How Hashmap data structure works internally? How hashcode and equals method work in hashmap? Explain with real time example.

IN JAVA - ON 12:28:00 - NO COMMENTS

Hashmap data structure works internally?
How hashcode and equals method work in hashmap?

This is the famous interview question for the beginners as well as for experienced, So Let's see what it is all about.

Hashmap is very popular data structure and found useful for solving many problems due to O(1) time complexity for both get and put operation. Before getting into Hashmap internals, Please read Hashmap basics and Hashcode.

Internal working of Get and Put operation.

Hashmap store objects in key-value pair in a table.

1. Objects are stored by method **hashmap.put(key, value)** and

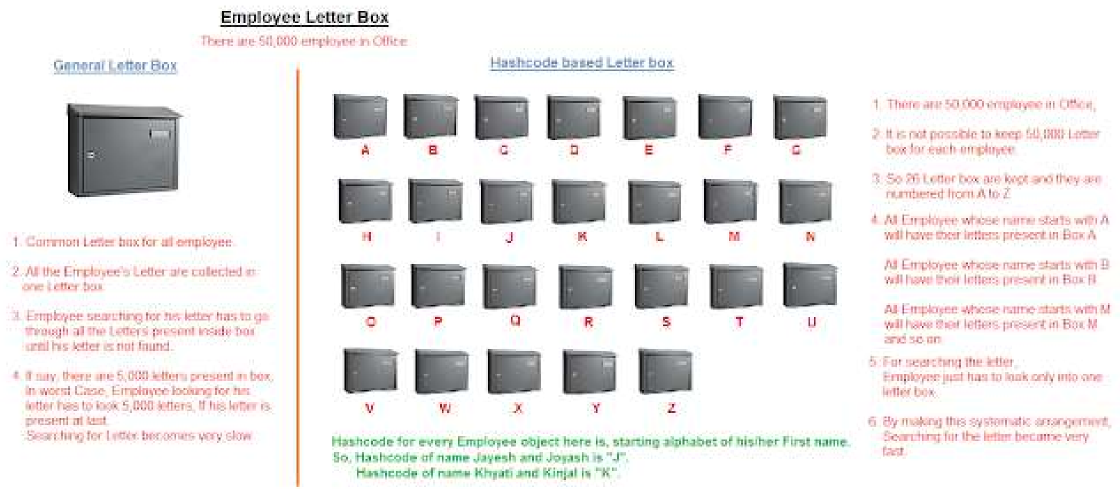
2. Objects are retrieved by calling **hashmap.get(key)** method.

For detail explanation on hashmap get and put API, Please read this post How Hashmap put and get API works.

Put Operation

Hashmap works on principle of hashing and internally uses hashcode as a base, for storing key-value pair. With the help of hashcode, Hashmap stores objects and retrieves it in constant time O(1).

Lets recap **"Employee Letter Box" example**, we saw in last post on Hashcode.



There are 50,000 employee in Office

1. Common Letter box for all employee.
2. All the Employee's Letter are collected in one Letter box
3. Employee searching for his letter has to go through all the Letters present inside box until his letter is not found.
4. If say, there are 5,000 letters present in box, In worst Case, Employee looking for his letter has to look 5,000 letters, If his letter is present at last. Searching for Letter becomes very slow.

Hashcode for every Employee object here is, starting alphabet of his/her First name.
So, Hashcode of name Jayesh and Joyash is "J".
Hashcode of name Khyati and Kinjal is "K".

1. There are 50,000 employee in Office,
2. It is not possible to keep 50,000 Letter box for each employee
3. So 26 Letter box are kept and they are numbered from A to Z
4. All Employee whose name starts with A will have their letters present in Box A

   All Employee whose name starts with B will have their letters present in Box B

   All Employee whose name starts with M will have their letters present in Box M and so on
5. For searching the letter, Employee just has to look only into one letter box.
6. By making this systematic arrangement, Searching for the letter became very fast.

