Tipo de dato: **MyHashMap<K, V>**

Operaciones:

* **MyHashMap()**: Crea un nuevo objeto **MyHashMap** con una capacidad predeterminada.
* **MyHashMap(capacity: int)**: Crea un nuevo objeto **MyHashMap** con la capacidad especificada.
* **size() -> int**: Retorna el número de pares clave-valor en el mapa.
* **isEmpty() -> bool**: Retorna **true** si el mapa está vacío, **false** de lo contrario.
* **containsKey(key: K) -> bool**: Retorna **true** si el mapa contiene la clave especificada, **false** de lo contrario.
* **containsValue(value: V) -> bool**: Retorna **true** si el mapa contiene el valor especificado, **false** de lo contrario.
* **get(key: K) -> V**: Retorna el valor asociado a la clave especificada, o **null** si la clave no está presente en el mapa.
* **put(key: K, value: V) -> V**: Inserta un nuevo par clave-valor en el mapa. Si la clave ya está presente, actualiza el valor y retorna el valor antiguo. Si el mapa está lleno según el factor de carga, se redimensiona.
* **remove(key: K) -> V**: Elimina la entrada con la clave especificada del mapa. Retorna el valor asociado a la clave, o **null** si la clave no está presente en el mapa.
* **hash(key: K) -> int**: Retorna el índice de la tabla de hashing asociado a la clave especificada. (Método privado)

Atributos:

* **size: int**: El número de pares clave-valor en el mapa.
* **capacity: int**: La capacidad actual de la tabla de hashing.
* **table: Entry<K, V>[]**: La tabla de hashing que almacena las entradas de clave-valor.

Tipos de dato auxiliares:

* **Entry<K, V>**: Una entrada en el mapa, que contiene una clave, un valor y un puntero a la siguiente entrada en la lista enlazada (en caso de colisión).

Here's the TAD for the given Java class **MyLinkedList<E>**:

* **MyLinkedList()**:
  + Constructs an empty **MyLinkedList** object.
  + Complexity: O(1).
* **size()**:
  + Returns the number of elements in this list.
  + Complexity: O(1).
* **isEmpty()**:
  + Returns **true** if this list contains no elements.
  + Complexity: O(1).
* **addFirst(E element)**:
  + Inserts the specified element at the beginning of this list.
  + Complexity: O(1).
* **addLast(E element)**:
  + Appends the specified element to the end of this list.
  + Complexity: O(1).
* **removeFirst()**:
  + Removes and returns the first element from this list.
  + Returns **null** if this list is empty.
  + Complexity: O(1).
* **removeLast()**:
  + Removes and returns the last element from this list.
  + Returns **null** if this list is empty.
  + Complexity: O(1).
* **remove(E element)**:
  + Removes the first occurrence of the specified element from this list, if it is present.
  + Returns **true** if this list contained the specified element, otherwise **false**.
  + Complexity: O(n), where n is the number of elements in the list.
* **getFirst()**:
  + Returns the first element in this list.
  + Returns **null** if this list is empty.
  + Complexity: O(1).
* **getLast()**:
  + Returns the last element in this list.
  + Returns **null** if this list is empty.
  + Complexity: O(1).
* **contains(E element)**:
  + Returns **true** if this list contains the specified element.
  + Complexity: O(n), where n is the number of elements in the list.

Type name:

* **MyMap<K, V>**

Initialization operation:

* **MyMap()**: Initializes an empty map.

Access operations:

* **get(K key)**: Returns the value associated with the given key in the map, or **null** if the key is not present.
* **containsKey(K key)**: Returns **true** if the given key is present in the map, or **false** otherwise.

Update operations:

* **put(K key, V value)**: Adds or updates the value associated with the given key in the map.
* **remove(K key)**: Removes the key-value pair associated with the given key from the map, and returns **true** if it was present, or **false** otherwise.

Observation operation:

* **size()**: Returns the number of key-value pairs in the map.

Internal representation:

* The map is implemented using an array of linked lists, where each linked list contains the key-value pairs that have the same index in the array. Each node in the linked list has a key, a value, and a reference to the next node in the list.

Node:

* **private static class Node<K, V>**: Represents a node in a linked list of key-value pairs.
* Fields:
  + **private final K key**: The key of the node.
  + **private V value**: The value of the node.
  + **private Node<K, V> next**: The next node in the linked list.
* Constructor:
  + **private Node(K key, V value)**: Initializes a new node with the given key and value.

Representation invariants:

* The **buckets** array is never null.
* Each node in the linked list has a non-null key.
* The **size** field reflects the actual number of key-value pairs in the map.
* The load factor (i.e., the ratio between the number of key-value pairs and the length of the **buckets** array) is less than or equal to a constant value, to ensure good performance of the map operations.

TAD para MyPriorityQueue<E>:

Tipo de datos:

* MyPriorityQueue<E>

Operaciones:

* MyPriorityQueue(Comparator<E> comparator): constructor que inicializa una cola de prioridad vacía con un comparador dado.
* MyPriorityQueue(int capacity, Comparator<E> comparator): constructor que inicializa una cola de prioridad vacía con una capacidad y un comparador dados.
* boolean add(E element): agrega un elemento a la cola de prioridad. Devuelve verdadero si el elemento fue agregado con éxito.
* E peek(): devuelve el elemento de mayor prioridad sin eliminarlo de la cola. Si la cola está vacía, devuelve null.
* E poll(): devuelve el elemento de mayor prioridad y lo elimina de la cola. Si la cola está vacía, devuelve null.
* boolean isEmpty(): devuelve verdadero si la cola de prioridad está vacía.
* int size(): devuelve la cantidad de elementos en la cola de prioridad.

Atributos:

* private final Comparator<E> comparator: comparador utilizado para determinar la prioridad de los elementos en la cola de prioridad.
* private int size: cantidad de elementos en la cola de prioridad.
* private E[] elements: arreglo de elementos en la cola de prioridad.
* private static final int DEFAULT\_CAPACITY: capacidad por defecto de la cola de prioridad.