Map:

- Map is not the child interface of collection.
- If we want to represent a group of objects as key value pair then we should go for Map.
- Each associated key value pair is known as one entry.

E.g:

Keys	Values
101	Ravi
1001	Tusar
102	Sangram
106	Rocky

- Both keys and values are objects only.
- Duplicate keys are not allowed but values can be duplicated.
- Each key value pair is called **entry**.
- Hence Map is called as a collection of entry objects.

Map Interface Methods:

object put(Object key,Object value);

- To add one key value pair to the Map.
- If the key already present old value will replaced with new value and returns old value.
- Void putAll(Map m);
- object get(Object key);//return the value associated with specified key
- object remove(Object key);//remove the vale associated with specified key.
- Boolean containsKey(Object key);
- Boolean containsValue(Object value);
- Boolean isEmpty();

- Int size();
- Void clear();

The below 3 methods are known as collection views of map

- Set keyset();
- Collection values();
- Set entrySet();

Entry Interface:

- Map is represented as a group of key value pairs and each key and its associated value is known as a entry.
- Also map is known as a group of entry objects.
- Without existing of map object there is no chance of existing entry object, Hence entry interface is defined inside map interface.

Interface map{

Interface entry{

Object getKey()

Object getValue()

Object setValue(Object newo)//Highlighted methods are known as entry specific methods and we can apply it only on entry objects.

}

}

HashMap:

- Underlying data structure is HashTable.
- Insertion order is not preserved and it is based on hashcode of keys.
- Duplicate keys are not allowed but values can be duplicated.
- Heterogeneous and homogeneous key and values are allowed.
- Null key allowed only once.
- Null value allowed any number of times.

- It implements serializable and clonable interface but not Random access.
- It is best choice if our frequent operation is search operation.

Constructors of HashMap:

- HashMap m=new HashMap();
- Will create a new hashmap object with default capacity of 16 and default fill ratio of 0.75.
- HashMap m=new HashMap(int size);
- Will create a new HashMap with specified capacity with default fill ratio of 0.75.
- HashMAp m=new HashMap(int size, float fillratio);
- Creates a new hashmap with specified fillratio and size.
- HashMap m=new HashMap(map M);
- Will convert a new hashmap from another map.

E.g:

```
    public static void main(String[] args) {
        HashMap <Integer,String> hm=new HashMap<>();
        hm.put(1, "Rocky");
        hm.put(2, "Sangram");
        hm.put(3,"Tusar");
        System.out.println(hm);
        Set<Integer> keys=hm.keySet();
        System.out.println(keys);
        Collection<String> values=hm.values();
        System.out.println(values);
        Set m=hm.entrySet();//To get the all entries
present in the HashMap object
        Iterator itr=m.iterator();//It will itrate over
the entryset
        while(itr.hasNext()) {//iterator hasNext method
till entries present or not
            Map.Entry<Integer, String>
m1=(Entry<Integer, String>)itr.next();//creates an entry
object to access keys and values
```

```
    System.out.println(m1.getKey()+"="+m1.getValue());//P
rinting the values present in Entryset
                if (m1.getKey().equals(1)) {
                    m1.setValue("Ram");
                    System.out.println(m1);
                }
            }
            System.out.println(hm);// {1=Ram, 2=Sangram,
3=Tusar}
```

Difference between "==" operator and .equals() method:

- In general "==" operator meant for reference comparison or address comparison whereas ".equals()" method is meant for content comparison in object.

  e.g:

```
Integer i1=new Integer (10);

Integer i2= new Integer (10);

Sop(i1==i2);//False

Sop(i1.equals(i2));//True
```

Identity HashMap:

- It is exactly same as hash map including methods and constructors except the following difference.
- In case of normal HashMap jvm will use .equals method to identify duplicate keys which is meant for content comparison.
- In case of identity hash map jvm will apply "==" operator to identify the duplicate keys which is meant for reference comparison.

e.g:

```
    public static void main(String[] args) {
IdentityHashMap<Integer, String>
m=newIdentityHashMap<Integer, String>();
          Integer I=new Integer(10);
          Integer I1=new Integer(10);
          m.put(I, "Ram");
          m.put(I1,"Laxman");
          System.out.println(m);// {10=Ram, 10=Laxman}
```

WeakHashMap:

- It is exactly same as HashMap except the following difference.
- In the case of HashMap even though object does not have any reference it is not eligible for gc if it is associated with hashMap that is hashMap dominates garbage collector.
- In case of weak HashMap its object does not contain any references the object is eligible for gc even though object associated with weak hashmap that is garbage collector dominates weak hashmap.

