


- Algorithm
- Apache Tomcat Server
- Collection Framework
- Collection programs
- Collections implementation
- Core Java
- Core Java Differences
- Core Java Tutorials
- Custom implementation of Data Structures
- Data Structure
- Date tutorials
- eclipse
- Exceptions
- File IO/File handling
-  Garbage collection in java



HashMap Custom implementation in java - How HashMap works internally with diagrams and full program

You are here : [Home](#) / [Core Java Tutorials](#) / [Data structures](#) / [Collection framework](#)

Contents of page :

- 1) Custom HashMap in java >
- 2) Entry<K,V>
- 3) Putting 5 key-value pairs in custom/own HashMap (step-by-step) in java>
- 4) Methods used in custom HashMap in java >
- 5) What will happen if map already contains mapping for key?
- 6) Full Program/SourceCode for implementing custom HashMap in java >



Search This Blog

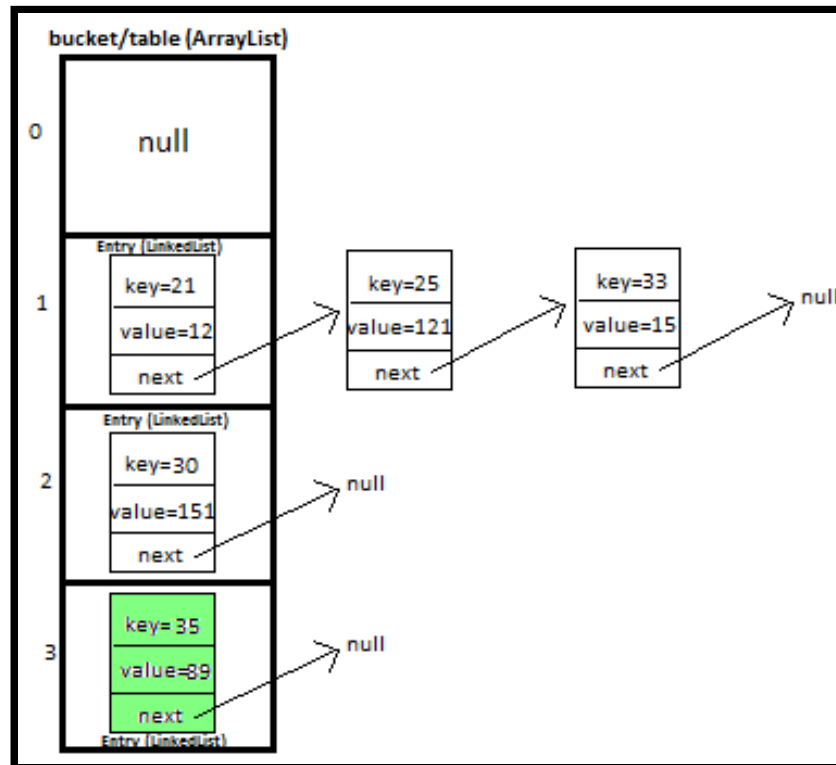
Search



- [Generics](#)
- [Hashing](#)
- [Interview questions](#)
- [iText Pdf tutorial](#)
- [Java 7](#)
- [Java 8](#)
- [Java 9](#)
- [Java QUIZ](#)
- [JavaScript](#)
- [JDBC](#)
- [JUNIT](#)
- [JVM](#)
- [Level1 programs \(beginners\)](#)
- [Level2 programs \(intermediate\)](#)
- [Level3 programs \(advanced\)](#)
- [Linked List](#)
- [Linux](#)
- [Matrix programs](#)
- [MongoDB](#)
- [MongoDB Java](#)
- [MsSql](#)
- [MultiThreading](#)
- [MySQL](#)
- [Notepad++](#)
- [Oracle](#)

- [7\) Complexity calculation of put and get methods in HashMap in java >](#)
 - [7.1\) put method - worst Case complexity >](#)
 - [7.2\) put method - best Case complexity >](#)
 - [7.3\) get method - worst Case complexity >](#)
 - [7.4\) get method - best Case complexity >](#)
- [8\) Summary of complexity of methods in HashMap in java >](#)

1) Custom HashMap in java >



In this tutorial we will learn how to create and implement own/custom [HashMap](#) in java with full working source code.



- [OutOfMemoryErr or](#)
- [Overriding equals and hashCode](#)
- [PostgreSQL](#)
- [Producer Consumer problem/pattern](#)
- [Pyramid generation](#)
- [RegEx](#)
- [Serialization](#)
- [Thread Concurrency](#)
- [Threads](#)
- [TUTORIALS](#)
- [Windows](#)

Join our
Groups>

- [FACEBOOK](#)
- [LINKED IN](#)

This is very **important** and **trending** topic in java. In this post i will be explaining **HashMap** custom implementation in lots of detail with diagrams which will help you in **visualizing** the HashMap implementation. ***This is must prepare topic for interview and from knowledge point of view as well.***

I will be explaining how we will **put** and **get** key-value pair in HashMap by overriding-
>**equals** method - helps in checking equality of entry objects.
>**hashCode** method - helps in finding bucket's index on which data will be stored.

We will maintain **bucket** ([ArrayList](#)) which will store **Entry** ([LinkedList](#)).

2) *Entry<K,V>*

We store key-value pair by using **Entry<K,V>**
Entry contains

- K **key**,
- V **value** and
- Entry<K,V> **next** (i.e. next entry on that location of bucket).

```
static class Entry<K, V> {  
    K key;  
    V value;  
    Entry<K,V> next;  
  
    public Entry(K key, V value, Entry<K,V> next){  
        this.key = key;  
        this.value = value;  
        this.next = next;  
    }  
}
```



Subscribe-Our exclusive posts
for **FREE**. Stay on top! Join
2693+ subscribers

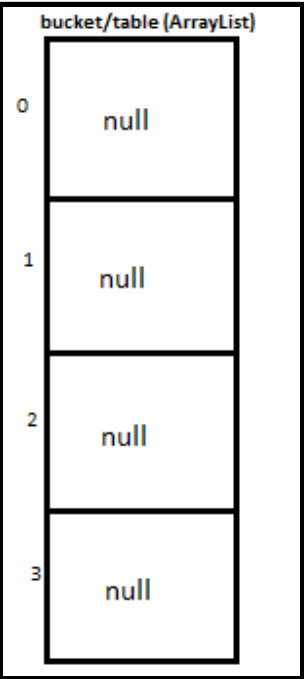
Email address..

