Hashtable class

Hashtable is *legacy* class and based on *hashcode of keys* where *keys* are *unique* and it is exactly same as *HashMap* with few differences with respect to following points

- Synchronized methods
- Performance
- Null key insertion
- Null value insertion

Hashtable:

- Java Hashtable is the *implementation* class of Map interface (i.e.; Hashtable implements Map)
- Java Hashtable is *legacy class* introduced in *Java 1.0* version
- Doesn't allows *NULL insertion* for *keys* and *values*
- Hashtable is *synchronized,* all methods of Java Hashtable is thread-safe (i.e.; only one thread is allowed to access, at a any given point of time)
- Present in *java.util* package and extends *java.util.Dictionary* abstract class implements *java.util.Map* interface
- Also, implements *java.lang.Cloneable*, *java.io.Serializable* marker interfaces which provides special ability to Hashtable (provided by JVM at run time)

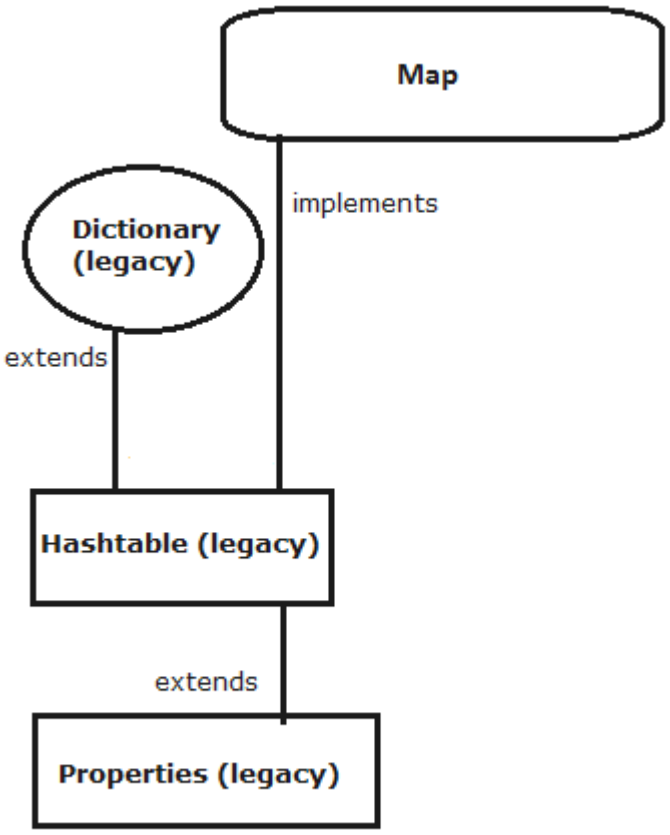

Figure: Hashtable in Java

Hashtable constructors:

Hashtable ht = new Hashtable();

- creates an empty Hashtable object of ***size 11*** with ***default fill ratio of 0.75***

Hashtable ht = new Hashtable(int initialCapacity);

- creates an empty Hashtable object of ***specified size*** (or initial capacity) with ***default fill ratio of 0.75***

Hashtable ht = new Hashtable(int initialCapacity, float loadFactor);

- creates an empty Hashtable object of ***specified size*** (or initial capacity) and ***specified fill ratio*** (for example 0.85)

Hashtable ht = new Hashtable(Map m);

- creates an ***equivalent Hashtable*** object for the ***specified map***
- it is basically used for ***inter-conversion*** between map objects

Fill ratio (or Load factor)

- Fill ratio is also known as ***Load factor***
- This factor determines ***when to increase the size*** of Java Hashtable automatically
- For example, for the 1st two constructors the ***default load factor is 0.75*** –> which means after filling 75% of original Java Hashtable, ***new Hashtable of bigger size*** will be created
- For 3rd constructor, ***programmer can define load factor*** while creating Hashtable object. If programmer defines it to be 0.95, then after filling 95% of Hashtable, size of Hashtable will be increased automatically
- The value of Load factor should be in between ***0 to 1.0***

Hashtable examples:

```
1  package in.bench.resources.java.collection;
2
3  import java.util.Hashtable;
4  import java.util.Iterator;
5  import java.util.Map;
6  import java.util.Set;
7
8  public class HashtableAddAndRemove {
9
10     public static void main(String[] args) {
11
12         // creating Hashtable object of type <Integer, String>
13         Hashtable<Integer, String> hm = new Hashtable<Integer, String>();
14
15         // adding key-value pairs to Hashtable object
16         hm.put(1, "Google");
17         hm.put(2, "Facebook");
18         hm.put(3, "Yahoo");
19         hm.put(4, "Amazon");
20         hm.put(5, "Reddit");
21
22         System.out.println("Printing all key-value pairs inside {}\n"
23                 + hm + "\n");
24
25         System.out.println("\nIterating using keySet\n");
```

```
26
27          // Iterating key-pairs using keySet
28          Set<Integer> keys = hm.keySet();
29          for(Integer key : keys) {
30              System.out.println(key + "   " + hm.get(key));
31          }
32
33          System.out.println("\n\nIterating using Map Entry interface\n");
34
35          // Iterating key-pairs using Map entry
36          Set set = hm.entrySet();
37          Iterator iterator = set.iterator();
38
39          while(iterator.hasNext()) {
40
41              Map.Entry mapEntry = (Map.Entry)iterator.next();
42              System.out.println(mapEntry.getKey() + "   "
43                      + mapEntry.getValue());
44          }
45
46          // removing map entry at 4th position
47          System.out.println("\n\nEntry removed at 4th position : "
48                  + hm.remove(4));
49      }
50  }
```

Output:

```
1   Printing all key-value pairs inside {}
2   {5=Reddit, 4=Amazon, 1=Google, 3=Yahoo, 2=Facebook}
3
4
5   Iterating using keySet
6
7   5   Reddit
8   4   Amazon
9   1   Google
10  3   Yahoo
11  2   Facebook
12
13
14  Iterating using Map Entry interface
15
16  5   Reddit
17  4   Amazon
18  1   Google
19  3   Yahoo
20  2   Facebook
21
22
23  Entry removed at 4th position : Amazon
```