SortedMap:

- It is the child interface of Map interface.
- If we want to represent a group of object as a group of key value pairs according some sorting order of keys then we should go for sortedmap.
- **Sorting is based on the key but not based on value.**

SortedMap specific methods:

- Object firstKey()
- Object lastKey()
- SortedMap headMap(Object key)
- SortedMap tailMap(Object key)
- SortedMap subMap(Object key1,Object key2)
- Comparator comparator()

E.G:

```
        SortedMap<String,String> sp=new TreeMap<>();
        sp.put("Rocky", "Tusar");
        sp.put("Tusar", "Rocky");
        sp.put("Ram", "Sita");
        System.out.println(sp);//{Ram=Sita, Rocky=Tusar,
Tusar=Rocky}
        String firstKey = sp.firstKey();
        System.out.println(firstKey);//Ram
        String lastKey=sp.lastKey();
        System.out.println(lastKey);//Tusar
        SortedMap<String, String> headMap =
sp.headMap(lastKey);
        System.out.println(headMap);//{Ram=Sita,
Rocky=Tusar}

    SortedMap<String,String>tailMap=sp.tailMap("Rocky");
        System.out.println(tailMap);//{Rocky=Tusar,
Tusar=Rocky}
        SortedMap<String, String> subMap =
sp.subMap(firstKey, lastKey);
        System.out.println(subMap);//{Ram=Sita,
  Rocky=Tusar}
```

TreeMap:

- Underlying data structure is Red-Black Tree.
- Duplicate keys are not allowed but values can be duplicated.
- If we are depending on default natural sorting order then keys should be homogeneous and comparable otherwise we will get runtime exception saying class cast exception.
- If we are defining our own sorting by comparator then the keys need not be homogeneous and comparable, we can take heterogeneous and non-comparable objects also.

- Wheather we are depending on default natural sorting order or customized sorting order there is no restriction for values, we can take heterogeneous non comparable objects also.

Null-Acceptance:

- Null key is not allowed but for values null is allowed any number of times.

Constructors for TreeMap:

- TreeMap t=new TreeMap()//Meant for creating a empty treemap according to natural sorting order
- TreeMap t=new TreeMap(Comparator c())//Meant for creating a treemap according to coustomized sorting order.
- TreeMap t=new TreeMap(SortedMap m);
- TreeMap t=new TreeMap(Map m);

HashTable:

- The underlying data structucture is hashtable only.
- Insertion order is not preserved and it is based on hashcode of keys.
- Duplicate keys are not allowed but values can be duplicated.
- Heterogeneous are allowed for both keys and values.
- Null is not allowed for both key and value otherwise we will get runtime exception saying null pointer exception.
- It implements serializable and clonable interfaces but not random access.
- Every method present in hashtable is synchronized and hence hashtable object is thread safe.
- Hashtable is the best choice if our frequent operation is search operation.

Constructors:

- Hashtable h=new Hashtable ();//Creates a new hashtable object with default initial capacity 11 and default fill ratio 0.75.
- Hashtable h=new HashTable (int initial capacity);
- Hashtable h=new Hashtable (int initialcapacity, float fill ratio)
- Hashtable h=new Hashtable (Map m)

E.g:

```
    public static void main(String[] args) {
         Hashtable<Integer,String> h=new Hashtable<>();
         h.put(1, "Rocky");
         h.put(11, "Tusar");
         h.put(12, "Snagram");
         h.put(56, "Rose");
         System.out.println(h);// {56=Rose, 12=Snagram,
1=Rocky, 11=Tusar}
```

Proporties:

- In our program if anything which changes frequently(Like user name,password,mailed,mobile number etc) are not recommended to hardcode in java program because if there is any change to reflect that change recompilation,redeploy application are required even some times server restart also required which creates a big business impact to the client.
- We can overcome this problem by using properties file such type of variable things we have to configure in the properties file, from the properties file we have to read into java program and we can use those properties, The main advantage of this approach is if there is a change in properties file to reflect that change just redeployment is enough which won't create any business impact to the client.
- We can use java properties object to hold properties which are coming from properties file.

- In normal map(like hashmap,hashtable,treemap)key and value can be any type but in case of properties key and value should be string type.

Constructor:

- Properties p=new Properties();

Methods:

-