

METHOD REFERENCES


```
List<String> lista = Arrays.asList("ab", "b", "ccc");
```

```
lista.sort((o1, o2) -> o1.length() - o2.length());
```

```
class Utilidades {  
    public static int compare(String o1, String o2) {  
        return o1.length() - o2.length();  
    }  
}
```



```
List<String> lista = Arrays.asList("ab", "b", "ccc");
```


```
lista.sort((o1,o2) -> Utilidades.compare(o1, o2));
```

```
class Utilidades {  
    public static int compare(String o1, String o2) {  
        return o1.length() - o2.length();  
    }  
}
```



```
List<String> lista = Arrays.asList("ab", "b", "ccc");
```

```
lista.sort(Utilidades::compare);
```



```
class Utilidades {  
    public static int compare(String o1, String o2) {  
        return o1.length() - o2.length();  
    }  
}
```

method reference

method reference

Utilidades::compare

Clase

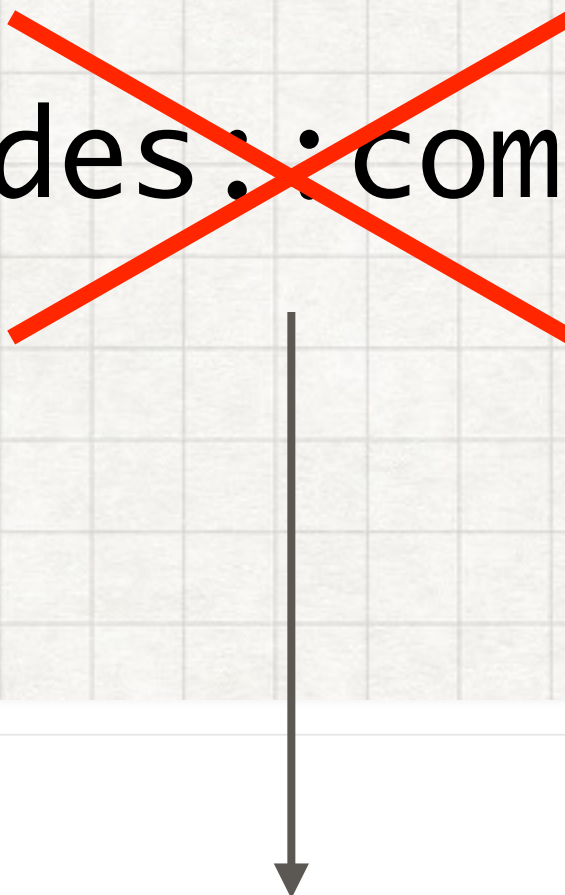
método estático



¿Que sucede si el método de Utilidades no es estático?

```
List<String> lista = Arrays.asList("ab", "b", "ccc");
```

```
lista.sort(Utilidades::compare);
```



```
class Utilidades {  
    public static int compare(String o1, String o2) {  
        return o1.length() - o2.length();  
    }  
}
```

La sintaxis anterior es para una referencia a un método estático

Referencia a un método de una instancia particular

```
Utilidades util = new Utilidades();  
lista.sort(util::compare);
```

```
class Utilidades {  
    public int compare(String o1, String o2) {  
        return o1.length() - o2.length();  
    }  
}
```


Referencia a un método de una instancia arbitraria

```
lista.sort(String::compareToIgnoreCase);
```

```
int compare(String o1, String o2);
```

```
public final class String ... {
```

```
...
```

```
public int compareToIgnoreCase(String str)
```


Referencia a un método de una instancia arbitraria

```
int compare(String o1, String o2);
```

(o1, o2) String::compareToIgnoreCase

Diagram illustrating the mapping of parameters from the function signature to the method call:

- The parameter `o1` from the signature `(o1, o2)` is mapped to the receiver `o1` in the call `o1.compareToIgnoreCase(o2);`.
- The parameter `o2` from the signature `(o1, o2)` is mapped to the argument `o2` in the call `o1.compareToIgnoreCase(o2);`.
- The method name `compareToIgnoreCase` in the call corresponds to the method name in the signature.

```
o1.compareToIgnoreCase(o2);
```


Referencia a un constructor

```
Function<String, Integer> converter = Integer::new;
```

```
converter.apply("3")
```

```
new Integer("3")
```



The diagram consists of two arrows. The first arrow starts at the 'new' keyword in the first line and points to the 'new' keyword in the second line. The second arrow starts at the '3' in the first line and points to the '3' in the second line.