


# Java HashMap Interview Questions

---

 [codespaghetti.com/java-hashmap-interview-questions/](https://codespaghetti.com/java-hashmap-interview-questions/)

## HashMap

---

Java HashMap Interview Questions, Algorithms and Programs.

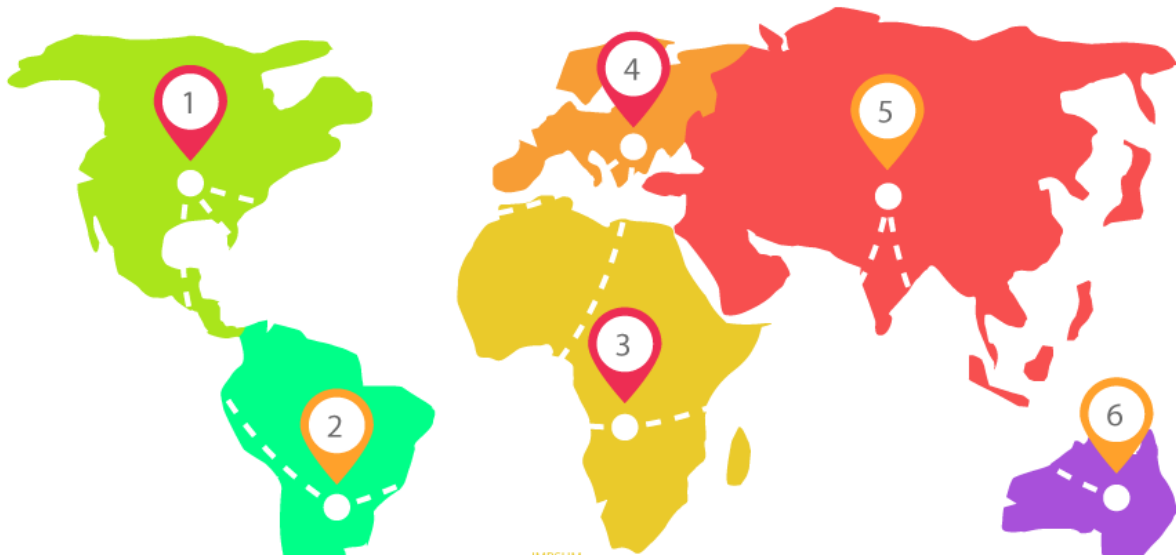


Table of Contents:

---

CHAPTER 1: Java HashMap Interview Questions

---

CHAPTER 2: Java HashMap Comparisons

---

CHAPTER 3: HashMap Equivalent In C#

---

CHAPTER 4: HashMap In A Nutshell

---

CHAPTER 5: Keys To Interview Success

---

Top 3 Java HashMap Interview Questions

---

---



## How To Describe HashMap in Interview?

---

1. HashMap implements Map interface and maintains key and value pairs.
2. HashMap internally works on the principle of Hashing
3. HashMap can only contain unique keys and only one null key.
4. HashMap methods are non-synchronozed.
5. HashMap lookups are  $O(1)$  in the average case, and  $O(n)$  in the worst case

## Question: How HashMap works internally ? [V. Imp]

---

HashMap works on the principle of Hashing. To understand Hashing , we should understand the three terms first

1. Hash Function
2. Hash Value
3. Bucket

## What is Hash function?

---

hashCode() method which returns an integer value is the Hash function. The important point to note that, this method is present in Object class.

This is the code for the hash function(also known as hashCode method) in Object Class :

```
public native int hashCode();
```

## What is Hash value?

---

HashCode method return an int value. So the Hash value is just that int value returned by the hash function.

## What is bucket?

---

A bucket is used to store key value pairs. A bucket can have multiple key-value pairs. In hash map, bucket used is simply a linked list to store objects .

## So how does hashMap works internally?

---

We know that Hash map works on the principle of hashing which means

HashMap get(Key k) method calls hashCode method on the key object and applies returned hashCode to its own static hash function to find a bucket location(backing array)

**Question: How to design a good key for HashMap or Why String, Integer and other wrapper classes are considered good keys? [V. Imp]**

---

The very basic need for designing a good key is that “we should be able to retrieve the value object back from the map without failure”, right??

Otherwise no matter how fancy data structure you build, it will be of no use. To decide that we have created a good key

Key’s hash code is used primarily in conjunction to its equals() method, for putting a key in map and then searching it back from map.

So if hash code of key object changes after we have put a key-value pair in map, then its almost impossible to fetch the value object back from map.

It is a case of memory leak. To avoid this, map keys should be immutable. These are few things to create an immutable of class.

This is the main reason why immutable classes like String, Integer or other wrapper classes are a good key object candidate.

But remember that immutability is recommended and not mandatory. If you want to make a mutable object as key in hashmap, then you have to make sure that state change for key object does not change the hash code of object.

This can be done by overriding the hashCode() method. Also, key class must honor the hashCode() and equals() methods contract to avoid the undesired and surprising behavior on run time.

**Question: How null key is handled in HashMap? Since equals() and hashCode() are used to store and retrieve values, how does it work in case of the null key?**

---

The null key is handled specially in HashMap, there are two separate methods for that putForNullKey(V value) and getForNullKey(). Later is offloaded version of get() to look up null keys.

Null keys always map to index 0. This null case is split out into separate methods for the sake of performance in the two most commonly used operations (get and put), but incorporated with conditionals in others. In short, equals() and hashCode() method are not used in case of null keys in HashMap.

here is how nulls are retrieved from HashMap

```
private V getForNullKey() {
    if (size == 0) {
        return null;
    }
    for (Entry<K,V> e = table[0]; e != null; e = e.next) {
        if (e.key == null)
            return e.value;
    }
    return null;
}
```

---

**Question: Can we use any custom object as a key in HashMap? [V. Imp]**

---

This is an extension of previous questions. Of course you can use any Object as key in Java HashMap provided it follows equals and hashCode contract.

And its hashCode should not vary once the object is inserted into Map. If the custom object is Immutable than this will be already taken care because you can not change it once created.

---

**Question: What if when two different keys have the same Hashcode? [Imp]**

---

In this situation equals() method comes to rescue.

Since bucket is one and we have two objects with the same hashcode. The bucket is the linked list effectively.

Its not a LinkedList as in a java.util.LinkedList – It's a separate (simpler) implementation just for the map

So we traverse through linked list , comparing keys in each entries using keys.equals() until it return true. Then the corresponding entry object Value is returned .

---

**Question: What if when two keys are same and have the same Hashcode? [Imp]**

---

If key needs to be inserted and already inserted hashkey's hashcodes are same, and keys are also same(via reference or using equals() method) then override the previous key value pair with the current key value pair. The other important point to note is that in Map ,Any class(String etc.) can serve as a key if and only if it overrides the equals() and hashCode() method.

---

**Question: What is collision in HashMap? [V. Imp]**

---

Collisions happen when 2 distinct keys generate the same hashCode() value. Multiple collisions are the result of bad hashCode() algorithm.

The more the collisions the worse the performance of the hashMap.

There are many collision-resolution strategies – chaining, double-hashing, clustering.

However, java has chosen chaining strategy for hashMap, so in case of collisions, items are chained together just like in a linkedList.

## Question: What is ConcurrentHashMap in Java? [V. Imp]

---

In concurrentHashMap, the difference lies in internal structure of how its stores key-value pairs.

ConcurrentHashMap has an addition concept of segments. You can imagine one segment equal to one HashMap.

A concurrentHashMap is divided into number of segments [default 16] on initialization.

ConcurrentHashMap allows similar number (16) of threads to access these segments concurrently so that each thread work on a specific segment during high concurrency.

This way, if when your key-value pair is stored in segment 10; code does not need to block other 15 segments additionally. This structure provides a very high level of concurrency.

