# Introducción a Set



**¡Comencemos!**

## ¿Qué aprenderás?

- Aplicar el uso de Set para resolver problemas cotidianos dentro del mundo de la programación.
- Crear distintas implementaciones de tipo Set para utilizar métodos como agregar, eliminar, interceptar y recorrer datos.

## Introducción

Set es una interfaz que extiende desde Collections. Es uno de los tipo de colecciones más desordenado en relación a los objetos, en donde a diferencia de List no se pueden almacenar valores duplicados. Set también agrega un contrato más fuerte sobre el comportamiento de las operaciones equals y hashCode, permitiendo que las instancias de Set se comparen significativamente incluso si sus tipos de implementación difieren. Dos instancias de Set son iguales si contienen los mismos elementos.

La plataforma Java contiene tres implementaciones de Set de propósito general: HashSet, TreeSet y LinkedHashSet. Además, Set tiene varios métodos de acceso posicional como add, remove, clear y size, entre muchos otros, que permiten mejorar el uso de esta interfaz.

Más adelante veremos cómo se usa cada una de estas operaciones, pero primero partiremos creando un HashSet para entender cómo funcionan y desde donde se implementan.

**¡Vamos con todo!**

## Creando un Set

Un Set es una interfaz presente en Java que extiende de Collections, se comporta como una colección de objetos desordenada en donde los valores duplicados no pueden ser almacenados. Además, dentro de las posibles implementaciones presentadas por Set, encontramos a HashSet. Esta implementación almacena elementos en una tabla llamada hash, cuyos rendimientos son mejores que el resto de las implementaciones, sin embargo, no garantiza el orden de iteración.

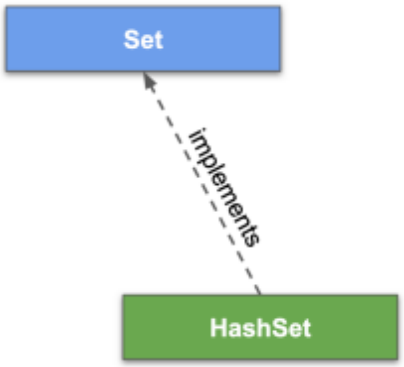


Imagen 1. Implementación de HashSet en Set.
Fuente: Desafio Latam

Ahora, vayamos a un ejemplo de cómo crear un HashSet en Java. Para esto, partiremos definiendo una Set del tipo String, cuyo nombre será capitales e instanciamos como una HashSet. Luego le agregaremos algunos lenguajes de programación para mostrar que el elemento “Java” no se repite.

```
import java.util.Set;
import java.util.HashSet;
       Set<String> languages = new HashSet<>();
       languages.add("Java");
       languages.add("Go");
       languages.add("Erlang");
       languages.add("Java");
       languages.add("Elixir");
       languages.add("Fortran");
       System.out.println(languages); //[Java, Go, Erlang, Elixir,
Fortran]
```

Se ha ingresado "Java" dos veces, pero no se muestra en la salida. Además, se pueden ordenar directamente las entradas pasando el conjunto desordenado como parámetro de TreeSet. A diferencia de un HashSet los elementos del TreeSet van ordenados.

## Creando un LinkedHashSet

LinkedHashSet se implementa como una tabla hash con una lista vinculada que lo ejecuta, ordena sus elementos según el orden en que se insertaron en el conjunto (orden de inserción).