Submit

3) Putting 5 key-value pairs in own/custom HashMap (step-by-step) in java>

I will explain you the whole concept of HashMap by putting 5 key-value pairs in HashMap.

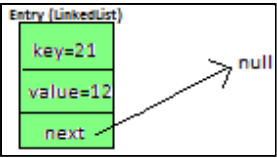
Initially, we have bucket of **capacity=4**. (all indexes of bucket i.e. 0,1,2,3 are pointing to null)



Let's put first key-value pair in HashMap-

Key=21, value=12

newEntry Object will be formed like this >

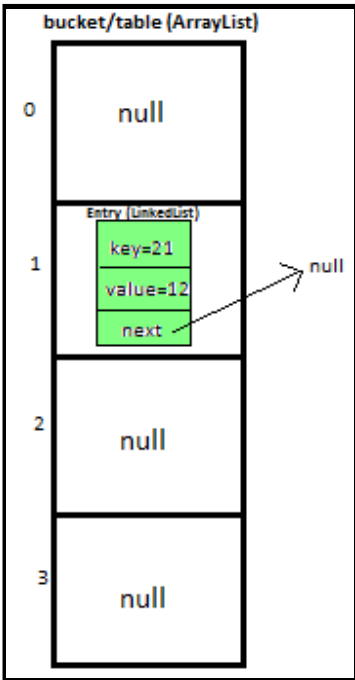


All Labels

[Algorithm](#)(34) [Apache Tomcat Server](#)(4) [Arrays](#)
[Java](#)(30) [Autoboxing](#)(5) [Basic java programs for beginners](#)(15) [Binary Trees](#)(6) [Collection Framework](#)(68) [Collection programs](#)(105) [Collections implementation](#)(16) [Comparable and Comparator program](#)(22) [Core Java](#)(1032) [core java Basics](#)(38) [Core java Conversions](#)(21) [Core Java Differences](#)(11) [Core Java Tutorials](#)(12) [CRUD MongoDB java](#)(5) [CRUD operations MongoDB](#)(3) [Custom implementation of Data Structures](#)(11) [Data Structure](#)(26) [Database](#)(1) [Database Tutorials](#)(3) [Date tutorials](#)(11) [DeSerialization](#)(19) [eclipse](#)(27) [Exceptions](#)(71) [File IO/File handling](#)(71) [Garbage collection in java](#)(43) [Generics](#)(5) [Hashing](#)(8) [Hibernate](#)(1) [Interview questions](#)(14) [iText Pdf tutorial](#)(45) [Java 4](#)(1) [java 5](#)(3) [Java 7](#)(32) [Java 8](#)(80) [Java 9](#)(12)

We will calculate hash by using our **hash(K key)** method - in this case it returns **key/capacity= 21%4= 1**.
So, **1** will be the **index of bucket** on which **newEntry object** will be stored.
We will go to **1st** index as it is pointing to null we will **put our newEntry object there**.

At completion of this step, our HashMap will look like this-



Let's put second key-value pair in HashMap-
Key=**25**, value=121

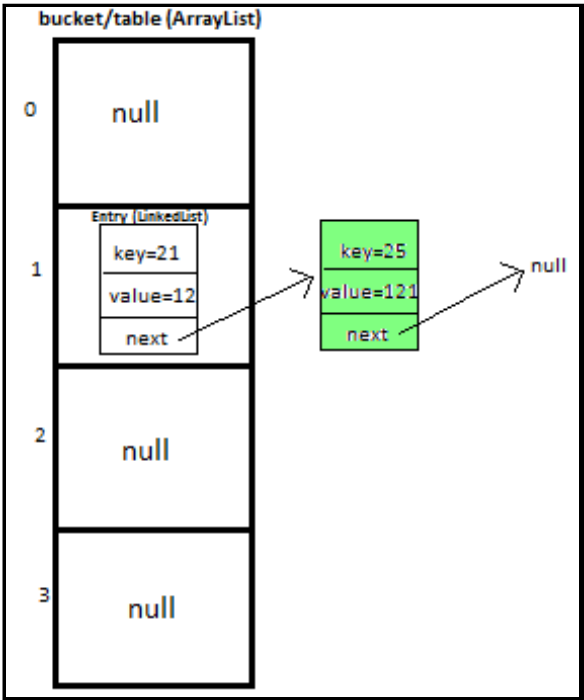
[Java keywords](#)(11) [Java Mcq\(Multiple choice questions\)](#)(5) [Java Programs](#)(8) [Java QUIZ](#)(7) [JavaScript](#)(2) [JDBC](#)(74) [JUNIT](#)(7) [JVM](#)(22) [Level1 programs \(beginners\)](#)(34) [Level2 programs \(intermediate\)](#)(23) [Level3 programs \(advanced\)](#)(13) [Linux](#)(6) [Matrix programs](#)(10) [MongoDB](#)(87) [MongoDB Java](#)(20) [MsSql](#)(1) [MultiThreading](#)(97) [MySql](#)(7) [Notepad++](#)(6) [Oracle](#)(18) [OutOfMemoryError](#)(13) [Overriding equals and hashCode](#)(11) [Pattern generating](#)(14) [PostgreSQL](#)(1) [Producer Consumer problem/pattern](#)(10) [Pyramid generation](#)(14) [Recursion](#)(10) [RegEx](#)(7) [Serialization](#)(20) [Serialization top interview questions and answers in java](#)(18) [Thread Concurrency](#)(35) [Threads](#)(73) [TUTORIALS](#)(22) [Windows](#)(8)





We will calculate hash by using our **hash(K key)** method - in this case it returns **key/capacity= 25%4= 1**.
So, 1 will be the **index of bucket** on which **newEntry object** will be stored.
We will go to **1st index**, it contains **entry with key=21**, we will compare two keys(i.e. **compare 21 with 25** by using **equals method**), as **two keys are different** we check whether entry with key=21's **next is null or not**, if **next is null** we will put our **newEntry object** on **next**.

