HashMap changes in Java 8

Hash elements use balanced trees instead of linked lists under certain circumstances now.

number of items in a hash is larger than a certain value, the hash will change from using a linked list of elements or entries to a balanced tree, this will improve the worst case performance from O(n) to O(log n).  
   
 

The fix has been implemented in the classes java.util.HashMap, java.util.LinkedHashMap and java.util.concurrent.ConcurrentHashMap. No interfaces or methods specifications have been changed, only the behavior in the implementation of the concurrent hash map is different. So there is no need to change the applications using these classes

Here is a list of classes implementing hash maps have not changed in relation to this fix:

* java.util.concurrent.ConcurrentHashMap already contains this implementation. Portions of the code already used in this class have been reused in the changes explained above.
* The java.util.HashTable class (present since java 1) has not been changed with this new technique. The main reason for that is that some legacy code uses and expects the historical iteration order of the java.util.Hashtable class.
* The class java.util.WeakHashMap does not contain this change neither because of the complexity would be to high and is not worth it.
* The class java.util.IdentityHashMap does not need this improvement. This class generates hash codes by using the System.identityHashCode() method and collisions are very rare or non existent.

java.util.LinkedHashMap should be used, since this class guarantees the iteration order.

2.1. O(n) performance

The big-O notation is a measure of complexity for a given algorithm. “n” is the amount of data used in the algorithm. It indicates the amount of time the algorithm will take when n tends to infinitive. O(2n) or O(constant \* n) do not exist, O(1) means constant time (performance is not related to the data that is processed) and O(n) means that the performance is directly related or proportional to the amount of data that is processed.

2.2. O(log n) performance

In this case, it means that the algorithm will perform better when the amount of data is larger. Performance is not directly proportional to the large of the processed data but in a log n relation. O(log n) performs better than O(n).

You can find several good articles, discussions and books about algorithm performance and measures, here are a couple of links:

* <http://stackoverflow.com/questions/4776219/algorithm-performance-explanation-ex-on>
* <http://www.perlmonks.org/?node_id=227909>

2.3. Balanced trees

A tree is balanced if the left and the right sub trees are balanced (recursion!) and their height differ by at most one. The main goal is to keep the depths of all nodes to be O(log n). The maintenance of the balanced tree has a penalty in the insertion of new elements but improve the indexing and accessing performance.

This article contains a lot of information about balanced trees: <http://webdocs.cs.ualberta.ca/~holte/T26/balanced-trees.html>.

4. Links

More information about this Java 8 enhancement, its causes and consequences and details related to Maps improvements and changes in Java8:

* <http://openjdk.java.net/jeps/180>
* <http://hg.openjdk.java.net/jdk8/jdk8/jdk/rev/43bd5ee0205e>
* <https://docs.oracle.com/javase/8/docs/technotes/guides/collections/changes8.html>

Download the Source Code

So in this example we show some improvements about HashMap implementation in Java 8.

**Download**  
You can download the full source code of this example here: [**hashMapJava8.zip**](http://examples.javacodegeeks.com/wp-content/uploads/2014/11/hashMapJava8.zip)