1) How does get(Key key) method works internally in HashMap, and Hashtable in Java?

Here are steps, which happens, when you call get() method with key object to retrieve corresponding value from hash based collection

a) Key.hashCode() method is used to find the bucket location in backing array. (Remember HashMap is backed by array in Java) Though hashcode() is not used directly, but they are passed to internal hash() function.

b) In backing array or better known as bucket, key and values are stored in form of a nested class called Entry. If there is only one Entry at bucket location, than value from that entry is returned. Pretty straightforward right?

Things get little tricky, when Interviewer ask second question, What happens if two keys has same hashCode? If multiple keys has same hashCode, than during put() operation collision had occurred, which means multiple Entry object stored in a bucket location. Each Entry keep track of another Entry, forming a [linked list data structure](http://javarevisited.blogspot.com/2013/05/find-if-linked-list-contains-loops-cycle-cyclic-circular-check.html) there. Now, if we need to retrieve value object in this situation, following steps will be followed:

1) Call hashCode() method of key to find bucket location.

2) Traverse thought linked list, comparing keys in each entries using keys.equals() until it return true.

So, we use equals() method of key object to find correct entry and then return value from that. Remember key.equals() method, and this is what Interviewer want to know. I have seen many programmer mentioning value.equals(), which may be due to interview nervousness, but that’s incorrect. Since you don't have value object passed to get() method, there is no question of calling equals and hashCode method on value object.

That's all on these two HashMap questions guys. Remember to mention about key.hashCode() and key.equals(), whenever some one ask how get method of HashMap or Hashtable works in Java. Value object is not used, it's just exist in Entry, so that get can return it.

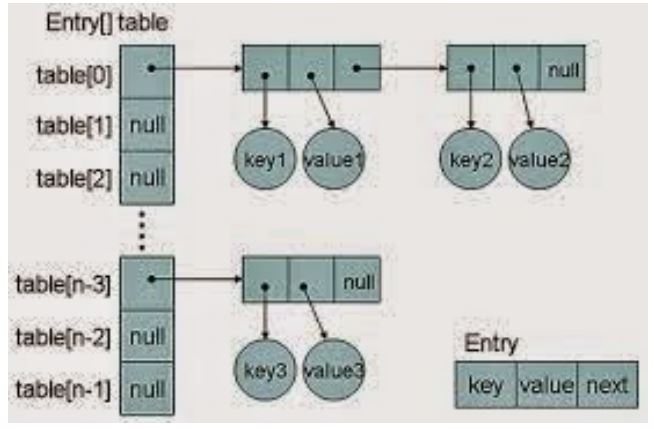
Objects are stored by calling put(key, value) method of HashMap and retrieved by calling get(key) method. When we call put method, hashcode() method of the key object is called so that hash function of the map can find a bucket location to store value object, which is actually an index of the internal array, known as the table. HashMap internally stores mapping in the form of Map.Entry object which contains both key and value object. When you want to retrieve the object, you call [the get() method](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) and again pass the key object. This time again key object generate same hash code (it's mandatory for it to do so to retrieve the object and that's why HashMap keys are immutable e.g. String) and we end up at same bucket location. If there is only one object then it is returned and that's your value object which you have stored earlier.

Since the internal array of HashMap is of fixed size, and if you keep storing objects, at some point of time hash function will return same bucket location for two different keys, this is called collision in HashMap. In this case, a linked list is formed at that bucket location and a new entry is stored as next node.  
If we try to retrieve an object from this linked list, we need an extra check to search correct value, this is done by equals() method. Since each node contains an entry, HashMap keeps comparing entry's key object with the passed key using equals() and when it return true, Map returns the corresponding value.

Since searching in lined list is O(n) operation, in worst case hash collision reduce a map to linked list. This issue is recently addressed in Java 8 by replacing linked list to the tree to search in O(logN) time.

**HashMap works on the principle of hashing**, we have put(key, value) and get(key) method for storing and retrieving Objects from HashMap. When we pass Key and Value object  to put() method on Java HashMap, HashMap implementation calls [hashCode method](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html) on Key object and applies returned hashcode into its own hashing function to find a bucket location for storing Entry object, important point to mention is that HashMap in Java stores both key and value object as Map.Entry in a bucket which is essential to understand the retrieving logic.   
 If people fail to recognize this and say it only stores Value in the bucket they will fail to explain the retrieving logic of any object stored in Java HashMap. This answer is very much acceptable and does make sense that interviewee has a fair bit of knowledge on how hashing works and how HashMap  works in Java. But this is just start of story and confusion increases when you put interviewee on scenarios faced by Java developers on day by day basis. Next question could be about collision detection and collision resolution in Java HashMap e.g.

HashMap uses Key Object's hashcode to find out bucket location and retrieves Value object but then you need to remind him that there are two Value objects are stored in same bucket , so they will say about [traversal in LinkedList](http://javarevisited.blogspot.sg/2010/10/how-do-you-find-length-of-singly-linked.html) until we find the value object , HashMap  stores both Key and Value in LinkedList node or as Map.Entry, we will call keys.equals() method to identify a correct node in LinkedList and return associated value object for that key in Java HashMap. Perfect this is the correct answer.



