

Operações sobre coleções

DigitalHouse>



**Certified Tech
Developer**

The Ultimate Degree

Índice

1. Criar uma coleção
2. Incluir elementos
3. Eliminar elementos
4. Buscar elementos

“

Nesta apresentação, teremos as operações mais **importantes** quando falamos em **coleções**.

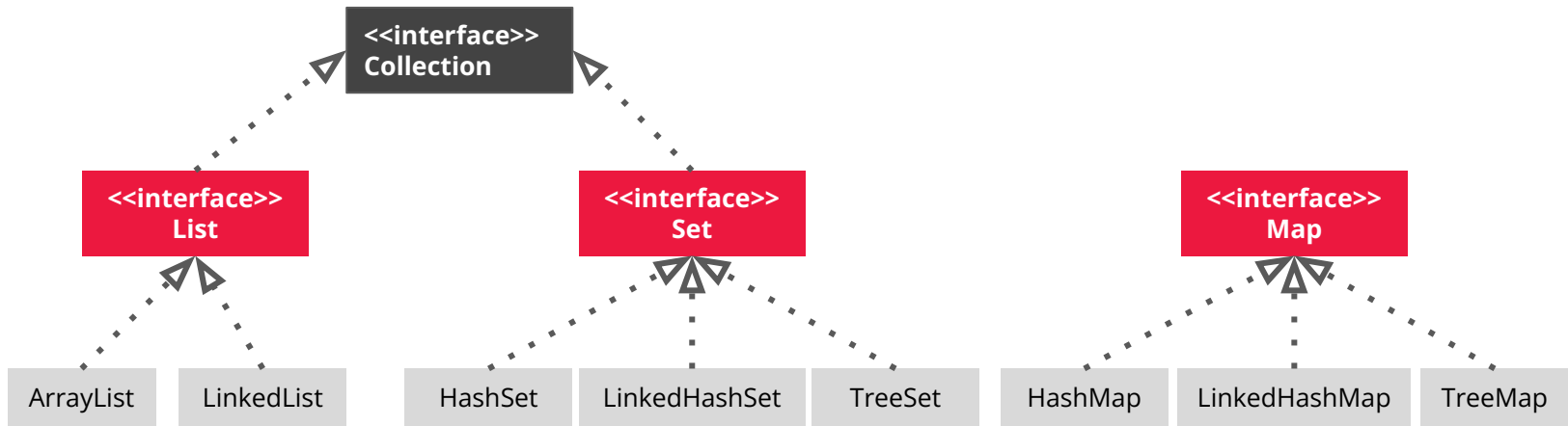
”



1 | Criar uma coleção

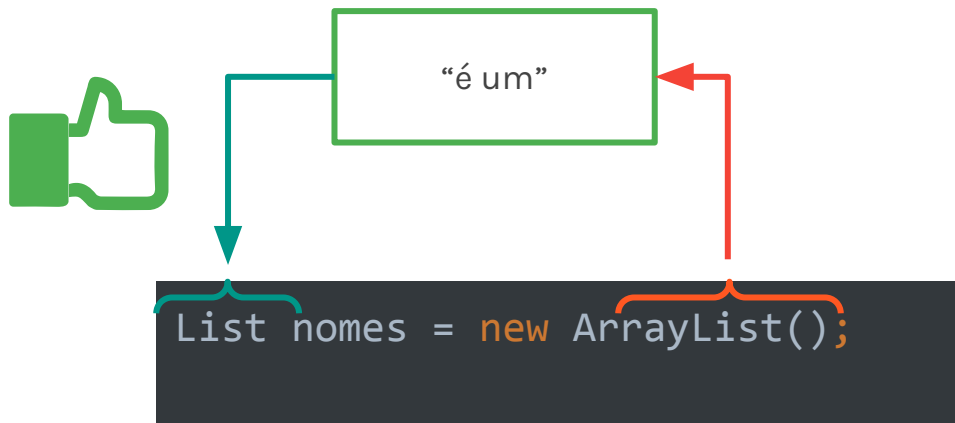
Criar uma coleção

As coleções em Java são implementadas por meio dessa família de **classes e interfaces**. Saber disso permitirá criar as coleções de forma genérica.