At completion of this step our HashMap will look like this-



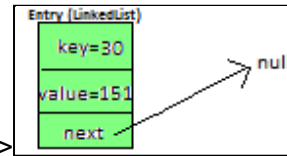
Popular Posts of JavaMadeSoEasy

- [HashMap Custom implementation in java - How HashMap works internally with diagrams and full program](#)
- [Collection Quiz in Java - MCQ - Multiple choice questions](#)
- [CORE JAVA - Top 120 most interesting and important interview questions and answers in core java](#)
- [THREADS - Top 80 interview questions and answers in java for freshers and experienced\(detailed explanation with programs\) Set-1 > Q1- Q60](#)
- [Core Java Tutorial in detail with diagram and programs - BEST EXPLANATION EVER](#)
- [COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50](#)

Let's put third key-value pair in HashMap-

Key=30, value=151

newEntry Object will be formed like this >

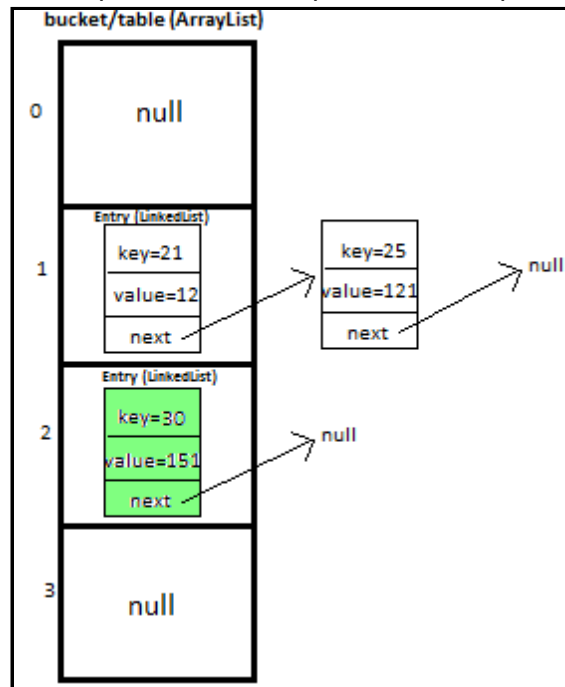


We will calculate hash by using our **hash(K key)** method - in this case it returns **key/capacity= 30%4= 2**.

So, 2 will be the **index of bucket** on which **newEntry object** will be stored.

We will go to **2nd** index as it is pointing to null we will **put our newEntry object there**.

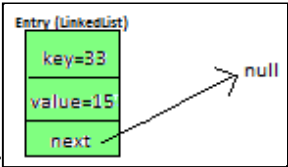
At completion of this step, our HashMap will look like this-



Let's put fourth key-value pair in HashMap-

- [Find sum of both diagonals in matrix - program in java](#)
- [LinkedList Custom implementation in java - How LinkedList works internally with diagrams and full program](#)
- [Thread/multi threading Quiz in Java - MCQ - Multiple choice questions](#)
- [EXCEPTIONS - Top 60 interview questions and answers in java for fresher and experienced - detailed explanation with diagrams Set-1 > Q1- Q25](#)

Key=33, value=15



Entry Object will be formed like this >

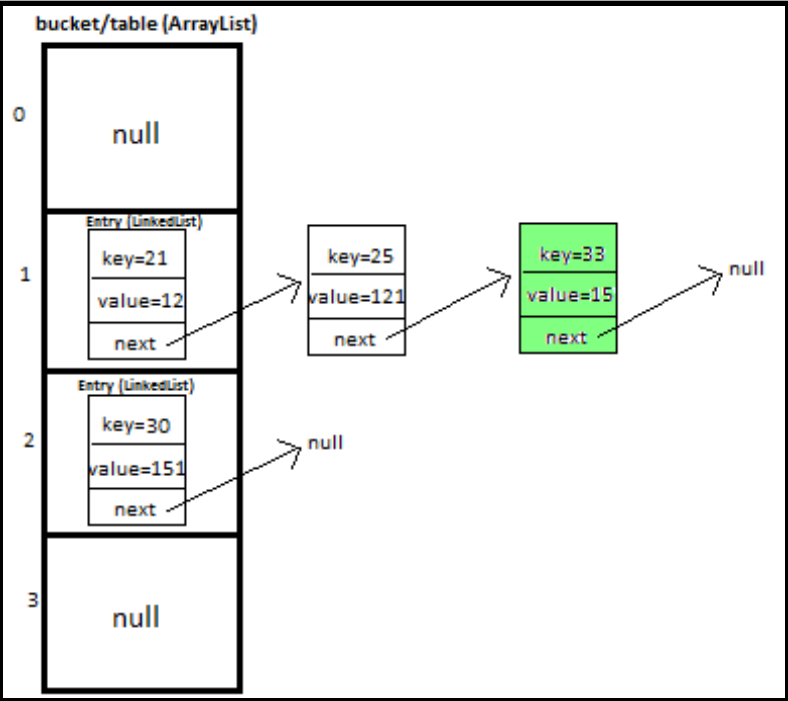
We will calculate hash by using our **hash(K key)** method - in this case it returns **key/capacity= 33%4= 1**,
So, 1 will be the **index of bucket** on which **newEntry object** will be stored.

We will go to 1st index -
>it contains **entry with key=21**, we will **compare** two keys (i.e. **compare 21 with 33** by using **equals method**, as **two keys are different**, proceed to next of **entry with key=21** (proceed only if **next is not null**).
>now, next contains **entry with key=25**, we will **compare** two keys (i.e. **compare 25 with 33** by using **equals method**, as **two keys are different**, now **next of entry with key=25** is pointing to **null** so we won't proceed **further**, we will **put our newEntry object on next**.

