<https://examples.javacodegeeks.com/core-java/util/hashmap/hashmap-changes-in-java-8/>

<http://winterbe.com/posts/2015/05/22/java8-concurrency-tutorial-atomic-concurrent-map-examples/>

The way java.util.HashMap entries are indexed and stored has changed in the Java 8 update.  
Hash elements use balanced trees instead of linked lists under certain circumstances now.  
http://coding-geek.com/how-does-a-hashmap-work-in-java/

Till Java7, HashMap used to keep an array of buckets. Each bucket used to have elements stored as linked list. LinkedList takes O(n) for access.

In Java8, code has become more complex. Number of lines in the code has been doubled. So, what's good about Java8's HashMap?  
It keeps Balanced Search Trees(BSTs) in the buckets instead of LinkedLists. Search in BST takes O(log n), which is a lot less than LinkedList.  
  
- If for a given index (bucket) in the inner table there are more than 8 nodes, the linked list is transformed into a red black tree

- If for a given index (bucket) in the inner table there are less than 6 nodes, the tree is transformed into a linked list  
  
The fix has been implemented in the classes  
 java.util.HashMap,  
 java.util.LinkedHashMap and  
 java.util.concurrent.ConcurrentHashMap  
They haven't considered WeakHashMap for this optimization.  
  
newly added methods

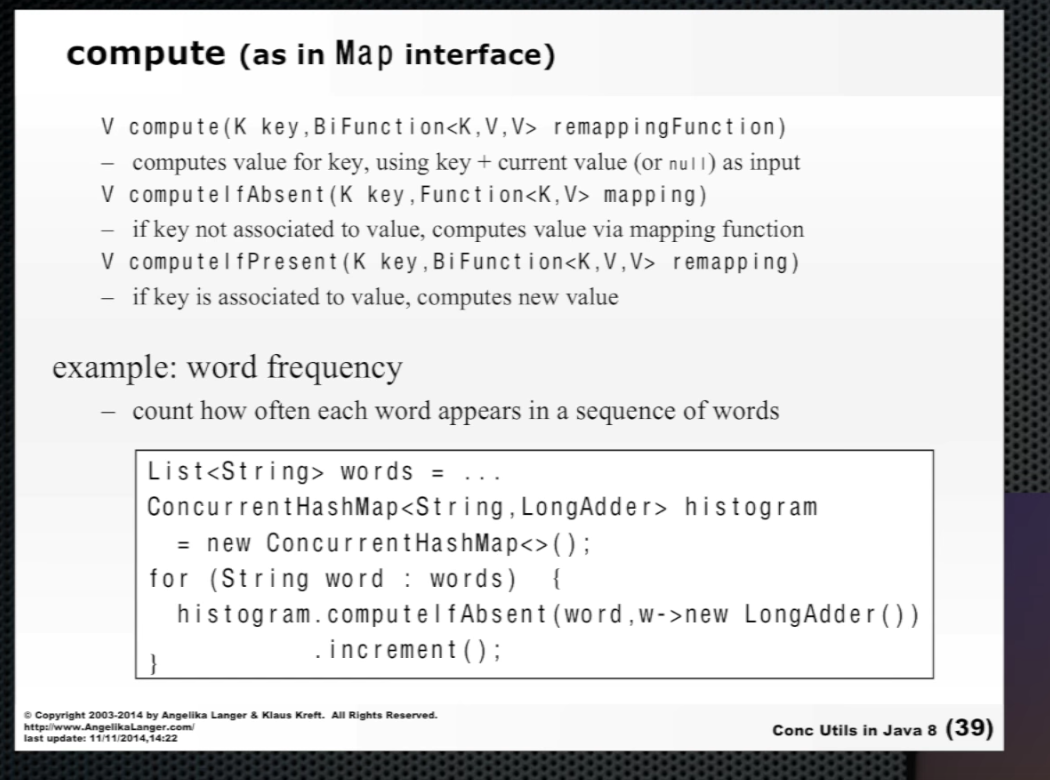
forEach(consumer)

search

reduce  
 replace(key, value)  
 replaceAll(key, function)  
 compute(key, BiFunction)  
 computeIfAbsent(key, Function)  
 computeIfPresent(key, BiFunction)  
 merge

compute methods take a function as parameter that takes key and value and returns a computed value.

<https://www.youtube.com/watch?v=Q_0_1mKTlnY>



computeIfAbsent is a special method that is related to Memoization.

Integer doubleValue(Integer x) {  
 return x \* 2;  
}  
  
Memoized form of above method:

Normally, you use Map to memorize the result.  
  
Map<Integer, Integer> cache = new ConcurrentHashMap<>();  
Integer doubleValue(Integer x) {  
 if (cache.containsKey(x)) {  
 return cache.get(x);  
 } else {  
 Integer result = x \* 2;  
 cache.put(x, result) ;  
 return result;  
 }  
}  
  
Using Java 8, you can even make it simpler.  
  
Map<Integer, Integer> cache = new ConcurrentHashMap<>();  
Integer doubleValue(Integer x) {  
 return cache.computeIfAbsent(x, x1 -> x1 \* 2);  
}

**merge**

<http://www.buggybread.com/2014/10/java-8-map-merge-method.html>

Merge the specified value to the existing Value using the Specified function for the Specified Key.

merge([Key],[New Value],[Function([Current Value],[New Value]]);

Map<String,String> strMap = new HashMap<String,String>();  
strMap.put("Key1","Value1");  
strMap.put("Key2", "Value2");

String str = strMap.merge("Key1","Value56",(v1,v2)->v1.substring(3).concat(v2));

System.out.println(str); // prints ue1Value56  
System.out.println(strMap); // prints {Key2=Value2, Key1=ue1Value56}