Criar uma coleção

Ao criar uma coleção ou qualquer tipo de objeto, é uma **boa prática** digitar a referência do modo mais genérico possível.



Criar uma coleção

Dado que **ArrayList** e **LinkedList** implementam a interface **List**, sempre trataremos essas coleções como uma lista, uma vez que as operações que precisamos fazer nessas coleções são estabelecidas nesta interface.

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

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

Criar uma coleção

Ao contrário do ArrayList e LinkedList, o **HashSet**, **LinkedHashSet** e **TreeSet** implementam a interface **Set**, portanto, sempre trataremos essas coleções como um Set.

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

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

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


Criar uma coleção

HashMap, **LinkedHashMap** e **TreeMap** implementam a interface do **Map**, portanto, sempre trataremos essas coleções como um Map.

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

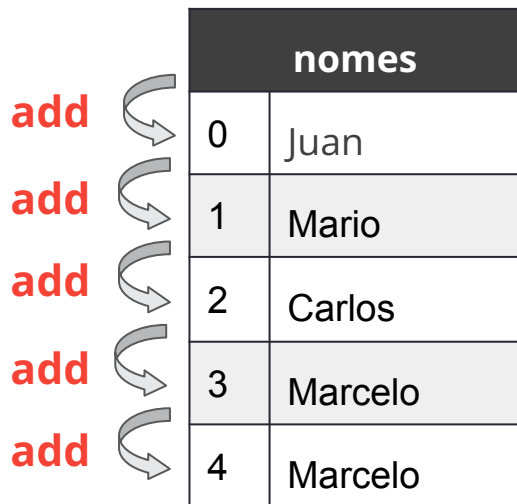
```
Map nomes = new LinkedHashMap();
```

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

2 | Incluindo elementos

Incluindo elementos

Ambas as interfaces **List** e **Set** nos fornecem o método **add** que recebe um **Object** como parâmetro e, como toda classe herda de **Object**, podemos armazenar qualquer tipo de objeto nelas. Vamos começar com **ArrayList**.

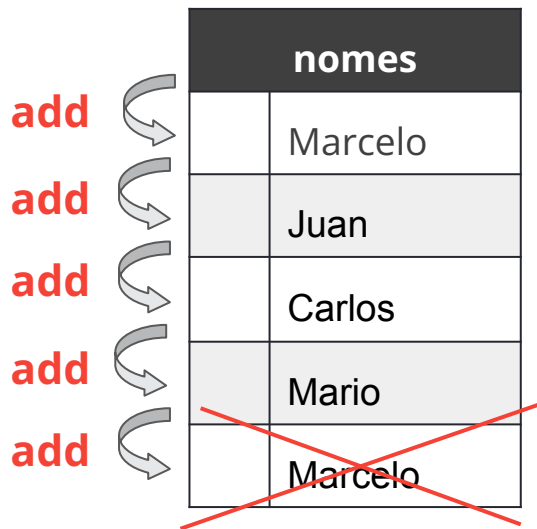


nomes	
0	Juan
1	Mario
2	Carlos
3	Marcelo
4	Marcelo

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

Incluindo elementos

No caso de **Set**, embora tenham o mesmo método de **add**, eles se comportam de maneira muito diferente. Eles não armazenam valores **repetidos ou nulos** e, no caso de **HashSet**, não respeitam a ordem de inserção.

