**Java HashMap/Hashtable, LinkedHashMap and TreeMap**

[**<<Previous**](http://www.w3resource.com/java-tutorial/java-linked-hashset.php)[**Next>>**](http://www.w3resource.com/java-tutorial/java-collections-utility-class.php)

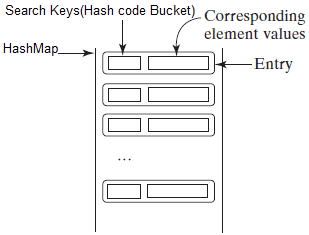
**Introduction**

The basic idea of a map is that it maintains key-value associations (pairs) so you can look up a value using a key. In this tutortial we will discuss Java HashMap/Hashtable, LinkedHashMap and TreeMap.

**HashMap/Hashtable**

HashMap has implementation based on a hash table. (Use this class instead of Hashtable which is legacy class) .The HashMap gives you an unsorted, unordered Map. When you need a Map and you don't care about the order (when you iterate through it), then HashMap is right choice. Keys of HashMap is like Set means no duplicates allowed and unordered while values can be any object even null or duplicate is also allowed. HashMap is very much similar to Hashtable only difference is Hashtable has all method synchronized for thread safety while HashMap has non-synchronized methods for better performance.

We can visualize HashMap as below diagram where we have keys as per hash-code and corresponding values.



HashMap provides constant-time performance for inserting and locating pairs. Performance can be adjusted via constructors that allow you to set the capacity and load factor of the hash table.

**HashMap Constructors**

HashMap( )

Default HashMap Constructor (with default capacity of 16 and load factor 0.75)

HashMap(Map<? extends KeyObject, ? extends ValueObject> m)

This is used to create HashMap based on existing map implementation m.

HashMap(int capacity)

This is used to initialize HashMap with capacity and default load factor.

HashMap(int capacity, float loadFactor)

This is used to initialize HashMap with capacity and custom load factor.

The basic operations of HashMap (put, get, containsKey, containsValue, size, and is Empty) behave exactly like their counterparts in Hashtable. HashMap has**toString( )** method overridden to print the key-value pairs easily. The following program illustrates HashMap. It maps names to salary. Notice how a set-view is obtained and used.

**Java Code**

[view plaincopy to clipboardprint?](http://www.w3resource.com/java-tutorial/java-maps.php)

1. **package** hashmap;


5. **import** java.util.\*;

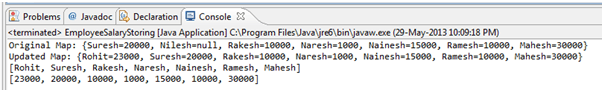

9. **public** **class** EmployeeSalaryStoring {


13. **public** **static** **void** main(String[] args) {
15. //Below Line will create HashMap with initial size 10 and 0.5 load factor
17. Map<String, Integer>empSal = **new** HashMap<String, Integer>(10, 0.5f);
19. //Adding employee name and salary to map
21. empSal.put("Ramesh", 10000);
23. empSal.put("Suresh", 20000);
25. empSal.put("Mahesh", 30000);
27. empSal.put("Naresh", 1000);
29. empSal.put("Nainesh", 15000);
31. empSal.put("Rakesh", 10000); // Duplicate Value also allowed but Keys should not be duplicate
33. empSal.put("Nilesh", **null**); //Value can be null as well


37. System.out.println("Original Map: "+ empSal);// Printing full Map


41. //Adding new employee the Map to see ordering of object changes
43. empSal.put("Rohit", 23000);
45. //Removing one key-value pair
47. empSal.remove("Nilesh");
49. System.out.println("Updated Map: "+empSal);// Printing full Map
51. //Printing all Keys
53. System.out.println(empSal.keySet());
55. //Printing all Values
57. System.out.println(empSal.values());
59. }
61. }

**Output**



**Java LinkedHashMap**

LinkedHashMap extends HashMap. It maintains a linked list of the entries in the map, in the order in which they were inserted. This allows insertion-order iteration over the map. That is,when iterating through a collection-view of a LinkedHashMap, the elements will be returned in the order in which they were inserted. Also if one inserts the key again into the LinkedHashMap the original orders is retained. This allows insertion-order iteration over the map. That is, when iterating a LinkedHashMap, the elements will be returned in the order in which they were inserted. You can also create a LinkedHashMap that returns its elements in the order in which they were last accessed.

**Constructors**

LinkedHashMap( )

This constructor constructs an empty insertion-ordered LinkedHashMap instance with the default initial capacity (16) and load factor (0.75).

LinkedHashMap(int capacity)

This constructor constructs an empty LinkedHashMap with the specified initial capacity.

LinkedHashMap(int capacity, float fillRatio)

This constructor constructs an empty LinkedHashMapwith the specified initial capacity and load factor.

LinkedHashMap(Map m)

This constructor constructs a insertion-ordered Linked HashMap with the same mappings as the specified Map.

LinkedHashMap(int capacity, float fillRatio, boolean Order)

This constructor construct an empty LinkedHashMap instance with the specified initial capacity, load factor and ordering mode.

Important methods supported by LinkedHashMap

Class clear( )

Removes all mappings from the map.

containsValue(object value )>

Returns true if this map maps one or more keys to the specified value.

get(Object key)

Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

removeEldestEntry(Map.Entry eldest)

Returns true if this map should remove its eldest entry.

