# UD2

## LENGUAJE DE PROGRAMACIÓN JAVA

MP_0485
Programación

## 4.3 Conjuntos

# Introducción

Los arrays nos ofrecen una interesante forma de estructurar datos en la memoria, pero tienen el problema de que es necesario saber previamente la longitud o número de elementos que tendrá cada array. Para resolver este problema podemos utilizar otro recurso que nos ofrece Java; las clases que representan colecciones y las clases que representan mapas

Colecciones y mapas sirven para almacenar en la memoria un grupo o conjunto de elementos, al igual que los arrays, pero tienen la gran ventaja de crecer dinámicamente según las necesidades del programa en cada momento.

Java cuenta con muchísimas clases que sirven para representar colecciones de datos y mapas, clases que forman parte de una compleja jerarquía. En la siguiente imagen te mostramos la jerarquía que corresponde a las clases con las que vamos a trabajar.

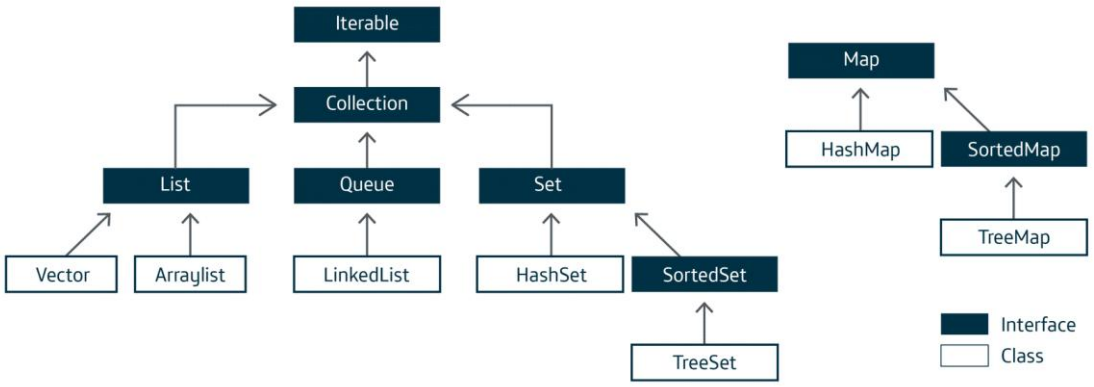



Figura 1: Estructura jerárquica de las interfaces Coleccion y Map.

En la imagen hemos simplificado el árbol jerárquico de las colecciones y mapas, eliminando algunas ramas y dejando lo más relevante. Lo importante es que comprendas que cuando utilizas una clase, esta no está aislada, sino que forma parte de una estructura jerárquica más compleja.

Ten en cuenta que en la imagen los rectángulos con fondo negro son interfaces y los rectángulos con fondo blanco son clases. Así puedes fácilmente deducir que la clase Vector implementa la interfaz List, e indirectamente, también las interfaces Collection e Iterable. También puedes comprobar que la clase LinkedList implementa las interfaces List y Queue, y también, de manera indirecta, las interfaces Collection e Iterable.

## Los mapas

Los mapas son un tipo especial de colecciones que asocian una clave a cada elemento, que servirá después como llave para acceder a dicho elemento. Las clases que representan mapas derivan de la interfaz Map (ver figura 1).

En este apartado trabajaremos con las clases HashMap y TreeMap. Es importante tener en cuenta que estas clases no derivan de Iterable y Collection, por lo que muchos de los métodos que te has acostumbrado a usar no están disponibles.

Los mapas tienen ciertas similitudes con los conjuntos:

- Igual que ocurre con los conjuntos, los elementos no se colocan en el orden en que se han añadido.
- Los objetos TreeMap se colocan ordenados según la clave.
- Los mapas no admiten valores duplicados en la clave o llave. Si añadimos un elemento con la misma clave que otro existente, sobrescribirá el elemento existente.

