MANEJO DE UNA ESTRUCTURA HASHMAP

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

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

		Map<Integer, String> mapa = new HashMap<Integer, String>();

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

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

		mapa.put(100, "a");                                  // Ingresa <100,"a">
		mapa.put(10, "b");                                   // Ingresa <10,"b">
		mapa.put(50, "c");                                   // Ingresa <50,"c">
		mapa.put(1, "d");                                    // Ingresa <1,"d">
		mapa.put(13, "e");                                   // Ingresa <13,"e">
		System.out.println("mapa: " + mapa);   // Imprime

		mapa.remove(10);                                     // Elimina <10, "b">

		System.out.println("claves: " + mapa.keySet());     // Imprime [1, 50, 100, 13]
		System.out.println("valores: " + mapa.values());    // Imprime [d, c, a, e]
		System.out.println("containsKey(50): " + mapa.containsKey(50)); // ¿Contiene?
		System.out.println("get(50): " + mapa.get(50));     // Obtiene

		// Muestra todos los elementos
		for (Map.Entry<Integer, String> e : mapa.entrySet())
			System.out.print("[" + e.getKey() + "," + e.getValue() + "] ");
	}
}
```

MANEJO DE UNA ESTRUCTURA LINKEDHASHMAP

```java
import java.util.LinkedHashMap;
import java.util.Map;

/**
 * Manejo de LinkedHashMap.
 *
 * LinkedHashMap se encuentra entre HashMap y TreeMap. Se implementa como una
 * tabla hash, con una lista doblemente enlazada que se ejecuta a través de
 * ella. Al momento de visualizar los datos (clave -> dato) 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 EjemploLinkedHashMap {
	public static void main(String[] args) {

		Map<Integer, String> mapa = new LinkedHashMap<Integer, String>();

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

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

		mapa.put(100, "a");                                  // Ingresa <100,"a">
		mapa.put(10, "b");                                   // Ingresa <10,"b">
		mapa.put(50, "c");                                   // Ingresa <50,"c">
		mapa.put(1, "d");                                    // Ingresa <1,"d">
		mapa.put(13, "e");                                   // Ingresa <13,"e">
		System.out.println("mapa: " + mapa);   // Imprime

		mapa.remove(10);                                     // Elimina <10, "b">

		System.out.println("claves: " + mapa.keySet());     // Imprime [100, 50, 1, 13]
		System.out.println("valores: " + mapa.values());    // Imprime [a, c, d, e]
		System.out.println("containsKey(50): " + mapa.containsKey(50)); // ¿Contiene?
		System.out.println("get(50): " + mapa.get(50));     // Obtiene

		// Muestra todos los elementos
		for (Map.Entry<Integer, String> e : mapa.entrySet())
			System.out.print("[" + e.getKey() + "," + e.getValue() + "] ");

	}
}
```

MANEJO DE UNA ESTRUCTURA TREEMAP

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

/**
 * Manejo de TreeMap.
 *
 * TreeMap se implementa en un árbol binario de búsqueda (BST) balanceado por
 * sólo claves. Los elementos del conjunto están ordenados por clave. 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 EjemploTreeMap {
	public static void main(String[] args) {

		TreeMap<Integer, String> arbol = new TreeMap<Integer, String>();

		arbol.put(100, "a");                          // Ingresa <100,"a">
		arbol.put(10, "b");                           // Ingresa <10,"b">
		arbol.put(50, "c");                           // Ingresa <50,"c">
		arbol.put(1, "d");                            // Ingresa <1,"d">
		arbol.put(13, "e");                           // Ingresa <13,"e">
		System.out.println("arbol: " + arbol); // Imprime

		// Obtiene el BST balanceado entre 13 <= clave < 100
		SortedMap<Integer, String> res = arbol.subMap(13, 100);
		System.out.println("Sub arbol (entre 13 y menor a 100):");
		System.out.print("\tclaves: " + res.keySet());      // Imprime [13, 50]
		System.out.println("\tvalores: " + res.values());   // Imprime [e, c]

		// Muestra todos los elementos
		for (Map.Entry<Integer, String> e : arbol.entrySet())
			System.out.print("[" + e.getKey() + "," + e.getValue() + "]");
		System.out.println();

		// Métodos de la interface SortedMap
		System.out.println("firstKey(): " + arbol.firstKey());    // Primera clave
		System.out.println("lastKey(): " + arbol.lastKey());      // Ultima clave
		System.out.println("headMap(50): " + arbol.headMap(50));  // claves < a 50
		System.out.println("tailMap(50): " + arbol.tailMap(50));  // claves >= a 50

		// Métodos de la interface NavigableMap
		System.out.println("lowerKey(50): " + arbol.lowerKey(50));       // sig. < a 50
		System.out.println("higherKey(50): " + arbol.higherKey(50));     // sig. > a 50
		System.out.println("floorKey(40): " + arbol.floorKey(40));       // sig. <= a 40
		System.out.println("ceilingKey(40): " + arbol.ceilingKey(40));   // sig. >= a 40
		System.out.println("pollFirstEntry(): " + arbol.pollFirstEntry()); // Eli. el primero
		System.out.println("pollLastEntry(): " + arbol.pollLastEntry());// Eli. el último
		System.out.println("Nuevo arbol: " + arbol); // Imprime
	}
}
```