75 - Colecciones: HashSet, TreeSet y LinkedHashSet

Listado completo de tutoriales

La diferencia fundamental entre las clases HashSet, TreeSet, LinkedHashSet con respecto a las listas ArrayList y LinkedList es que no puede haber elementos repetidos en las colecciones que implementan la interfaz Set.

A su vez se han creado estas tres clases que tienen pequeñas diferencias entre una y otras:

- HashSet: El conjunto de datos no se almacena en un orden específico, si bien se garantiza que no hay duplicados.
- TreeSet: Los elementos del conjunto se almacenan de menor a mayor.
- LinkedHashSet: Los elementos del conjunto se encuentran en el orden que se insertan, similar a una lista pero sin dejar ingresar valores repetido.

El siguiente programa muestra la sintaxis para crear objetos de estas clases y los métodos principales que disponen:

Programa:

Ver video

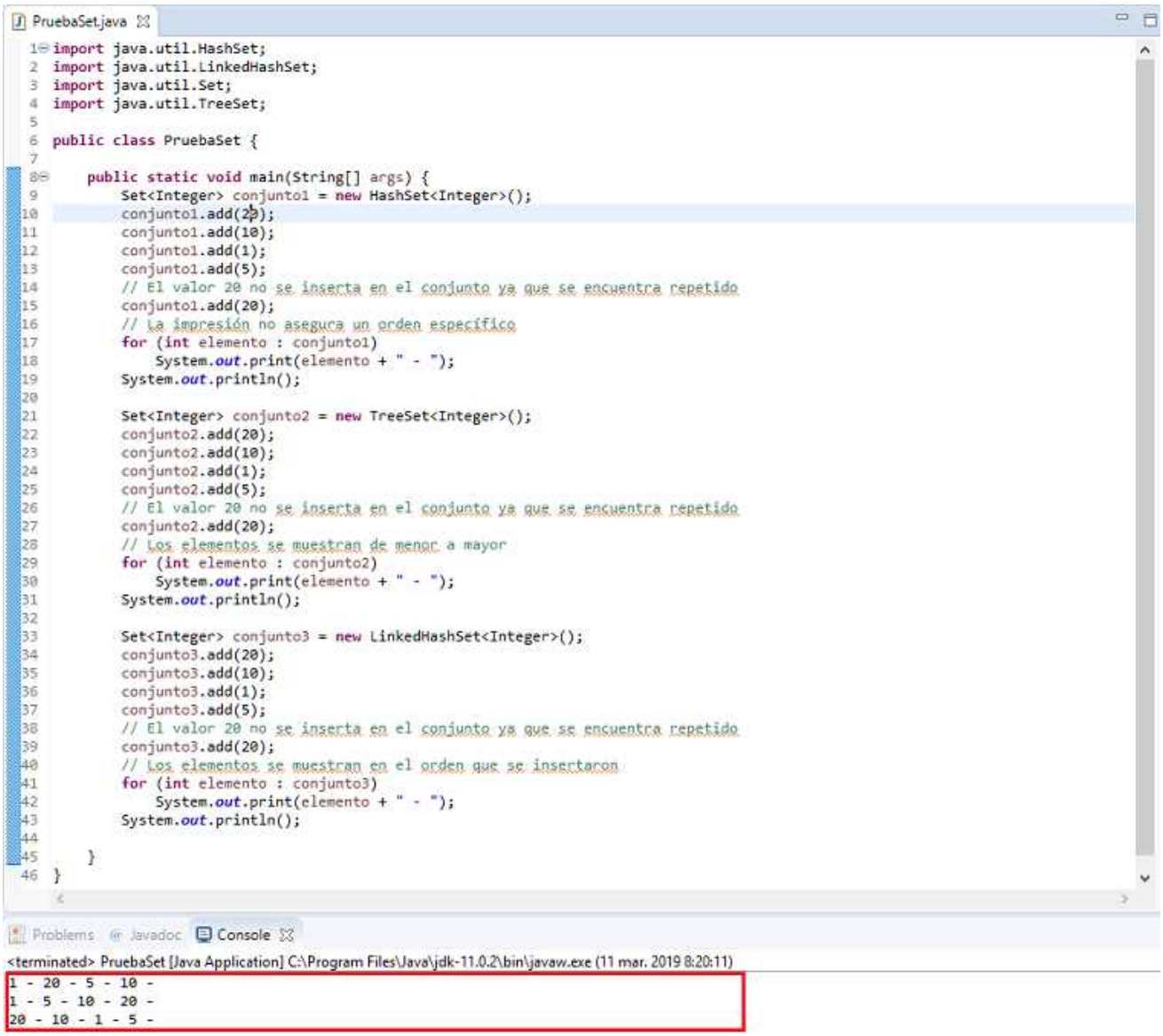
```
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.Set;
import java.util.TreeSet;

public class PruebaSet {

    public static void main(String[] args) {
        Set<Integer> conjunto1 = new HashSet<Integer>();
        conjunto1.add(20);
        conjunto1.add(10);
        conjunto1.add(1);
        conjunto1.add(5);
        // El valor 20 no se inserta en el conjunto ya que se encuentra repetido
        conjunto1.add(20);
        // La impresión no asegura un orden específico
        for (int elemento : conjunto1)
            System.out.print(elemento + " - ");
        System.out.println();

        Set<Integer> conjunto2 = new TreeSet<Integer>();
        conjunto2.add(20);
        conjunto2.add(10);
        conjunto2.add(1);
        conjunto2.add(5);
        // El valor 20 no se inserta en el conjunto ya que se encuentra repetido
        conjunto2.add(20);
        // Los elementos se muestran de menor a mayor
```

El resultado de ejecutar el programa es similar a:



Métodos más columnes

Los métodos más comunes que tienen estas clases son:

- size: Retorna la cantidad de elementos del conjunto.
- clear: Elimina todos los elementos.
- remove: Elimina el elemento si existe en el conjunto:

```
lista1.remove(20);
```

- isEmpty: Nos informa si la lista está vacía.
- contains: Le pasamos como parámetro el dato a buscar en el conjunto:

```
if (conjunto1.contains(20))
...
```

Más datos podemos conseguir visitando la documentación oficial de las clases HashSet, TreeSet y LinkedHashSet.

Problema

Generar una lista de 10 valores enteros comprendidos entre 1 y 100. Validar que no se repitan, para esto utilizar la ayuda de una de las colecciones de conjuntos visto en este concepto.

Programa:

Ver video

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

public class Lista10Valores {
    public static void main(String[] args) {
        Set<Integer> conjunto1 = new TreeSet<Integer>();
        while (conjunto1.size() < 10) {
            int aleatorio = (int) (Math.random() * 100) + 1;
            conjunto1.add(aleatorio);
        }
        System.out.println(conjunto1);
    }
}
```

Dentro de un while mientras el objeto 'conjunto1' tenga menos de 10 elementos, procedemos a generar otro valor aleatorio y lo agregamos al conjunto, como sabemos si el valor ya existe en el conjunto1 luego el método 'add' no lo agrega:

```
while (conjunto1.size() < 10) {
    int aleatorio = (int) (Math.random() * 100) + 1;
    conjunto1.add(aleatorio);
}
```

Podemos recorrer el conjunto para imprimirlo mediante un for o inclusive utilizar el método 'println':

```
System.out.println(conjunto1);
```

Retornar