Todas las clases que derivan directa o indirectamente de la interfaz Set pertenecen a un tipo especial de colecciones llamadas conjuntos. Estas colecciones están basadas en la teoría de conjuntos de matemáticas. La característica más importante de este tipo de colecciones es que no admiten duplicados.

## HashMap

Un objeto HashMap es una colección donde cada elemento tiene asociada una clave que sirve para distinguirlo de los demás elementos de la colección. Vamos a comenzar por crear una colección de nombres, es decir, de objetos String.

```java
import java.util.Map;
import java.util.HashMap;
import java.util.Set;

public class Principal {
	public static void main(String args[]) {
		Map<String, String> nombres = new HashMap<String, String>()

		nombres.put("51666443R", "Carlos Maldonado Gómez");
		nombres.put("51666443R", "Luis Santos Gómez");
		nombres.put("52664443A", "Alicia Torres Durán");
		nombres.put("31234443H", "Soledad Delgado Perico");
		nombres.put("45666443R", "Miguel Rubio gonzález");
		nombres.put("82333333T", "Alicia Pimiento Pérez");
		nombres.put("51777788Z", "Angel Ruiz Califato");
		nombres.put("91549494P", "Fernándo García Solera");

		// Acceder a un elemento por la clave
		System.out.println(nombres.get("31234443H"));

		// Recorrer el mapa
		Set<String> claves = nombres.keySet();

		for (String key : claves) {
			System.out.println(key + " - " + nombres.get(key));
		}

		System.out.println("Hay " + nombres.size() + " personas");
	}
}
```

Vamos a analizar el ejemplo paso a paso:

**1. Construir el objeto HasMap**

Primero hemos creado el objeto HashMap nombres utilizando el siguiente formato:

```
Map<tupoClave, tipoElemento> nombreObjeto = new HashMap<>();
```

HashMap es una clase genérica, maneja internamente dos objetos cuyo tipo puede variar. tipoClave corresponde al tipo de dato de las claves y tipoElementos corresponde al tipo de dato de los elementos.

```
Map<String, String> nombres = new HashMap<>();
```

En nuestro ejemplo tenemos una colección de elementos de tipo String (nombres), cuya clave es un DNI, que es también un texto. Lo más habitual es que la clave sea de tipo Integer o de tipo String.

**2. Insertar elementos**

Luego hemos añadido varios elementos utilizando el método put, que tiene el siguiente formato:

```
objMap.put(Object key, Object element);
```

El método put() recibe la clave (como objeto) y el elemento al que se asocia dicha clave (como objeto) y retorna el elemento que acaba de añadirse.

```
nombres.put("51666443R", "Carlos Maldonado Gómez");
nombres.put("51666443R", "Luis Santos Gómez");
nombres.put("52664443A", "Alicia Torres Durán");
nombres.put("31234443H", "Soledad Delgado Perico");
nombres.put("45666443R", "Miguel Rubio González");
nombres.put("82333333T", "Alicia Pimiento Pérez");
nombres.put("51777788Z", "Angel Ruiz Califato");
nombres.put("91549494P", "Fernándo García Solera");
```

Se han añadido dos elementos con la misma clave ("51666443R") a propósito para que puedas comprobar lo que ocurre en este caso. El método put() añade un elemento si la clave especificada no existe, pero si existe lo que hace es una modificación, es decir, sobrescribe el elemento asociado con dicha clave. Puedes comprobar que "Carlos Maldonado Gómez" no aparece en el listado, ha sido sustituido por "Luis Santos Gómez".

