



# NIVEL 6

## Colecciones

Las colecciones son estructuras de datos que nos permiten almacenar y recuperar objetos de cualquier clase.[1]

Su funcionamiento es similar a cuando usábamos arrays para almacenar números, cadenas de caracteres, booleanos, etc. Con la diferencia de que las colecciones son dinámicas ya que su tamaño puede variar como consecuencia de que permiten añadir o eliminar objetos.[2]

Las colecciones son útiles cuando queremos trabajar con un gran número de objetos y necesitamos que estos queden almacenados en el programa de manera temporal, para ello se usa interfaz genérica *Collection*, está a su vez cuenta con tres subtipos principales de estructuras para almacenar datos que son. [3]

### Set

La principal característica de los *Set* es que no permiten que se almacenen valores duplicados, por lo que es útil usarlos cuando queremos que un programa tenga registros de objetos sin valores duplicados.

- **HashSet:** Son conjuntos no ordenados ni clasificados, es decir, cuando añadimos un elemento a la lista este se guarda en una ubicación aleatoria.

```

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

public class Main {

    public static void main(String[] args) {
        Set<String> listaCompras = new HashSet<String>();
        listaCompras.add("Freijoa");
        listaCompras.add("Arandanos");
        listaCompras.add("Tomates");
        listaCompras.add("papa");
        listaCompras.forEach(System.out::println);
    }
}

```

Importamos la librerías de Set y HashSet

Creamos la lista

Añadimos elementos a la lista

Iteramos la lista para mostrar su contenido

#### Nota

Por simplicidad del ejemplo la lista almacena 'Strings', sin embargo se saca más provecho cuando las usamos con Objetos de clases propias del proyecto.

## Consola

```

<terminated> Main (7) [Java Application] C:\Users\aleja\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre
Tomates
Freijoa
Arandanos
papa
|

```

Como los elementos no están ordenados, no es posible obtener el valor de un índice específico, limitando su uso a almacenar los valores al azar y mostrar la lista completa.

- **TreeSet:** Son conjuntos ordenados y clasificados, sus elementos cuentan con un índice para consultarlos luego y además al ser clasificados va organizando los valores entrantes en orden alfabético, llegando a tener lo siguiente:

```

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

public class Main {

    public static void main(String[] args) {
        Set<String> listaCompras = new TreeSet<String>();
        listaCompras.add("Freijoa");
        listaCompras.add("Arandanos");
        listaCompras.add("Tomates");
        listaCompras.add("papa");
        listaCompras.add("Tomates");

        listaCompras.forEach(System.out::println);
    }
}

```

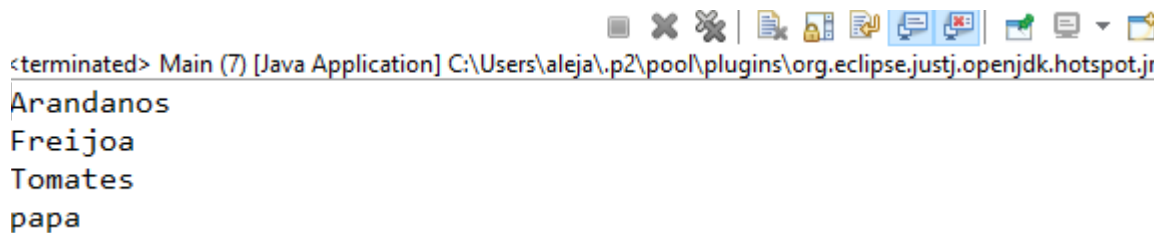
Importamos la librerías de Set y TreeSet

Creamos la lista

Añadimos elementos a la lista

Iteramos la lista para mostrar su contenido

## Consola



```

<terminated> Main (7) [Java Application] C:\Users\aleja\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jr
Arandanos
Freijoa
Tomates
papa

```

## Nota:

Cuando queremos usar los TreeSet con objetos de clases definidas, es necesario implementar la interfaz Comparable para indicarle al programa el atributo que se usará para ordenar la lista (más información [4]).