[final object](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) with proper equals() and hashcode() implementation would act as perfect Java HashMap  keys and **improve the performance of Java HashMap  by reducing collision**.

"**What happens On HashMap in Java if the size of the HashMap exceeds a given threshold defined by load factor?** Until you know how HashMap works exactly you won't be able to answer this question. If the size of the Map exceeds a given threshold defined by load-factor e.g. if the load factor is .75 it will act to re-size the map once it filled 75%. Similar to other collection classes like [ArrayList](http://javarevisited.blogspot.sg/2011/05/example-of-arraylist-in-java-tutorial.html), Java HashMap re-size itself by creating a new bucket array of size twice of the previous size of HashMap and then start putting every old element into that new bucket array. This process is called rehashing because it also applies the hash function to find new bucket location.

WHY MAP KEY SHOULD BE IMMUTABLE OBJECT?

If immutable, the object's hashcode won’t change and it allows caching the hashcode of different keys which makes the overall retrieval process very fast. Also for mutable objects, the hashCode() might be dependent on fields that could change, if this happens you won’t be able to find the key (and its value) in the HashMap since hashCode() returns different value.

## PROBLEM WITH MUTABLE KEYS IN HASHMAP:

*If key’s hash code changes after the key-value pair (Entry) is stored in HashMap, the map will not be able to retrieve the Entry.*

Key’s hashcode can change if the key object is mutable. Mutable keys in HahsMap can result in data loss.

Following example will show you the problem with mutable keys in HashMap.3

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import java.util.HashMap;  import java.util.Map;  public class MutableDemo1 {      public static void main(String[] args) {          // HashMap          Map<MutableKey, String> map = new HashMap<>();          // Object created          MutableKey key = new MutableKey(10, 20);          // Insert entry.          map.put(key, "Robin");          // This line will print 'Robin'          System.out.println(map.get(key));          // Object State is changed after object creation.          // i.e. Object hash code will be changed.          key.setI(30);          // This line will print null as Map would be unable to retrieve the          // entry.          System.out.println(map.get(key));      }  } |

**Output: Robin**

**null**

How to fix it?

*Use immutable objects as keys in HashMap. String, Integer etc are considered as good keys for HashMap since objects of these classes are immutable.*  
We can write your own immutable class also.  
If mutable objects are used as keys in HashMap, then care should be taken such that change in the key object state should not change its hashcode.

**Can we use any custom object as a key in HashMap?**

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](http://javarevisited.blogspot.sg/2011/12/how-to-traverse-or-loop-hashmap-in-java.html). If the custom object is Immutable than this will be already taken care because you cannot change it once created.

**"do you see any problem with resizing of HashMap  in Java"** , you might not be able to pick the context and then he will try to give you hint about multiple thread accessing the Java HashMap and potentially looking for **race condition on HashMap  in Java**.

So the answer is Yes there is potential [race condition](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) exists while resizing HashMap in Java, if two [thread](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) at the same time found that now HashMap needs resizing and they both try to resizing. on the process of resizing of HashMap in Java, the element in the bucket which is stored in linked list get reversed in order during their migration to new bucket because Java HashMap  doesn't append the new element at tail instead it append new element at the head *to avoid tail traversing*. If race condition happens then you will end up with an infinite loop. Though this point, you can potentially argue that what the hell makes you think to use HashMap  in multi-threaded environment to interviewer :)

A couple days ago a coworker and I were working on a piece of code that at first seemed like a mystery to us. We were working with a set. Every attempt at removing the value from the set resulted in false. After hours of looking at the code we discovered that the value was changing, which was the source of our problem.  
Actual problem was the new Hashcode for the object doesn't match the Hashcode the set used to store the value.

### Source Code with Problem and Results

1. public class SetTest {
2. public static void main(String[] args) {
3. //Both Hash code and equals method
4. // are properly implemented in the Person Class.
5. Set&ltPerson&gt set = new HashSet&ltPerson&gt();
7. Person p1 = new Person("Jain Lokesh", new Date());
8. set.add(p1);
9. Person p2 = new Person("Jain", new Date());
10. set.add(p2);
11. Person p3 = new Person("Lokesh", new Date());
12. set.add(p3);
14. System.out.println(set.size()); //Prints 3
16. //Changing the object values
17. p1.setName(null); p1.setDob(null);
18. p2.setName(null); p2.setDob(null);
19. p3.setName(null); p3.setDob(null);
21. set.add(p1); //P1 added Again
22. System.out.println(set.size()); //Prints 4
24. System.out.println(set.remove(p1)); //Returns true
25. System.out.println(set.remove(p2)); //Returns false
26. System.out.println(set.remove(p3)); //Returns false
27. System.out.println(set.size()); // Prints 3
28. }
29. }

**Key points to remember**:

* Hash set is backed by Hash Map
* Hash Map uses hashing algorithm to store object as a Key value for the Map.

**When hash value is calculated for the Hash map objects**:

* Hash value for an object is calculated at the time we add it into Hash Map as a key. And store this value in the Hash Map as a hashing bucket for this key.
* Any time we try to retrieve values from the Map against the key it does following operations:
* Calculate hash value from the object
* Find the bucket from the hash map for the objects hash value.
* If bucket contains multiple object it uses equals method to compare two objects and return proper key value.
* If bucket only contain one object it return this value directly without any another compare.

