**Java Collections – Map**

A Map is an object that maps keys to values. A map cannot contain duplicate keys. There are three main implementations of Map interfaces: HashMap, TreeMap, and LinkedHashMap.  
HashMap: it makes no guarantees concerning the order of iteration  
TreeMap: It stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashMap.  
LinkedHashMap: It orders its elements based on the order in which they were inserted into the set (insertion-order).

**HashMap in Java with Example**

HashMap is a Map based collection class that is used for storing Key & value pairs, it is denoted as HashMap<Key, Value> or HashMap<K, V>. This class makes no guarantees as to the order of the map. It is similar to the Hashtable class except that it is unsynchronized and permits nulls(null values and null key).

It is not an ordered collection which means it does not return the keys and values in the same order in which they have been inserted into the HashMap. It does not sort the stored keys and Values. You must need to import java.util.HashMap or its super class in order to use the HashMap class and methods.

**HashMap Example in Java:**

In this example we have demonstrated almost all the important methods of HashMap class.

import java.util.HashMap;

import java.util.Map;

import java.util.Iterator;

import java.util.Set;

public class Details {

public static void main(String args[]) {

/\* This is how to declare HashMap \*/

HashMap<Integer, String> hmap = new HashMap<Integer, String>();

/\*Adding elements to HashMap\*/

hmap.put(12, "Chaitanya");

hmap.put(2, "Rahul");

hmap.put(7, "Singh");

hmap.put(49, "Ajeet");

hmap.put(3, "Anuj");

/\* Display content using Iterator\*/

Set set = hmap.entrySet();

Iterator iterator = set.iterator();

while(iterator.hasNext()) {

Map.Entry mentry = (Map.Entry)iterator.next();

System.out.print("key is: "+ mentry.getKey() + " & Value is: ");

System.out.println(mentry.getValue());

}

/\* Get values based on key\*/

String var= hmap.get(2);

System.out.println("Value at index 2 is: "+var);

/\* Remove values based on key\*/

hmap.remove(3);

System.out.println("Map key and values after removal:");

Set set2 = hmap.entrySet();

Iterator iterator2 = set2.iterator();

while(iterator2.hasNext()) {

Map.Entry mentry2 = (Map.Entry)iterator2.next();

System.out.print("Key is: "+mentry2.getKey() + " & Value is: ");

System.out.println(mentry2.getValue());

}

}

}

**Output:**

key is: 49 & Value is: Ajeet

key is: 2 & Value is: Rahul

key is: 3 & Value is: Anuj

key is: 7 & Value is: Singh

key is: 12 & Value is: Chaitanya

Value at index 2 is: Rahul

Map key and values after removal:

Key is: 49 & Value is: Ajeet

Key is: 2 & Value is: Rahul

Key is: 7 & Value is: Singh

Key is: 12 & Value is: Chaitanya

**HashMap Class Methods**

Here is the list of methods available in HashMap class. I have also covered examples using these methods at the end of this post.

1. **void clear()**: It removes all the key and value pairs from the specified Map.
2. **Object clone()**: It returns a copy of all the mappings of a map and used for cloning them into another map.
3. **boolean containsKey(Object key)**: It is a boolean function which returns true or false based on whether the specified key is found in the map.
4. **boolean containsValue(Object Value)**: Similar to containsKey() method, however it looks for the specified value instead of key.
5. **Value get(Object key)**: It returns the value for the specified key.
6. **boolean isEmpty()**: It checks whether the map is empty. If there are no key-value mapping present in the map then this function returns true else false.
7. **Set keySet()**: It returns the Set of the keys fetched from the map.
8. **value put(Key k, Value v)**: Inserts key value mapping into the map. Used in the above example.
9. **int size()**: Returns the size of the map – Number of key-value mappings.
10. **Collection values()**: It returns a collection of values of map.
11. **Value remove(Object key)**: It removes the key-value pair for the specified key. Used in the above example.
12. **void putAll(Map m)**: Copies all the elements of a map to the another specified map.

**TreeMap in Java with Example**

TreeMap is Red-Black tree based NavigableMap implementation. It is sorted according to the natural ordering of its keys.  
**TreeMap class** implements Map interface similar to HashMap class. The main difference between them is that HashMap is an unordered collection while TreeMap is sorted in the ascending order of its keys. TreeMap is unsynchronized collection class which means it is not suitable for thread-safe operations until unless synchronized explicitly.

**TreeMap Example**

In this example we are storing the key and value mappings into the TreeMap and we are getting a sorted key-value mapping upon fetching the data from TreeMap.

import java.util.TreeMap;

import java.util.Set;

import java.util.Iterator;

import java.util.Map;

public class Details {

public static void main(String args[]) {

/\* This is how to declare TreeMap \*/

TreeMap<Integer, String> tmap =

new TreeMap<Integer, String>();

/\*Adding elements to TreeMap\*/

tmap.put(1, "Data1");

tmap.put(23, "Data2");

tmap.put(70, "Data3");

tmap.put(4, "Data4");

tmap.put(2, "Data5");

/\* Display content using Iterator\*/

Set set = tmap.entrySet();

Iterator iterator = set.iterator();

while(iterator.hasNext()) {

Map.Entry mentry = (Map.Entry)iterator.next();

System.out.print("key is: "+ mentry.getKey() + " & Value is: ");

System.out.println(mentry.getValue());

}

}

}

**Output:**

key is: 1 & Value is: Data1

key is: 2 & Value is: Data5

key is: 4 & Value is: Data4

key is: 23 & Value is: Data2

key is: 70 & Value is: Data3

As you can see that we have inserted the data in random order however when we displayed the TreeMap content we got the sorted result in the ascending order of keys.