At completion of this step our HashMap will look like this-

JavaMadeSoeasy Archive

- ▶ [2018](#) (17)
- ▶ [2017](#) (211)
- ▶ [2016](#) (240)
- ▶ [2015](#) (722)
 - ▶ [December](#) (61)
 - ▶ [November](#) (96)
 - ▶ [October](#) (14)
 - ▶ [September](#) (13)
 - ▶ [August](#) (73)
 - ▶ [July](#) (54)
 - ▶ [June](#) (29)
 - ▶ [May](#) (53)
 - ▶ [April](#) (109)



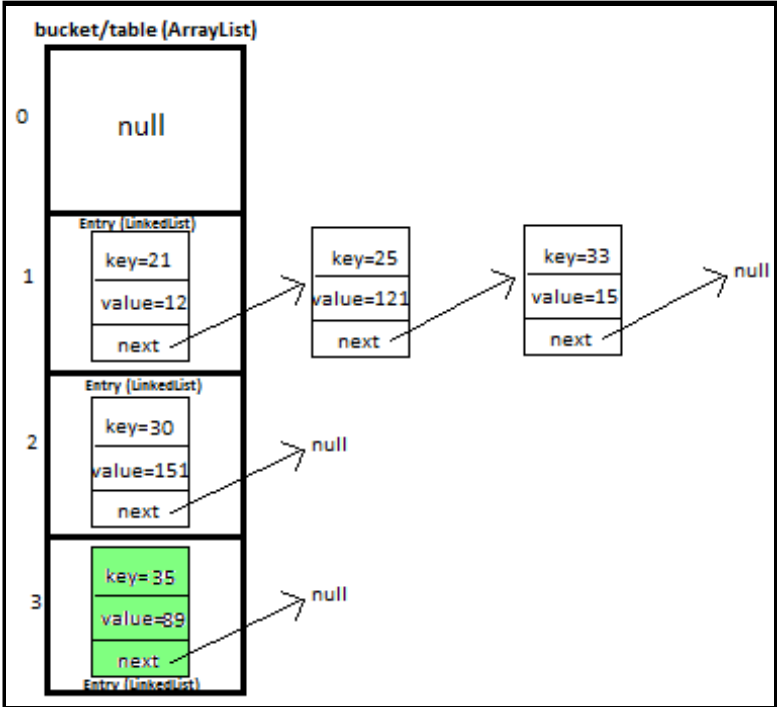
- ▶ March (93)
- ▶ February (99)
- ▶ January (28)

Let's put fifth key-value pair in HashMap-

Key=35, value=89

Repeat above mentioned steps.

At completion of this step our HashMap will look like this-



Must read: [LinkedHashMap Custom implementation](#)
[LinkedHashMap Custom implementation - put, get, remove Employee object.](#)

4) *Methods used in custom HashMap in java >*

public void put (K newKey, V data)	-Method allows you put key-value pair in HashMap -If the map already contains a mapping for the key, the old value is replaced. -provide complete functionality how to override equals method. -provide complete functionality how to override hashCode method.
public V get (K key)	Method returns value corresponding to key.
public boolean remove (K	Method removes key-value pair from HashMapCustom.

deleteKey)	
public void display()	-Method displays all key-value pairs present in HashMapCustom., -insertion order is not guaranteed, for maintaining insertion order refer LinkedHashMapCustom .
private int hash(K key)	-Method implements hashing functionality, which helps in finding the appropriate bucket location to store our data. -This is very important method, as performance of HashMapCustom is very much dependent on this method's implementation.

REFER: [HashMap Custom implementation - put, get, remove Employee object](#).

5) *What will happen if map already contains mapping for key?*

If the map already contains a mapping for the key, the old value is replaced.

6) *Full Program/SourceCode for implementing custom HashMap in java >*

```
package com.ankit;
/**
 * @author AnkitMittal, JavaMadeSoEasy.com
 * Copyright (c), AnkitMittal . All Contents are copyrighted and must not be
 * reproduced in any form.
 * This class provides custom implementation of HashMap(without using java api's)-
 * which allows us to store data in key-value pair form.
 * insertion order of key-value pairs is not maintained.
 * @param <K>
 * @param <V>
 */
class HashMapCustom<K, V> {

    private Entry<K,V>[] table; //Array of Entry.
```

```
private int capacity= 4; //Initial capacity of HashMap
```

```
static class Entry<K, V> {
```

```
    K key;
```

```
    V value;
```

```
    Entry<K,V> next;
```

```
    public Entry(K key, V value, Entry<K,V> next){
```

```
        this.key = key;
```

```
        this.value = value;
```

```
        this.next = next;
```

```
    }
```

```
}
```

```
@SuppressWarnings("unchecked")
```

```
public HashMapCustom(){
```

```
    table = new Entry[capacity];
```

```
}
```

```
/**
```

```
 * Method allows you put key-value pair in HashMapCustom.
```

```
 * If the map already contains a mapping for the key, the old value is replaced.
```

```
 * Note: method does not allows you to put null key though it allows null values.
```

```
 * Implementation allows you to put custom objects as a key as well.
```

```
 * Key Features: implementation provides you with following features:-
```

```
 *     >provide complete functionality how to override equals method.
```

```
 *     >provide complete functionality how to override hashCode method.
```

```
 * @param newKey
```

```
 * @param data
```

```
 */
```

```
public void put(K newKey, V data){
```

```
    if(newKey==null)
```

```
        return; //does not allow to store null.
```

```
    //calculate hash of key.
```

```
    int hash=hash(newKey);
```

```
    //create new entry.
```



```
Entry<K,V> newEntry = new Entry<K,V>(newKey, data, null);
```

```
//if table location does not contain any entry, store entry there.
```

```
if(table[hash] == null){
```

```
    table[hash] = newEntry;
```

```
}else{
```

```
    Entry<K,V> previous = null;
```

```
    Entry<K,V> current = table[hash];
```

```
while(current != null){ //we have reached last entry of bucket.
```

```
    if(current.key.equals(newKey)){
```

```
        if(previous==null){ //node has to be insert on first of bucket.
```

```
            newEntry.next=current.next;
```

```
            table[hash]=newEntry;
```

```
            return;
```

```
        }
```

```
        else{
```

```
            newEntry.next=current.next;
```

```
            previous.next=newEntry;
```

```
            return;
```

```
        }
```

```
    }
```

```
    previous=current;
```

```
    current = current.next;
```

```
}
```

```
previous.next = newEntry;
```

```
}
```

```
}
```

```
/**
```

```
 * Method returns value corresponding to key.
```

```
 * @param key
```

```
 */
```

```
public V get(K key){
```

```
    int hash = hash(key);
```

```
    if(table[hash] == null){
```

```
        return null;
```

```
    }else{
```

```
        Entry<K,V> temp = table[hash];
```

```
        while(temp!= null){
```

```
        if(temp.key.equals(key))
            return temp.value;
        temp = temp.next; //return value corresponding to key.
    }
    return null; //returns null if key is not found.
}

/**
 * Method removes key-value pair from HashMapCustom.
 * @param key
 */
public boolean remove(K deleteKey){

    int hash=hash(deleteKey);

    if(table[hash] == null){
        return false;
    }else{
        Entry<K,V> previous = null;
        Entry<K,V> current = table[hash];

        while(current != null){ //we have reached last entry node of bucket.
            if(current.key.equals(deleteKey)){
                if(previous==null){ //delete first entry node.
                    table[hash]=table[hash].next;
                    return true;
                }
                else{
                    previous.next=current.next;
                    return true;
                }
            }
            previous=current;
            current = current.next;
        }
        return false;
    }
}
```

```
}

/**
 * Method displays all key-value pairs present in HashMapCustom.,
 * insertion order is not guaranteed, for maintaining insertion order
 * refer LinkedHashMapCustom.
 * @param key
 */
public void display(){

    for(int i=0;i<capacity;i++){
        if(table[i]!=null){
            Entry<K, V> entry=table[i];
            while(entry!=null){
                System.out.print("{ "+entry.key+"="+entry.value+" }" + " ");
                entry=entry.next;
            }
        }
    }

}

/**
 * Method implements hashing functionality, which helps in finding the appropriate
 * bucket location to store our data.
 * This is very important method, as performance of HashMapCustom is very much
 * dependent on this method's implementation.
 * @param key
 */
private int hash(K key){
    return Math.abs(key.hashCode()) % capacity;
}

}

/**
 * Main class- to test HashMap functionality.
 */
```

```
public class HashMapCustomApp {  
  
    public static void main(String[] args) {  
        HashMapCustom<Integer, Integer> hashMapCustom = new HashMapCustom<Integer, Integer>();  
        hashMapCustom.put(21, 12);  
        hashMapCustom.put(25, 121);  
        hashMapCustom.put(30, 151);  
        hashMapCustom.put(33, 15);  
        hashMapCustom.put(35, 89);  
  
        System.out.println("value corresponding to key 21=" + hashMapCustom.get(21));  
        System.out.println("value corresponding to key 51=" + hashMapCustom.get(51));  
  
        System.out.print("Displaying : ");  
        hashMapCustom.display();  
  
        System.out.println("\n\nvalue corresponding to key 21 removed: " + hashMapCustom.remove(21));  
        System.out.println("value corresponding to key 51 removed: " + hashMapCustom.remove(51));  
  
        System.out.print("Displaying : ");  
        hashMapCustom.display();  
    }  
}
```