**3. Buscar un elemento aleatoriamente**

El método get() permite el acceso aleatorio a través de la clave. Si la clave buscada no existe devuelve null.

```
// Acceder a un elemento por la clave
System.out.println(nombres.get("31234443H"));
```

**4. Iterando el mapa**

Las colecciones de tipo Map no descienden de Iterable, luego es fácil deducir que no son colecciones iterables. No podemos hacer ninguna de estas operaciones con nuestro objeto nombres:

```
Iterator nombres = nombres.iterator();
for (String n : nombres) { }
```

Sin embargo, hay una solución. Las clases que implementan la interfaz Map tienen un par de métodos que vienen a salvarnos de esta situación:

```
objMap.keySet(); // Devuelve un objeto de tipo Set con las claves.
objMap.values(); // Devuelve un objeto de tipo Set con los valores, sin las claves.
```

Los objetos devueltos por estos métodos sí son iterables, ya que son conjuntos.

```
// Recorrer el mapa
Set claves = nombres.keySet();
for (String key : claves) {
	System.out.println(key + " - " + nombres.get(key));
}
```

**5. Obtener el número de elementos de la colección**

```
System.out.println("Hay " + nombres.size() + " personas");
```

El método size() devuelve un valor int con el número de elementos que contiene el mapa.

## TreeMap

Un objeto TreeMap funciona igual que un objeto HashMap, **pero sus elementos estarán ordenados según la clave o llave**.

Como prueba, vamos a utilizar el mismo ejemplo anterior, pero ahora con un objeto TreeMap.

```
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

public class Principal {
	public static void main(String args[]) {
		Map<String, String> nombres = new TreeMap<String, String>()

		nombres.put("51666443R", "Carlos Maldonado Gómez");
		nombres.put("51666443R", "Luis Santos Gómez");
		nombres.put("52664443A", "Alicia Torres Durán");
		nombres.put("31234443H", "Soledad Delgado Perico");
		nombres.put("45666443R", "Miguel Rubio gonzález");
		nombres.put("82333333T", "Alicia Pimiento Pérez");
		nombres.put("51777788Z", "Angel Ruiz Califato");
		nombres.put("91549494P", "Fernándo García Solera");

		// Acceder a un elemento por la clave
		System.out.println(nombres.get("31234443H"));

		// Recorrer el mapa
		Set<String> claves = nombres.keySet();

		for (String key : claves) {
			System.out.println(key + " - " + nombres.get(key));
		}

		System.out.println("Hay " + nombres.size() + " personas");
	}
}
```

En la ejecución puedes comprobar que los elementos aparecen ordenados por DNI.

Aprovecharemos la clase Triangulo creada anteriormente para crear un objeto TreeMap, que contendrá una colección de objetos Triangulo.

```
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;


public class Principal {
        public static void main(String args[]) {
                Map<Integer, Triangulo> trians = new TreeMap<Integer, Trian

                trians.put(1, new Triangulo(1, 2, 2));
                trians.put(2, new Triangulo(2, 2, 3));
                trians.put(3, new Triangulo(1, 2, 3));
                trians.put(4, new Triangulo(4, 4, 4));

                Set<Integer> claves = trians.keySet();
                for (Integer key : claves) {
                        System.out.println(trians.get(key).verTipo());
                }

        }
}
```