In other words, ConcurrentHashMap uses a multitude of locks, each lock controls one segment of the map. When setting data in a particular segment, the lock for that segment is obtained. So essentially update operations are synchronized.

When getting data, a volatile read is used without any synchronization. If the volatile read results in a miss, then the lock for that segment is obtained and entry is again searched in synchronized block.

## Question: HashMap Constructors in Java.

---

### Java HashMap provides four constructors.

---

**public HashMap():** Most commonly used HashMap constructor. This constructor will create an empty HashMap with default initial capacity 16 and load factor 0.75

**public HashMap(int initialCapacity):** This HashMap constructor is used to specify the initial capacity and 0.75 load factor. This is useful in avoiding rehashing if you know the number of mappings to be stored in the HashMap.

**public HashMap(int initialCapacity, float loadFactor):** This HashMap constructor will create an empty HashMap with specified initial capacity and load factor.

You can use this if you know the maximum number of mappings to be stored in HashMap. In common scenarios you should avoid this because load factor 0.75 offers a good tradeoff between space and time cost.

`public HashMap(Map? extends K, ? extends V> m):` Creates a Map having same mappings as the specified map and with load factor 0.75

## Examples:

---

```
Map<String, String> mapA = new HashMap<>();
```

```
Map<String, String> mapB = new HashMap<>(2^5);
```

```
Map<String, String> mapC = new HashMap<>(32,0.80f);
```

```
Map<String,String> mapD = new HashMap<>(map1);
```

## Question: How to iterate over hashMap in Java?

---

```
public static void printMap(Map mp) {  
    Iterator it = mp.entrySet().iterator();  
    while (it.hasNext()) {  
        Map.Entry pair = (Map.Entry)it.next();  
        System.out.println(pair.getKey() + " = " + pair.getValue());  
        it.remove(); // avoids a ConcurrentModificationException  
    }  
}
```

## Question: How will you measure the performance of HashMap?

---

An instance of HashMap has two parameters that affect its performance:

1. Initial Capacity
2. Load Factor

The capacity is the number of buckets in the hash table( HashMap class is roughly equivalent to Hashtable.

Except that it is unsynchronized and permits nulls.), and the initial capacity is simply the capacity at the time the hash table is created.

The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased.

When the number of entries in the hash table exceeds the product of the load factor and the current capacity.

The hash table is rehashed (that is, internal data structures are rebuilt) so that the hash table has approximately twice the number of buckets.

In HashMap class, the default value of load factor is (.75)

Question: What is the time complexity of Hashmap get() and put() method ? or Is hashMap really O(1)?

---

A particular feature of a HashMap is that unlike, say, balanced trees, its behavior is probabilistic.

In these cases it's usually most helpful to talk about complexity in terms of the probability of a worst-case event occurring would be.

For a hash map, that of course is the case of a collision with respect to how full the map happens to be. A collision is pretty easy to estimate.

$$p_{\text{collision}} = n / \text{capacity}$$

So a hash map with even a modest number of elements is pretty likely to experience at least one collision. Big O notation allows us to do something more compelling. Observe that for any arbitrary, fixed constant  $k$ .

$$O(n) = O(k * n)$$

We can use this feature to improve the performance of the hash map. We could instead think about the probability of at most 2 collisions.

$$p_{\text{collision}} \times 2 = (n / \text{capacity})^2$$

This is much lower. Since the cost of handling one extra collision is irrelevant to Big O performance.

We've found a way to improve performance without actually changing the algorithm! We can generalize this to

$$p_{\text{collision}} \times k = (n / \text{capacity})^k$$

And now we can disregard some arbitrary number of collisions and end up with vanishingly tiny likelihood of more collisions than we are accounting for. You could get the probability to an arbitrarily tiny level by choosing the correct  $k$ .

All without altering the actual implementation of the algorithm.

