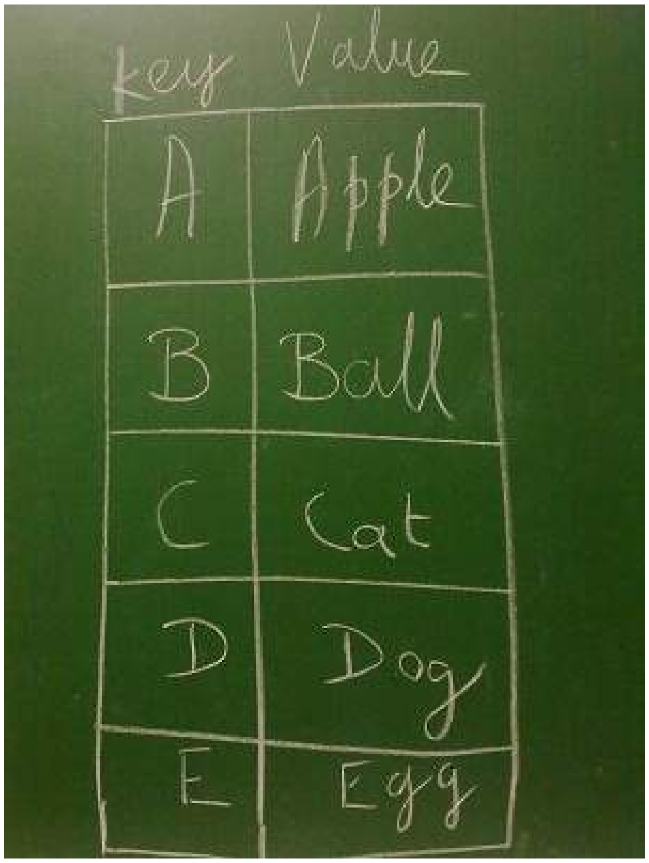
Java Hashtable

Hashtable is an implementation of a key-value pair data structure in java. You can store and retrieve a 'value' using a 'key' and it is an identifier of the value stored. It is obvious that the 'key' should be unique.

java.util.Hashtable extends Dictionary and implements Map. Objects with non-null value can be used as a key or value. Key of the Hashtable must implement hashcode() and equals() methods. By the end of this article you will find out the reason behind this condition.



Key	Value
A	Apple
B	Ball
C	Cat
D	Dog
E	Egg

Generally a Hashtable in java is created using the empty constructor Hashtable(). Which is a poor decision and an often repeated mistake. Hashtable has two other constructors

```
Hashtable(int initialCapacity)
```

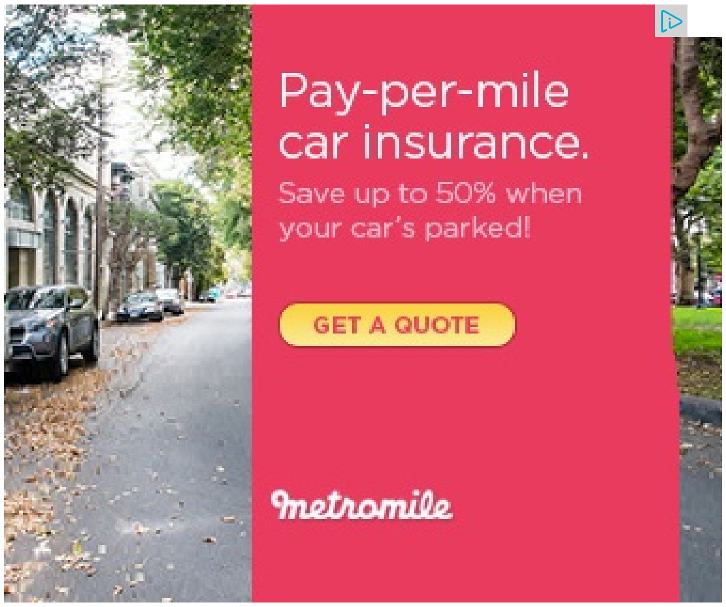
and

```
Hashtable(int initialCapacity, float loadFactor)
```

. Initial capacity is number of buckets created at the time of Hashtable instantiation. Bucket is a logical space of storage for Hashtable.