From above letter box example we can say,

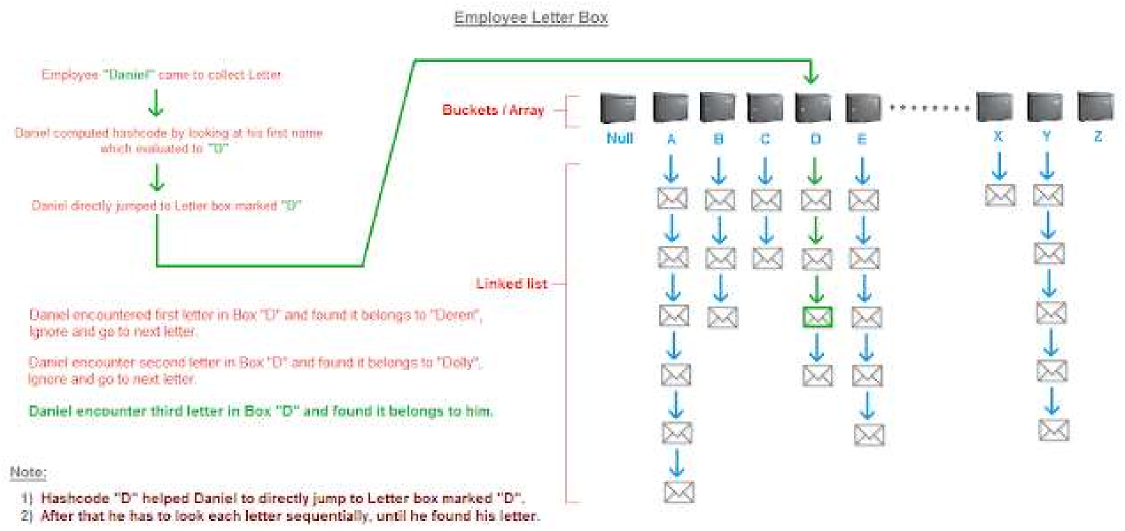
With the help of hashcode(start alphabet of First name), all letters are distributed across several letter box and due to which searching for the particular Letter within Letter box became very fast.

Employee just need to look into appropriate Letter box and search letter only within that box, ignoring other boxes. This make searching for Letter very fast.

Bit Manipulation Database
Datastructure Design Pattern
Docker Graph hashmap
hashtable Heap Interviews Java
Java8 Javascript Linked List
Matrix Miscellaneous Multithreading
N-ary-tree OOPS PriorityQueue
Puzzle Quartz Queue Recursion
Sorting Spring Stack Stream
Strings Typescript Web Service
Windows

> *Hashcode is basically used to distribute the objects systematically, so that searching can be done faster.*

Let's try to shape Letter box example in way Hashmap works.



Employee "Daniel" came to collect Letter

Daniel computed hashcode by looking at his first name which evaluated to "D"

Daniel directly jumped to Letter box marked "D"

Daniel encountered first letter in Box "D" and found it belongs to "Deren", Ignore and go to next letter.

Daniel encounter second letter in Box "D" and found it belongs to "Dolly", Ignore and go to next letter.

Daniel encounter third letter in Box "D" and found it belongs to him.

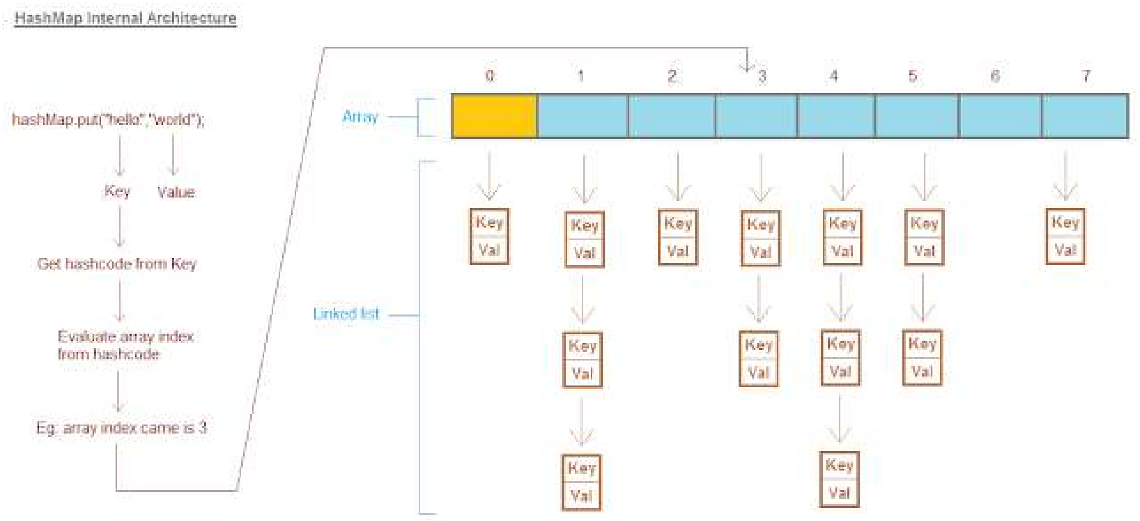
Note:

