HowToDoInJava

Java – HashMap

Last Updated: December 26, 2020

HashMap in Java in a collection class which implements **Map** interface. It is used to store **key & value** pairs. Each key is mapped to a single value in the map.

Keys are unique. It means we can insert a key 'K' only once in a map. Duplicate keys are not allowed. Though a value `'V'` can be mapped to multiple keys.

1. java.util.HashMap Class

1.1. HashMap class declaration

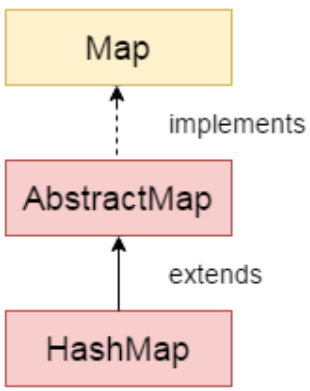
HashMap has been declared as following:

```
HashMap class declaration

public class HashMap&lt;K,V&gt; extends AbstractMap&lt;K,V&gt;
        implements Map&lt;K,V&gt;, Cloneable, Serializable
```

1.2. HashMap class Hierarchy

As shown above, HashMap implements `Map` interface and extends `AbstractMap` class.



HashMap Hierarchy

2. Java HashMap Features

- HashMap cannot contain duplicate keys.
- HashMap allows multiple `null` values but only one `null` key.
- HashMap is an **unordered collection**. It does not guarantee any specific order of the elements.
- HashMap is **not thread-safe**. You must explicitly synchronize concurrent modifications to the HashMap. Or you can use **Collections.synchronizedMap(hashMap)** to get the synchronized version of HashMap.
- A value can be retrieved only using the associated key.
- HashMap stores only object references. So primitives must be used with their corresponding wrapper classes. Such as `int`
- will be stored as `Integer`.
- HashMap implements **Cloneable** and **Serializable** interfaces.

3. HashMap internal implementation

HashMap works on principle of hashing. Hashing is a way to assigning a unique code for any variable/object after applying any formula/algorithm on its properties. Each object in java has it's **hash code** in such a way that two equal objects must produce the same hash code consistently.

3.1. HashMap.Entry class

The key-value pairs are stored as instance of inner class `HashMap.Entry` which has key and value mapping stored as attributes. key has been marked as `final`.

```
HashMap.Entry class

static class Entry&lt;K ,V&gt; implements Map.Entry&lt;K, V&gt;
{
    final K key;
    V value;

    Entry&lt;K ,V&gt; next;
    final int hash;
```

```
    ...//More code goes here
}
```

3.2. Internal working

All instances of Entry class are stored in an array declard as `'transient Entry[] table'`. For each key-value to be stored in HashMap, a hash value is calculated using the key's hash code. This hash value is used to calculate the **index** in the array for storing Entry object.

In case of **collision**, where multiple keys are mapped to single index location, a **linked list** of formed to store all such key-value pairs which should go in single array index location.

While retrieving the value by key, first index location is found using key's hashcode. Then all elements are iterated in the linkedlist and correct value object is found by identifying the correct key using it's **equals()** method.

4. Java HashMap Example

Let's quickly go through some examples to work with HashMap in Java.

4.1. Add key-value – HashMap.put()

```
HashMap Example

import java.util.HashMap;

public class HashMapExample
{
    public static void main(String[] args) throws CloneNotSupportedException
    {
        HashMap&lt;Integer, String&gt; map = new HashMap&lt;&gt;();

        map.put(1,  &quot;A&quot;);
        map.put(2,  &quot;B&quot;);
        map.put(3,  &quot;C&quot;);

        System.out.println(map);
    }
}
```

Program output.

```
Console

{1=A, 2=B, 3=C}
```

4.2. Get value by key – HashMap.get()

```
HashMap Example

HashMap&lt;Integer, String&gt; map = new HashMap&lt;&gt;();

map.put(1,  &quot;A&quot;);
map.put(2,  &quot;B&quot;);
map.put(3,  &quot;C&quot;);

String value = map.get(2);

System.out.println(&quot;The value is :: &quot;+  value );
```

Program output.

```
Console

The value is :: B
```

4.3. Remove pair by key – HashMap.remove()

```
HashMap Example

HashMap&lt;Integer, String&gt; map = new HashMap&lt;&gt;();

map.put(1, &quot;A&quot;);
map.put(2, &quot;B&quot;);
map.put(3, &quot;C&quot;);

System.out.println(map);

map.remove(3);

System.out.println(map);
```

Program output.

```
Console

{1=A, 2=B, 3=C}
{1=A, 2=B}
```

4.4. Iterate a HashMap

Please note that iterators of this class are **fail-fast** and if any structure modification is done after creation of iterator, it will throw `ConcurrentModificationException`.

```
HashMap Example

HashMap&lt;Integer, String&gt; map = new HashMap&lt;&gt;();

map.put(1, &quot;A&quot;);
map.put(2, &quot;B&quot;);
map.put(3, &quot;C&quot;);

System.out.println(&quot;//Iterate over keys&quot;);

Iterator&lt;Integer&gt; itr = map.keySet().iterator();

while (itr.hasNext())
{
    Integer key = itr.next();
    String value = map.get(key);

    System.out.println(&quot;The key is :: &quot; + key + &quot;, and value is ::
}

System.out.println(&quot;//Iterate over entries set&quot;);

Iterator&lt;Entry&lt;Integer, String&gt;&gt; entryIterator = map.entrySet().itera

while (entryIterator.hasNext())
{
    Entry&lt;Integer, String&gt; entry = entryIterator.next();

    System.out.println(&quot;The key is :: &quot; + entry.getKey() + &quot;, and
}
```

Program output.

```
Console
```

```
//Iterate over keys
The key is :: 1, and value is :: A
The key is :: 2, and value is :: B
The key is :: 3, and value is :: C

//Iterate over entries set
The key is :: 1, and value is :: A
The key is :: 2, and value is :: B
The key is :: 3, and value is :: C
```

5. HashMap Methods

List of methods in HashMap class and their short description.