**Drawbacks of the mutable objects**:

* If object is updated via any other thread or method the behavior will be unexpected.
* We get unexpected behavior because Hash value for object will be changed even though user is changing same Object which is stored to Map.
* As the hash value changes for the object, we are not able to find the hashing bucket for this object.
* Once object changes some methods start giving unpredictable results.
* Hash Set remove method (Object can’t be removed).
* Hash Set add method (Object added again).
* Hash Set contains method (Returns false for the object)
* Hash Map put (Key will be added again)
* Hash Map remove (Object can’t be removed)
* Hash Map contains (Returns false for the object)

Read more: <http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html#ixzz44GTlQs3B>

HOW TO GET KEY FROM VALUE:

//finding key corresponding to value in hashtable - one to one mapping

String key**=** *null*;

String value**=**"Lumia";

**for**(Map**.**Entry entry**:** table**.**entrySet()){

**if**(value**.**equals(entry**.**getValue())){

key **=** entry**.**getKey();

**break**; //breaking because its one to one map

}

}

System**.**out**.**println("got key from value in hashtable key: "**+** key **+**" value: " **+** value);

//finding key corresponding to value in hashtable - one to many mapping

table**.**put("HTC", "Lumia");

Set keys **=** **new** HashSet();

**for**(Map**.**Entry entry**:** table**.**entrySet()){

**if**(value**.**equals(entry**.**getValue())){

keys**.**add(entry**.**getKey()); //no break, looping entire hashtable

}

}

System**.**out**.**println("keys : " **+** keys **+**" corresponding to value in hash table: "**+** value);

# [WHY STATIC CODE ANALYSIS IS IMPORTANT?](http://javarevisited.blogspot.in/2014/02/why-static-code-analysis-is-important.html)

In my opinion, [unit testing](http://javarevisited.blogspot.sg/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html), [code review](http://javarevisited.blogspot.com/2011/09/code-review-checklist-best-practice.html) and **static code analysis** makes a nice combo, along with **continuous integration**.

Many organizations, including investment banks are making it mandatory to pass static code analysis test, penetration testing and security testing before you deploy your code in production. Static analysis tools like findbugs and fortify are getting popular

Pen testing is actually more realistic than static code analysis, because test cases are provided by user and they are more close to real world use case scenario, while static code analysis, only look for patterns, which can cause bugs. If there is no pattern, it doesn't mean no bugs, so ideally you need to do both pen testing and static code analysis to push your application in production.

Read more: <http://javarevisited.blogspot.com/2014/02/why-static-code-analysis-is-important.html#ixzz44MikzUAA>

# [TIPS TO IMPROVE PROGRAMMING SKILL AND BECOME BETTER PROGRAMMER, DEVELOPER](http://javarevisited.blogspot.in/2014/01/10-tips-to-improve-programming-skill-become-better-programmer.html)

Read more: <http://javarevisited.blogspot.com/2014/01/10-tips-to-improve-programming-skill-become-better-programmer.html#ixzz44MlosXSc>

# [WHY STRING IS IMMUTABLE OR FINAL IN JAVA](http://javarevisited.blogspot.in/2010/10/why-string-is-immutable-in-java.html)

Since Strings are very popular as HashMap key, it's important for them to be immutable so that they can retrieve the value object which was stored in HashMap. Since [HashMap works in the principle of hashing](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html), which requires same has value to function properly. Mutable String would produce two different hashcodes at the time of insertion and retrieval if contents of String was modified after insertion, potentially losing the value object in the map.

The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk, where one client's action would affect all another client. For example, if one client changes the value of String "Test" to "TEST", all other clients will also see that value as explained in the first example. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable. At the same time, [*String was made final*](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html) so that no one can compromise invariant of String class e.g. Immutability, Caching, hashcode calculation etc. by extending and overriding behaviors. Another reason of *why String class is immutable* could die due to HashMap.

Read more: <http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html#ixzz44NFNX9y0>

GENERICS TUTORIAL

**Type-safety:**

Advantage of Generics in Java is **type-safety**. Collections prior to JDK1.5 are not type-safe because they accept Object type argument which allows them to catch all type of objects instead of only required the type of object.

**No Casting:**

With Generics you don’t need to cast object, Generics will automatically do that for you. For example here is the code for adding and retrieving an element in List with and without Generics in Java:

**List**  items = **new** **ArrayList**();  
items.add("chocolates");  
**String** item = (**String**) items.get(0)  
  
**List**<**String**> items = **new** **ArrayList**();  
items.add("biscuits");  
**String** item = items.get(0) *//no cast required*

Since no cast required, the result is clear and robust code.

**No ClassCastException:**

With Generics compiler ensures that correct types are added into Java collection classes and no cast is required while retrieving an element, so there is no risk of ClassCastException at [runtime](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html).