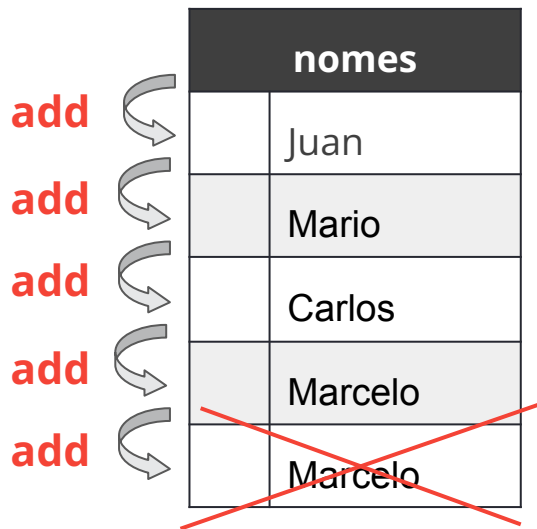


nomes	
add	Marcelo
add	Juan
add	Carlos
add	Mario
add	Marcelo

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

Incluindo elementos

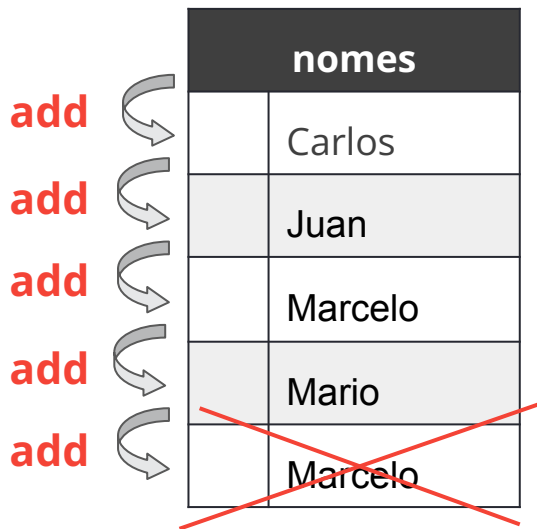
O **LinkedHashSet**, como todos os Sets, **não** armazena valores repetidos ou nulos, mas, ao contrário do **HashSet**, eles respeitam a **ordem de inserção**.



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

Incluindo elementos

TreeSet, como todos os **Sets**, **não** armazena valores repetidos ou nulos e os insere de maneira ordenada. No exemplo a seguir, sendo elementos **String**, ele os insere em ordem alfabética.



nomes	
add	Carlos
add	Juan
add	Marcelo
add	Mario
add	Marcelo

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

Incluindo elementos

Os **Maps** não têm um método **add**, em vez disso, eles têm um método chamado **put** que recebe dois parâmetros: **uma chave e um valor**. Eles permitem valores duplicados, mas não chaves duplicadas. **HashMap** também não respeita a ordem de inserção.

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

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

Incluindo elementos

LinkedHashMap tem o mesmo comportamento que **Map**, mas ao contrário de **HashMap**, eles respeitam a ordem de inserção.

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

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


Incluindo elementos

O **TreeMap** tem o mesmo comportamento de um **Map**, mas ao contrário dos demais ele os insere em ordem de acordo com a **chave**. Nesse caso, a chave é um número inteiro, portanto, ordene-os **do menor para o maior**.

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

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

3 | Eliminar elementos

Eliminar elementos

Todas as coleções têm um método de **remoção**. Para listas, como **ArrayList** e **LinkedList**, eles podem ser removidos por índice ou por valor.

remove

nomes	
0	Juan
1	Mario
2	Carlos
3	Marcelo
4	Marcelo

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

```
nomes.remove(2);
```

Eliminar elementos

No caso de todas as implementações de **Set**, os elementos só podem ser removidos passando o **valor armazenado** como um parâmetro para o método `remove`.

remove

nomes	
	Marcelo
	Juan
	Carlos
	Mario

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

Eliminar elementos

No caso de **Maps**, os elementos são removidos por **Key**. Ou seja, o método **remove** recebe como parâmetro a **chave do elemento** que queremos remover.

remove

nomes	
27888999	Carlos
29888999	Juan
30888999	Mario
40888999	Marcelo
50888999	Marcelo

```
nomes.remove(27888999);
```

4 | Buscar elementos

Buscar elementos

No caso de **List**, como **ArrayList** e **LinkedList**, se quisermos obter um valor e conhecermos o índice, podemos usar o método **get** que recebe o índice da posição como parâmetro.

nomes	
0	Juan
1	Mario
2	Carlos
3	Diego
4	Marcelo

get(2)

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

Buscar elementos

No caso de **Sets**, para obter um elemento devemos procurá-lo **percorrendo** a coleção, uma vez que **Sets** não possuem índice.

nomes	
	Juan
	Mario
	Carlos
	Diego
	Marcelo

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


Buscar elementos

No caso do **Maps**, para obter um elemento, podemos fazê-lo através de sua **Key** com o método **get**.

get

nomes	
27888999	Carlos
29888999	Juan
30888999	Mario
40888999	Marcelo
50888999	Marcelo

```
nomes.get(30888999);
```

DigitalHouse>
Coding School