- **LinkedHashSet:** Son conjuntos ordenados pero no clasificados, es decir, sus elementos están ordenados únicamente de acuerdo al orden en que añadimos un objeto, a diferencia del TreeSet este no organiza los elementos alfabéticamente.

```

import java.util.Set;
import java.util.LinkedHashSet;
}
public class Main {

    public static void main(String[] args) {
        Set<String> listaCompras = new LinkedHashSet<String>();
        listaCompras.add("Freijoa");
        listaCompras.add("Arandanos");
        listaCompras.add("Tomates");
        listaCompras.add("papa");
        listaCompras.add("Tomates");
        listaCompras.forEach(System.out::println);
    }
}

```

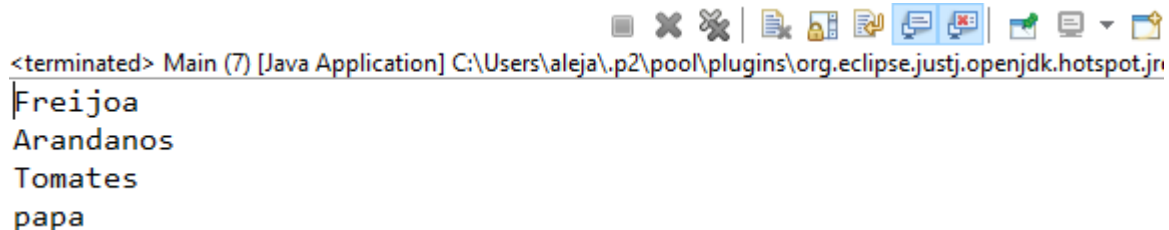
Importamos la librerías de Set y LinkedHashSet

Creamos la lista

Añadimos elementos a la lista

Iteramos la lista para mostrar su contenido

Consola



```

<terminated> Main (7) [Java Application] C:\Users\aleja\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jr
Freijoa
Arandanos
Tomates
papa
Tomates

```

## List

La interfaz List define una sucesión de elementos. Esta interfaz permite el manejo de elementos duplicados. Además, permite añadir métodos que permiten mejorar:

- Acceso posicional a elementos: En el cual manipula elementos en función de su posición en la lista.
- Búsqueda de elementos: Busca un elemento concreto de la lista y devuelve su posición.
- Iteración sobre elementos: En el cual mejora el Iterator por defecto.
- Rango de operaciones: Donde permite realizar ciertas operaciones sobre rangos de elementos de la propia lista.

A continuación, veremos algunos tipos de implementaciones realizadas en Java.

ArrayList: Esta implementación se basa en un arreglo redimensionable que aumenta su tamaño según crece la colección de elementos. Tiene el mejor rendimiento sobre otras situaciones.

```
public class Media {  
    public static void main(String[] args) {  
  
        int nums[] = {3,4,7,8,4,5,6};  
        int suma=0;  
        double media;  
  
        for (int i=0; i<nums.length; i++) {  
            suma=suma+nums[i];  
        }  
        media=(double)suma / nums.length;  
        System.out.println("La media es "+media);  
        System.out.println("Longitud es "+nums.length);  
    }  
}
```

LinkedList: Esta implementación permite la mejora del rendimiento en ciertas ocasiones por medio de una lista doblemente enlazada de los elementos, teniendo cada uno de los elementos un puntero al anterior y al siguiente elemento.

```
LinkedList.add("Facebook");
```

**3 Ways to Iterate through LinkedList**

```
// ListIterator approach  
System.out.println("ListIterator Approach: =====");  
ListIterator<String> listIterator = linkedList.listIterator();  
while (listIterator.hasNext()) {  
    System.out.println(listIterator.next());  
}
```

```
System.out.println("\nLoop Approach: =====");  
// Traditional for loop approach  
for (int i = 0; i < linkedList.size(); i++) {  
    System.out.println(linkedList.get(i));  
}
```

```
// Java8 Loop  
System.out.println("\nJava8 Approach: =====");  
linkedList.forEach(System.out::println);
```



**Nota:** Ninguna de estas dos implementaciones son sincronizadas, es decir, no se garantiza el estado del List si dos o más hilos acceden de forma concurrente al mismo.

```
1 final List arrayList = new ArrayList();  
2 final List linkedList = new LinkedList();
```

## Map

Esta interfaz asocia claves a valores. No puede tener claves duplicadas y, cada una de estas claves, sólo puede tener un valor asignado como máximo.

Dentro de esta interfaz, existen varios tipos de implementaciones realizadas en Java.