Hashing and Hashtable

Before seeing java's Hashtable in detail you should understand hashing in general. Assume that, v is a value to be stored and k is the key used for storage / retrieval, then h is a hash function where v is stored at h(k) of table. To retrieve a value compute h(k) so that you can directly get the position of v. So in a key-value pair table, you need not sequentially scan through the keys to identify a value.


Pay-per-mile car insurance.
Save up to 50% when your car's parked!
GET A QUOTE
metromile

h(k) is the hashing function and it is used to find the location to store the corresponding value v. h(k) cannot compute to a indefinite space. Storage allocated for a Hashtable is limited within a program. So, the hasing function h(k) should return a number within that allocated spectrum (logical address space).

Hashing in Java

Java's hashing uses uses hashCode() method from the key and value objects to compute. Following is the core code from Hashtable where the hashCode 'h' is computed. You can see that both key's and value's hashCode() method is called.

```
h += e.key.hashCode() ^ e.value.hashCode();
```

It is better to have your hashCode() method in your custom objects. String has its own hashCode methode and it computes the hashcode value as below:

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

If you don't have a hashCode() method, then it is derived from Object class. Following is javadoc comment of hashCode() method from Object class:

> *Returns a hash code value for the object. This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable.*

If you are going to write a custom hashCode(), then follow the following contract:

> *The general contract of hashCode is: Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified.*

The following is to improve performance of the Hashtable.

> *If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.*

hashCode() guarantees distinct integers by using the internal address of the object.

Collision in Hashtable

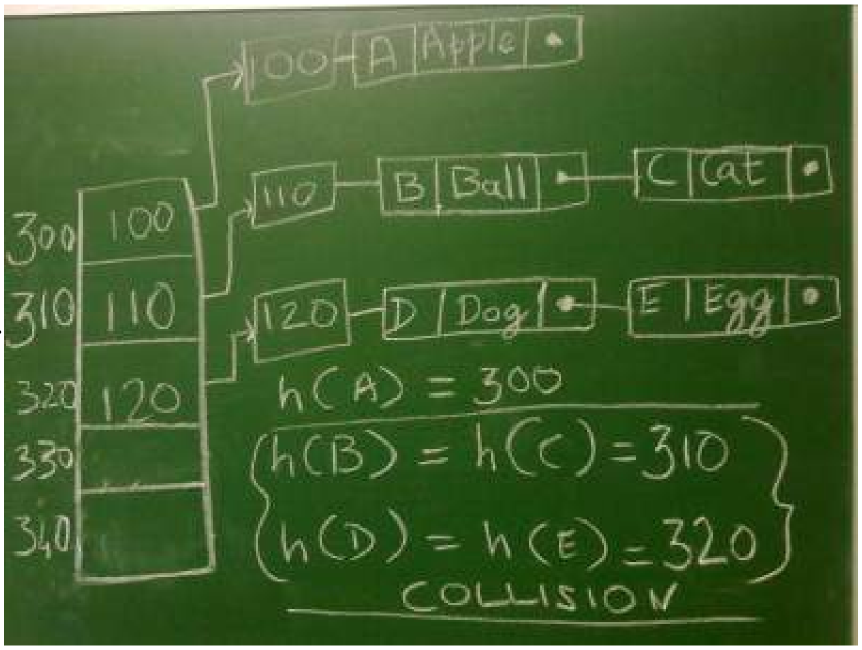
When we try to restrict the hashing function's output within the allocated address spectrue limit, there is a possibility of a collision. For two different keys k1 and k2, if we have h(k1) = h(k2), then this is called collision in hashtable. What does this mean, our hashing function directs us store two different values (keys are also different) in the same location.