**LinkedHashMap in Java**

LinkedHashMap is a Hash table and linked list implementation of the Map interface, with predictable iteration order. This implementation differs from HashMap in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is normally the order in which keys were inserted into the map (insertion-order). In last few tutorials we have discussed about HashMap and TreeMap. This class is different from both of them:

* HashMap doesn’t maintain any order.
* TreeMap sort the entries in ascending order of keys.
* LinkedHashMap maintains the insertion order.

Let’s understand the LinkedHashMap with the help of an example:

import java.util.LinkedHashMap;

import java.util.Set;

import java.util.Iterator;

import java.util.Map;

public class LinkedHashMapDemo {

public static void main(String args[]) {

// HashMap Declaration

LinkedHashMap<Integer, String> lhmap =

new LinkedHashMap<Integer, String>();

//Adding elements to LinkedHashMap

lhmap.put(22, "Abey");

lhmap.put(33, "Dawn");

lhmap.put(1, "Sherry");

lhmap.put(2, "Karon");

lhmap.put(100, "Jim");

// Generating a Set of entries

Set set = lhmap.entrySet();

// Displaying elements of LinkedHashMap

Iterator iterator = set.iterator();

while(iterator.hasNext()) {

Map.Entry me = (Map.Entry)iterator.next();

System.out.print("Key is: "+ me.getKey() +

"& Value is: "+me.getValue()+"\n");

}

}

}

**Output:**

Key is: 22& Value is: Abey

Key is: 33& Value is: Dawn

Key is: 1& Value is: Sherry

Key is: 2& Value is: Karon

Key is: 100& Value is: Jim

As you can see the values are returned in the same order in which they got inserted.

**Hashtable in java with example**

This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value. Hashtable is similar to [HashMap](https://beginnersbook.com/2013/12/hashmap-in-java-with-example/" \o "HashMap in Java with Example" \t "_blank) except it is synchronized. There are few more differences between HashMap and Hashtable class, you can read them in detail at: [Difference between HashMap and Hashtable](https://beginnersbook.com/2014/06/difference-between-hashmap-and-hashtable/).

**Example**

import java.util.Hashtable;

import java.util.Enumeration;

public class HashtableExample {

public static void main(String[] args) {

Enumeration names;

String key;

// Creating a Hashtable

Hashtable<String, String> hashtable =

new Hashtable<String, String>();

// Adding Key and Value pairs to Hashtable

hashtable.put("Key1","Chaitanya");

hashtable.put("Key2","Ajeet");

hashtable.put("Key3","Peter");

hashtable.put("Key4","Ricky");

hashtable.put("Key5","Mona");

names = hashtable.keys();

while(names.hasMoreElements()) {

key = (String) names.nextElement();

System.out.println("Key: " +key+ " & Value: " +

hashtable.get(key));

}

}

}

**Output:**

Key: Key4 & Value: Ricky

Key: Key3 & Value: Peter

Key: Key2 & Value: Ajeet

Key: Key1 & Value: Chaitanya

Key: Key5 & Value: Mona

**Methods of Hashtable class:**

1) void clear(): Removes all the key-value mappings from Hashtable and makes it empty. Clears this hashtable so that it contains no keys..

2) Object clone(): Creates a shallow copy of this hashtable. All the structure of the hashtable itself is copied, but the keys and values are not cloned. This is a relatively expensive operation.

3) boolean contains(Object value): Tests if some key maps into the specified value in this hashtable. This operation is more expensive than the containsKey method.  
Note that this method is identical in functionality to containsValue, (which is part of the Map interface in the collections framework).

4) boolean isEmpty(): Tests if this hashtable maps no keys to values.

5) Enumeration keys(): Returns an enumeration of the keys contained in the hash table.

6) Object put(Object key, Object value): Maps the specified key to the specified value in this hashtable.

7) void rehash(): Increases the size of the hash table and rehashes all of its keys.

8) Object remove(Object key): Removes the key (and its corresponding value) from this hashtable.

9) int size(): Returns the number of key-value mappings present in Hashtable.

10) String toString(): Returns the string equivalent of a hash table.

11) boolean containsKey(Object key): Tests if the specified object is a key in this hashtable.

12) boolean containsValue(Object value): Tests if the specified object is a value in this hashtable. Returns true if some value equal to value exists within the hash table. Returns false if the value isn’t found.

13) Enumeration elements(): Returns an enumeration of the values contained in the hash table.

14) Object get(Object key): Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.