HowToDoInJava

Java Hashtable class

Last Updated: December 26, 2020 By: Lokesh Gupta Java Collections Java Concurrency, Java HashTable

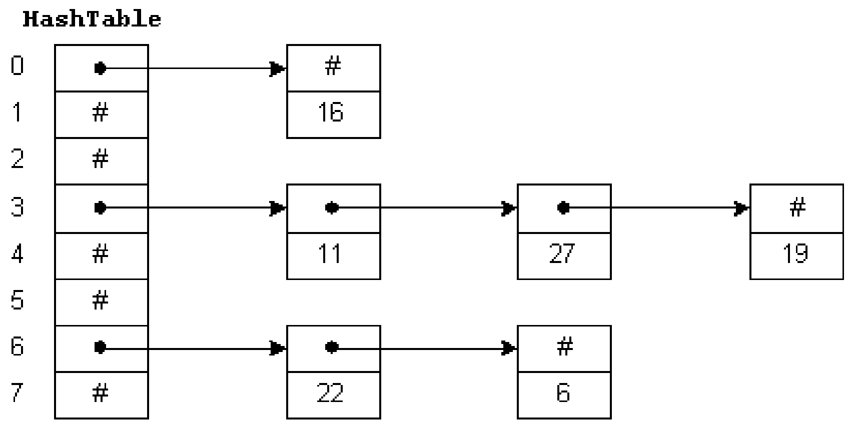
Java Hashtable class is an implementation of hash table data structure. It is very much similar to HashMap in Java, with most significant difference that Hashtable is **synchronized** while HashMap is not.

In this **Hashtable tutorial**, we will learn it's internals, constructors, methods, use-cases and other important points.

Table of Contents

1. How Hashtable Works?

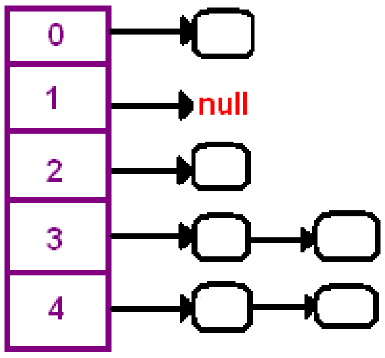
Hashtable internally contains buckets in which it stores the key/value pairs. The Hashtable uses the key's hashcode to determine to which bucket the key/value pair should map.



Java Hashtable

The function to get bucket location from Key's hashcode is called **hash function**. In theory, a hash function is a function which when given a key, generates an address in the table. A hash function always returns a number for an object. Two equal objects will always have the same number while two unequal objects might not always have different numbers.

When we put objects into a hashtable, it is possible that different objects (by the equals() method) might have the same hashcode. This is called a **collision**. To resolve collisions, hashtable uses an **array of lists**. The pairs mapped to a single bucket (array index) are stored in a list and list reference is stored in array index.



Hashtable collision

1.1. Hashtable Declaration

The Hashtable class is declared as following in Java. It **extends Dictionary** class and **implements Map**, `Cloneable` and `Serializable` interfaces. Here `'K'` is the type of keys and `'V'` is the type of mapped values to keys.

```
Hashtable.java
```

```
public class Hashtable<K,V>
    extends Dictionary<K,V>
    implements Map<K,V>, Cloneable, java.io.Serializable
{
  //implementation
}
```

2. Hashtable Features

The important things to learn about Java Hashtable class are:

1. It is similar to HashMap, but it is synchronized while HashMap is not synchronized.
2. It does not accept `null` key or value.
3. It does not accept duplicate keys.
4. It stores key-value pairs in hash table data structure which internally maintains an array of list. Each list may be referred as a bucket. In case of collisions, pairs are stored in this list.
5. Enumerator in Hashtable is not fail-fast.

3. Hashtable Constructors

Hashtable class has four constructors.

- **Hashtable():** It is the default constructor. It constructs a new, empty hashtable with a default initial capacity (11) and load factor (0.75).
- **Hashtable(int size):** It constructs a new, empty hashtable of specified initial size.
- **Hashtable(int size, float fillRatio):** It constructs a new, empty hashtable of specified initial size and fill ratio.
- **Hashtable(Map m):** It constructs a hashtable that is initialized with the key-value pairs in specified map.

Please note that **initial capacity** refers to number of buckets in hashtable. An optimal number of buckets is required to store key-value pairs with minimum collisions (to improve performance) and efficient memory utilization.

The **fill ratio** determines how full hashtable can be before it's capacity is increased. It's Value lie between 0.0 to 1.0.

4. Hashtable Methods

The methods in Hashtable class are very similar to HashMap. Take a look.