**Conclusion:** We talk about this by saying that the hash-map has  $O(1)$  access with high probability'

**An overall comparison chart:**

---

Property	HashMap	TreeMap	LinkedHashMap
Order	no guarantee order will remain constant over time	sorted according to the natural ordering	insertion-order
Get/put remove containsKey	$O(1)$	$O(\log(n))$	$O(1)$
Interfaces	Map	NavigableMap Map SortedMap	Map
Null values/keys	allowed	only values	allowed
Fail-fast behavior	Fail-fast behavior of an iterator cannot be guaranteed impossible to make any hard guarantees in the presence of unsynchronized concurrent modification		
Implementation	buckets	Red-Black Tree	double-linked buckets
Is synchronized	implementation is not synchronized		

## Java HashMap Comparisons



## HashMap Vs Concurrent HashMap



## HashMap

## ConcurrentHashMap

HashMap is not thread-safe	ConcurrentHashMap is thread-safe
HashMap can be synchronized by using <code>synchronizedMap(HashMap)</code> method	ConcurrentHashMap synchronizes or locks on the certain portion of the Map . To optimize the performance of ConcurrentHashMap , Map is divided into different partitions depending upon the Concurrency level . So that we do not need to synchronize the whole Map Object.
HashMap there can only be one null key	ConcurrentHashMap does not allow NULL values . So the key can not be null in ConcurrentHashMap
In multiple threaded environment HashMap is usually faster than ConcurrentHashMap	As only single thread can access the certain portion of the Map and this reduces the performance of ConcurrentHashMap

### Question: HashMap vs HashTable

	HashMap	HashTable
<b>performance</b>	<b>FAST</b> ● ● ●	<b>Slow in Comparison</b> ● ●
<b>Iteration</b>	<b>Fail Fast Iterator</b>	<b>Fail Safe Iterator</b>
<b>Thread Safe</b>	<b>No</b>	<b>Yes</b>
<b>Structure</b>	<b>One null key Any null values</b>	<b>No null key and values</b>

### Question: HashMap vs HashSet

HashMap	HashSet
HashMap contains key - value pairs	HashSet contains the objects(elements or values)
HashMap allows one null key and multiple null values.	HashSet permits to have a single null value.
HashMap cannot contain duplicate keys but can contain multiple duplicate values	HashMap does not allow duplicate keys however it allows to have duplicate values.
HashMap class implements the Map interface	HashSet class implements the Set interface.

## Question: HashMap vs ArrayList

HashMap	ArrayList
HashMap implements the Map interface	ArrayList implements the List interface
In HashMap, the elements are not maintained in the order in which they are inserted	In a ArrayList, the elements will be maintained in the order in which they are inserted
Duplicate values can be stored in ArrayList.	HashMap allows duplicate values. However, HashMap does not allow duplicate keys. The keys must be unique.
keys of HashMap must implements equals and hashCode method correctly	ArrayList doesn't have that requirement but its good to have that because <b>contains()</b> method of ArrayList will use equals() method to see if that object already exists or not.
The <b>HashMap</b> has O(1) performance for every search (on average), so for n searches its performance will be O(n).	The <b>ArrayList</b> has O(n) performance for every search, so for n searches its performance is O(n <sup>2</sup> ).

## HashMap Equivalent In C#

In C# programming language, the nearly exact equivalent to Java's HashMap is the .NET Dictionary collection.

**Dictionary** implements the **IDictionary** interface (which is similar to Java's **Map** interface).

Some notable differences that you should be aware of:

- **Adding/Getting items**
  - Java's HashMap has the **put** and **get** methods for setting/getting items
    - `myMap.put(key, value)`
    - `MyObject value = myMap.get(key)`
  - C#'s Dictionary uses **[]** indexing for setting/getting items
    - `myDictionary[key] = value`