/*Output

value corresponding to key 21=12
value corresponding to key 51=null
Displaying : {21=12} {25=121} {33=15} {30=151} {35=89}

value corresponding to key 21 removed: true
value corresponding to key 51 removed: false
Displaying : {25=121} {33=15} {30=151} {35=89}

*/

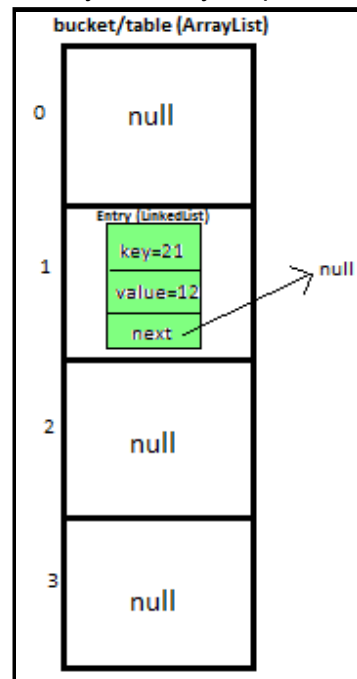
7) Complexity calculation of put and get methods in HashMap in java >

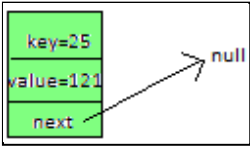
7.1) put method - worst Case complexity >

$O(n)$.

But how complexity is $O(n)$?

Initially, let's say map is like this -





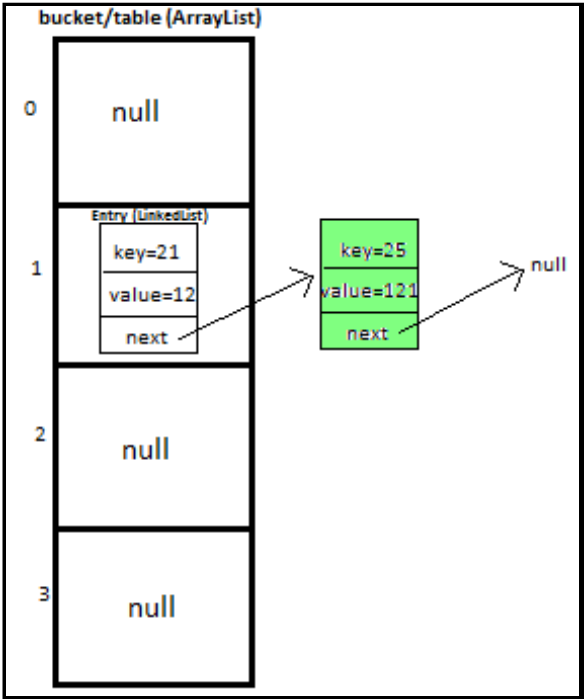
And we have to insert **newEntry Object** with **Key=25, value=121**

We will calculate hash by using our **hash(K key)** method - in this case it returns **key/capacity= 25%4= 1**.

So, **1** will be the **index of bucket** on which **newEntry object** will be stored.

We will go to **1st index**, it contains **entry with key=21**, we will compare two keys(i.e. **compare 21 with 25** by using **equals method**), as **two keys are different** we check whether entry with key=21's **next is null or not**, if **next is null** we will **put our newEntry object** on **next**.

At completion of this step our HashMap will look like this-



Now let's do complexity calculation -

Earlier there was 1 element in HashMap and for putting **newEntry Object** we iterated on it. Hence complexity was $O(n)$.