1) Hashcode "D" helped Daniel to directly jump to Letter box marked "D".
2) After that he has to look each letter sequentially, until he found his letter.

Hashmap works very similar to above Employee Letter box example.

1. It make use of **array** in place of Letter Boxes.
2. It make use of **linked list** for storing Key-Value pair and each Node of linked list corresponds to mail in above example.

Hashmap uses Array and Linked list datastructure internally for storing key-value pair.


HashMap Internal Architecture
hashMap.put("hello","world");
Key
Value
Get hashcode from Key
Evaluate array index from hashcode
Eg: array index came is 3
Array
Linked list
0
1
2
3
4
5
6
7
Key
Val

Lets try to put below key-value pair in hashmap,

1. hashMap.put("**hello**","**world**");
2. hashMap.put("**jayesh**","**patel**");
3. hashMap.put("**khyati**","**patel**");
4. hashMap.put("**khyati**","**pokar**");

Before going into details, Let understand what is hashcode,

In Employee Letter box example, hashcode of Employee is "First character of Name".

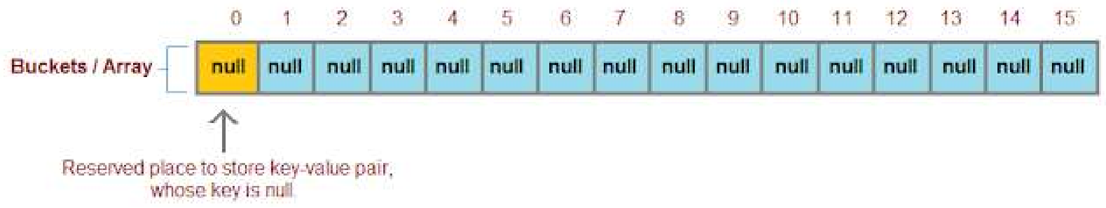
In case of Object, hashcode is a Integer value which represents object.

hashcode is calculated using hashcode() method and which uses object properties to compute it.

Please refer this link for detail explanation on Details on Hashcode

Note:

1. Default Hashmap size is 16. It means, default array size is 16, starting from index 0 to 15, like shown below,


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Buckets / Array
null null null null null null null null null null null null null null null null
Reserved place to store key-value pair, whose key is null

2. Let's try to put key-value pair hashmap.put("hello","world");
Above line says, **value "world"** needs to be stored against **key "hello".**

Step 1 : Hashmap will compute the hashcode of key "hello",
Step 2 : Say, hashcode of key "hello" computed to integer value "12345".

[In Employee Letter Box example, hashcode was "first alphabet of employee name", So employee directly jumps to corresponding Letter box as he/she was aware that Letter box with same alphabet will be present.]

In case of Hashmap, after computing hashcode 12345, we can not jump directly to array index 12345, because array size is 16(from 0 to 15) and index 12345 is not present.

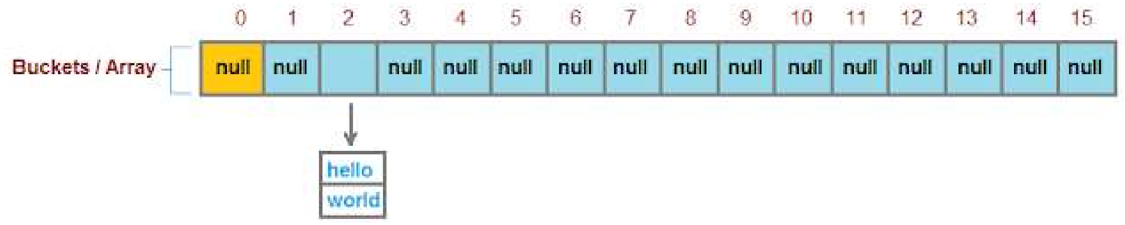
So, hashcode 12345 need to be converted into number between 0 to 15 to put it in array. So here comes the use of **indexFor()** method.

indexFor(hash, table.length) is used to calculate exact index in table array for storing the Key-Value pair.

