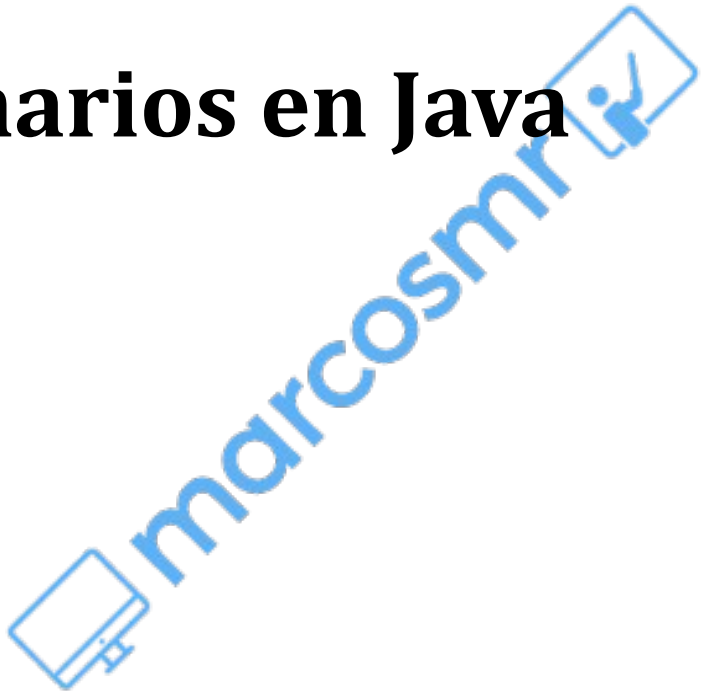
Diccionarios en Java



ÍNDICE

1. Introducción

Los diccionarios son estructuras de datos que nos van a permitir almacenar pares de clave-valor, es decir, dada una clave podemos recuperar el valor asociado a dicha clave (similar a lo que hacemos con el índice de una lista). Por ejemplo, que dado un DNI (valor entero) podamos recuperar el nombre y apellidos de una persona (cadena de texto).

Para definirlos se utiliza la interfaz `Map<K,V>`. En este caso se trabaja con dos clases una que se utiliza como clave (K) y otra para almacenar los valores (V). La idea es que cada elemento se almacena mediante un par de objetos (K,V). Esta estructura de datos nos permite obtener el objeto V muy rápidamente, a partir de su clave K. Esta interfaz no puede contener claves duplicadas y cada una de dichas claves, únicamente puede tener asociado un valor como máximo.

Al igual que sucede con las colecciones, en Java tenemos que trabajar con las clases *wrapper* si deseamos manejar tipos primitivos ya que lo que almacenan son objetos (que deben ser de una determinada clase). Por ello, en el caso de querer manejar tipos de datos primitivos debemos hacer uso de las clases `Integer`, `Float`, `Boolean`, etcétera.

Nosotros, para el trabajo con diccionarios nos vamos a centrar en el uso de `HashMap`. A continuación se indican una serie de métodos pero hay muchos más… ¡investiga!.

2. Declaración e inicialización

El trabajo con diccionarios en Java no se limita a la interfaz `HashMap`. No obstante, es la más utilizada y, por ello, nos centraremos en ella.

Para declarar diccionarios en Java podemos utilizar la siguiente sintaxis:

```
Map<tipoClave, tipoValor> diccionario = new HashMap<tipoClave, tipoValor>();
```

Para poder trabajar con diccionarios, en Java, es necesario importar el paquete `java.util` que es donde se encuentra la interfaz `Map` (entre otras), como ya sabemos.

Por ejemplo, para implementar el ejemplo anterior sobre DNI y nombre completo de una persona, podemos hacer uso del siguiente código:

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

3. Métodos

A continuación, vamos a ver los principales métodos de la interfaz `Map<K,V>` que nos van a permitir trabajar con los diccionarios. Los métodos que más vamos a usar son los siguientes:

- `V` **`put`**`(K key, V value)`. Añade un nuevo par clave-valor al diccionario.
- `V` **`get`**`(Object key)`. Da el valor asociado a una clave o **`null`** si no se encontró.
- `V` **`remove`**`(Object key)`. Elimina el par clave-valor que corresponde a la clave.
- `boolean` **`containsKey`**`(Object key)`. Comprueba si está la clave especificada.
- `boolean` **`containsValue`**`(Object value)`. Comprueba si está el valor.
- `Set` **`keySet`**`()`. Devuelve un conjunto con las claves contenidas en el diccionario.
- `Collection` **`values`**`()`. Devuelve una colección con únicamente los valores.
- `Set` **`entrySet`**`()`. Devuelve un conjunto de clave-valor contenidas en el diccionario.
- `boolean` **`isEmpty`**`()`. Comprueba si la colección está vacía.
- `int` **`size`**`()`. Devuelve el número de elementos que contiene la colección.
- `void` **`clear`**`()`. Elimina todos los elementos de la colección.

4. Tipos de Map

Podemos utilizar diferentes implementaciones para el uso de diccionarios haciendo uso de la interfaz `Map`. Las siguientes clases implementan la interfaz `Map`:

4.1 HashMap

Con esta implementación, las claves se almacenan en una tabla *hash* y, por ello, conseguimos las siguientes características:

- Tiene el mejor rendimiento de todas.
- No garantiza ningún orden a la hora de realizar iteraciones (bucles).
- Proporciona tiempos constantes en las operaciones básicas siempre y cuando la función *hash* disperse de forma correcta los elementos dentro de la tabla hash.

4.2 TreeMap

Con esta implementación conseguimos almacenar las claves ordenadas en función de sus valores.

- Es bastante más lento que `HashMap`.
- Garantiza el orden.
- Las claves almacenadas deben implementar la interfaz `Comparable` (ahora mismo no sabemos lo que es).

4.3 LinkedHashMap

Esta implementación almacena las claves en función del orden de inserción (tal y como sucedía en las listas) y es un poco más costosa que `HashMap`.

5. Sincronización

Aunque para nosotros no es importante, dado nuestro estado actual de aprendizaje, os quiero avisar que ninguna de estas implementaciones son sincronizadas; es decir, no se garantiza el estado del `Map` si dos o más hilos acceden de forma concurrente al mismo.

Esto se puede solucionar empleando una serie de métodos que actúan de `wrapper` para dotar a estas colecciones de esta falta de sincronización:

- `Map<K,V> map = Collections.synchronizedMap(new HashMap<K,V>());`
- `SortedMap<K,V> sortMap = Collections.synchronizedSortedMap(new TreeMap<K,V>());`
- `Map map<K,V> = Collections.synchronizedMap(new LinkedHashMap<K,V>());`