When we have a collision, there are multiple methodologies available to resolve it. To name a few hashtable collision resolution technique, 'separate chaining', 'open addressing', 'robin hood hashing', 'cuckoo hashing', etc. Java's hashtable uses 'separate chaining' for collision resolution in

Hashtable.

Collision Resolution in java's Hashtable

Java uses separate chaining for collision resolution. Recall a point that Hashtable stores elements in buckets. In separate chaining, every bucket will store a reference to a linked list. Now assume that you have stored an element in bucket 1. That means, in bucket 1 you will have a reference to a linked list and in that linked list you will have two cells. In those two cells you will have key and its corresponding value.



Why do you want to store the key? Because when there is a collision i.e., when two keys results in same hashcode and directs to the same bucket (assume bucket 1) you want to store the second element also in the same bucket. You add this second element to the already created linked list as the adjacent element.

Now when you retrieve a value it will compute the hash code and direct you to a bucket which has two elements. You scan those two elements alone sequentially and compare the keys using their equals() method. When the key mathches you get the respective value. Hope you have got the reason behind the condition that your object must have hashCode() and equals() method.

Java has a private static class Entry inside Hashtable. It is an implementation of a list and you can see there, it stores both the key and value.

Hashtable performance

To get better performance from your java Hashtable, you need to
1) use the initialCapacity and loadFactor arguments
2) use them wisely
while instantiating a Hashtable.

initialCapacitiy is the number of buckets to be created at the time of Hashtable instantiation. The number of buckets and probability of collision is inversly proportional. If you have more number of buckets than needed then you have lesser possibility for a collision.

For example, if you are going to store 10 elements and if you are going to have initialCapacity as 100 then you will have 100 buckets. You are going to calculate hashCoe() only 10 times with a spectrum of 100 buckets. The possibility of a collision is very very less.

But if you are going to supply initialCapacity for the Hashtable as 10, then the possibility of collision is very large. loadFactor decides when to automatically increase the size of the Hashtable. The default size of initialCapacity is 11 and loadFactor is .75 That if the Hashtable is 3/4 th full then the size of the Hashtable is increased.

New capacity in java Hashtable is calculated as follows:

```
    int newCapacity = oldCapacity * 2 + 1;
```

If you give a lesser capacity and loadfactor and often it does the rehash() which will cause you performance issues. Therefore for efficient performance for Hashtable in java, give initialCapacity as 25% extra than you need and loadFactor as 0.75 when you instantiate.

This Core Java tutorial was added on 17/08/2011.

Share This Tutorial:

Twitter Facebook Google+

PREVIOUS

Decorator Design Pattern

NEXT

Java String

Comments on "Java Hashtable" Tutorial:

joshi says: *17/08/2011 at 9:04 pm*

if you're really interested in performance you should stick to hashmap if multithreading is no issue. You really should add a section about Hastable vs. Hashmap in this article

Joe says: *17/08/2011 at 9:09 pm*

yes Joshi you are right. Between Hashtable and Hashmap, Hashmap is preferred in non-threaded environment.

When you are forced to use Hashtable, consider these performance tips.

Sure I will come up with an article about Hashtable vs Hashmap.

Tito says: *18/08/2011 at 6:54 am*

good info. keep posting. can u write some info on the DAO patterns.

Jigar Joshi says: *18/08/2011 at 6:58 am*

Nicely explained..!

Jigar Joshi says: *18/08/2011 at 7:13 am*

and implementation of hashCode for String is

```
h = 31*h + val[off++];
```

in standard jdk 6 & 7

Praveen says: *18/08/2011 at 1:26 pm*

Good article after a long time…
Kindly start posting on JEE6 features.

