HashMap

Map 🡪 Collection of “Key-Value” Pairs.

Internal Working of a HashMap

1. Key = Unique identifier to which we map a value in a map.
2. Value = Something that is mapped to a unique key in a map.
3. Bucket = Basically an Array, where the key-value pairs get stored in the memory at runtime.
4. HashFunction = Basically a method/function which generates a hashCode for a given key passed as an input to it.

**Hash Function** – An algorithm which takes in an input of an arbitrary size and always returns an output of a fixed size known as HashCode or HashValue or simply Hash.

**Properties of a HashFunction:**

1)Deterministic: Must generate the same output for a given same input.

2)Fixed Output Size: Regardless of the input Size, the output of a HashFunction is always of a fixed size, typically 16 bits or 32 bits.

2)Computationally very Efficient.

How Data is Stored inside a HashMap?

1. Hashing the key

At First, the key is passed to a HashFunction which generates a fixed size HashCode which is later used to calculate the index of the bucket(array), in which the key-value pair would get stored.

1. Calculating the Index

Once the HashCode is computed in Step 1, the next step involves calculating the Index which is often calculated as follows:

**int index = hashCode%arraySize**

1. **Storing in the Bucket**

The next step is storing the key-value pair in the calculated index of the bucket(array). An index can store multiple key-value pairs. (this is called collision handling and is discussed later)

How HashMap Retrieves Data?

Data Retrieval from a HashMap involves the same process as storing a Data into it. First the key is hashed by the HashFunction to generate a HashCode. Then the HashCode is used to find the index at which the particular key-value pair is stored. Then finally once the correct index is found, it checks for the key and if the key is found, the associated value with the key is returned.

2 Important Questions to ask at this point of time,

Q1) What is actually stored in the bucket(array)?

Q2) If we are storing a key-value pair at a particular index of the bucket, then why are we searching?

Let’s start off with answering Q2

As already discussed, a HashMap is defined as an algorithm that generates an output of fixed size (within a set of finite numbers) known as HashCode, HashValue or Hash. Now it may so happen that, for two given inputs (keys), the HashFunction generates the same output. This condition is termed as **COLLISION** in hashing.

Example:

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

1) map.put(“apple”,20); // hashCode1

2) map.put(“orange”,30); // hashCode2

3) map.put(“grape”,40); // hashCode1

We can see 1 & 3 are having the same hashCode generated by the hashFunction. This means that both these entries are projected to be stored at the same index inside the bucket. Thus an index inside the bucket can contain multiple key-value pairs. Hence a data-structure is required to achieve this. This is usually done through a LinkedList Data Structure as follows.

(“apple”,20) -----------------🡪(“grape”,40)

This answers our Q1.

Further Search Optimization

Since we know searching in a LinkedList takes O(n) Time, in order to further optimize the searching, after Java 8, the LinkedList is converted to a Self-Balancing Binary Search Tree (mostly a Red-Black Tree) which taking O(logn) time for search operations just significantly reducing the search time.

Time Complexity

HashMap provides constant time O(1) performance for basic operations like put() and get()(assuming no collisions).

However, if there are too many collisions and too many entries are stored in the same bucket, the performance can degrade to O(n).  
  
But After Java 8, if there are too many elements in the bucket, HashMap switches to a balancing tree instead of a linked list to ensure better performance O(logn).

**Role of equals() and hashCode() method :**

As we have discussed that during a collision situation, hashmap internally uses a linked list or a self-balancing bst to store multiple key-value pairs. The comparison of the key with the searched key is done using the equals() method.

equals() and hashCode() method belongs to the Object class. hashCode() method normally returns a number by computing the memory address of a particular object while equals() method is used to compare the references of any two given objects.

This causes some functional ambiguity for keys of Custom DataTypes, which we shall look into shortly.

As demonstrated in the practical

**C:\JavaImportantTopics\JavaCollections\src\hashmaps\EqualsAndHashMethod.java** , it is always recommended to override the equals() and hasCode() method for such scenarios.