# PRÁCTICA 3

## Estructura de datos

Grupo 201 GII

María Guzmán Valdezate
Guillermo López de Arechavaleta Zapatero

# Contenido

# Introducción

En esta actividad se ha creado una estructura de datos personalizada basada en la interfaz Map, llamada MultiMapa, que permite vincular varios valores a una misma clave. Esta ejecución no solo facilita almacenar y recuperar listas de valores por clave, sino que también permite acceder a los elementos de manera cíclica y eliminarlos siguiendo el orden FIFO (Primero en entrar, primero en salir).

El objetivo de este documento es evaluar el diseño y la complejidad algorítmica de los métodos utilizados en MultiMapa, explicando su eficacia y comportamiento frente a tareas comunes como insertar, acceder y eliminar. También se analiza cómo el uso correcto de estructuras de datos especializadas como los multimapa puede mejorar tanto el rendimiento como la claridad en la solución de problemas computacionales.

## Descripción y Análisis de Métodos

**entrySet ()**

Descripción: Devuelven el conjunto de pares clave-valor.

- Complejidad algorítmica: O(n), ya que se recorren todas las listas para obtener el Set de los valores.

**put (K key, V value)**

Descripción: Añade un valor a la lista asociada a la clave. Si la clave no existe, se crea una nueva lista.

- Complejidad algorítmica: O(1) en promedio, gracias al acceso constante al mapa y al uso de ArrayList. Pero en el peor de los casos, si hay colisiones, es O(n).

**putAllMappings (K key, List<V> values)**

Descripción: Inserta todos los valores de una lista en la clave especificada.

- Complejidad algorítmica: O(n) en total, siendo n el tamaño de la lista, ya que el acceso a la clave es O(1) y cada inserción es O(1).

**getAllMappings (Object key)**

Descripción: Devuelve todos los valores asociados a una clave como una lista.

- Complejidad algorítmica: O(1) en promedio, acceso directo a través del mapa. Si hay colisiones es O(n).

**removeAllMappings (Object key)**

Descripción: Elimina todos los valores asociados a una clave.

- Complejidad algorítmica: O(1) en promedio, ya que HashMap.remove() opera en tiempo constante y la actualización del tamaño depende de la longitud de la lista. En el peor caso es O(n), cuando hay muchas colisiones.

**clear ()**

Descripción: Elimina todos los pares clave-valor del multimap, vaciando tanto el mapa principal como el de índices de acceso.

Complejidad algorítmica: Es O(n), ya que la operación clear() en un HashMap debe recorrer todas las entradas para eliminarlas.

**get (Object key)**

Descripción: Devuelve el siguiente valor asociado a una clave, de forma cíclica.

- Complejidad algorítmica: O(1), ya que el acceso al índice y a la lista es constante. En el peor de los casos, si hay colisiones, es O(n).

**remove (Object key)**

Descripción: Elimina el primer valor (FIFO) de la lista asociada a la clave.

- Complejidad algorítmica: O(n) en el peor caso, ya que ArrayList.remove(0) requiere desplazamiento de elementos.

**containsKey (Object key)**

Descripción: Verifican si existe una clave dentro del multimapa.

- Complejidad algorítmica: O(n*m), donde n es el número de claves y m es el tamaño de las listas. El método lista.contains en el peor de los casos tiene que recorrer toda la lista, si no encuentra el elemento habrá recorrido toda la lista.

**containsValue (Object value)**

Descripción: Verifican si algún valor está presente.

- Complejidad algorítmica: O(n) donde n es el número total de valores almacenados en todas las listas asociadas.

**values ()**

Descripción: Devuelven una colección de todos los valores.

- Complejidad algorítmica: O(n), ya que se recorren todas las listas para obtener la colección con los valores.

**size ()**

Descripción: Devuelve el número total de valores almacenados en el multimap (no el número de claves, sino la suma de todos los elementos de todas las listas asociadas).

- Complejidad algorítmica: O(1), ya que se mantiene una variable tamano actualizada durante las operaciones de inserción y eliminación.

**isEmpty ()**

Descripción: Indica si el multimap está vacío, es decir, si no contiene ningún valor asociado a ninguna clave.

- Complejidad algorítmica: O(1), ya que se verifica directamente si el tamaño total es igual a cero.

## Conclusiones

El avance en el desarrollo del MultiMapa ha facilitado un uso más profundo de mapas y listas en Java. Al permitir que haya varios valores por clave, esta estructura se ajusta a situaciones donde se necesita una relación uno-a-muchos, como en agrupaciones, vínculos o registros.

Se ha comprobado que, en líneas generales, los métodos presentan una eficiencia aceptable, siendo la mayoría O(1) o O(n) según el tamaño de las listas. Sin embargo, se ha encontrado un caso particular en la eliminación FIFO (remove(Object key)), que puede llegar a O(n) debido a la estructura de ArrayList. Este problema se podría solucionar utilizando una estructura diferente como LinkedList.

La implementación de un índice de acceso cíclico ofrece una utilidad interesante, ideal para sistemas que necesitan rotar tareas o acceder a datos de manera secuencial y repetida.

En resumen, esta experiencia ha mostrado cómo elegir bien las estructuras y hacer un análisis detallado de su comportamiento algorítmico puede llevar a soluciones sólidas, adaptativas y eficaces para una variedad de problemas.