Java Program demonstrate use of LinkedHashMap:

**Java Code**

[view plaincopy to clipboardprint?](http://www.w3resource.com/java-tutorial/java-maps.php)

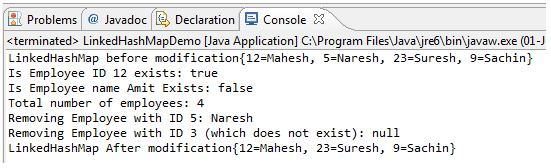
1. **package** linkedhashmap;


5. **import** java.util.LinkedHashMap;
7. **import** java.util.Map;


11. **public** **class** LinkedHashMapDemo {
13. **public** **static** **void** main (String args[]){
15. //Here Insertion order maintains
17. Map<Integer, String>lmap = **new** LinkedHashMap<Integer, String>();


21. lmap.put(12, "Mahesh");
23. lmap.put(5, "Naresh");
25. lmap.put(23, "Suresh");
27. lmap.put(9, "Sachin");
29. System.out.println("LinkedHashMap before modification" + lmap);
31. System.out.println("Is Employee ID 12 exists: " +lmap.containsKey(12));
33. System.out.println("Is Employee name Amit Exists: "+lmap.containsValue("Amit"));
35. System.out.println("Total number of employees: "+ lmap.size());
37. System.out.println("Removing Employee with ID 5: " + lmap.remove(5));
39. System.out.println("Removing Employee with ID 3 (which does not exist): " + lmap.remove(3));
41. System.out.println("LinkedHashMap After modification" + lmap);
43. }
45. }

**Output**



**Java TreeMap**

A TreeMap is a Map that maintains its entries in ascending order, sorted according to the keys' natural ordering, or according to a Comparator provided at the time of the TreeMap constructor argument.The TreeMap class is efficient for traversing the keys in a sorted order. The keys can be sorted using the Comparable interface or the Comparator interface. SortedMap is a subinterface of Map, which guarantees that the entries in the map are sorted. Additionally, it provides the methods firstKey() and lastKey() for returning the first and last keys in the map, and headMap(toKey) and tailMap(fromKey) for returning a portion of the map whose keys are less than toKey and greater than or equal to fromKey.

**TreeMap Constructors**

TreeMap( )

Default TreeMap Constructor

TreeMap(Map m)

This is used to create TreeMap based on existing map implementation m.

TreeMap(SortedMap m)

This is used to create TreeMap based on existing map implementation m.

TreeMap(Comparator ()

This is used to create TreeMap with ordering based on comparator output.

Java Program which explain some important methods of tree map.

**Java Code**

[view plaincopy to clipboardprint?](http://www.w3resource.com/java-tutorial/java-maps.php)

1. **package** treemap;


5. **import** java.util.Map;
7. **import** java.util.TreeMap;

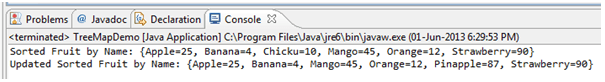

11. **public** **class** TreeMapDemo {


15. **public** **static** **void** main(String[] args) {
17. //Creating Map of Fruit and price of it
19. Map<String, Integer> tMap = **new** TreeMap<String, Integer>();
21. tMap.put("Orange", 12);
23. tMap.put("Apple", 25);
25. tMap.put("Mango", 45);
27. tMap.put("Chicku", 10);
29. tMap.put("Banana", 4);
31. tMap.put("Strawberry", 90);


35. System.out.println("Sorted Fruit by Name: "+tMap);
37. tMap.put("Pinapple", 87);
39. tMap.remove("Chicku");
41. System.out.println("Updated Sorted Fruit by Name: "+tMap);


45. }
47. }

**Output**



**Java Code**

[view plaincopy to clipboardprint?](http://www.w3resource.com/java-tutorial/java-maps.php)

1. **package** treemap;


5. **import** java.util.\*;

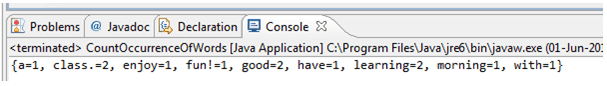

9. **public** **class** CountOccurrenceOfWords {
11. **public** **static** **void** main(String[] args) {
13. // Set text in a string
15. String text = "Good morning class. Have a good learning class. Enjoy learning with fun!";


19. // Create a TreeMap to hold words as key and count as value
21. TreeMap<String, Integer> map = **new** TreeMap<String, Integer>();


25. String[] words = text.split(" "); //Splitting sentance based on String
27. **for** (**int** i = 0; i < words.length; i++) {
29. String key = words[i].toLowerCase();


33. **if** (key.length() > 0) {
35. **if** (map.get(key) == **null**) {
37. map.put(key, 1);
39. } **else** {
41. **int** value = map.get(key).intValue();
43. value++;
45. map.put(key, value);
47. }
49. }
51. }
53. System.out.println(map);
55. }
57. }

**Output**



**Summary**

* Map is collection of key-value pair (associate) objects collection
* HashMap allows one null key but Hashtable does not allow any null keys.
* Values in HashMap can be null or duplicate but keys have to be unique.
* Iteration order is not constant in case of HashMap.
* When we need to maintain insertion order while iterating we should use LinkedHashMap.
* LinkedHashMap provides all the methods same as HashMap.
* LinkedHashMap is not thread safe.
* TreeMap has faster iteration speed compare to other map implementation.
* TreeMap is sorted order collection either natural ordering or custom ordering as per comparator.