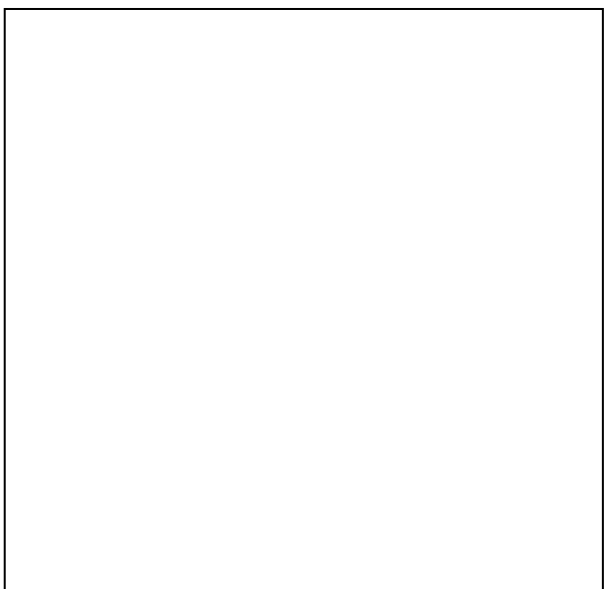
1. **void clear()** : removes all the key-value pairs from the HashMap.
2. **Object clone()** : returns a shallow copy of the specified HashMap.
3. **boolean containsKey(Object key)** : returns `true` or `false` based on whether the specified key is found in the map or not.
4. **boolean containsValue(Object Value)** : Similar to containsKey() method, it looks for the specified value instead of key.
5. **Object get(Object key)** : returns the value for the specified key in the HashMap.
6. **boolean isEmpty()** : checks whether the map is empty.
7. **Set keySet()** : returns the **Set** of the all keys stored in the HashMap.
8. **Object put(Key k, Value v)** : Inserts key-value pair into the HashMap.
9. **int size()** : returns the size of the map which is equal to the number of key-value pairs stored in the HashMap.
10. **Collection values()** : returns a collection of all values in the map.
11. **Value remove(Object key)** : removes the key-value pair for the specified key.
12. **void putAll(Map m)** : copies all the elements of a map to the another specified map.

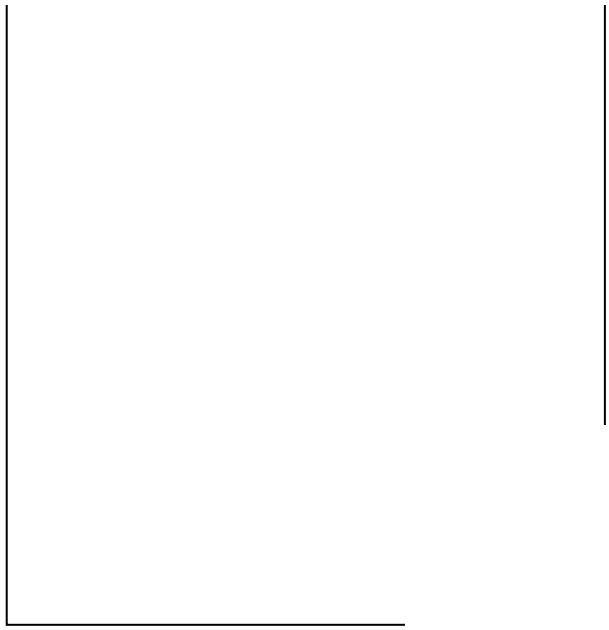
6. HashMap tutorials and examples

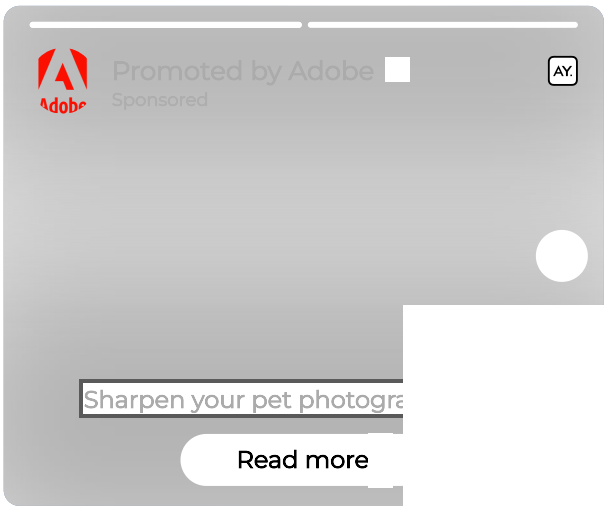
- How HashMap works in Java
- Performance Comparison of Different Ways to Iterate over HashMap
- How to design good custom key object for HashMap
- Difference between HashMap and Hashtable in Java
- Java sort Map by keys (ascending and descending orders)
- Java sort Map by values (ascending and descending orders)
- Java hashCode() and equals() – Contract, rules and best practices
- HashMap and ConcurrentHashMap Interview Questions
- Java ConcurrentHashMap Best Practices
- Convert JSON to Map and Map to JSON
- Marshal and Unmarshal HashMap in Java
- How to Find Duplicate Words in String using HashMap
- Compare two hashmaps
- Synchronize HashMap
- Merge two HashMaps

  How to clone a HashMap

Happy Learning !!






Promoted by Adobe
Sponsored
AY.
Sharpen your pet photogra
Read more

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