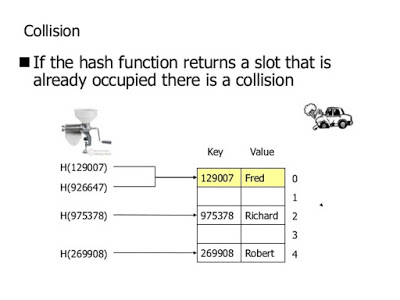
**[How does Java HashMap or LinkedHahsMap handles collisions?](https://javarevisited.blogspot.com/2016/01/how-does-java-hashmap-or-linkedhahsmap-handles.html" \o "How does Java HashMap or LinkedHahsMap handles collisions?)**

Prior to Java 8, HashMap and all other hash table based Map implementation classes in Java handle collision by *chaining*, i.e. they use [linked list](http://javarevisited.blogspot.com/2015/02/simple-junit-example-unit-tests-for-linked-list-java.html) to store map entries which ended in the same bucket due to a collision. If a key end up in same bucket location where an entry is already stored then this entry is just added at the head of the linked list there. In the worst case this degrades the performance of the [get() method of HashMap](http://java67.blogspot.com/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) to O(n) from O(1). In order to address this issue in the case of frequent HashMap collisions, Java8 has started using a balanced tree instead of linked list for storing collided entries. This also means that in the worst case you will get a performance boost from **O(n)** to **O(log n)**.

The threshold of switching to the balanced tree is defined as TREEIFY\_THRESHOLD constant in java.util.HashMap JDK 8 code.  Currently, it's value is 8, which means if there are more than 8 elements in the same bucket than HashMap will use a tree instead of linked list to hold them in the same bucket.

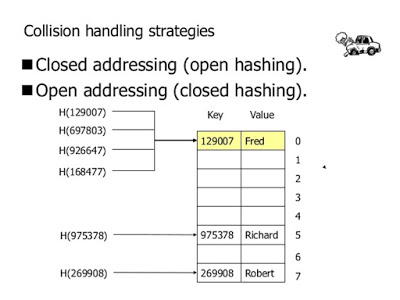
This change in continuation of efforts to improve most used classes. If you remember earlier in JDK 7 they have also introduced a [change](http://javarevisited.blogspot.com/2014/07/java-optimization-empty-arraylist-and-Hashmap-cost-less-memory-jdk-17040-update.html)so that empty ArrayList and HashMap will take less memory by postponing the allocation of the underlying array until an element is added.

This is a dynamic feature which means HashMap will initially use the linked list but when the number of entries crosses a certain threshold it will replace the linked list with a balanced binary tree. Also, this feature will not available to all hash table based classes in Java e.g. **Hashtable will not have this feature** because of its legacy nature and given that this feature can change the traditional legacy [iteration order of Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html). Similarly, **WeakHashMap** will also not include this feature.  
  
So far (until JDK 8) only ConcurrentHashMap, LinkedHashMap and HashMap will use the balanced tree in case of a frequent collision.This is a dynamic feature which means [HashMap will initially use the linked list](http://java67.blogspot.com/2013/02/10-examples-of-hashmap-in-java-programming-tutorial.html)but when the number of entries crosses a certain threshold it will replace the linked list with a balanced binary tree.

[](https://4.bp.blogspot.com/-h97jzqdIfK0/Vo6HLxwSDYI/AAAAAAAAEhE/RK8U_n4eA6w/s1600/What%2Bis%2BCollision%2Bin%2BJava.jpg)

**When does collision occur in HashMap**

There are several class in JDK which are based upon the hash table data structure e.g.  
[HashMap](http://javarevisited.blogspot.com/2011/12/how-to-traverse-or-loop-hashmap-in-java.html),  
[LinkedHashMap](http://javarevisited.blogspot.com/2015/08/difference-between-HashMap-vs-TreeMap-vs-LinkedHashMap-Java.html),  
[Hashtable](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html),  
[WeakHashMap](http://javarevisited.blogspot.com/2014/03/difference-between-weakreference-vs-softreference-phantom-strong-reference-java.html),  
[IdentityHashMap](http://javarevisited.blogspot.com/2013/01/difference-between-identityhashmap-and-hashmap-java.html),  
[ConcurrentHashMap](http://javarevisited.blogspot.com/2013/02/concurrent-collections-from-jdk-56-java-example-tutorial.html),   
[TreeMap](http://javarevisited.blogspot.com/2011/12/treemap-java-tutorial-example-program.html), and  
[EnumMap](http://javarevisited.blogspot.com/2012/09/what-is-enummap-in-java-example-tutorial.html).  
  
Underlying working of all these Map is pretty much same as discussed in [How does HashMap internally works in Java](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html), except some minor differences in their specific behaviors. Since hash table data structure is subject to collision all these implementations are required to handle the collision.  
  
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 as shown in below code snippet from the HashMap class of JDK 1.7 (jkd1.7.0\_60) update.  
  
Ignoring the first two lines, which was the performance improvement done for String keys in JDK 7, you can see that computation of hash is totally based upon the hashCode method.  
  
A collision will occur when two different keys have the same hashCode, which can happen because two unequal objects in Java can have the same hashCode.

[](https://2.bp.blogspot.com/-N_JzqRSxAlg/Vo6HRVsklDI/AAAAAAAAEhM/JIyM1yKjhUQ/s1600/Collision%2Bhandling%2Bstrategies%2Bin%2BHashMap%2BJava.jpg)

**Summary**

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.  
  
  
That's all about **how HashMap in Java handles collisions**. In general, this method is called chaining because all objects stored in the same bucket are chained as linked list. In general, all hash table based classes in Java e.g. HashMap, HashSet, LinkedHashSet, LinkedHashMap, ConcurrentHsahMap, Hashtable, IdentityHashMap and WeakHashMaap uses linked list to handle collisions.  
  
From JDK 8, a balanced tree will be used to implement chaining instead of linked list to improve worst case performance of HashMap from **O(n)** to O(log n) for HashMap, LinkedHashMap, and ConcurrentHashMap. Since HashSet internally uses HashMap and LinkedHashSet internally uses LinkedHashMap they will also benefit from this performance improvement.