**Generics wild cards Example in Java**

There are generally two kinds of wildcards in Generics, Bounded and unbounded. Bounded wildcards can be written in two ways to denote upper bound and lower bound. <?> is called unbounded wildcards because it can accept any Type while <? extends T> and <? super T> are bounded wildcards. To know more about them see my post:  [Bounded vs Unbounded wildcards in Generics](http://javarevisited.blogspot.sg/2012/04/what-is-bounded-and-unbounded-wildcards.html) .  
Now let’s see example of different wildcards in Generics:

**<?>**

*"*?" denotes any unknown type, It can represent any Type at in code for. Use this wildcard if you are not sure about Type. for example, if you want to have an ArrayList which can work with any type than declare it as  "ArrayList<?> unknownList" and it can be assigned to any type of ArrayList as shown in following an example of generics in Java:

**ArrayList**<?> unknownList = **new** **ArrayList**<**Number**>();  
unknownList = **new** **ArrayList**<**Float**>();

**<? extends T>**

This is little restrictive than the previous one it will allow All Types which are either "T" or extends T means a subclass of T. for example List<? extends Number> can hold List<Number> or List<Integer>

**ArrayList**<? **extends** **Number**> numberList = **new** **ArrayList**<**Number**>();  
numberList = **new** **ArrayList**<**Integer**>();  
numberList = **new** **ArrayList**<**Float**>();

**<T super ?>**

This is just opposite of previous one, It will allow T and super classes of T, e.g. List<? super Integer>can hold List<Integer> or List<Number>.

**ArrayList**<? **super** **Integer**> numberList = **new** **ArrayList**<**Number**>();  
numberList = **new** **ArrayList**<**Integer**>();  
numberList = **new** **ArrayList**<**Float**>(); *//compilation error*

T – Used to denote type

E – Used to denote element

K – Keys

V - Values

N – for numbers

|  |  |
| --- | --- |
| Generic Term | Meaning |
| Set<E> | Generic Type , E is called formal parameter |
| Set<Integer> | Parametrized type , Integer is actual parameter here |
| <T extends Comparable> | Bounded type parameter |
| <T super Comparable> | Bounded type parameter |
| Set<?> | Unbounded wildcard |
| <? extends T> | Bounded wildcard type |
| <? Super T> | Bounded wildcards |
| Set | Raw type |
| <T extends Comparable<T>> | Recursive type bound |

Read more: <http://javarevisited.blogspot.com/2011/09/generics-java-example-tutorial.html#ixzz44ODaTfbA>

## WHAT IS HEAP SPACE IN JAVA?

When a Java program started Java Virtual Machine gets some memory from Operating System. Java Virtual Machine or JVM uses this memory for all its need and part of this memory is call java heap memory. Heap in Java generally located at bottom of address space and move upwards. whenever we create object using new operator or by any another means object is allocated memory from Heap and When object dies or garbage collected ,memory goes back to Heap space in Java.

### OUTOFMEMORYERROR IN JAVA HEAP

When JVM starts JVM heap space is equal to the initial size of Heap specified by -Xms parameter, as application progress more objects get created and heap space is expanded to accommodate new objects. JVM also run garbage collector periodically to reclaim memory back from dead objects. JVM expands Heap in Java somewhere near to Maximum Heap Size specified by -Xmx and if there is no more memory left for creating new object in java heap , JVM throws java.lang.OutOfMemoryError and your application dies. Before throwing [OutOfMemoryError No Space in Java Heap](http://javarevisited.blogspot.sg/2011/09/javalangoutofmemoryerror-permgen-space.html), JVM tries to run garbage collector to free any available space but even after that not much space available on Heap in Java it results into OutOfMemoryError. To resolve this error you need to understand your application object profile i.e. what kind of object you are creating, which objects are taking how much memory etc. you can use profiler or heap analyzer to troubleshoot OutOfMemoryError in Java. "java.lang.OutOfMemoryError: Java heap space" error messages denotes that Java heap does not have sufficient space and cannot be expanded further while "java.lang.OutOfMemoryError: PermGen space" error message comes when the permanent generation of Java Heap is full, the application will [fail to load a class](http://javarevisited.blogspot.sg/2011/08/classnotfoundexception-in-java-example.html) or to allocate an interned string.

## HOW TO INCREASE HEAP SIZE IN JAVA

Default size of Heap space in Java is 128MB on most of 32 bit Sun's [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) but its highly varies from JVM to JVM e.g. default maximum and start heap size for the 32-bit Solaris Operating System (SPARC Platform Edition) is -Xms=3670K and -Xmx=64M and Default values of heap size parameters on 64-bit systems have been increased up by approximately 30%.

By the way you can increase size of java heap space based on your application need and I always recommend this to avoid using default JVM heap values. if your application is large and lots of object created you can change size of heap space by using JVM options -Xms and -Xmx. Xms denotes starting size of Heap while -Xmx denotes maximum size of Heap in Java. There is another parameter called -Xmn which denotes Size of new generation of Java Heap Space. Only thing is you cannot change the size of Heap in Java dynamically, you can only provide Java Heap Size parameter while starting JVM. I have shared some more useful JVM options related to Java Heap space and Garbage collection on my post [10 JVM options Java programmer must know](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html), you may find useful.