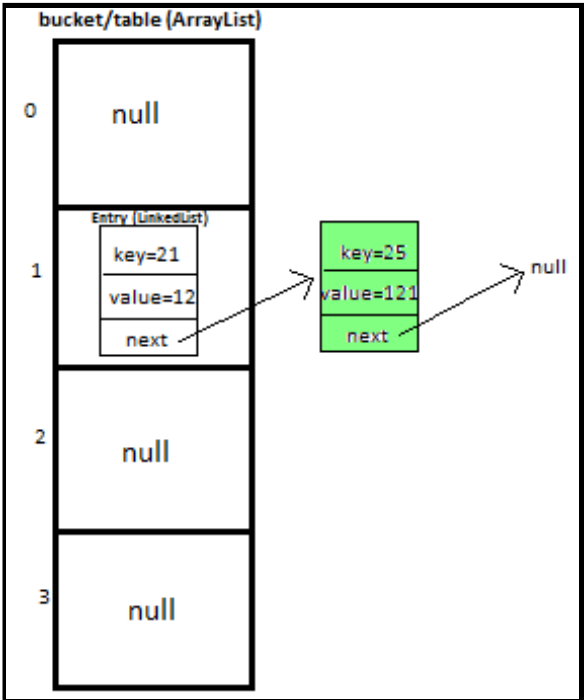
Note: We may calculate complexity by adding more elements in HashMap as well, but to keep explanation simple i kept less elements in HashMap.

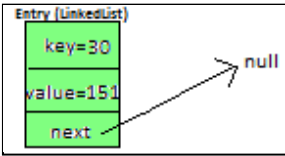
7.2) put method - best Case complexity >

$O(1)$.

But how complexity is $O(n)$?

Let's say map is like this -





And we have to insert **newEntry Object** with **Key=30, value=151**

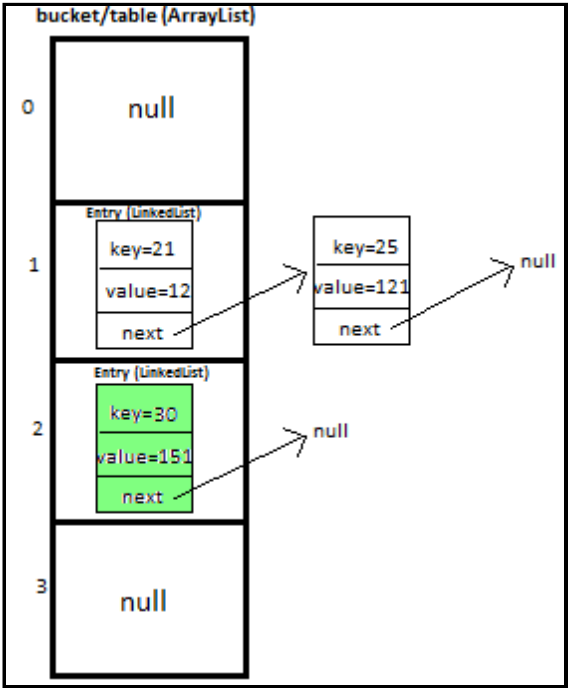
We will calculate hash by using our **hash(K key)** method - in this case it returns

key/capacity= 30%4= 2.

So, 2 will be the **index of bucket** on which **newEntry object** will be stored.

We will go to **2nd** index as it is pointing to null we will **put our newEntry object there**.

At completion of this step our HashMap will look like this-



Now let's do complexity calculation -

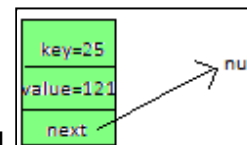
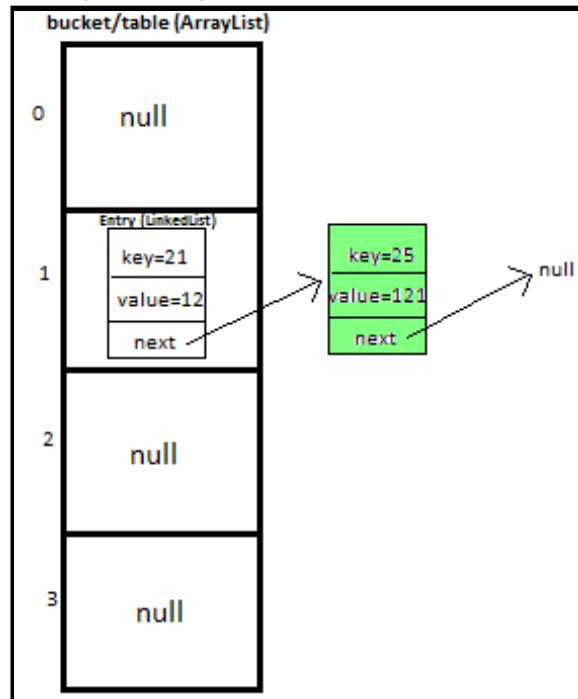
Earlier there 2 elements in HashMap but we were able to put **newEntry Object** in first go. Hence complexity was **O(1)**.

7.3) *get method - worst Case complexity >*

$O(n)$.

But how complexity is $O(n)$?

Initially, let's say map is like this -



And we have to get **Entry Object** with **Key=25, value=121**

We will calculate hash by using our **hash(K key)** method - in this case it returns **key/capacity= 25%4= 1**.

So, 1 will be the **index of bucket** on which **Entry object** is stored.

We will go to **1st** index, it contains **entry with key=21**, we will compare two keys(i.e. **compare 21 with 25** by using **equals method**), as **two keys are different** we check whether entry with key=21's **next is null or not**, **next is not null** so we will repeat same process and ultimately will be able to get **Entry object**.

Now let's do complexity calculation -

There were 2 elements in HashMap and for getting **Entry Object** we iterated on both of them. Hence complexity was $O(n)$.