- **void clear()** : It is used to remove all pairs in the hashtable.
- **boolean contains(Object value)** : It returns `true` if specified value exist within the hash table for any pair, else return `false`. Note that this method is identical in functionality to `containsValue()` function.
- **boolean containsValue(Object value)** : It returns `true` if specified value exist within the hash table for any pair, else return `false`.
- **boolean containsKey(Object key)** : It returns `true` if specified key exist within the hash table for any pair, else return `false`.
- **boolean isEmpty()** : It returns `true` if the hashtable is empty; returns `false` if it contains at least one key.
- **void rehash()** : It is used to increase the size of the hash table and rehashes all of its keys.
- **Object get(Object key)** : It returns the value to which the specified key is mapped. Returns null if no such key is found.
- **Object put(Object key, Object value)** : It maps the specified `key` to the specified `value` in this hashtable. Neither the key nor the value can be `null`.
- **Object remove(Object key)** : It removes the key (and its corresponding value) from hashtable.
- **int size()** : It returns the number of entries in the hash table.

5. Hashtable Example

Let’s see a example for how to use Hashtable in java programs.

```
Hashtable Example

import java.util.Hashtable;
import java.util.Iterator;

public class HashtableExample
{
    public static void main(String[] args)
    {
        //1. Create Hashtable
        Hashtable<Integer, String> hashtable = new Hashtable<>();

        //2. Add mappings to hashtable
        hashtable.put(1,  "A");
        hashtable.put(2,  "B" );
        hashtable.put(3,  "C");

        System.out.println(hashtable);

        //3. Get a mapping by key
        String value = hashtable.get(1);        //A
        System.out.println(value);

        //4. Remove a mapping
        hashtable.remove(3);            //3 is deleted

        //5. Iterate over mappings
        Iterator<Integer> itr = hashtable.keySet().iterator();

        while(itr.hasNext())
        {
            Integer key = itr.next();
            String mappedValue = hashtable.get(key);

            System.out.println("Key: " + key + ", Value: " + mappedValue);
        }
    }
}
```

Program Output.

```
Console
```

```
{3=C, 2=B, 1=A}
A
Key: 2, Value: B
Key: 1, Value: A
```

6. Hashtable Performance

Performance wise HashMap performs in O(log(n)) in comparion to O(n) in Hashtable for most common operations such as get(), put(), contains() etc.

The naive approach to thread-safety in Hashtable ("synchronizing every method") makes it very much worse for threaded applications. We are better off externally synchronizing a HashMap. A well thought design will perform much better than Hashtable.

Hashtable is obsolete. Best is to use ConcurrentHashMap class which provide much higher degree of concurrency.

7. Hashtable vs HashMap

Let's quickly list down the **differences between a hashmap and hashtable in Java**.

1. HashMap is non synchronized. Hashtable is synchronized.
2. HashMap allows one null key and multiple null values. Hashtable doesn't allow any null key or value.
3. HashMap is fast. Hashtable is slow due to added synchronization.
4. HashMap is traversed by Iterator. Hashtable is traversed by Enumerator and Iterator.
5. Iterator in HashMap is fail-fast. Enumerator in Hashtable is not fail-fast.
6. HashMap inherits AbstractMap class. Hashtable inherits Dictionary class.

8. Conclusion

In this tutorial, we learned about Java Hashtable class, it's constructors, methods, real life usecases and compared their performances. We also learned how a hastable is different from hashmap in Java.

Do not use Hashtable in your new applications. Use HashMap if you do not need councurrency. In concurrent environment, prefer to use ConcurrentHashMap.

Drop me your questions in comments.

Happy Learning !!

Reference:

Hashtable Java Docs

Recommended Reading:

1. Difference between HashMap vs Hashtable in Java
2. [Solved]: javax.xml.bind.JAXBException: class java.util.ArrayList nor any of its super class is known to this context
3. Java TransferQueue – Java LinkedTransferQueue class
4. Java LinkedHashMap class
5. Java TreeMap class

6. Java HashSet class
7. Java LinkedHashSet class
8. Java ArrayBlockingQueue class
9. Java CopyOnWriteArrayList class
0. Java CopyOnWriteArraySet class

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

* *We do not spam !!*

6 thoughts on “Java Hashtable class”

josiah williams

July 10, 2020 at 7:16 pm

Why is it essential for a HashTable class to allow the user to determine the initial size?

Reply

Lokesh Gupta

July 13, 2020 at 11:37 am

Resizing is an expensive operation. So, if I can predict the initial size based on number of elements I am going to put in it, it saves some resources.

Reply

ravi

January 12, 2020 at 10:50 am

Why hashtable is slow?

Reply

Harsha Narayan

January 23, 2020 at 10:07 am

Because Hashtable is synchronized.

Reply

Jessica

December 10, 2019 at 6:21 pm

performance of HashMap should be O(1) instead of O(logn) right?

Reply

Vishnu

March 11, 2020 at 8:52 am

O(1) is the best case. However, worst case is O(logn) for HashMap's as if more entries are present in a bucket, then the bucket is transformed from a linked list to a tree. Traversing a tree takes O(logn) time.

Reply

Leave a Comment

Name *

Email *

Website

☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial