Lesson:

Hashmap in JAVA

Pre Requisites:

- Basic Java Syntax

List of concepts involved :

- What are hashmaps
- Various functions of hashmap
- Collision in Hashmap
- Types of hashmap
- Two Sum problem
- First unique character in String

What are Hashmaps

First of all let's understand Maps.Maps is a interface in Java which is present in java.util.* package represents a key and value pair in java. A key should always be unique in the map.

Maps can be used on various occasions where we need to have a key -value sort of relationship. Various examples of map usage are.

- Dictionaries having key value relationships. Word is key and meaning is value
- A map of zip codes and cities. Cities are key and list of zip codes are values.
- A map of class of students. Class is a key and list of students are values.

Now HashMap<K,V> is a part of Java's collection. The class is also found in java.util.* package. It provides the basic implementation of the map interface in java. It stores the data in (key, value) pairs, and you can access them by a index of key. A hashmap always contains unique key. If we try to insert a duplicate key, then it replaces the old key. Hashmap also allows us to store NULL keys. But it only contains one NULL key. If we try to insert another, it will update the value of previous null key.

Syntax to declare Hashmap in Java

HashMap<K,V> hashmap = new HashMap<>();

K,V can be of any data structure. For eg.

HashMap<Integer, String> hashmap = new HashMap<>();

Various functions of HashMap

put()

Put is used to insert elements into the hashmap. We need to specify the arguments of key, value type and it will insert into the hashmap.

For eg.

hashmap.put(1, "Piyush");
hashmap.put(2, "Raghav");

Now map contains: {1=Piyush, 2= Raghav}
If we again update put the same key in the map, it will update the value of hashmap key.

hashmap.put(2, “Akash”);
Now map contains: {1=Piyush, 2= Akash}

get()

get() is used to get the value of a particular key inside the map. If the map doesn’t contains the key it will return null as the output.

For eg,

String answer = hashmap.get(2);

gives us the output in String as Akash.

remove()

In order to remove a element from hashmap we use remove function. We provide key as the argument to the function and it removes the key-value pair from the map if it is present in the map. For eg.

hashmap.remove(1);

Now map contains: {2= Akash}

entrySet()

To traverse a complete map we use the entrySet function. It helps in traversing over the map in order of keys.

For eg.

For the previous created hashmap,

```
for (Map.Entry<Integer, String> e : hashmap.entrySet()) {
        System.out.println(“Key :”+ e.getKey() + “, value: ”+ e.getValue());

}
```

Now we have created an iterator e which will traverse every key of map. When we use getKey(), it gives us the element which is acting as a key in map and similarly when we use .getValue(), it gives the value stored for the corresponding key.

containsKey()

This function is used to check if map contains a following key or not. We pass the key inside this function and it will return true if key is present in map.

For eg.

Boolean result = hashmap.containsKey(1);

Code: LP_code1.java

Output

```
Value of Map is: {1=Piyush, 2=Athar, 3=Ajay, 4=Anil}
Value of Map is: {1=Rajesh, 2=Athar, 3=Ajay, 4=Anil}
Value of key = 3 is: Ajay
Value of Map is: {1=Rajesh, 3=Ajay, 4=Anil}
Key: 1 Value: Rajesh
Key: 3 Value: Ajay
Key: 4 Value: Anil
```

Collision in Hashmap

What is the Hash function?

A hash function is basically used to map a data of large size into a data of fixed size. The values returned by a hash function are called hash values.
The hash values are used to store keys in the map.

For eg a simple hash function could be

```
Integer hashFunction(Integer key) {
        return key%10;
}
```

Now if you pass any sort of data to this hash function it will return modulus 10. So all the hash values will be in order of 0-9.

What is collision

When a hash function starts returning same hash value for more than one key, then it results in collision. for eg for the above hash function, if we pass 10, 20 to the hash function it will return as 0 in both cases. Now we won't be able to store value for different keys as both are being mapped to same hash key.So it's very necessary to have a strong hash function.

How to handle collision

- **Separate Chaining.**
  a. Separate Chaining with linked lists.
- **Open addressing**
  a. Linear Probing
  b. Quadratic probing
  c. Double hashing.

Separate Chaining

Separate chaining is used when for the same hash value we start storing the keys in a chain. The chain can be of any type. Generally Linked list is preferred as the type of chain.

Now for eg. for they following keys 31,33,12, 13 ,10,20,30 the hashing would look like

{
0 -> [10]->[20]->[30]
1 -> [31]
2 -> [12]
3 -> [33,]->[13]

}

Now the time complexity would be to first find the hash key and then linearly search the list for that particular hash key.

Open Addressing

- **Linear probing**

In linear probing,

- When collision occurs, we linearly probe for the next bucket.
- We keep probing until an empty bucket is found.

Advantage-

- It is easy to compute.

Disadvantage-

- The main problem with linear probing is clustering.
- Many consecutive elements form groups.
- Then, it takes time to search an element or to find an empty bucket.

- **Quadratic Probing-**

In quadratic probing,

- When collision occurs, we probe for i2'th bucket in ith iteration.
- We keep probing until an empty bucket is found.