Jacob says: *18/08/2011 at 8:27 am*

See this also ,,http://stackoverflow.com/questions/6493605/how-does-java-hashmap-work

---

Madhusudhan says: *18/08/2011 at 3:44 pm*

Great. Clear & Simple. Thank you.

Dev Ghotkule says: *18/08/2011 at 11:23 am*

Most use full information.

KiranJyothi says: *18/08/2011 at 10:17 pm*

It is very nice and clear. Can you take some real time examples and explain (like yellow pages). That would be great!!

Thank you for this post!!

Cheers,
Jyothi

---

shashi says: *24/08/2011 at 5:15 am*

nice artical

---

shyam says: *24/08/2011 at 7:10 am*

this kind of explanation is clear

Born says: *25/08/2011 at 9:33 am*

"Hashtable is preferred in non-threaded environment." – you mixed it up? ;) – hashtable is the synchronized version, so use hashmap instead, if multi-threading is not needed. And besides – one should always consider the synchronizedXYZ Methods of the Collections class instead of using special synchronized Collection types. So use: Collections.synchronizedMap if you want a synchronized version of HashMap.

But – nevertheless – the theory the blog post is talking about is of course the same for the hashmap and considered as useful information ;)

Joe says: *25/08/2011 at 10:27 am*

Oops! Born, thats a typo (very costly one) :0 I have fixed it. Thanks for pointing out.

Anjali says: *07/09/2011 at 7:03 am*

Nice article Joe. Well explained. Thanks

Elakkiya says: *29/09/2011 at 9:22 am*

Such a nice article i have ever seen..

Anant Choubey says: *08/10/2011 at 8:51 pm*

Worth Reading. Nice post. Looking forward for more :) keep posting

---

Partha says: *10/10/2011 at 12:02 pm*

Instead of using HashTable, If we use HashMap and put those block in a synchronized block..what you say on performance?

Joe says: *10/10/2011 at 12:24 pm*

@Partha, using HashMap enclosed in a synchronized block is costlier than using a Hashtable. Since, the synchronization in Hashtable is fine-grained than this approach.

The best possible approach would be analyze the need for synchronization and avoid using Hashtable.

---

satesh says: *14/10/2011 at 9:00 am*

Very good explanation.Thanks alot :-)

Anonymous says: *24/10/2011 at 8:30 am*

awesome CSS !! i like it very much :D contents are lucid and to the point.. :)

---

prabha says: *04/11/2011 at 9:47 am*

It is very simple language that you used in this blog which makes me very impressive to read a lot of stuff about java from this blog. I am fortunate to take look at this website. Hopefully you will be coming up with better and essential information regarding Java.
Thank you so much
from
Prabha

Grasshopper Network says: *07/11/2011 at 2:13 am*

This is a lucid and nice Article. You can always keep adding the features and extra information. We are also java developers and provide various projects for free. Would you give us an opportunity for a Guest Posting to write about Image Processing in Java? We will wait for your positive response.

---

mangesh says: *15/11/2011 at 2:15 pm*

Nice post…….please try to elaborate it with some examples….

---

haripriya says: *16/11/2011 at 7:51 am*

very nice site and helpful for beginners :)

CAL says: *17/11/2011 at 3:48 am*

Hey joe the explanation is really excellent you can also post on struts, spring, hibernate.

---

muthu says: *22/11/2011 at 11:25 am*

Joe, Could you explain the Collision resolution for below scenario

Map map = new HashMap();

I- two keys are same .But it does not throw any exception. How it finds collision resolution?

map.put(“abc”,12);
map.put(“abc”,13);

&

II- two keys are different .But hascode values are same for two keys. How it finds collision resolution?

map.put(“abc”,15);
map.put(“cba”,18);

JRZ says: *27/11/2011 at 4:46 pm*

Your point to this line when discussing the use of value.hashCode():

h += e.key.hashCode() ^ e.value.hashCode();

A line much like this exists (Hashtable.java:948 in jdk 6u23). However, this line is NOT used in the regular business of put() and get(). Rather, it is the hashCode Entry inner class, which must override hashCode() because it overrides equals().