- `MyObject value = myDictionary[key]`
- **null keys**
  - Java's `HashMap` allows null keys
  - .NET's `Dictionary` throws an `ArgumentNullException` if you try to add a null key
- **Adding a duplicate key**
  - Java's `HashMap` will replace the existing value with the new one.
  - .NET's `Dictionary` will replace the existing value with the new one if you use `[]` indexing. If you use the `Add` method, it will instead throw an `ArgumentException` .
- **Attempting to get a non-existent key**
  - Java's `HashMap` will return null.
  - .NET's `Dictionary` will throw a `KeyNotFoundException` . You can use the `TryGetValue` method instead of the `[]` indexing to avoid this: `MyObject value = null; if (!myDictionary.TryGetValue(key, value)) { /* key doesn't exist */ }`

## HashMap In a Nutshell

---

### 1- HashMap Syntax:

```
HashMap<String, Object> hashMap = new HashMap<>();
hashMap.put("key", "value");
```

### 2- HashMap Lookup Process:

HashMap lookup process consists of 2 steps:

Step# 1: Quickly determine the bucket number in which this element may reside (using `key.hashCode()`).

Step# 2: Go over the mini-list and return the element that matches the key (using `key.equals()`).

### 3- HashMap Load factor and resize:

When a hashMap resizes, it will double in size and create a new instance and populate it. Default load factor for hashMap is 0.75.

### 4- HashMap Worst-Case Performance:

In the worst case scenario, a hashMap reduces to a linkedList.

However with Java 8, there is a change,

Java 8 determines if we are running in the worst-case scenario and converts the list into a binary search tree instead of linked list.

### 5- Collisions in HashMap:

Collisions happen when 2 distinct keys generate the same hashCode() value. Multiple collisions are the result of bad hashCode() algorithm.

There are many collision-resolution strategies like

- chaining
- Double-hashing
- Clustering

6- Adding Duplicate Entries Into HashMap:

If you attempt to put the same key with a different value, it will overwrite the old value.

### 7- Concurrency:

Do not use standard HashMap in multi-threaded environment. Instead ConcurrentHashMap must be used in multithreaded applications.

8- HasMap of HashMaps:

HashMap of hashMaps is very popular.

```
Map<String, Map<String, Object>> multiHashMap = new HashMap<>();  
Map<String, Object> hashMapA;  
Map<String, Object> hashMapB;  
multiHashMap.put("A", hashMapA);  
multiHashMap.put("B", hashMapB);
```

9- EnumMap

HashMap with Enum values as keys.

10- LinkedHashMap

HashMap with predictable iteration order (great for FIFO/LIFO caches)

## Java HashMap Resources

---

- Java HashMap Official documentation

---

<https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>

- HashMap and Java8

---

<https://dzone.com/articles/hashmap-performance>

- HashMap Infographic

---

# HASHMAP/

---

## TOP 5 HashMap Questions

/Hashing

/ Internal working

/ HashMap vs LinkedList

/ HashMap vs TreeMap

/ Concurrent HashMap

## How To Describe HashMap in Interviews?

- HashMap implements **Map interface** and maintains key and value pairs.
- HashMap works on **Hashing**
- HashMap can only contain **unique** keys and only one null key.
- HashMap methods are **non-synchronozed**.
- HashMap lookups are  **$O(1)$**  in the average case, and  **$O(n)$**  in the worst case

---

source: [www.codespaghetti.com](http://www.codespaghetti.com)

["HashMap Interview Questions" Click to Tweet](#)

## Keys to interview success

---

The technical interview process is a rough journey. And you will often find yourself in trouble in these interviews. But your preparation and readiness will pave the path for your success.

Understanding data structures, like arrays, linked lists and specially the hash maps is one of those paths that can get you out of troubles.

Learn them, practice them and you will be at ease in any interview.

## About The Author:

---

## References:

---

- <https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html>
- <https://www.linkedin.com/pulse/10-things-java-developer-should-know-hashmap-chinmay-parekh>
- [www.amazon.com/Data-Structures-Algorithms-Java-2nd](http://www.amazon.com/Data-Structures-Algorithms-Java-2nd)
- <https://stackoverflow.com/questions/1273139/c-sharp-java-hashmap-equivalent/4855802#4855802>
- <https://www.linkedin.com/pulse/10-things-java-developer-should-know-hashmap-chinmay-parekh/>