HashMap en Java

Los HashMap son estructuras de datos que permiten almacenar y procesar información mediante una clave y un valor, a diferencia de estructuras como las listas estos no gestionan la información basados en un indice sinó que lo hacen mediante un numero hash que genera indices aleatorios y no en orden de llegada.

Se relaciona con un diccionario en el que se tiene un código (palabra) y un valor (significado), a diferencia de elementos similares como el **hashtable** este si puede almacenar valores null, un hashMap no es sincronizado permitiendo el acceso de más de una tarea al mismo tiempo, al ser no sincronizado se tienen mejoras en rendimiento a nivel de tiempo de acceso.

Video referente: https://www.youtube.com/watch?v=TX5Sucd1CRA&list=PLAg6Lv5BbjjfOJJN2Wla7_Axoynrt0i3b_&index=6

DESARROLLO DEL LABORATORIO

Uso de HashMap

para el desarrollo de esta guía se debe crear un proyecto simple y una clase con el método main con la siguiente estructura.

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

mapaNombres.put(123, "Cristian");
mapaNombres.put(3, "maria");
mapaNombres.put(13, "Pepe");
mapaNombres.put(23, "Camila");

System.out.println(mapaNombres);

System.out.println(mapaNombres.keySet());
System.out.println(mapaNombres.values());

System.out.println(mapaNombres.get(123));
```

Vemos que se crea el mapa Nombres el cual es declarado con un tipo de clave Integer que permite valores de tipo String (**como clave no se pueden usar datos primitivos**)

y vemos también que para agregar elementos hacemos uso del método put() con la estructura (clave, valor)

posteriormente vemos como imprimir el mapa completo o las claves y los valores de forma independiente o el método get para obtener un valor especifico.

```
{3=maria, 23=Camila, 123=Cristian, 13=Pepe}
[3, 23, 123, 13]
[maria, Camila, Cristian, Pepe]
Cristian
```

Como estamos haciendo uso de estructuras de datos tenemos a nuestra disposición una cantidad de funciones que podemos utilizar, ya vimos como agregar elementos, obtener un elemento, obtener la lista de claves o la lista de valores, también podríamos eliminar un dato en específico o todos los datos, así como consultar datos específicos sin tener que consultar uno a uno los elementos.

En este caso agregamos mapaNombres.clear() lo que permitirá eliminar todos los datos de la lista y si consultamos en el if si el mapa contiene el valor “**camilas**” en este punto el resultado será lo que imprima en el else, pero si no limpiamos la lista entonces se presentaría la información sin problema.

```java
mapaNombres.put(123, "Cristian");
mapaNombres.put(3, "maria");
mapaNombres.put(13, "Pepe");
mapaNombres.put(23, "Camila");

System.out.println(mapaNombres);

System.out.println(mapaNombres.keySet());
System.out.println(mapaNombres.values());

mapaNombres.clear();

if (mapaNombres.containsValue("Camilas")) {
    System.out.println("SI CONTIENE A CAMILA");
}else{
    System.out.println("NO CONTIENE A CAMILA");
}
```

Al imprimir tendríamos el siguiente resultado.

```
{3=maria, 23=Camila, 123=Cristian, 13=Pepe}
[3, 23, 123, 13]
[maria, Camila, Cristian, Pepe]
{}
NO CONTIENE A CAMILA
```

Para finalizar podemos recorrer todos los elementos del mapa haciendo uso de un iterador que generará los indices con el orden necesario para recorrer el mapa usando el método hasNext()

este proceso lo hicimos modificando el código anterior y creando un método específico para imprimir el mapa, teniendo la clase completa así

```java
import java.util.HashMap;
import java.util.Iterator;

public class Main {

    public static void main(String[] args) {
        // Crear el HashMap
        HashMap<Integer, String> mapaNombres = new HashMap<>();

        // Agregar los valores al mapa
        mapaNombres.put(123, "Cristian");
        mapaNombres.put(3, "Maria");
        mapaNombres.put(13, "Pepe");
        mapaNombres.put(23, "Camila");
        mapaNombres.put(132, null);

        // Llama al método que imprime el mapa
        ImprimirMapa(mapaNombres);
    }

    /* Método para imprimir el mapa usando un Iterator sobre las claves*/
    public static void ImprimirMapa(HashMap<Integer, String> mapa) {
        // Obtiene un iterator sobre las claves del mapa
        Iterator<Integer> iterator = mapa.keySet().iterator();

        // Recorre el mapa usando el iterator sobre las claves
        while (iterator.hasNext()) {
            Integer llave = iterator.next();  // Obtener la clave
            String valor = mapa.get(llave);   // Obtener el valor asociado a la clave
```

```
            // Imprime la clave y el valor asociado
            System.out.println(llave + " - " + valor);
        }
    }
}
```

teniendo el siguiente resultado.