### JAVA HEAP DUMP

Java Heap dump is a snapshot of Java Heap Memory at a particular time. This is very useful to analyze or troubleshoot any memory leak in Java or any java.lang.OutOfMemoryError. There are tools available inside JDK which helps you to take heap dump and there are heap analyzer available tool which helps you to analyze java heap dump. You can use "jmap" command to get java heap dump, this will create heap dump file and then you can use "jhat - Java Heap Analysis Tool" to analyze those heap dumps. You should also read [Java Performance The Definitive Guide By Scott Oaks](http://www.amazon.com/Java-Performance-The-Definitive-Guide/dp/1449358454?tag=javamysqlanta-20) to learn more about Java performance tuning and profiling. It is one of the must read Java performance book for any senior Java developers.

## POINTS ABOUT JAVA HEAP SPACE

1. Java Heap Memory is part of memory allocated to JVM by Operating System.
2. Whenever we create objects they are created inside Heap in Java.
3. Java Heap space is divided into three regions or generation for sake of garbage collection called New Generation, Old or tenured Generation or Perm Space. Permanent generation is garbage collected during full GC in hotspot JVM.
4. You can increase or change size of Java Heap space by using JVM command line option -Xms, -Xmx and -Xmn. don't forget to add word "M" or "G" after specifying size to indicate Mega or Gig. For example you can set java heap size to 258MB by executing following command java -Xmx256m HelloWord.
5. You can use either JConsole or Runtime.maxMemory(), Runtime.totalMemory(), Runtime.freeMemory() to query about Heap size programmatic in Java. See my post [How to find memory usage in Java program](http://javarevisited.blogspot.sg/2012/01/find-max-free-total-memory-in-java.html) for more details.
6. You can use command "jmap" to take Heap dump in Java and "jhat" to analyze that heap dump.
7. Java Heap space is different than Stack which is used to store call hierarchy and local variables.
8. Java Garbage collector is responsible for reclaiming memory from dead object and returning to Java Heap space.
9. Don’t panic when you get java.lang.OutOfMemoryError, sometimes it’s just matter of increasing heap size but if its recurrent then look for [memory leak in Java](http://javarevisited.blogspot.sg/2012/01/tomcat-javalangoutofmemoryerror-permgen.html).
10. Use Profiler and Heap dump Analyzer tool to understand Java Heap space and how much memory is allocated to each object.

Read more: <http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html#ixzz44SDup4FE>

Read more: <http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html#ixzz44SA78Sjk>

# [HOW GARBAGE COLLECTION WORKS IN JAVA](http://javarevisited.blogspot.in/2011/04/garbage-collection-in-java.html)

Garbage collection works by employing several GC algorithm e.g. Mark and Sweep. There are different kinds of garbage collector available in Java to collect different area of heap memory e.g. you have serial, parallel and concurrent garbage collector in Java.

A new collector called G1 (Garbage first) are also introduced in JDK 1.7.  The first step to learning about GC is to understand when an object becomes eligible to garbage collection? Since JVM provides memory management, Java developers only care about creating an object, they don't care about cleaning up, that is done by the garbage collector, but it can only collect objects which have no live strong reference or it's not reachable from any thread.

## IMPORTANT POINTS ABOUT GARBAGE COLLECTION IN JAVA

* 1. Objects are created on the heap in Java irrespective of their scope e.g. local or member variable. while it's worth noting that class variables or static members are created in method area of [Java memory space](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html) and both heap and method area is shared between different thread.
  2. Garbage collection is a mechanism provided by Java Virtual Machine to reclaim heap space from objects which are eligible for Garbage collection.
  3. Garbage collection relieves Java programmer from memory management which is an essential part of C++ programming and gives more time to focus on business logic.
  4. Garbage Collection in Java is carried by a daemon thread called Garbage Collector.
  5. Before removing an object from memory garbage collection thread invokes [finalize() method](http://javarevisited.blogspot.com/2012/03/finalize-method-in-java-tutorial.html) of that object and gives an opportunity to perform any sort of cleanup required.
  6. You as Java programmer cannot force garbage collection in Java; it will only trigger if JVM thinks it needs a garbage collection based on Java heap size.
  7. There are methods like System.gc() and Runtime.gc() which is used to send request of Garbage collection to JVM but it’s not guaranteed that garbage collection will happen.
  8. If there is no memory space for creating a new object in Heap Java Virtual Machine throws OutOfMemoryError or [java.lang.OutOfMemoryError heap space](http://javarevisited.blogspot.com/2011/05/java-heap-space-memory-size-jvm.html)

## WHEN AN OBJECT BECOMES ELIGIBLE FOR GARBAGE COLLECTION

An object becomes eligible for Garbage collection or GC if it's not reachable from any live threads or by any static references. In other words, you can say that an object becomes eligible for garbage collection if its all references are null. Cyclic dependencies are not counted as the reference so if object A has a reference to object B and object B has a reference to Object A and they don't have any other live reference then both Objects A and B will be eligible for Garbage collection. Generally, an object becomes eligible for garbage collection in Java on following cases:

1. All references to that object explicitly set to null e.g. object = null
2. The object is created inside a block and reference goes out scope once control exit that block.
3. Parent object set to null if an object holds the reference to another object and when you set container object's reference null, child or contained object automatically becomes eligible for garbage collection.
4. If an object has only [lived weak references](http://javarevisited.blogspot.sg/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html) via WeakHashMap it will be eligible for garbage collection.