- Hashmap: Esta implementación almacena las claves en una tabla hash. Es la implementación con el mejor rendimiento aunque no garantiza ningún orden a la hora de realizar iteraciones. Proporciona tiempos constantes en las operaciones básicas siempre y cuando la función hash disperse de forma correcta los elementos dentro de la tabla. Según la definición del tamaño de la tabla, dependerá del rendimiento dentro de la implementación.

```
public class CrunchifyHashMapContainsKey {
    static Map<String, String> crunchifyComapnies = new HashMap<>();


    private static void checkIfValueExist(String value) {
        // Let's checkout if Value exist
        String result = crunchifyComapnies.containsKey(value) ? ("Value (" + value + ") exist")
            : ("Value (" + value + ") doesn't exist");
        log(result);
    }

    private static void checkIfKeyExist(String key) {
        // Let's checkout if Key exist
        String result = crunchifyComapnies.containsKey(key) ? (crunchifyComapnies.get(key))
            : ("Key (" + key + ") doesn't exist");
        log(result);
    }

    public static void main(String[] args) {

        crunchifyComapnies.put("Google", "Mountain View, CA");
        crunchifyComapnies.put("Yahoo", "Santa Clara, CA");
        crunchifyComapnies.put("Microsoft", "Redmond, WA");
    }
}
```

**Java Hashmap - containsKey(Object key) and containsValue(Object value)  
Check if Key/Value Exists in Map**



- Treemap: Esta implementación almacena las claves organizándose en función de sus valores. Es más lento que Hashmap. Esta implementación garantiza un rendimiento logarítmico en las operaciones básicas, debido a la estructura de árbol empleada para manejar los elementos.

```
TreeMapExample.java
1 package com.kscodes.collections;
2
3 import java.util.TreeMap;
4
5 public class TreeMapExample {
6
7     public static void main(String args[]) {
8
9         // TreeMap with Keys as Integer
10        TreeMap<Integer, String> treeMap1 = new TreeMap<>();
11        treeMap1.put(300, "Steve Smith");
12        treeMap1.put(100, "John Doe");
13        treeMap1.put(50, "Rick Martin");
14
15        System.out.println("Printing the TreeMap with Integers");
16        System.out.println(treeMap1.toString());
17
18        System.out.println("-----");
19    }
20 }
```

<terminated> TreeMapExample [Java Application] C:\Program Files\Java\jre1.8.0\_111\bin\javaw.exe  
Printing the TreeMap with Integers  
{50=Rick Martin, 100=John Doe, 300=Steve Smith}  
-----  
Printing the TreeMap with Strings  
{A=Apple, B=Ball, D=Dog, Z=Zebra}

- LinkedHashMap: Esta implementación almacena las claves en función del orden de inserción.

```
LinkedHashMapExample.java
8
9 public class LinkedHashMapExample {
10     public static void main(String args[]) {
11
12         // Create a linkedHashMap and add elements to it.
13         LinkedHashMap<String, String> linkedHashMap = new LinkedHashMap<>();
14         linkedHashMap.put("Z", "Zebra");
15         linkedHashMap.put("A", "Apple");
16         linkedHashMap.put("D", "Dog");
17         linkedHashMap.put("B", "Ball");
18
19         // Display the Elements by Iterating over the linkedHashMap
20         System.out.println("Displaying the Contents of linkedHashMap");
21         System.out.println("-----");
22         Set<Entry<String, String>> set = linkedHashMap.entrySet();
23         Iterator<Entry<String, String>> iterator = set.iterator();
24         while (iterator.hasNext()) {
25             Map.Entry mapEntry = (Map.Entry) iterator.next();
26             System.out.print(mapEntry.getKey() + " : " + mapEntry.getValue());
27             System.out.println("");
28         }
29     }
30 }
```

<terminated> LinkedHashMapExample [Java Application] C:\Program Files\Java\jre1.8.0\_111\bin\javaw.exe (Dec 28, 201  
Displaying the Contents of linkedHashMap  
-----  
Z : Zebra  
A : Apple  
D : Dog  
B : Ball

