MANEJO DE UNA ESTRUCTURA HASHSET

```java
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

/**
 * Manejo del HashSet.
 *
 * HashSet se implementa utilizando una tabla hash, por lo que al momento de
 * visualizar los datos se proporcionarán en el orden en el que fueron
 * almacenados en la tabla. Desde ya, los elementos no están ordenados. No se
 * aceptan claves duplicadas.
 * Las operaciones de add, remove y contains se realizan en O(1).
 *
 */
public class EjemploHashSet {
	public static void main(String[] args) {

		Set<Integer> claves = new HashSet<Integer>();

		claves.clear();                              // Elimina todos los elementos.

		/* ¿La tabla está vacía? */
		System.out.println("isEmpty(): " + claves.isEmpty()); // true

		claves.add(100);                             // Ingresa 100
		claves.add(10);                              // Ingresa 10
		claves.add(50);                              // Ingresa 50
		claves.add(1);                               // Ingresa 1
		claves.add(13);                              // Ingresa 13
		claves.add(100);                             // Ingresa nuevamente 100

		System.out.println("set: " + claves);        // Imprime

		claves.remove(10);                           // Elimina clave 10

		// Muestra todos los elementos
		for (Integer e : claves) {
			System.out.print(e + " ");
		}
		System.out.println();

		System.out.println("contains(50): " + claves.contains(50)); // ¿Contiene clave = 50?

		// Muestra todos los elementos
		Iterator<Integer> it = claves.iterator();
		while (it.hasNext()) {
			System.out.print(it.next() + " ");
		}
	}
}
```

MANEJO DE UNA ESTRUCTURA LINKEDHASHSET

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

/**
 * Manejo del LinkedHashSet.
 *
 * LinkedHashSet se encuentra entre HashSet y TreeSet. Se implementa como una
 * tabla hash, con una lista doblemente enlazada que se ejecuta a través de
 * ella. Al momento de visualizar los datos se proporcionarán en el orden en el
 * que fueron insertados los datos. No se aceptan claves duplicadas.
 * Las operaciones de add, remove y contains se realizan en O(1).
 *
 */
public class EjemploLinkedHashSet {
	public static void main(String[] args) {

		Set<Integer> claves = new LinkedHashSet<Integer>();

		claves.clear();                                 // Elimina todos los elementos.

		/* ¿La tabla está vacía? */
		System.out.println("isEmpty(): " + claves.isEmpty()); // true

		claves.add(100);                                // Ingresa 100
		claves.add(10);                                 // Ingresa 10
		claves.add(50);                                 // Ingresa 50
		claves.add(1);                                  // Ingresa 1
		claves.add(13);                                 // Ingresa 13
		claves.add(100);                                // Ingresa nuevamente 100

		System.out.println("set: " + claves);           // Imprime

		claves.remove(10);                              // Elimina clave 10

		System.out.println("contains(50): " + claves.contains(50)); // ¿Contiene clave = 50?

		// Muestra todos los elementos
		for (Integer e : claves) {
			System.out.print(e + " ");
		}
	}
}
```

MANEJO DE UNA ESTRUCTURA TREESET

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

/**
 * Manejo del TreeSet.
 *
 * TreeSet se implementa en un árbol binario de búsqueda (BST) balanceado de
 * sólo claves. Los elementos del conjunto están ordenados. No se aceptan claves
 * duplicadas.
 * Las operaciones de add, remove y contains se realizan en O(log n).
 * Ofrece varios métodos para tratar con el conjunto ordenado, tales como
 * first(), last(), headSet(), tailSet(), etc.
 *
 */
public class EjemploTreeSet {
	public static void main(String[] args) {

		Set<Integer> set = new HashSet<Integer>();

		set.clear();                                    // Elimina todos los elementos.

		/* ¿La tabla está vacía? */
		System.out.println("isEmpty(): " + set.isEmpty());  // true

		set.add(100);                                   // Ingresa 100
		set.add(10);                                    // Ingresa 10
		set.add(50);                                    // Ingresa 50
		set.add(1);                                     // Ingresa 1
		set.add(13);                                    // Ingresa 13
		set.add(100);                                   // Ingresa nuevamente 100

		System.out.println("set: " + set);              // Imprime

		TreeSet<Integer> treeset = new TreeSet<Integer>(set);

		System.out.println("Ordenado treeset: " + treeset); // Imprime

		// Métodos de la interface SortedSet
		System.out.println("first(): " + treeset.first());         // Primer elemento
		System.out.println("last(): " + treeset.last());           // Ultimo elemento
		System.out.println("headSet(50): " + treeset.headSet(50)); // claves < a 50
		System.out.println("tailSet(50): " + treeset.tailSet(50)); // claves >= a 50

		// Métodos de la interface NavigableSet
		System.out.println("lower(50): " + treeset.lower(50));     // sig. < a 50
		System.out.println("higher(50): " + treeset.higher(50));   // sig. > a 50
		System.out.println("floor(40): " + treeset.floor(40));     // sig. <= a 40
		System.out.println("ceiling(40): " + treeset.ceiling(40)); // sig. >= a 40
		System.out.println("pollFirst(): " + treeset.pollFirst()); // Elimina el primero
		System.out.println("pollLast(): " + treeset.pollLast());   // Elimina el último
		System.out.println("Nuevo treeset: " + treeset);           // Imprime
	}
}
```