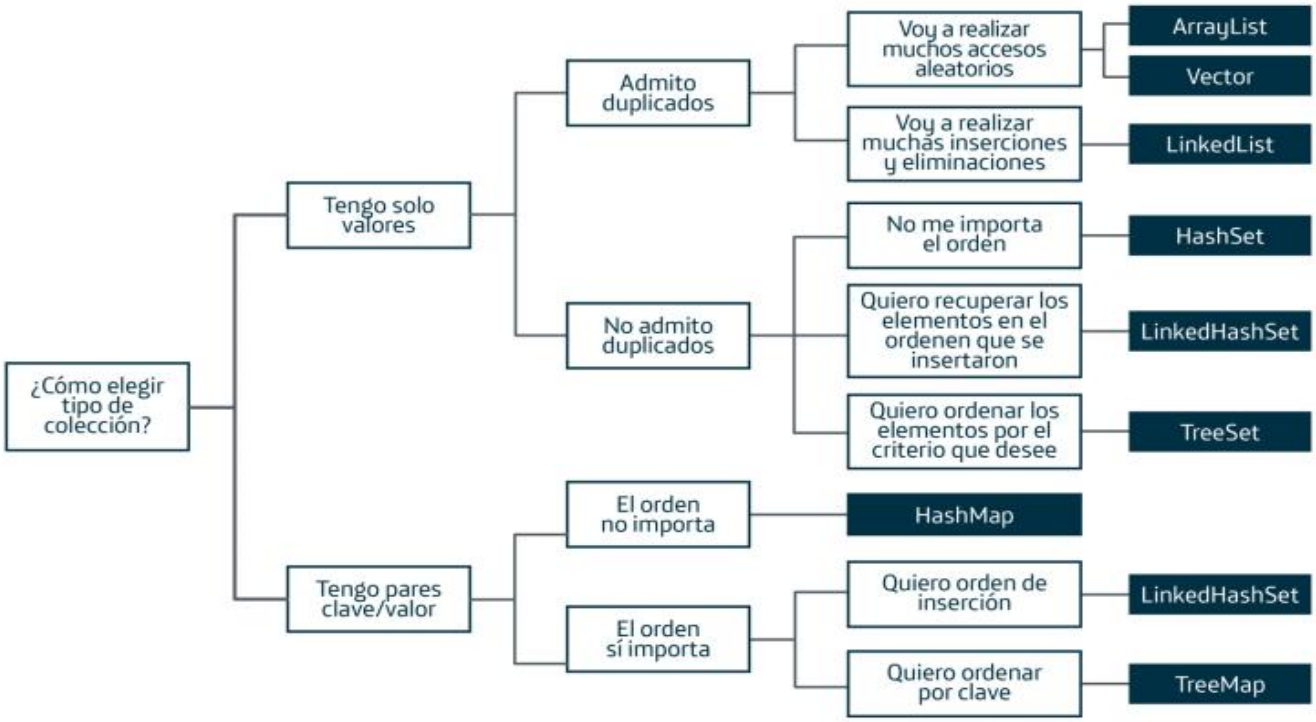
## ¿Cómo elegir el tipo de colección?

En esta lección hemos visto buena parte de las clases Java que puedes utilizar para construir colecciones de objetos, pero aún hay más.

Ahora podemos, de manera opcional, investigar por tu cuenta y utilizar algún tipo más de colección de las que no hemos visto en esta unidad. Por ejemplo, puedes probar con:

- **LinkedHashMap:** lista enlazada de elementos clave/valor (mapa). Puedes utilizarla igual que HashMap, pero al realizar una iteración los elementos se recuperan en el mismo orden en que se insertaron.
- **LinkedHashSet:** lista enlazada de elementos que no admite duplicados (conjunto). Puedes utilizarla igual que HashSet, pero al realizar una iteración los elementos se recuperan en el mismo orden en que se insertaron.
- **Stack:** lista de elementos basada en un array de tipo pila o estructura LIFO (Last In, First Out), donde el último en entrar es el primero en salir.

Con tantas clases disponibles para construir colecciones de datos, ¿cómo saber la que más nos conviene? La siguiente imagen resume de manera esquemática cómo decidir el tipo de colección que necesitamos:




## Referencias

Apuntes elaborados a partir de la siguiente documentación:

[1] Apuntes Fernando Barber y Ricardo Ferris. Universidad de Valencia.

[2] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.

[3] Apuntes de Programación de Carlos Cacho y Raquel Torres. Ceedcv.

[4] Apuntes de Programación Edix Digital Workers.

Licencia

CC BY-NC-SA 3.0 ES **Reconocimiento – No Comercial – Compartir Igual (by-nc-sa)**
No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.
NOTA: Esta es una obra derivada de la original realizada por Carlos Cacho y Raquel Torres.