Map:

* Map is not a part of collection, why?
* Because collection framework is based on individual objects but map is based on key value pair.

When do we go for maps?

* If we want to represent a data/group of objects as key, value pairs, eg. Rollnum name pair, attributename attributevalue pair
* Both keys and values are object only
* Key can’t be duplicate, value can be duplicate
* Each key,value pair is known as **Entry**, it is a technical word, **Entry** interface
* A collection of **Entry** objects is called as a map

MAP METHODS:

Map has it’s own specific methods. We can’t apply collection methods in map.

In collection we used add(Object o) , why one argument, because as mentioned earlier collection is a concept based on individual objects.

1. **PUT**  
   🡪 So, in map we have put(Object key, Object value)

So, let me now tell you something, as we know that set can’t have a duplicate value right? So, whenever we use a set and call the add(10), add(10), so for the 2nd add call it will just return false and it would not add.

But for map, let’s say we are using map.put(101,”Nilim”), map.put(101, “Sarma”).

As key value can’t be duplicate Nilim would get replaced by sarma, and returns Nilim

* The return type of the put method is Object, if the key is not available already it will return null and put the value.
* If the key is already present, it will replace the new value and will return the old value.

1. **Get:** get (Object key): Object 🡪 returns the value associated with the specified key.
2. **Remove:** remove(Object key): Object 🡪removes the entry associated with the specified key
3. **containsKey:** containsKey(Object key): Boolean🡪 checks whether the key is available or not.
4. **containsValue:** containsValue(Object key): Boolean🡪 checks whether the value is available or not.
5. **isEmpty:boolean**
6. **size:int**
7. **clear:void**

**IMPORTANT METHODS:**

1. **keySet():Set 🡪** to get only keys we use this method, why the return type is set? because in set duplicates are not allowed. It returns only the keys.
2. **values():collection🡪** to get only values we , why the return type is collection, because in values duplicates are allowed.
3. **entrySet():Set 🡪** to list out the set of entry objects, why the return type is set? because in set duplicates are not allowed.

These methods are not map methods right, but we need to use it in map, so these methods are known as collection views of map.

**Entry Interface:**

A map is a group of key,value pairs and each key,value pair is called an entry. Hence, map is considered as a collection of entry objects.  
Without existing a map object, there is no use of existing entry object. Hence, **Entry interface** is a inner interface of **Map Interface**.

Entry interface in Java provides certain methods to access the entries in the Map. By gaining access to the entry of the Map we can easily manipulate them.

Methods of Entry Interface:

1. Object getKey();
2. Object getValue();
3. Object setValue(Object newValue)

These are entry specific methods and we can apply only on entry objects.

package com.te.setexamples;

import java.util.Collection;

import java.util.Map;

import java.util.Set;

import java.util.TreeMap;

public class entryS {

public static void main(String[] args) {

Map<String, Integer> map = new HashMap<String, Integer>();

map.put("scd", 1029);

map.put("jhc", 331);

map.put("acc", 131);

System.out.println(map);

// Collection collection = map.entrySet();

// for (Object object : collection) {

// System.out.println(object);

// }

Set<Map.Entry<String, Integer>> entry = map.entrySet();

for (Map.Entry<String, Integer> entry2 : entry) {

System.out.println(entry2.getKey() + "---> " + entry2.getValue());

if(entry2.getKey().equals("jhc")) {

entry2.setValue(456);

}

}

System.out.println(map);

}

}

**HashMap:**

* Underlying data structure, hashtable
* Insertion order not guaranteed, It is based on hashcode of keys
* Duplicate keys not allowed, but values duplicate allowed
* Null is allowed for key only once, for value duplicate null allowed
* Introduced in jdk 1.2

EG.

**import** java.util.Collection;

**import** java.util.HashMap;

**import** java.util.Iterator;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Set;

**public** **class** map {

**public** **static** **void** main(String[] args) {

Map map = **new** HashMap<Integer, String>();

System.***out***.println(map.put(101, "Nilim"));

System.***out***.println(map.put(102, "Prasad"));

System.***out***.println(map.put(101, "Sarma"));

System.***out***.println("-------------");

System.***out***.println(map.get(101));

System.***out***.println("------------");

Set set = map.keySet();

System.***out***.println(set);

System.***out***.println("------------");

Collection collection = map.values();

System.***out***.println(collection);

System.***out***.println("----------");

Set set2 = map.entrySet();

System.***out***.println(set2);

System.***out***.println("-------------");

Set set3 = map.entrySet();

Iterator iterator = set3.iterator();

**while** (iterator.hasNext()) {

Map.Entry<Integer, String> entry = (Entry<Integer, String>) iterator.next();

System.***out***.println(entry.getKey()+ " "+ entry.getValue());

**if**(entry.getKey().equals(102)) {

entry.setValue("Surjit");

}

System.***out***.println(map);

}

}

}

Another eg.

