**How HashMap Internally Works in Java**

[How HashMap works internally || Popular java interview question on collection (HashMap)](https://www.youtube.com/watch?v=CojCE-ojdGY)

[How HashMap Internally Works in Java With Animation | Popular Java Interview QA | Java Techie](https://www.youtube.com/watch?v=1CJbB6SzjVw)

* When we create a HashMap,

it internally creates a bucket like structure with initial capacity 16, i.e., 0-15 buckets

|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |

* Each bucket internally has a linked list & this each node of the Linked list contains Key, Value, Hash & Next

Map<Employee, String> empMap = new HashMap<>();

Employee emp1 = new Employee(101, “Mahesh”);

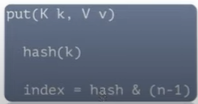
Employee emp2 = new Employee(102, “Varun”);

Employee emp3 = new Employee(103, “Bharath”);

Employee emp4 = new Employee(104, “Akhil”);

* We have a map and few emp objs.

EmpMap.put(emp1, “Employee 1”);



* The put Method has two tasks:
  + Calculates hashCode w.r.t key.
  + Create index using hash value and ‘&’ modular operator, index = hashCode & (n-1);
* If the hash is 1011 and index is 6, it stores in the manner below. The 4th value is null, because at 6th index position, the Linked list contains only one node.

|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6   |  |  |  |  | | --- | --- | --- | --- | | emp1 | Employee1 | 1011 | null | |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |

* Let's add emp2, EmpMap.put(emp2, “Employee 2”); say emp2 has hash 2345 and index as 8.

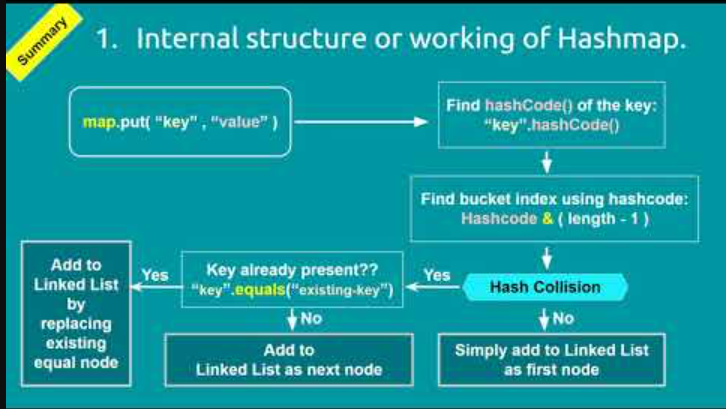
|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6   |  |  |  |  | | --- | --- | --- | --- | | emp1 | Employee1 | 1011 | null | |
| 7 |
| 8   |  |  |  |  | | --- | --- | --- | --- | | emp2 | Employee2 | 2345 | null | |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |

* Let's add emp3, EmpMap.put(emp3, “Employee 3”); say emp3 has hash 7976 and index as 6.
  + If the index is different, simply add it.
  + As index is same as existing one(emp1), this is called Hash Collision, Here HashMap won’t directly add/override the new entry.
    - It checks for the equals method, emp1.equals(emp3) == true ?
    - If true, HashMap replaces the existing node with newone.
  + If false, it appends to the existing node in LinkedList. The next of the first node gets changed and adds a pointer to the newly added node.

This is for false condition.

|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6   |  |  |  |  | | --- | --- | --- | --- | | emp1 | Employee1 | 1011 | Links to next node |  |  |  |  |  | | --- | --- | --- | --- | | emp3 | Employee3 | 7976 | null | |
| 7 |
| 8   |  |  |  |  | | --- | --- | --- | --- | | emp2 | Employee2 | 2345 | null | |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |
| 14 |
| 15 |

* If the key is null, it directly goes to 0th index position. As only one null key can be accommodated, there won't be any further discussion. If a second null key is added, as usual the value will be overridden.
* Enhancement to HashMap in Java8.
  + if a hash collision in any bucket occurs the threshold no. of times, the LinkedList gets converted to a balanced tree.



* Re-iterating something obvious -- If the hashCode is same for two objects, it goes to same index
  + If the equals is true, then it over- rides the value
  + Else, same object is added to the linked list as key, but the value will be different.
  + However in the second case, retrieving the value will be complex. Hence we usually follow the equals and hashCode contract, i.e., if two objects are equal as per equals method, we have to override hashCode() as well and they should return the same hashCode.
* After all these, the get method should not seem difficult.

Map.get(key) ---cals hash, finds index

-- only one node, return the value in that node.

-- multiple nodes ? Then iterate over the linked list. Returns the value of the node whose hash is same as the hash of the key.