Step 3: So, hashcode is further computed to get array index also known as Bucket in hashmap terminology. Once, bucket is known, Key-Value pair is placed in that bucket.

Lets say bucket evaluated from hashcode 12345 is 2,

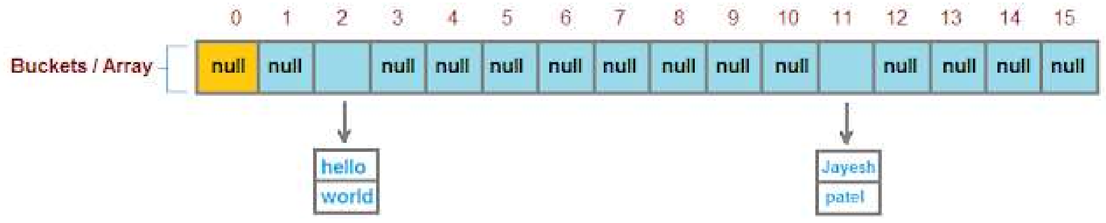
After placing object in hashmap, it will look like below.



3. Let's try to put key-value pair hashmap.put("jayesh","patel");

Step 3 will be repeated, hashcode of key "jayesh" is computed and let's say hashcode evaluated is 450, and bucket or index of array evaluated is 11.

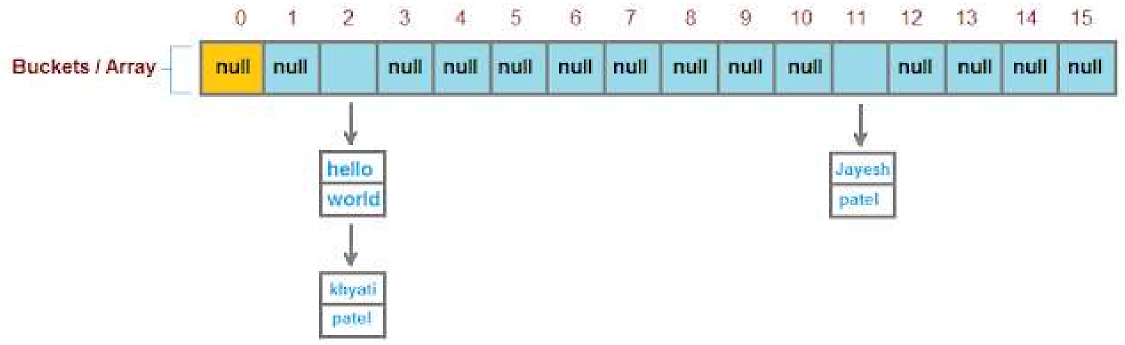
After placing object in hashmap, it will look like below.



<u>Note:</u>
Items that have the same hashcode will fall in same bucket,
So if more than one item is falling in the same bucket then those items will be stored in Linked list like shown below.

4. Let's try to put key-value pair hashmap.put("khyati","patel");
Step 3 will be repeated, hashcode of key "khyati" is computed and let's say hashcode evaluated is 1200, and bucket or index of array evaluated is 2.

After placing object in hashmap, it will look like below.



5. Let's try to put key-value pair hashmap.put("khyati","pokar");
Step 3 will be repeated, hashcode() method will be called to compute hashcode of "khyati".

In Step 4, hashcode evaluated for "khyati" gave 1200, then second time if you calculate hashcode of "khyati", it will always return same hashcode 1200. and ultimately, bucket will also be same for same key. So in our example it will be Bucket 2.

Point to note here is, hashcode() method always return same hashcode for same key, irrespective of number of times it is called.

If we compare the same with our Employee Letter box example, then hashcode of Employee "Daniel" will be evaluate to "D", irrespective of number of times it is calculated.