**import** java.util.ArrayList;

**import** java.util.Collection;

**import** java.util.Collections;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Set;

**import** java.util.TreeSet;

**public** **class** HashMapImpl {

**public** **static** **void** main(String[] args) {

HashMap<String, Integer> hashmap = **new** HashMap<String, Integer>();

hashmap.put("Naveen", 12345);

hashmap.put("Nilim", 45678);

hashmap.put("Akash", 34567);

hashmap.put("Dilip", 23456);

hashmap.put("Bunny", 98765);

System.***out***.println(hashmap);

System.***out***.println("----get based on key----");

System.***out***.println(hashmap.get("Nilim"));

Set<Map.Entry<String, Integer>> entries = hashmap.entrySet();

System.***out***.println("----entries-----");

**for** (Map.Entry<String, Integer> entry : entries) {

System.***out***.println(entry.getKey());

System.***out***.println(entry.getValue());

System.***out***.println("----------");

}

Set<String> set = hashmap.keySet();

System.***out***.println(set);

System.***out***.println("---------");

Collection<Integer> collection =hashmap.values();

System.***out***.println(collection);

System.***out***.println("----sort according to values----");

Collection<Integer> collection2 = hashmap.values();

TreeSet<Integer> treeset = **new** TreeSet<Integer>(collection2);

System.***out***.println(treeset);

System.***out***.println("-----sort according to keys-----");

Set<String> set2 = hashmap.keySet();

ArrayList<String> arrayList = **new** ArrayList<String>(set2);

Collections.*sort*(arrayList);

**for** (String string : arrayList) {

System.***out***.println(string);

}

}

}

**LinkedHashMap:**

* Child class of HashMap.
* Insertion order is preserved
* Underlying data structure is a combination of LinkedList and HashTable, hybrid data structure
* Introduced in jdk 1.4

**TreeMap🡪**

* underlying data structure is red black tree
* insertion order not preserved
* it is based on sorting order of keys
* duplicate keys not allowed, values can be duplicate
* if we are depending on default natural sorting order, then keys should be homogeneous and comparable. Otherwise we will get ClassCastException
* if we are defining our own sorting by comparator then keys need not be homogeneous.
* For treemap no null key values allowed, we will get null pointer exception.

**Simple eg for treeMap 1st:**

import java.util.\*;

import java.util.Map.Entry;

import java.util.TreeMap;

public class StudentTreeMapExec {

public static void main(String[] args) {

TreeMap<String, Integer> treemap = new TreeMap<String, Integer>();

treemap.put("Nilim", 30);

treemap.put("Nil", 330);

treemap.put("Bunny", 80);

treemap.put( "Naveen", 100);

treemap.put("Akash", 40);

System.out.println(treemap);

System.out.println("---------------------");

//ITERATING BY PUTTING TREEMAP INTO A SET

Collection<Map.Entry<String, Integer>> entries = treemap.entrySet();

for (Entry<String, Integer> entry : entries) {

System.out.println(entry.getKey() + " " + entry.getValue());

}

}

}

**Eg for comparable for id and comparator for name**

**Comparable class**

**public** **class** Student **implements** Comparable<Student> {

**int** id;

String name;

**public** Student() {

**super**();

}

**public** Student(**int** id, String name) {

**super**();

**this**.id = id;

**this**.name = name;

}

@Override

**public** **int** compareTo(Student o) {

**return** **this**.id - o.id;

}

@Override

**public** String toString() {

**return** "Student [id=" + id + ", name=" + name + "]";

}

}

**Comparator class**

**import** java.util.Comparator;

**public** **class** SortByNameStud **implements** Comparator<Student> {

@Override

**public** **int** compare(Student o1, Student o2) {

return o1.name.compareTo(o2.name);

}

}