Juan
put	30888999	Mario
put	40888999	Marcelo
put	50888999	Marcelo

```
Map nombres = new TreeMap();  
nombres.put(29888999,"Juan");  
nombres.put(30888999,"Mario");  
nombres.put(27888999,"Carlos");  
nombres.put(40888999,"Marcelo");  
nombres.put(50888999,"Marcelo");
```

3 | Eliminar elementos

Eliminar elementos

Todas las colecciones poseen un método **remove**. En el caso de las **List**, como **ArrayList** y **LinkedList**, se pueden eliminar por índice o por valor.

remove

nombres	
0	Juan
1	Mario
2	Carlos
3	Marcelo
4	Marcelo

```
nombres.remove("Carlos");
```

```
nombres.remove(2);
```

Eliminar elementos

En el caso de todas las implementaciones de **Set** solo se pueden eliminar elementos pasando como parámetro al método `remove` el valor almacenado.

remove

nombres	
	Marcelo
	Juan
	Carlos
	Mario

```
nombres.remove("Carlos");
```

Eliminar elementos

En el caso de las **Map**, los elementos se eliminan por Key. Es decir, remove recibe como parámetro la Key del elemento que queremos eliminar.

remove

nombres	
27888999	Carlos
29888999	Juan
30888999	Mario
40888999	Marcelo
50888999	Marcelo

```
nombres.remove(27888999);
```

4 | Obtener o buscar elementos

Obtener o buscar elementos

En el caso de las **List**, como **ArrayList** y **LinkedList**, si queremos obtener un valor y conocemos el índice, podemos utilizar el método `get` que recibe como parámetro el índice de la posición.

nombres	
0	Juan
1	Mario
2	Carlos
3	Diego
4	Marcelo

`get(2)`

```
System.out.println(nombres.get(2));
```

Obtener o buscar elementos

En el caso de las **Set**, para obtener un elemento debemos buscarlo recorriendo la colección, ya que las Set no tienen índice.

nombres	
	Juan
	Mario
	Carlos
	Diego
	Marcelo

```
boolean encontrado = false;
String nombre = null;
Iterator it = nombres.iterator();
while(it.hasNext() && !encontrado) {
    nombre = (String) it.next();
    if(nombre == "Carlos")
        encontrado = true;
}
System.out.println("Encontramos a " + nombre);
```


Obtener o Buscar elementos

En el caso de las **Map**, para obtener un elemento, podemos hacerlo a través de su **Key** con el método **get**.

get

nombres	
27888999	Carlos
29888999	Juan
30888999	Mario
40888999	Marcelo
50888999	Marcelo

```
nombres.get(30888999);
```

DigitalHouse>
Coding School