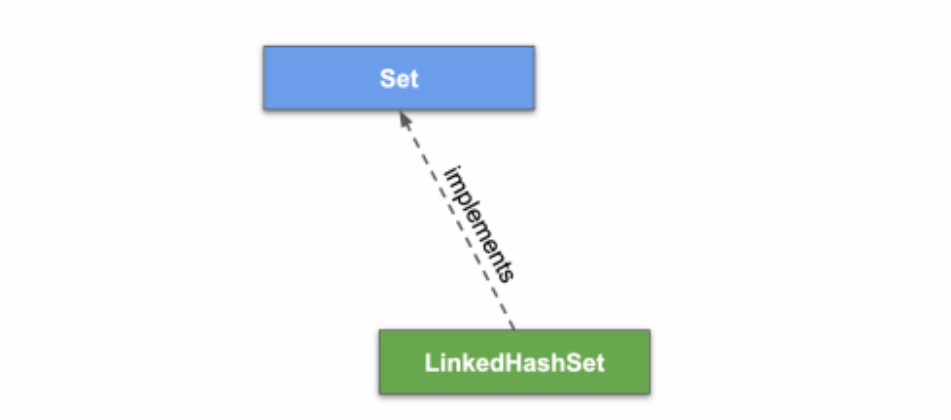


Imagen 2. Implementación de LinkedHashSet en Set.
Fuente: Desafío Latam

En el ejemplo que se muestra a continuación, se puede ver cómo estos programadores a medida que se van incorporando, se van imprimiendo en consola.

```
import java.util.LinkedHashSet;

       Set<String> programmers = new LinkedHashSet<>();
    programmers.add("James Gosling");
    programmers.add("Martin Odersky");
    programmers.add("Rich Hickey");
    programmers.add("Larry Wall");
    programmers.add("Graydon Hoare");
    System.out.println(programmers);
//[James Gosling, Martin Odersky, Rich Hickey, Larry Wall, Graydon
Hoare]
```

## Creando un TreeSet

Como se mencionó en el párrafo previo, TreeSet almacena sus elementos en un árbol rojo-negro, es decir, ordena sus elementos en función de sus valores. Es sustancialmente más lento que HashSet.

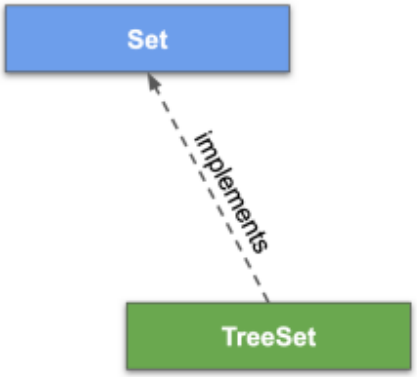


Imagen 3.Implementación de TreeSet en Set.
Fuente: Desafío Latam

## Ejercicio guiado: Capitales del mundo

- **Paso 1:** Para explicar lo que acabamos de mencionar, realizaremos un ejemplo guiado con las capitales sudamericanas. Cabe destacar que repetiremos "Brasilia" a propósito para mostrar que al imprimir capitales, no aparecerá repetida.

```
import java.util.TreeSet;

        Set<String> capitales = new TreeSet<>();
        capitales.add("Buenos Aires");
        capitales.add("Brasilia");
        capitales.add("Asunción");
        capitales.add("Lima");
        System.out.println(capitales); //[Asunción, Brasilia, Buenos
Aires, Lima]
```

- **Paso 2**: Ahora, veremos cómo se conforma la unión de 2 colecciones mediante `TreeSet`. Para ello crearemos otro `HashSet` llamado capitales2 y le incorporaremos 5 capitales más. Luego con un `TreeSet` juntaremos las capitales e imprimimos que resulta del Set capitalesUnidas.

```
      Set<String> capitales = new TreeSet<>();
      capitales.add("Buenos Aires");
      capitales.add("Brasilia");
      capitales.add("Asunción");
      capitales.add("Lima");

      Set<String> capitales2 = new HashSet<>(Arrays.asList("Caracas",
"Bogotá", "Montevideo","Quito", "Brasilia"));

      Set<String> capitalesUnidas = new TreeSet<>(capitales);
      capitalesUnidas.addAll(capitales2);
      System.out.println(capitalesUnidas);
      ---------------------------
      Impresión en pantalla:

      [Asunción, Bogotá ,Brasilia, Buenos Aires, Caracas, Lima,
Montevideo, Quito]
```

- **Paso 3:** Si queremos borrar una colección completa de la lista, debemos usar el método `removeAll()`. Al igual que en `ArrayList()`, remove es un método de acceso posicional que sirve para eliminar elementos desde una colección general, en este caso para eliminar una colección llamada capitales2 (visto en el ejemplo anterior).

```
      Set<String> removerCapitales = new HashSet<>(capitales);
      removerCapitales.removeAll(capitales2);
      System.out.println(removerCapitales);
      ---------------------------
      Impresión en pantalla:

      [Asunción,Brasilia, Buenos Aires, Lima]
```

- **Paso 4**: Para encontrar valores en común entre colecciones, se puede usar el método retainAll(). Para nuestro caso, usaremos las colecciones previas de capitales y capitales2. Es decir, buscaremos la intersección de ambas colecciones.

```
Set<String> interseccionCapitales = new HashSet<>(capitales);
interseccionCapitales.retainAll(capitales2);
System.out.println(interseccionCapitales);
---------------------------
Impresión en pantalla:
[Brasilia]
```