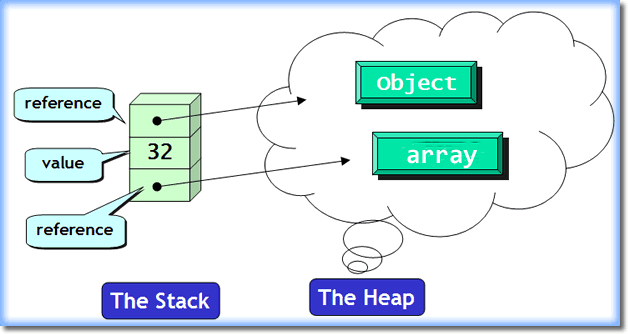
### SUMMARY ON GARBAGE COLLECTION IN JAVA

1. Java Heap is divided into three generation for the sake of garbage collection. These are a young generation, tenured or old generation, and Perm area.
2. New objects are created by young generation and subsequently moved to the old generation.
3. String pool is created in [PermGen area of Heap](http://javarevisited.blogspot.com/2012/01/tomcat-javalangoutofmemoryerror-permgen.html#uds-search-results), garbage collection can occur in perm space but depends upon JVM to JVM. By the way from JDK 1.7 update, String pool is moved to heap area where objects are created.

Read more: <http://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html#ixzz44ZMXe3D3>

## DIFFERENCE BETWEEN STACK VS HEAP IN JAVA

1. The main difference between heap and stack is that stack memory is used to store [local variables](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) and function call while heap memory is used to store objects in Java. No matter, where the object is created in code e.g. as a member variable, local variable or class variable, they are always created inside heap space in Java.
2. Each [Thread in Java](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) has their own stack which can be specified using -Xss JVM parameter, similarly, you can also specify heap size of Java program using JVM option -Xms and -Xmx where -Xms is starting size of the heap and -Xmx is a maximum size of java heap. to learn more about JVM options see my post [10 JVM option Java programmer should know](http://javarevisited.blogspot.com/2011/11/hotspot-jvm-options-java-examples.html).



1. If there is no memory left in the stack for storing function call or local variable, JVM will throw java.lang.StackOverFlowError, while if there is no more heap space for creating an object, JVM will throw java.lang.OutOfMemoryError: Java Heap Space. Read more about how to deal with java.lang.OutOfMemoryError in my post [2 ways to solve OutOfMemoryError in Java](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html).
2. If you are using [Recursion](http://javarevisited.blogspot.com/2012/12/recursion-in-java-with-example-programming.html), on which method calls itself, you can quickly fill up stack memory. Another difference between stack and heap is that size of stack memory is a lot lesser than the size of heap memory in Java.
3. Variables stored in stacks are only visible to the owner Thread while objects created in the heap are visible to all thread. In other words, stack memory is kind of private memory of Java Threads while heap memory is shared among all threads.

Read more: <http://javarevisited.blogspot.com/2013/01/difference-between-stack-and-heap-java.html#ixzz44ZTyJfno>

# [HOW TO CHECK IF STRING IS NOT NULL AND EMPTY IN JAVA?](http://javarevisited.blogspot.in/2016/01/how-to-check-if-string-is-not-null-and-empty-in-java-example.html)

**1st solution - using isEmpty() method**

isEmpty() method is available from Java SE 6 onwards

if(stirng != null && !string.isEmpty()){

System.out.println("String is not null and not empty");

}

**2nd solution - Using length() function**  
if(stirng != null && string.length() > 0){

System.out.println("String is not null and not empty");

}

**3rd solution - Using trim() method**  
if(stirng != null && string.trim().length() > 0){

System.out.println("String is not null and not empty");

}

# DIFFERENCE BETWEEN STRING LITERAL AND NEW STRING OBJECT IN JAVA

String strObject = new String("Java");

and

String strLiteral = "Java";

Both expression gives you String object, but there is subtle difference between them. When you create String object using new() operator, it always create a new object in [heap memory](http://java67.blogspot.sg/2013/08/guide-of-javalangoutofmemoryerror-java-heap-space-tomcat-eclipse-minecraft-jboss.html). On the other hand, if you create object using String literal syntax e.g. "Java", it may return an existing object from String pool (a cache of String object in Perm gen space, which is now moved to heap space in recent Java release), if it's already exists. Otherwise it will create a new string object and put in string pool for future re-use.

## WHAT IS STRING LITERAL AND STRING POOL

Since String is one of the most used type in any application, Java designer took a step further to optimize uses of this class. They know that Strings will not going to be cheap, and that's why they come up with an idea to cache all String instances created inside double quotes e.g. "Java". These double quoted literal is known as *String literal* and the cache which stored these String instances are known as *String pool*. In earlier version of Java, I think up-to Java 1.6 String pool is located in permgen area of heap, but in Java 1.7 updates its moved to main heap area. Earlier since it was in PermGen space, it was always a risk to create too many String object, because it’s a very limited space, default size 64 MB and used to store class metadata e.g. .class files. Creating too many String literals can cause [java.lang.OutOfMemory: permgen space](http://javarevisited.blogspot.sg/2012/01/tomcat-javalangoutofmemoryerror-permgen.html). Now because String pool is moved to a much larger memory space, it's much safer. By the way, don't misuse memory here, always try to minimize temporary String object e.g. "a", "b" and then "ab". Always use StringBuilder to deal with temporary String object.