Bucket 2 is not empty, So for placing Key-Value pair, "khyati"-"pokar", it first goes to Bucket 2 and from there,

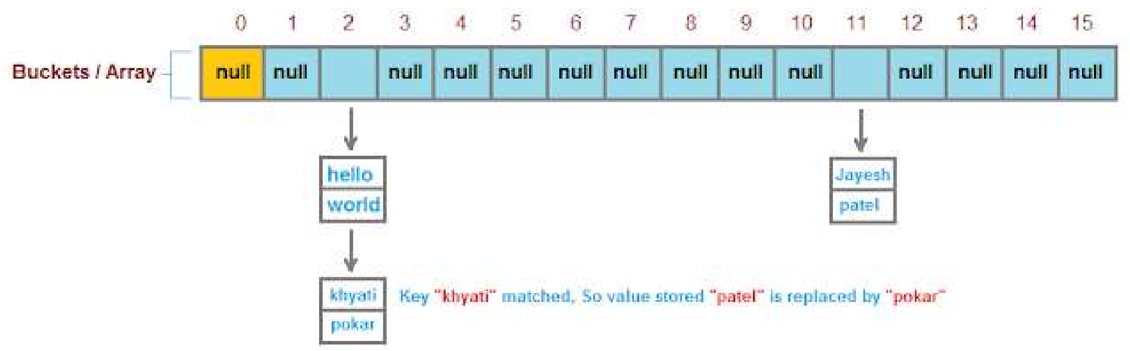
1. It will encounter first key-value pair having key **"hello"**. hashmap will compare, key "hello" with new key "khyati", and check, is it same??

(internally, equals() method does this comparison of two keys)
It is not same, so it will move on to next Key-Value pair.

2. Next it will encounter Key-Value pair having key, "**khyati**". hashmap will compare, key "khyati" with new key "khyati", and check, is it same??

It is same, so it will replace the value "patel" with "pokar" and return (and not move on to next Key-Value pair).

After placing object in hashmap, it will look like below.


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Buckets / Array
null null null null null null null null null null null null null null
hello
world
khyati
pokar
Jayesh
patel
Key "khyati" matched, So value stored "patel" is replaced by "pokar"

> ***Remember, new item that need to be stored, first is compared with every item already present in link list, if the item with same key is found then instead of creating new node/key-value pair, only value for matched node will be replaced.***

Note:

If two Objects have same hashcode, it doesn't mean both are same. hashcode is used to group all Objects having same hashcode in one bucket, So that searching within bucket can be done faster compare to all Objects dumped in one bucket.

GET Operation:

Lets try to get already stored key-value pair in hashmap using key,

1. hashMap.get("**khyati**");
Get operation will follow the same procedure which is used in put operation.

Step 1: First hashcode of key "khyati" is computed,

Step 2: Remember, hashcode of key "khyati" was evaluated to 1200 in put operation, If we calculate hashcode of "khyati" again, then it always evaluate to 1200.

Step 3: Now. bucket/array index need to be calculated from hashcode. indexFor() method will be used to get exact bucket/index for the key "khyati".

Hashcode for same key will always evaluate same, similarly bucket/index calculated for same hashcode will always return same index.

indexFor() method will always return same index 2 for hashcode 1200.

Irrespective of number of times it is called, indexFor() method will always return same index for same hashcode.

Step 4: Once the bucket is known, **equals()** method will come in picture for comparing keys, It will start looking all the items in the bucket 2 for key "khyati",
First key-value pair in bucket is "hello-world", equals method will compare key "hello" with search key "khyati", both are **NOT SAME**, It will ignore and check next key-value pair.

Second key-value pair is "khyati-pokar" and key "khyati" will be compared with search key "khyati", both are SAME, **So it will return value "pokar", stored against key "khyati" and return.**

> *If no item with matching key is found in the Bucket, then it will return value as null.*

hashcode and equals method

1. We saw, two in-different object's can have same hashcode, which ultimately have same bucket.

2. To get the object, each key-value pair in that bucket need to be to compared against the key until bject is not found.

That is where hashcode() and equals() method come into picture.

In our Employee Letter box example, we can say,

1. **hashcode()** methods helps in finding the correct Letter box and,
2. **equals()** method helps to look for each letter within that Letter box.