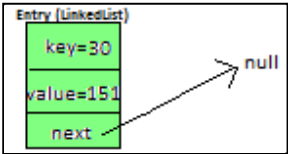
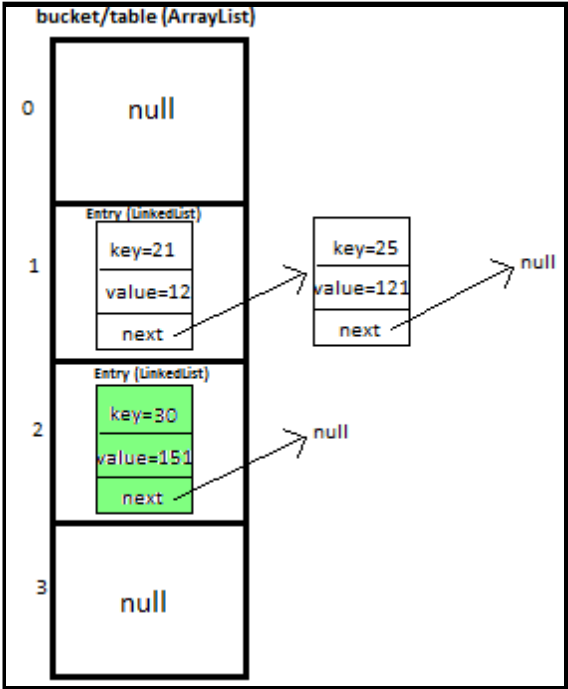
Note: We may calculate complexity by using HashMap of larger size, but to keep explanation simple i kept less elements in HashMap.

7.4) get method - best Case complexity >

$O(1)$.

But how complexity is $O(n)$?

Initially, let's say map is like this -



And we have to get **Entry Object** with **Key=30, value=151**
We will calculate hash by using our **hash(K key)** method - in this case it returns **key/capacity= 30%4= 2**.
So, **2** will be the **index of bucket** on which **Entry object** is stored.
We will go to **2nd** index and get **Entry object**.

Now let's do complexity calculation -
There were 3 elements in HashMap but we were able to get **Entry Object** in first go.
Hence complexity was O(1).

8) Summary of complexity of methods in HashMap in java >

Operation/ method	Worst case	Best case
<i>put(K key, V value)</i>	O(n)	O(1)
<i>get(Object key)</i>	O(n)	O(1)

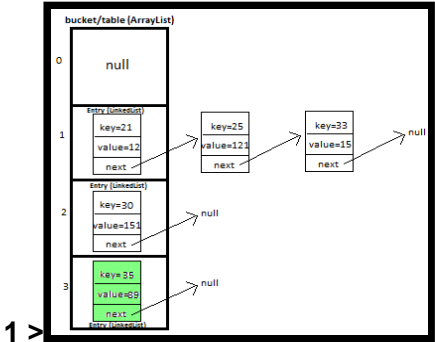
Summary of article >

In this tutorial we learned how to create and implement own/custom [HashMap](#) in java with full program, diagram and examples to insert and retrieve key-value pairs in it.

Having any doubt? or you liked the tutorial! Please comment in below section.

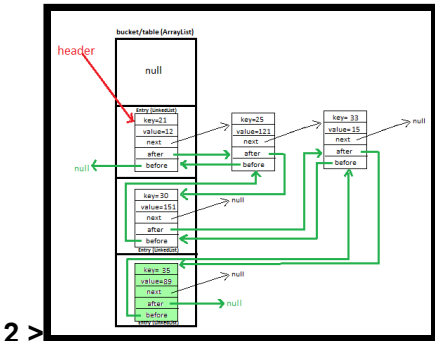
Please express your love by liking [JavaMadeSoEasy.com](#) ([JMSE](#)) on [facebook](#), following on [google+](#) or [Twitter](#).

RELATED LINKS>



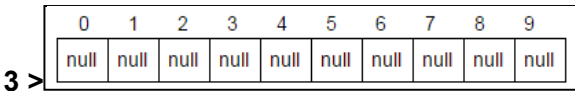
1 > [HashMap Custom implementation in java/ Develop own HashMap with full java code](#)

>[HashMap Custom implementation in java - put, get, remove Employee object](#)



2 > [LinkedHashMap own/Custom implementation/ with full code in java](#)

>[LinkedHashMap Custom implementation - put, get, remove Employee object](#)



3 > [ArrayList custom implementation in java/ Create own ArrayList with full code](#)

>[ArrayList custom implementation - add, get, remove Employee object](#)



4 > [Set Custom implementation in java](#)

>[Set Custom implementation - add, contains, remove Employee object](#)



5 > [LinkedHashSet Custom implementation in java](#)

>[LinkedHashSet Custom implementation - add, contains, remove Employee object](#)

6 > | | | | | | | | | | | |----|------|------|------|------|------|------|------|------|------| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | 71 | null | null | null | null | null | null | null | null | null | [Vector custom implementation in java](#)

[HashMap and Hashtable - Similarity and Differences](#)

[Collection - List, Set and Map all properties in tabular form in java](#)

/** Copyright (c), AnkitMittal [JavaMadeSoEasy.com](#) */

Labels: [Algorithm](#) [Collections implementation](#)

[Core Java](#) [Custom implementation of Data](#)

[Structures](#) [Custom implementation of Map](#)

[Hashing](#) [How map works internally](#) [Map](#)

Overriding equals and hashCode Single Linked List

Must read for you :

32 Comments

www.javamadesoeasy.com

 Login ▾

 Recommend 7

 Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name



Himanshu Choudhary • 3 years ago

Brilliant stuff. Thanx Mr Ankit.

1 ^ | v • Reply • Share ›



Narinder • 8 months ago

//short put method

```
public void put(K newKey,V data){  
  
    if (newKey==null)  
        return;  
  
    int hash=hash(newKey);  
  
    Entry<k,v> newEntry = new Entry<k,v>(newKey,data,null);  
  
    if (table[hash]==null) {  
        table[hash]=newEntry;  
    }else{  
  
        Entry<k,v> temp=table[hash];  
  
        while (temp!=null){ /* iterating till it reaches end, and once it reaches end,add new entry there */  
  
            if(temp.key.equals(newKey)){  
                temp.value=data;  
                return;  
            }  
            temp=temp.next;  
        }  
        temp.next=newEntry;  
    }  
}
```

temp.next=newEntry;

```
    }  
  
    }  
  
    ^ | v • Reply • Share ›
```



Narinder • 8 months ago

in case key matches, why cant we simply replace the value ?

```
"new code"  
if(current.key.equals(newKey)){  
    current.value=data;  
    return;  
}  
  
"old code"  
if(current.key.equals(newKey)){  
    if(previous==null){ //node has to be insert on first of bucket.  
        newEntry.next=current.next;  
        table[hash]=newEntry;  
        return;  
    }  
    else{  
        newEntry.next=current.next;  
        previous.next=newEntry;  
        return;  
    }  
}  
  
^ | v • Reply • Share ›
```



Sohail Ali • a year ago

if we pass null as key then how would we calculate hashCode for that null key in which bucket null value will store bcz hashmap can accept one null key..