### STRING INTERNING USING INTERN() METHOD

Java by default doesn't put all String object into String pool, instead they gives you flexibility to explicitly store any arbitrary object in String pool. You can put any object to String pool by calling intern() method of java.lang.String class. Though, when you create using *String literal notation* of Java, it automatically call intern() method to put that object into String pool, provided it was not present in the pool already. This is another *difference between string literal and new string*, because in case of new, interning doesn't happen automatically, until you call intern() method on that object. Also don't forget to use [StringBuffer](http://java67.blogspot.sg/2012/08/difference-between-string-and-stringbuffer-in-java.html) and [StringBuilder](http://java67.blogspot.sg/2014/05/difference-between-stringbuilder-and-StringBuffer-java.html) for string concatenation, they will reduce number.

# [WHAT IS AUTOBOXING AND UNBOXING IN JAVA](http://javarevisited.blogspot.in/2012/07/auto-boxing-and-unboxing-in-java-be.html)

Autoboxing and unboxing is introduced in Java 1.5 to automatically convert primitive type into Object or Wrapper class type called **autoboxing**, in opposite case is called **unboxing**.

**ArrayList**<**Integer**> intList = **new** **ArrayList**<**Integer**>();  
intList.add(1); *//autoboxing - primitive to object*  
intList.add(2); *//autoboxing*  
        
**int** number = intList.get(0); *// unboxing*  
**int** local = intLocal.get(); *// unboxing in Java*

*Or*

*Integer a = 2; //autoboxing - primitive to object*

*int s = a;// unboxing*

*or*

**public** **static** **Integer** show(**Integer** iParam){  
   **System**.out.println("autoboxing example - method invocation i: " + iParam);  
   **return** iParam;  
}

*//autoboxing and unboxing in method invocation*  
show(3); *//autoboxing*  
**int** result = show(3); *//unboxing because return type of method is Integer*

Read more: <http://javarevisited.blogspot.com/2012/07/auto-boxing-and-unboxing-in-java-be.html#ixzz44Ztoq1a2>

## COMPARING INTEGER OBJECT WITH == IN JAVA

Some of the JVM cache objects of some wrapper class e.g. Integer from -128 to 127 and return same object which if compare via “ ==” can return true but after this range this validity doesn't work and to make it worse this behavior is JVM dependent so better avoid this kind of check and use equals() method. e.g.

Integer i1 = **260**;

Integer i2 = **260**;

**if** (i1 == i2) {

System.out.println("i1 and i2 is equal");

} **else** {

System.out.println("i1 and i2 is not equal ");

}

Here you will most probably get **"i1 and i2 is not equal”** at least in my machine.  
Because in this case, unboxing operation is not involved. The literal 260 is boxed into two different Integer objects (again it varies between JVM to JVM), and then those objects are compared with ==. The result is false, as the two objects are different instances, with different memory addresses. Because both sides of the == expression contain objects, no unboxing occurs.

Integer i1 = **100**;

Integer i2 = **100**;

**if** (i1 == i2) {

System.out.println("i1 and i2 is equal");

} **else** {

System.out.println("i1 and i2 is not equal ");

}

Here, most probably you will get the text "i1 and i2 is equal". Because int values from -127 to 127 are in a range which most JVM will like to cache so the VM actually uses the same object instance (and therefore memory address) for both i1 and i2. As a result, == returns a true result.

IS STRING THREAD-SAFE IN JAVA?

If you are familiar with the concept of immutability and [thread-safety](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html) you can easily answer this String interview question in Java. Since [String is immutable](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html), it is thread-safe and it can be shared between multiple threads without external synchronization.

FINDING SUBSTRING IN STRING

String s = "abc xyz bbc anz";

System.***out***.println(s.contains("abc"));

System.***out***.println(s.indexOf("xyz") != -1);

[HOW DOES HASHMAP OR LINKEDHAHSMAP HANDLES COLLISIONS?](http://javarevisited.blogspot.in/2016/01/how-does-java-hashmap-or-linkedhahsmap-handles.html)

A collision occurs when a hash function returns same bucket location for two different keys. Since all hash based Map class e.g. HashMap uses [equals() and hashCode() contract](http://javarevisited.blogspot.com/2015/01/why-override-equals-hashcode-or-tostring-java.html) to find the bucket. HashMap calls the hashCode() method to compute the hash value which is used to find the bucket location.