```
{3=maria, 23=Camila, 123=Cristian, 132=null, 13=Pepe}
3 - maria
23 - Camila
123 - Cristian
132 - null
13 - Pepe
```

Aquí usamos `Iterator<Integer>` para recorrer las claves del `HashMap`. El `Iterator` se obtiene con `mapaNombres.keySet().iterator()`, que devuelve un conjunto de las claves.

En cada iteración del bucle `while`, obtenemos una clave (`Integer llave = iterator.next();`), y luego usamos esa clave para obtener el valor asociado en el `HashMap` mediante `mapa.get(llave)`.

Formas de recorrer e imprimir un HashMap

En Java, un `HashMap` es una estructura de datos muy útil que almacena pares clave-valor. Sin embargo, una vez que tienes datos en un `HashMap`, a menudo querrás imprimir o recorrer esos datos de manera eficiente. Hay **varias formas** de hacerlo, y la razón principal de esta diversidad es que Java proporciona múltiples enfoques para acceder a los elementos del mapa, dependiendo del caso de uso específico.

Cada método tiene sus propias ventajas, dependiendo de los requisitos de rendimiento, claridad del código y control sobre el proceso de iteración.

Las diferentes formas de imprimir un `HashMap` surgen de los diferentes mecanismos que Java ofrece para acceder a las claves, valores o pares clave-valor. Algunas razones clave son:

- A veces solo necesitas acceder a las **claves**.
- En otros casos, necesitas tanto las claves como los **valores** (pares clave-valor).
- También puede ser necesario obtener los valores basados en las claves, o iterar y modificar los elementos al mismo tiempo.
- Algunos métodos, como los basados en **Iterator**, te permiten tener un mayor control sobre la iteración, por ejemplo, para eliminar entradas durante el recorrido.
- Otros métodos, como los ciclos `for-each`, son más simples y directos, pero no permiten modificaciones.
- A partir de **Java 8**, se introdujeron características más avanzadas como expresiones **lambda** y el método `forEach()` para proporcionar una forma más concisa y funcional de recorrer colecciones.
- Algunas formas, como recorrer el `entrySet()` directamente, son más eficientes en términos de rendimiento, ya que no requieren múltiples búsquedas dentro del mapa.
- Otras formas, como usar un `Iterator` sobre `keySet()` y luego buscar los valores con `get()`, son más flexibles, pero pueden ser menos eficientes.

Veamos las formas:

1. **Usando un `for-each` con `entrySet()`:**
    - Este enfoque recorre el `entrySet` del mapa, lo que te permite obtener directamente tanto la clave como el valor de cada entrada.
    - Ventaja: Fácil de leer y usar, y es una forma directa de obtener clave-valor.

```
// 1. Usar un for-each para recorrer el entrySet del mapa
 public static void imprimirConForEach(HashMap<Integer, String> mapa) {
    System.out.println("Imprimiendo con for-each (entrySet):");
    for (Map.Entry<Integer, String> entry : mapa.entrySet()) {
       System.out.println("Clave: " + entry.getKey() + ", Valor: " + entry.getValue());
    }
 }
```

2. **Usando un `Iterator` con `entrySet()`**:
   - En este caso, se usa un `Iterator` sobre el `entrySet` del `HashMap` para recorrer cada par clave-valor.
   - Ventaja: Permite más control en caso de que necesites eliminar elementos mientras iteras.

```
// 2. Usar un iterator sobre el entrySet del mapa
 public static void imprimirConEntrySetIterator(HashMap<Integer, String> mapa) {
    System.out.println("\nImprimiendo con Iterator (entrySet):");
    Iterator<Map.Entry<Integer, String>> iterator = mapa.entrySet().iterator();
    while (iterator.hasNext()) {
       Map.Entry<Integer, String> entry = iterator.next();
       System.out.println("Clave: " + entry.getKey() + ", Valor: " + entry.getValue());
    }
 }
```

3. **Usando un `Iterator` sobre `keySet()` y obteniendo los valores manualmente**:
   - Este enfoque itera sobre las claves (`keySet()`) del mapa y luego obtiene los valores usando `get()`.
   - Ventaja: Útil si solo necesitas las claves y ocasionalmente los valores.

```
// 3. Usar un iterator sobre el keySet del mapa y obtener los valores manualmente
    public static void imprimirConKeySetIterator(HashMap<Integer, String> mapa) {
       System.out.println("\nImprimiendo con Iterator (keySet):");
       Iterator<Integer> iterator = mapa.keySet().iterator();
       while (iterator.hasNext()) {
          Integer key = iterator.next();
          String value = mapa.get(key);
          System.out.println("Clave: " + key + ", Valor: " + value);
       }
    }
```

4. **Usando la función `forEach` de Java 8**:
   - Esta es una forma moderna y concisa de recorrer un `HashMap` usando la funcionalidad de `forEach` introducida en Java 8.
   - Ventaja: Sintaxis compacta, ideal para recorridos rápidos y operaciones lambda.

```
// 4. Usar la función forEach de Java 8 para imprimir el mapa
public static void imprimirConForEachJava8(HashMap<Integer, String> mapa) {
   System.out.println("\nImprimiendo con forEach de Java 8:");
   mapa.forEach((key, value) -> {
      System.out.println("Clave: " + key + ", Valor: " + value);
   });
 }
```

Conclusión:

- **Elegir la técnica adecuada** depende de tus necesidades. Si solo quieres un recorrido sencillo, `for-each` o `forEach` de Java 8 es la mejor opción. Si necesitas más control o eliminar elementos mientras iteras, los `Iterator` son una mejor opción.

- Cada método en este ejemplo es independiente y te permite usar el que más se ajuste a tus necesidades en diferentes situaciones.

Instructor: Cristian David Henao Hoyos