In normal put() and get() operations, the value's hashCode is never used. (This must be the case, because the value is not known in a get operation.) Look at the get() method on line 332 and note that it only calls key.hashCode().

Thanks for posting this article! Since it's getting plenty of readership, I wanted to make sure that people don't get the wrong idea and think that the value's hashCode plays a role in the put operation.

Mariusz Lotko says: *28/11/2011 at 7:56 am*

To be honest, calculating hash using both key.hashCode() and value.hashCode() is done in Map.Entry.hashCode().

Hashtable uses _only_ the value of key.hashCode() to find value. It would be quite surprising to use value's hash for lookups.

I assume Map.Entry.hashCode() is implemented only for purpose of using Map.Entry as a key:

Map<Map.Entry, Integer>

Which maps par (String, Integer) -> Integer.

Anurag Bansal says: *03/12/2011 at 6:12 pm*

Great post Buddy.

Keep posting :)

Rajasekar says: *06/12/2011 at 3:57 am*

Very nice…..

Ramesh says: *13/12/2011 at 6:55 pm*

nice explanation! i was struggling to understand about hash table and hash map.Now i am clear.Thanks a lot.looking for more stuff….

Thangavel says: *18/12/2011 at 4:59 pm*

Hi Joe,

Its pretty good article for Hashtable. I have got more idea about on this. The Explanation is very nice..

Keep on posting more article, best of luck.

Thangavel L Nathan.

Vishnu says: *29/12/2011 at 6:59 am*

Good work dude………. very very easy to understand…………

Ghanshyam says: *29/12/2011 at 10:36 am*

Please give a same demo for HashMap also.

---

chandran says: *04/01/2012 at 2:21 pm*

Really very happy to see this article……Thanks a lot…

---

pavan says: *08/01/2012 at 11:59 am*

Information is very clear and in simple terminology. Thanks a lot.

---

chandra says: *18/01/2012 at 6:47 pm*

this blog is great but if u posting the examples with real time that would be better…

Manjunath says: *08/02/2012 at 5:46 pm*

why null is not allowed in Hastable..??

Arvind Singh says: *21/02/2012 at 2:16 pm*

Very nice discussion…it has almost full coverage of the topic,which is most feared one in java..if you please give some practical examples..then it will increase reader's understanding

---

Raajesh says: *13/03/2012 at 11:31 pm*

Hi Joe,

Worth going through this article – well done

Regards

Raajesh

---

dimuthu says: *25/03/2012 at 10:08 am*

This is great!

---

ahmed farag says: *26/05/2012 at 2:26 am*

i want hashcode for long string without collision please.

---

mr.hello says: *12/06/2012 at 3:35 pm*

excellent and super

BLOB says: *23/06/2012 at 3:16 pm*

You are awesome… Last thing I am gonna say (Actually the first thing too)..

---

sweta says: *20/07/2012 at 5:37 pm*

Thanks Joe for making Hashtable simple for me.All my queries are resolved after reading this short and simple article.You rock :)

Anonymous says: *23/07/2012 at 10:32 am*

your explanation was good..
also pl try to put simple example to construct a hashtable program in java……..

by

Reader

Anonymous says: *28/07/2012 at 3:55 pm*

Nice explanation, I've one question. Plz give two different Keys which produces a same hashcode?

Bhushan says: *03/08/2012 at 8:15 pm*

Please Post article on HashMap vs Hashtable

Anonymous says: *21/08/2012 at 12:38 pm*

Sir,
Please write an article on Thread and Comparable/comparator. I like the the way you are explaining every topic. This blog became as one of good references.

Muthuraj says: *27/08/2012 at 1:32 am*

Interesting….please write Hashtable vs Hashmap also. it will really help us

---

convex.naresh says: *30/08/2012 at 3:48 pm*

Thank you for this post. Very useful article for me.