1. HashMap handles collision by using linked list to store map entries ended up in same array location or bucket location.
2. From Java 8 onwards, HashMap, ConcurrentHashMap, and LinkedHashMap will use the balanced tree in place of [linked list](http://java67.blogspot.com/2016/01/how-to-implement-singly-linked-list-in-java-using-generics-example.html) to handle frequently hash collisions. The idea is to switch to the balanced tree once the number of items in a hash bucket grows beyond a certain threshold. This will improve the worst case get() method performance from O(n) to O(log n).
3. By switching from linked list to balanced tree for handling collision, the iteration order of HashMap will change. This is Ok because HashMap doesn't provide any guarantee on iteration order and any code which depends upon that are likely to break.
4. Legacy class Hashtable which exists in JDK from Java 1 will not use the balanced binary tree to handle frequent hash collision to keep its iteration order intact. This was decided to avoid breaking many legacy Java application which depends upon iteration order of Hashtable.
5. Apart from Hashtable, WeakHashMap and IdentityHashMap will also continue to use the linked list for handling collision even in the case of frequent collisions.
6. Collision in [HashMap](http://java67.blogspot.com/2016/01/how-to-initialize-hashmap-with-values-in-java.html) is possible because hash function uses hashCode() of key object and equals() and hashCode() contract doesn't guarantee different hashCode for different objects. Remember, they guarantee same hash code for the equal object but not the vice-versa.
7. A collision will occur on Hashtable or HashMap when hashCode() method of two different key objects will return same values.

[TOP 5 CONCURRENT COLLECTIONS FROM JDK 5 & 6 JAVA PROGRAMMER SHOULD KNOW](http://javarevisited.blogspot.in/2013/02/concurrent-collections-from-jdk-56-java-example-tutorial.html)

Several new Collection classes are added in Java 5 and Java 6 specially concurrent alternatives of standard [synchronized ArrayList](http://javarevisited.blogspot.com/2011/05/example-of-arraylist-in-java-tutorial.html), [Hashtable](http://javarevisited.blogspot.sg/2012/01/java-hashtable-example-tutorial-code.html) and [synchronized HashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html) collection classes. Many Java programmer still not familiar with these new collection classes from java.util.concurrent package and misses a whole new set of functionality which can be utilized to build more scalable and high performance Java application. In this Java tutorial we will some of useful collection classes e.g. [ConcurrentHashMap](http://javarevisited.blogspot.sg/2011/04/difference-between-concurrenthashmap.html), [BlockingQueue](http://javarevisited.blogspot.com/2012/12/blocking-queue-in-java-example-ArrayBlockingQueue-LinkedBlockingQueue.html) which provides some of the very useful functionalities to build concurrent Java application.

**1. ConcurrentHashMap**

ConcurrentHashMap is undoubtedly most popular collection class introduced in Java 5 and most of us are already using it. ConcurrentHashMap provides a concurrent alternative of [Hashtable or Synchronized Map](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) classes with aim to support higher level of concurrency by implementing fined grained locking. Multiple reader can access the Map concurrently while a portion of Map gets locked for write operation depends upon concurrency level of Map. ConcurrentHashMap provides better scalability than there synchronized counterpart. [Iterator](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) of ConcurrentHashMap are [fail-safe iterators](http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) which doesn't throw ConcurrencModificationException thus eliminates another requirement of locking during iteration which result in further scalability and performance.

### **Summary**

Now we know what is ConcurrentHashMap in Java and when to use ConcurrentHashMap, it’s time to know and revise some important points about CHM in Java.

1. ConcurrentHashMap allows concurrent read and thread-safe update operation.
2. During the update operation, ConcurrentHashMap only locks a portion of Map instead of whole Map.
3. The concurrent update is achieved by internally dividing Map into the small portion which is defined by concurrency level.
4. Choose concurrency level carefully as a significantly higher number can be a waste of time and space and the lower number may introduce thread contention in case writers over number concurrency level.
5. All operations of ConcurrentHashMap are [thread-safe](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html).
6. Since ConcurrentHashMap implementation doesn't lock whole Map, there is chance of read overlapping with update operations like put() and remove(). In that case result returned by get() method will reflect most recently completed operation from there start.
7. Iterator returned by ConcurrentHashMap is weekly consistent, [fail-safe](http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and never throw ConcurrencModificationException. In Java.
8. ConcurrentHashMap doesn't allow null as key or value.
9. You can use ConcurrentHashMap in place of [Hashtable](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html) but with caution as CHM doesn't lock whole Map.
10. During putAll() and clear() operations, the concurrent read may only reflect insertion or deletion of some entries.

**2. CopyOnWriteArrayList and CopyOnWriteArraySet**

CopyOnWriteArrayList is a concurrent alternative of synchronized List. CopyOnWriteArrayList provides better concurrency than [synchronized](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) List by allowing multiple concurrent reader and replacing the whole list on write operation. Yes, write operation is costly on CopyOnWriteArrayList but it performs better when there are multiple reader and requirement of iteration is more than writing. Since CopyOnWriteArrayList Iterator also don't throw ConcurrencModificationException it eliminates need to lock the collection during iteration. Remember both ConcurrentHashMap and CopyOnWriteArrayList doesn't provides same level of locking as Synchronized Collection and achieves [thread-safety by](http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html) there locking and mutability strategy. So they perform better if requirements suits there nature. Similarly, CopyOnWriteArraySet is a concurrent replacement to Synchronized Set. See [What is CopyOnWriteArrayList in Java](http://java67.blogspot.com/2012/09/what-is-copyonwritearraylist-in-java-example-vs-arraylist.html) for more details