So we can say,

> ***hashcode()*** *method generate hashcode of a object, which helps in identifying exact bucket.*
> ***equals()*** *method compares each element in the bucket until matching key is not found.*

In hashmap, for both **put and get operation**,
1. First hashcode() method will come in picture, for bucket(array index) **evaluation from key**.
2. Second equals() method will comes in picture, which compares each key-value pair in identified bucket.

In Put operation, equals() method checks "is there any matching key present",
If present, then it will replace the value at matched key.
If not present, then new key-value pair is placed at end.

In Get operation, equals() method checks "is there any matching key present",
If present, then it will return the value stored against matched key.
If not present, then it will return null.

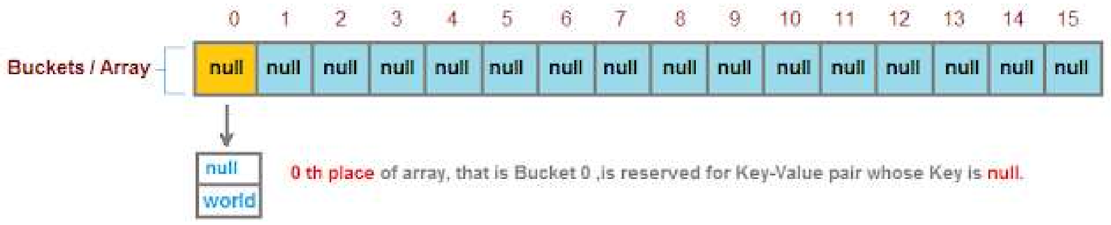
Null Key Placement

We just saw, how put operation of hashmap works, say **hashmap.put("hello","world");**

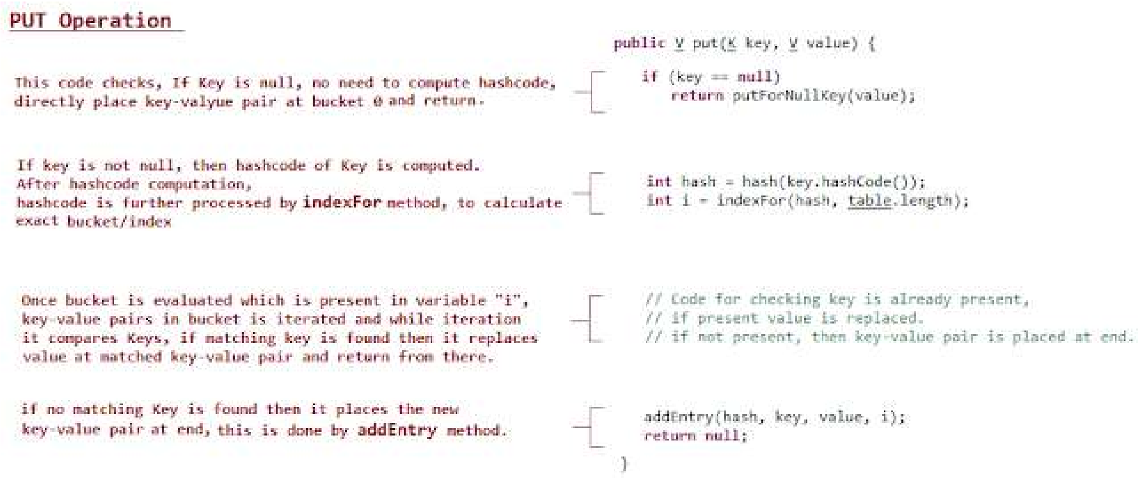
1. First **hashcode** of key "**hello**" is evaluated, which is supplied to **indexFor** method to identify exact bucket.

2. What if we try to put key-value pair like, hashmap.put(null,"world");
Above statement is perfectly valid and **key here is null.**

*If key is null then it will be stored at **first bucket / array[0]**, because hashcode of null key is always 0.*


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Buckets / Array
null null null null null null null null null null null null null null null null
null
world
0 th place of array, that is Bucket 0 ,is reserved for Key-Value pair whose Key is null.

Let's see internally how Put operation works for handling NULL Key.


PUT Operation

This code checks, If Key is null, no need to compute hashcode, directly place key-valyue pair at bucket 0 and return.