- **Double Hashing.**

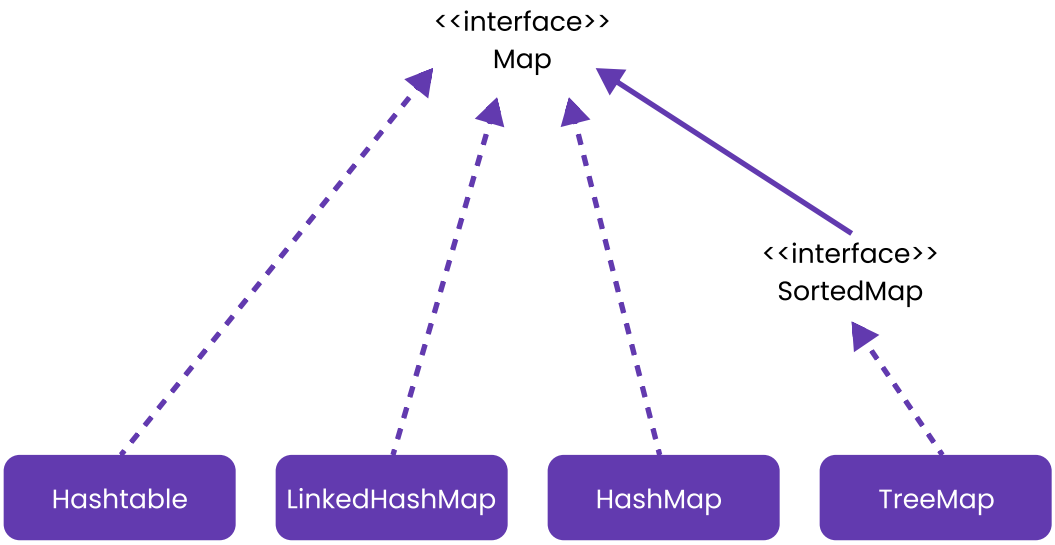
As the name suggests, to store a hash, we use a second hash function which acts as an offset and updates the current hash function.

The new hash function would somewhat look like

h(i,k) = (h1(k) + i* h2(k))%SOME_CONSTANT

So now we have hash function h1 and hash function h2 which acts as an offset and updates the key for the total hash.

Types of HashMap in Java



HashMap is implemented as a hash table, and there is no ordering on keys or values.

TreeMap is implemented based on red-black tree structure, and it is ordered by the key.

LinkedHashMap preserves the insertion order

Hashtable is synchronized, in contrast to HashMap. It has an overhead for synchronization.
This is the reason that HashMap should be used if the program is thread-safe.

Also concurrent hash map can be used if we want to achieve synchronization in a multi threaded environment

Now we will see an example of each type of Map in java and will see the output of map. The idea is to see the order of keys for different types of maps.

HashMap can store keys in any order.TreeMap sorts the keys and stores them in order.

LinkedHashMap preserves the order of insertion of keys.

Code: LP_Code2.java

```
Value of HashMap is: {1=Piyush, 2=Athar, 3=Ajay, 4=Anil}
Value of TreeMap is: {1=Piyush, 2=Athar, 3=Ajay, 4=Anil}
Value of LinkedHashMap is: {4=Anil, 2=Athar, 3=Ajay, 1=Piyush}
```

Interview problem: Two Sum

Ques: Given an array of integers and an integer target, return indices of the two numbers such that they add up to target. Assume only 1 valid answer exists.

Eg. Input = [2,7,11,15]. target =9
Output = [0,1]

Input: [3,2,4]. target = 6
Output: [1,2]

Code: LP_Code3.java

Output

```
Enter the number of elements you want to store: 5
Enter the elements of the array:
2 7 4 1 6
Enter the target element of the array:
9
The output array is: [0,1]
```

```
Enter the number of elements you want to store: 3
Enter the elements of the array:
3 2 4
Enter the target element of the array:
6
The output array is: [1,2]
```

Approach:

We would store every element in the hashmap in the form of (array[index], index). The idea is we will traverse the original array and will check in map if the remaining element i.e. target - currentElement is present in hashmap then we found our answer.
The answer would be currentIndex and index of the remaining element.

There can be 2 edge cases in this.

- What if we found exactly half of the target. Then to confirm we have one more element of the same value, we would check the currentIndex < map.get(target-currentElement). This condition would ensure that we have another element which is ahead of this as map would have already updated the index when we are entering the elements in the array.

Interview problem: First unique Character in String

Given a string s, find the first non-repeating character in it and return its index. If it does not exist, return -1.

eg. **Input:** s = "leetcode"
Output: 0

Input: s = "loveleetcode"
Output: 2

Input: s = "aabb"
Output: No Character is found: -1

Code: LP_code4.java

Output:

```
Enter the String: leetcode
The first non repeating character index is: 0
```

```
Enter the String: loveleetcode
The first non repeating character index is: 2
```

```
Enter the String: aabb
No character is found: -1
```

Approach:

We would create a hashmap of char, integer. Now for every character of String we would insert this into the hashmap and update the frequency of that element..

Now we would traverse the map and check the first character whose frequency is equal to 1. If we found any such element we found our answer, else we will return -1.

Next Class Teasers

- Stacks In Java
- Queue in Java