^ | v • Reply • Share ›



ILuv Billi ➔ Sohail Ali • 8 months ago

Sohail,

It key is null , then hashCode will not generated and it will store in index 0 of table with
hashCode '0'

^ | v • Reply • Share ›



Balu RajasekharReddy • a year ago

Thanks, good explanation....

^ | v • Reply • Share ›



Ankit Mittal Mod ➔ Balu RajasekharReddy • 9 months ago

Thanks **@Balu RajasekharReddy**

^ | v • Reply • Share ›



Bhargav Patel • a year ago

Hi Nice explanation.
One thing i would like to know, When the capacity will get increased ?

^ | v • Reply • Share ›



Ankit Mittal Mod ➔ Bhargav Patel • 9 months ago

Hi **@Bhargav Patel**

This was basic demonstration done to show implementation of HashMap. so capacity is not
increased here, you may add it :) Thanks

^ | v • Reply • Share ›



SUNIL KUMAR • a year ago

Hello Sir,
Can you share me a link in which explanation of spring boot with jpa,hibernate and service
,Model,DTO,DAO classes.

^ | v • Reply • Share ›



Ankit Mittal Mod • a year ago

Hi! Thanks Sachin, keep reading.

^ | v • Reply • Share ›



Sachin Srivastava • a year ago

Absolutely brilliant explanation



Absolutely brilliant explanation...

^ | v · Reply · Share ›



Harman Patel · a year ago

in put method if `current.key.equals(newKey)` then why can't we use `current.value=newEntry.value`. Is is necessary to have newEntry at front of Entry List??

^ | v · Reply · Share ›



Ankit Mittal Mod · a year ago

Hello **@Dhiraj Kumar**

Initially, we have bucket of capacity=4. (all indexes of bucket i.e. 0,1,2,3 are pointing to null)

So, that's why we use $21\%4$, please go through initial explanation <http://www.javamadesoeasy.c...>

It hope it will be clear than. :)

^ | v · Reply · Share ›



Ankit Mittal Mod · a year ago

Hello@ thanks **@Shubham** :)

^ | v · Reply · Share ›



Shubham · a year ago

Absolutely amazing explanation. Best explanation about internal working of Java Collections

^ | v · Reply · Share ›



Ankit Mittal Mod ➔ **Shubham** · 9 months ago

Thanks **@Shubham** :)

3 ^ | v · Reply · Share ›



Dhiraj Kumar · a year ago

Well, I did not understand one term here. As far as I know, HashMap implementation finds the hashCode value of the key and depending on that value, it finds the space in bucket to store the entry(key-value pair) object. As you've added the very first entry object where the key is 21. If we calculate the hashCode of 21, it comes 1599, But how can you calculate hash value as 1 using $\text{key}/\text{capacity} = 21\%4 = 1$?

And you've quoted it clearly that it returns 1. Could you please explain to me. Do drop me an email

And you've quoted it clearly that it returns 1. Could you please explain to me. Do drop me an email on dhirajn72@gmail.com

^ | v • Reply • Share ›



Rashmi Kanta Swain → Dhiraj Kumar • a year ago

Did not understand why we are calculating 21%4, not 21%7? Could you please explain clearly?

^ | v • Reply • Share ›



Anil Nivargi • 2 years ago

Good explanation ...thanks

^ | v • Reply • Share ›



Ankita Bharambe • 2 years ago

best explanation... very nice..

^ | v • Reply • Share ›



Ankit Mittal Mod • 2 years ago

nat505 Please ensure that you have called put method before calling the get method

Try these lines

```
hashMapCustom.put(21, 12);
```

```
//use scanner as per requirement.....
```

```
System.out.println("value corresponding to key 21="
```

```
+ hashMapCustom.get(21));
```

^ | v • Reply • Share ›



nat505 • 2 years ago

Is there a way to use the different functions, such as get, by using a scanner to read input that would say 21, and have the program return 12? I've been trying to mess with the code, and I keep getting null values returned. I am basically doing `int key = Scanner.nextInt()` and then `customHashMap.get(key)` and it is returning errors.

^ | v • Reply • Share ›



ravi • 2 years ago

Excellent stuff.

^ | v • Reply • Share ›



Dheeraj Kumar Gopali • 3 years ago

Clearly explained. Thanks Ankit

^ | v • Reply • Share ›



Raj Kathal • 3 years ago

In didn't understande put menthod 2 lines what is happing in that line
previous=current;

current = [current.next](#);

^ | v • Reply • Share ›



Ankit Mittal Mod • 3 years ago

In LinkedList values will be added at last.

^ | v • Reply • Share ›



Mahesh • 3 years ago

The best implementation example i have seen on google.Ankit you Rocks buddy

^ | v • Reply • Share ›



Ankit Mittal Mod • 3 years ago

Thanks for comments.

^ | v • Reply • Share ›



Ankit Mittal • 3 years ago

Thanks Mr. Mrinal Pandey.

^ | v • Reply • Share ›



Mrinal Pandey • 3 years ago

Good one. Thanks.

^ | v • Reply • Share ›



Ankit Mittal • 3 years ago

Thanks Mr Himanshu.

^ | v • Reply • Share ›

ALSO ON WWW.JAVAMADESOEASY.COM

JavaMadeSoEasy.com: How to Create and work with PDF files in java - iText library ...

2 comments • 2 years ago

Ankit Mittal — Thanks josep.

JavaMadeSoEasy: JDBC- Batch PreparedStatement example- Execute ...

1 comment • 3 years ago

Veera Sareddy — Add Rollback() in catch block.

JavaMadeSoEasy.com: JVM (java virtual machine)

1 comment • 3 years ago

Punyasmaran Panda — Thanks..nicely xplained

JavaMadeSoEasy.com: What is Implicit casting/promotion of primitive Data type ...

1 comment • 3 years ago

Kingshore Naidu — Please provide type casting with parent class and sub class - looks you covered all OOPs concepts.We all thanks to you ...

Subscribe **Add Disqus to your site**Add DisqusAdd **Disqus' Privacy Policy**Privacy PolicyPrivacy Policy

[Newer Post](#)

[Home](#)

[Older Post](#)