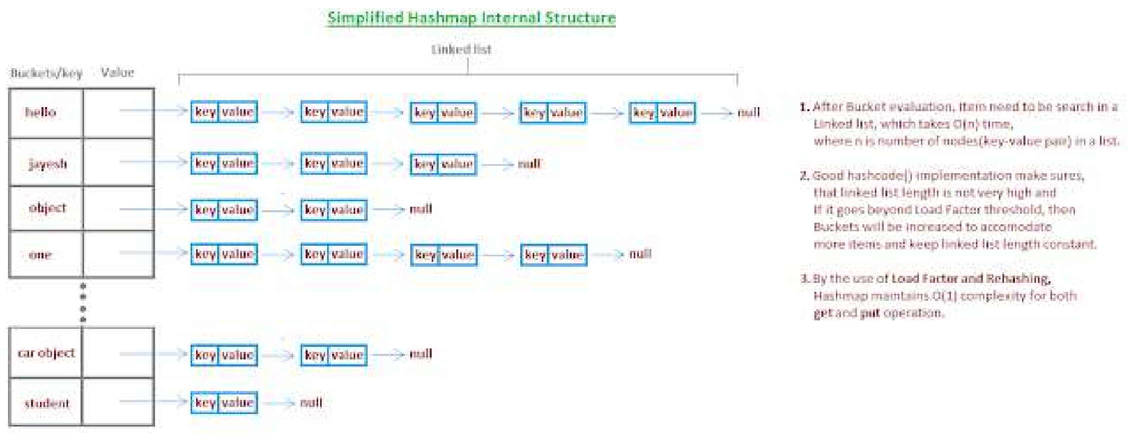
If key is not null, then hashcode of Key is computed.
After hashcode computation,
hashcode is further processed by **indexFor** method, to calculate exact bucket/index

Once bucket is evaluated which is present in variable "i", key-value pairs in bucket is iterated and while iteration it compares Keys, if matching key is found then it replaces value at matched key-value pair and return from there.

if no matching Key is found then it places the new key-value pair at end, this is done by **addEntry** method.

```
public V put(K key, V value) {

    if (key == null)
        return putForNullKey(value);

    int hash = hash(key.hashCode());
    int i = indexFor(hash, table.length);

    // Code for checking key is already present,
    // if present value is replaced.
    // if not present, then key-value pair is placed at end.

    addEntry(hash, key, value, i);
    return null;
}
```

Simplified HashMap Internal Architecture representation



1. After Bucket evaluation, Item need to be search in a Linked list, which takes O(n) time, where n is number of nodes(key-value pair) in a list.

2. Good hashcode() implementation make sures, that linked list length is not very high and If it goes beyond Load Factor threshold, then Buckets will be increased to accomodate more items and keep linked list length constant.

3. By the use of **Load Factor and Rehashing**, Hashmap maintains O(1) complexity for both **get** and **put** operation.

You may also like to see

[What is Hashmap data structure? What is the need of Hashmap? How Hashmap data structure works in Java? What is Hashcode? Can 2 objects have same hashcode?]

[How time complexity of Hashmap get() and put() operation is O(1)? Is it O(1) in any condition?]

[What is Load factor and Rehashing in Hashmap?]

[When get method go to infinite loop in HashMap?]

Enjoy !!!!

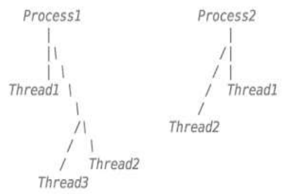
If you find any issue in post or face any error while implementing, Please comment.

Tweet Like 0 Share

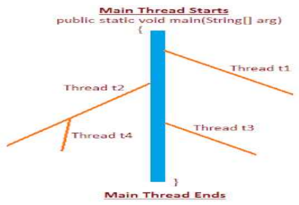
Sav
e

Sav
e

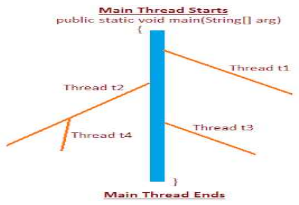
RELATED POSTS :



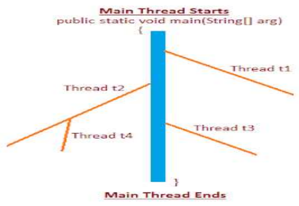
Difference between process and thre...



Difference between CountDownLatch a...



Difference between Join and CountDo...



Custom BlockingQueue implementation...