Swati says: *05/09/2012 at 12:47 pm*

Thanks Joe,
can you please provide a full article on java collection framework with example.

Anonymous says: *17/09/2012 at 2:05 am*

awesum………
thanks for the details…….

Bhanu Tekula says: *17/09/2012 at 12:52 pm*

nice article..

Pankaj says: *23/09/2012 at 1:42 pm*

Very good article Joe!

---

azer says: *26/09/2012 at 3:01 pm*

very awesum…. article joe

Pankaj says: *29/10/2012 at 8:12 am*

yeah… site is very useful for beginner as well as for those who want to know about any specific topic. great! Thank you very much

---

Neeraj says: *31/10/2012 at 11:26 pm*

Also put all Hashing Techniques like Bucket all that.

Tamawy says: *15/11/2012 at 3:43 am*

Quick and briefly useful. Thank you for that. We need easy examples to start with and increase our skills in hashing.

Thank you

kamal sharma says: *30/11/2012 at 2:33 pm*

Thank you very much.
very well explained !!!

Anonymous says: *15/12/2012 at 11:58 am*

Very wonder full article….
I like very much

Srinu D says: *15/12/2012 at 11:59 am*

Very good article..
I like that….Thank you very much

neha says: *04/01/2013 at 2:49 am*

thank you.

venkat says: *07/01/2013 at 3:31 pm*

Hi Joe,Can you please clarify my doubt about HashTable.

the locking will happen only for adding/removing the object or every operation (reading also)

Ganesh says: *08/01/2013 at 2:02 pm*

Nice article Joe, helps understand the basics very well !

---

abhishek says: *08/01/2013 at 2:45 pm*

Hi Joe,

It was nice article

Thanks

Narpath says: *16/01/2013 at 8:44 pm*

Hi joe,

Please provide How will we handle the error code like 500 internal server error in application level ?

Kushagra Thapar says: *24/01/2013 at 1:10 pm*

Hey Joe!
It is a very nice article, it helped me a lot.
But it would be very nice if you explain about how buckets are created and handled in hashtables.

Anonymous says: *30/01/2013 at 3:01 pm*

It was nice article.

Thanks,
Subbu

Gangadhar siraveni says: *06/02/2013 at 10:54 am*

Hi Good Morning Sir,
I am getting confusion in ovberriding equals and hasshcode methods will please provide in detail information

---

Ashok says: *17/02/2013 at 9:24 pm*

Dear Joe,

Please explain how this hashCode() returns hash number and why attributes i and j are converted into String
int i,j;
public int hashCode()
{
String s1 += Integer.toString(i);
String s2 += Integer.toString(j);
return hash;
}

Ashok says: *17/02/2013 at 9:30 pm*

Dear Joe,

Please explain how this hashCode() returns hash number and why attributes i and j are converted into String

```
int i,j;
public int hashCode()
{
String s1 += Integer.toString(i);
String s2 += Integer.toString(j);
return hash;
}
```

Sanjay says: *17/02/2013 at 9:37 pm*

Dear Ashok i think u missing some codes how it will return hash whn u not defined

---

Joseph says: *26/02/2013 at 1:17 pm*

Really fantastic article, thanks, Joe

Chitta says: *11/03/2013 at 2:47 pm*

Beautiful article

---

kavitha says: *17/03/2013 at 9:49 pm*

hi joe,

can you tell me the formula to calculate "hash function"?

---

vamsee says: *09/04/2013 at 5:08 pm*

Hi Joe,

i have been following your Articles in your website
They are just awesome
Can u please come up with Struts explanation Saying what happens inside the ActionServlet and how RequestProcessor Handles

---

Rajkumar Chaudhary says: *26/04/2013 at 7:14 pm*

Fantastic explanation,You are really Astute person.

