COLLECTION:**MAP:** [**https://www.java67.com/2017/08/top-10-java-hashmap-interview-questions.html**](https://www.java67.com/2017/08/top-10-java-hashmap-interview-questions.html)

|  |  |
| --- | --- |
| Map interface: | Map interface is a special type of collection which is used to store key-value pairs.  It does not extend Collection interface for this reason  The implementing Map interface are: **HashMap, Hashtable, EnumMap, IdentityHashMap, LinkedHashMap and Properties.** |
| What are IdentityHashMap and WeakHashMap? | IdentityHashMap is similar to HashMap except that it uses reference equality when comparing elements.  Does not use equals() method when comparing objects. |
| **WeakHashMap?** | WeakHashMap is an implementation of the Map interface that stores only weak references to its keys. Storing only weak references allows a key-value pair to be garbage collected when its key is no longer referenced outside of the WeakHashMap. This class is intended primarily for use with key objects whose equals methods test for object identity using the == operator. |
| **How to design a good key for hashmap?** | 1. it most important constraint is you must be able to fetch the value object back in future.  2. a good key object must provide same hashCode() again and again |
| **Difference between HashMap and HashTable?** | 1. Hashtable is synchronized, whereas HashMap is not. 2. Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values. 3. The third significant difference between HashMap vs Hashtable is that Iterator in the HashMap is a fail-fast iterator while the enumerator for the Hashtable is not. |
| **What are different Collection views provided by Map interface?** | Map interface provides 3 views of key-values pairs stored in it:  key set view  value set view  entry set view  All the views can be navigated using iterators. |
| **Return type of hashCode()** | the HashCode() function returns an integer. If you have not defined a HashCode() function for your object, Java MAY transcode the memory address of the object to an integer and return that. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **HashMap or TreeMap?** | TreeMap is special form of HashMap. It maintains the ordering of keys which is missing in HashMap.  all keys inserted into the map must implement the Comparable interface otherwise it will throw class cast exception. |
| **What will happen if you try to store a key which is already present in HashMap?** | If you store an existing key in the HashMap, then it will override the old value with the new value and put() will return the old value. There will not be any exception or error. |
| **Can you store a null key in Java HashMap?** | Yes, HashMap allows one null key, which is stored at the first location of bucket array e.g., bucket[0] = value. The HashMap doesn't call hashCode() on the null key because it will throw NullPointerException, hence when a user call get() method with null, then the value of the first index is returned. |
| **Can you store a null value inside HashMap in Java?** | Yes, HashMap also allows null value; you can store as many null values as you want |
| **Which data structure HashMap represents?** | The HashMap is an implementation of hash table data structure, which is idle for mapping one value to other like id to name as you can search for value in O(1) time if you have the key. |
| **Which data structure is used to implement HashMap in Java?** | Even though HashMap represents a hash table, it is internally implemented by using an array and linked list data structure in JDK. The array data structure is used as a bucket, while a [linked list](http://javarevisited.blogspot.sg/2017/07/top-10-linked-list-coding-questions-and.html#axzz4qw7RoNvw) is used to store all mappings which land in the same bucket. From Java 8 onwards, the linked list is dynamically replaced by [binary search tree](https://javarevisited.blogspot.com/2015/10/how-to-implement-binary-search-tree-in-java-example.html), once a number of elements in the linked list cross a certain threshold to improve performance. |
| **Can you store a duplicate key in HashMap?** | No, you cannot insert duplicate keys in HashMap, it doesn't allow duplicate keys. If you try to insert an existing key with the new or same value, then it will override the old value, but the size of HashMap will not change i.e., it will remain the same. This is one of the reasons when you get all keys from the HashMap by calling [keySet()](http://www.java67.com/2016/05/keyset-vs-entryset-vs-values-in-java-map-example.html). It returns a Set, not a Collection, because Set doesn't allow duplicates |
| **Can you store the duplicate value in Java HashMap?** | Yes, you can put duplicate values in HashMap of Java. It allows duplicate values; that's why when you retrieve all values from the Hashmap by calling values() method, it returns a Collection and not Set. Worth noting is that it doesn't return List because HashMap doesn't provide any ordering guarantee for key or value. |
| **Is HashMap thread-safe in Java?** | No, HashMap is not thread-safe in Java. You should not share a HashMap with multiple threads if one or more thread is modifying the HashMap e.g., inserting or removing a map. Though, you can easily share a read-only HashMap. |
| **What are the different ways to iterate over HashMap in Java?** | Map<String,String> map = new HashMap<String,String>();  map.put("kk", "Kali");  map.put("aa", "Durga");  map.put("bb", "Shiva");  map.put("cc", "Shani");  **Set<String> allKeys = map.keySet();**  //**Using keyset**  for(String key : allKeys) {  System.out.println(key+": "+ map.get(key));  }    //**Using Entryset**  **Iterator<Entry<String, String>> itr5 = map.entrySet().iterator();**    while (itr.hasNext()) {  Entry<String, String> entry = itr5.next();  System.out.println(entry.getKey()+":"+entry.getValue());  } |
| **How do you remove a mapping while iterating over HashMap in Java?** | Even though HashMap provides remove() method to remove a key and a key/value pair, you cannot use them to remove a mapping while traversing a HashMap. Instead, you need to use the Iterator's remove method to remove a mapping |
| **In which order mappings are stored in HashMap?** | Random order because HashMap doesn't provide any ordering guarantee for keys, values, or entries. When you iterate over a HashMap, you may get a different order every time you iterate over it. |
| **Can you sort HashMap** | No, you cannot sort a HashMap because unlike List, it is not an ordered collection. Albeit, you can sort contents of HashMap by keys, values, or by entries by sorting and then storing the result into an ordered map like LinkedHashMap or a sorted map e.g., [TreeMap](http://javarevisited.blogspot.sg/2011/12/treemap-java-tutorial-example-program.html#axzz4qw7RoNvw). |
| **What is the load factor in**  **HashMap** | A load factor is a number that controls the resizing of HashMap when a number of elements in the HashMap cross the load factor e.g., if the load factor is 0.75 and when becoming more than 75% full then resizing trigger which involves array copy. |
| **How does resizing happens in HashMap?** | The resizing happens when the map becomes full or when the size of the map crosses the load factor. For example, if the load factor is 0.75 and then becomes more than 75% full, then resizing trigger, which involves an array copy. First, the size of the bucket is doubled, and then old entries are copied into a new bucket. |
| **How many entries you can store in HashMap? What is the maximum limit?** | There is no maximum limit for HashMap, you can store as many entries as you want because when you run out of the bucket, entries will be added to a linked list which can support an infinite number of entries, of course until you exhaust all the memory you have.  Btw, the size() method of HashMap return an int, which has a limit, once a number of entries cross the limit, size() will overflow, and if your program relies on that, then it will break.  This issue has been addressed in JDK 8 by introducing a new method called mappingCount(), which returns a long value. So, you should use mappingCount() for large maps. See [**Java SE 8 for Really Impatient**](http://www.amazon.com/Java-SE8-Really-Impatient-Course/dp/0321927761?tag=javamysqlanta-20) to learn more about new methods introduced in existing interfaces in JDK 8. |
| **What is the difference between the capacity and size of HashMap in Java?** | The capacity denotes how many entries HashMap can store, and size denotes how many mappings or key/value pair is currently present. |
| **What will happen if two different keys of HashMap return the same hashcode()?** | If two keys of HashMap return the same hash code, then they will end up in the same bucket; hence collision will occur. They will be stored in a linked list together. |
| Frequency of String  cats 2  dogs 2  horses 1 |  |
|  |  |
|  |  |
|  |  |