CATEGORIES: JAVA

ALSO ON **JAVA BY PATEL**

How Recursion works in Java with example

3 years ago

how recursion works internally in java with example. Recursion call ...

Why Hashtable doesn't allow null key/value ...

2 years ago · 2 comments

Why Hashtable doesn't allow null key/value while Hashmap does. ...

Find longest length bi-valued slice in an array

3 years ago

Find longest length bi-valu slice in an array in Java. given n integers, ...

11 Comments 1 Login ▾

G Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

♡ 5 **Share** **Best** **Newest** **Oldest**

F **Fall N Rise**

4 years ago

Can you plz explain this case ..i have overridden hashcode() method giving same bucket number or hashcode value everytime..so each time i add same object k("vaibhav ") it will have same bucket as hashcode is same . when it will traverse over Linkedlist to add new node at end of list in same bucket.....but equals method will say its not equal object as it always return false....so it should add 8 times same object because equals is returning false.why this is not happenning ..o/p:= {Key@1=20}

```
class Key {
String key;
Key(String key)
{
this.key = key;
}

@Override
public int hashCode()
{
return 1;
}
```

see more

0 0 Reply • Share ›

Prasanti

4 years ago

Thanks very much for very detailed and clear explanation. This post made me to understand hashMap in depth.

0 0 Reply • Share ›

Nitin Bhatt

5 years ago

Hello Jayesh ,
Thanks For Providing awesome Explanation . it will really good if you can add details about TreeSet and LinkedHashSet too in future.

Regards
Nitin

0 0 Reply • Share ›

Mohammad Khalid

5 years ago

@Jayesh Patel thanks jayesh

0 0 Reply • Share ›

Mohammad Khalid

5 years ago

Really such a good explanation hashmap

0 0 Reply • Share ›

Yogendra Chaudhari

6 years ago

Good information on HashMap

0 0 Reply • Share ›

Jayesh Patel Mod → Yogendra Chaudhari

6 years ago

Thanks Yogendra .. :)

0 0 Reply • Share ›

Vimal

6 years ago

Really nice explanation...Thanks Jayesh..

0 0 Reply • Share ›

Jayesh Patel Mod ➜ Vimal

6 years ago

Thanks Vimal.. :)

0 0 Reply • Share ›

Kumar Anil Yadav

7 years ago

good link regarding hashmap

0 0 Reply • Share ›

Jayesh Patel Mod ➜ Kumar Anil Yadav

7 years ago

Thank you. I am glad it helped.

0 0 Reply • Share ›

Powered by Blogger.

BLOG ARCHIVE

- 2021 (6)
- 2020 (15)
- 2019 (42)
- 2018 (2)
- 2017 (102)
- 2016 (63)
- 2015 (48)
  - December (7)
  - November (3)
  - October (9)
    - Check If two strings are Anagrams.
    - Find all subsets of a set (Power Set)
    - Given two words (beginWord and endWord), and a dic...
    - Print Nodes in Bottom View of Binary Tree.
    - Print Nodes in Top View of Binary Tree
    - Print a Binary Tree in Vertical Order. Find Vertic...

SEARCH THIS BLOG

Search

STAY IN TOUCH -:)

Java By Patel
1,207 followers

Follow Page

JavaByPatel

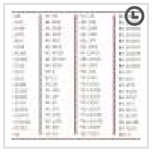
Contact us

POPULAR POSTS

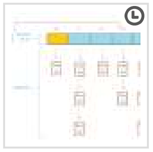
How time complexity of Hashmap get() and put() operation is O(1)? Is it O(1) in any condition?
How time complexity of Hashmap get() and put()...

Kill process on port 8080 in Windows
Kill process running on port 8080 in Windows. Kill...

What is Load factor and Rehashing in Hashmap?
What is Load factor and Rehashing in Hashmap? This is...



Convert Integer to Roman numeral in Java
Converting Integers to Roman Numerals equivalent in Java ...



How ConcurrentHashMap works and ConcurrentHashMap interview questions.
ConcurrentHashMap Interview Questions In Java. ...

- What is Load factor and Rehashing in Hashmap?
- How time complexity of Hashmap get() and put() ope...
- How Hashmap data structure works internally? How h...

- ► September (10)
- ► August (12)
- ► July (7)