Muralidhar N says: *17/06/2013 at 8:07 pm*

Here uses came twice: "Java's hashing uses uses hashCode() "

Thank you..

---

suresh atta says: *20/06/2013 at 12:38 pm*

Jigar Bhai ,Nice link :)

BK says: *20/06/2013 at 6:32 pm*

Nice article but I dont think this is correct
hashcode is calculated using
h += e.key.hashCode() ^ e.value.hashCode();

Simple logic: How will this work when you are doing a search on a hash table. You are just aware of the key and there is no way you will be directed to the right bucket in retrieval.
The bucket creation is done using only the hashcode of the key

---

sonam says: *19/07/2013 at 11:45 pm*

Nice article.. very usefull.thnx for sharing

Anwar says: *06/08/2013 at 4:08 pm*

Hi joe,

Provide some good Examples for Hashtable.I am able to attend easily.

Ashok says: *07/08/2013 at 10:41 am*

Hi Joe,
This is pretty awesome.
I didn’t get this statement.

“If you have more number of buckets than needed then you have lesser possibility for a collision”
The collision is purely depends on the symetric objects(Same in hash codes) right? how it is depended on size of bucket?

Nagarjuna says: *09/08/2013 at 1:09 pm*

Hi Joe,

I am fan of you, you are sharing the excellent knowledge from your experience. Thanks a lot for that.

I need some help from you that I just want a sample code (any class) which doesn’t override hashcode() method and shows different hashcode number in different context..

---

kushi says: *25/08/2013 at 7:26 pm*

Gr8t 1… Thanx

---

sandeep says: *29/08/2013 at 3:50 pm*

Please me give real time ussage of collections..suppose in bank project where we use hash table or hashmap or ArrayList ect..

Anonymous says: *12/09/2013 at 7:55 am*

hi Joe,

i didn't understand completely how the Collision Resolution. if h(B)=H(C)=310 if same hash code is returned, can you explain how the collision is resolved?

---

raj says: *12/09/2013 at 7:57 am*

didn't understand completely how the Collision Resolution. if h(B)=H(C)=310 if same hash code is returned, can you explain how the collision is resolved?

Gaurav Kaushik says: *12/09/2013 at 11:17 am*

There are several differences between HashMap and Hashtable in Java:

1. Hashtable is synchronized, whereas HashMap is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.
2. Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.
3. One of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you could easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.

Seema says: *19/09/2013 at 3:15 pm*

Hi can u provide some tutorial also on hsqldb.
Please do so

Seema says: *19/09/2013 at 3:47 pm*

Will u please let us know abt ring buffer

vikas ohlyan says: *20/09/2013 at 9:45 am*

good explanation on hashing…..

Ruks says: *27/09/2013 at 5:44 pm*

No need to covert into Integer here and can be done as below:-
public int hashCode()
{
int hash = 0, p = 31;
hash = hash * p + i;
hash = hash * p + j;
return hash;
}

Basic idea is to generate hashcode uniquely among the object and calculation logic should be simple enough.

Anonymous says: *05/10/2013 at 6:09 pm* very clear and easily understand………

Thank u very mush

Tarik Makhija says: *21/10/2013 at 1:53 pm*

Hi Joe, I have a question on HashMap below :

myMap.get(itr.next()) results output as “Four”
How is it possible that output is coming “Four” for the key provided as null.

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;

public class HashMapTest {
public static void main(String[] args) {

HashMap myMap = new HashMap();

myMap.put(“1″, “One”);
myMap.put(“2″, “Two”);
myMap.put(“3″, “Three”);

myMap.put(null, “Four”);

Set mySet = myMap.keySet();
Iterator itr = mySet.iterator();

while (itr.hasNext()) {
System.out.println(myMap.get(itr.next()));