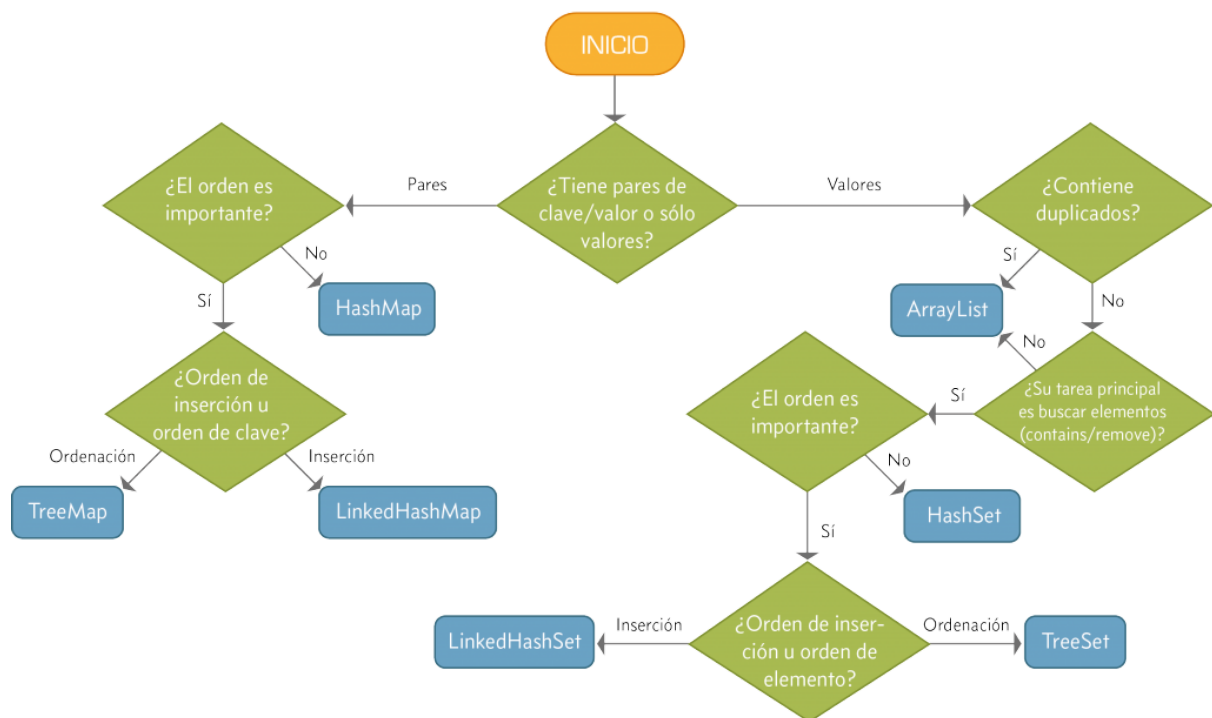


**Nota:** Ninguna de las implementaciones son sincronizadas, es decir, no se garantiza el estado del Map si dos o más hilos acceden de forma concurrente al mismo.

```
1 final Map<Integer, List> hashMap = new HashMap<Integer, List>();
2 final Map<Integer, List> treeMap = new TreeMap<Integer, List>();
3 final Map<Integer, List> linkedHashMap = new LinkedHashMap<Integer, List>();
```

Para conocer qué tipo de colección usar, podemos emplear el siguiente diagrama:

Diagrama de decisión para uso de colecciones Java



Anexo 1. Mapa conceptual para decidir qué tipo de colección usar.[3]



## Referencias

1. Máster en Desarrollo de Aplicaciones Android - Las colecciones en Java. (2017). Androidcurso.com. <http://www.androidcurso.com/index.php/tutoriales-java-esencial/462-las-colecciones-en-java>
2. Bartolomé Abellán. (2012). Colecciones. Blogspot.com. <http://picarcodigo.blogspot.com/2012/12/colecciones.html>
3. Rafael Vindel Amor. (2015, September 25). Introducción a Colecciones en Java - Adictos al trabajo. Adictos al Trabajo. <https://www.adictosaltrabajo.com/2015/09/25/introduccion-a-colecciones-en-java/>
4. MitoCode. (2016). Tutorial Java 7 SE Avanzado - 14 TreeSet [YouTube Video]. In YouTube. <https://www.youtube.com/watch?v=Gclb5YHli0o>
5. Rafael Vindel Amor. (2015, September 25). *Introducción a Colecciones en Java - Adictos al trabajo*. Adictos al Trabajo. <https://www.adictosaltrabajo.com/2015/09/25/introduccion-a-colecciones-en-java/>