}

}

}
```

```
/*
Output :
Four
Three
Two
One
*/
```

Muralidhar says: *24/10/2013 at 5:24 pm*

yes..nice differentiation..

---

mathew says: *07/11/2013 at 6:53 pm*

hi sir,
can we able to read and write excel(.xls) using hashtable ??

---

ratnesh says: *22/02/2014 at 2:44 pm*

Hi Joe,
This is pretty awesome.
I didn’t get this statement.

“If you have more number of buckets than needed then you have lesser possibility for a collision”
The collision is purely depends on the symetric objects(Same in hash codes) right? how it is depended on size of bucket?

Ramesh M says: *15/04/2014 at 1:01 am*

See the code below from HashMap

public V put(K key, V value) {
if (key == null)
return putForNullKey(value);

Basically it will return the hash code for null value as 0.So in your example the key is stored at 0 index and when you pass the key the value stored at index 0 will be returned which is “Four”

Thanks,

Ramesh M

Ruks says: *17/04/2014 at 1:36 pm*

In the first scenario, it overrides the value for the key “abc” from 12 to 13.

In the second scenario, it uses key’s equals() method for collision resolution.

Balbir says: *30/04/2014 at 3:49 am*

Good article!!!! Keep sharing such valuable info…

Harish Kumar says: *22/05/2014 at 2:21 pm*

Hi Joe,

In which scenario I should use HashTable. Can you please give scenarios for all Collection implementations. i.e., for ArrayList, LinkedList, HashMap, HashTable, HashSet, TreeSet etc.

Thanks in advance.

Siva Sankar says: *30/05/2014 at 10:39 am*

Excellent document …Trying to understand about the hashcode() and Equals() methods contract from long time. Finally got a nice and simple document which gives a clear explanation, thanku.

Anonymous says: *11/06/2014 at 7:47 pm*

Why Hashtable class does not allow null key and null values? hash value for "null" is zero and HashMap is storing null key (and corresponding value) in the 0th location. why hashtable is not using this ? Is there any specific reason ?

Selva Madhesh says: *23/07/2014 at 5:10 pm*

Very Nice,Can any one explain the difference between hashtable and hashmap with realtime examples?
Thanks in Advance

Ajay says: *06/08/2014 at 8:28 am*

Thanks for explaining in simple language..
My question is regarding size of HashMap/HashTable.
1.When will rehashing be done?.
suppose 16 is the initial size and i have added 12 ie. 16*0.75 element into hashmap. is it irrespective of element being added to same or different bucket?
2. cane we set default size? if yes than how?.

---

kishore says: *23/08/2014 at 12:09 pm*

Please explain these lines written above "The general contract of hashCode is: Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified."

note: please let me know where exactly(in real time) we are going to use hashtable(with example).. and advantages over It???

Comments are closed for this "Java Hashtable" tutorial.

↑ Go to top

Java

Introduction

Java Basics

Java and OOPS

Java String

Exception Handling

Java Serialization

Java Collections Framework

Java Iterator

Java Hashtable

Java PriorityQueue

Read Only Collections

Difference between Vector and ArrayList in java?

Java Generics

Java Util

Concurrent Util

Java JDBC

Java Security

Java Internals

Java Garbage Collection

Java 8

Java Puzzles

Third Party

Tools

Java Interview Questions

Others

Java Gallery

Android

Design Patterns

Hibernate

Spring

Web Services

Servlet

JavaPapers is a tutorials site and **Joe** *runs it passionately.*

Say hi: joe@javapapers.com

FREE UPDATES (Join 10,000 Enthusiasts)

Enter your email here Subscribe

Recommended Tutorials

- Java History
- Overloading Vs Overriding In Java
- Eclipse Shortcuts
- Java Serialization
- Difference Between Interface And Abstract Class
- Java Hashtable
- Difference Between Forward And SendRedirect
- Differentiate JVM JRE JDK JIT
- Java (JVM) Memory Types
- Why Multiple Inheritance Is Not Supported In Java

Java Android Design Patterns Spring

Web Services Servlet

Site Map