SortedMap<K,V>

It is the sub interface of Map<K,V> available from JDK 1.2V.

It maintains its keys in a sorted order.

By default It provides according to the natural ordering of the keys or by a custom Comparator provided at map creation time.

TreeMap<K,V>
----
public class TreeMap<K,V> extends AbstractMap<K,V> implements NavigableMap<K,V> , Clonable, Serializable

It is a predefined class avaialble in java.util package under Map interface available for 1.2V.

It is a sorted map that means it will sort the elements by natural sorting order based on the key (Comparable) or by using Comparator interface as a constructor parameter.

It does not allow non comparable keys.(ClassCastException)

It does not accept null key but null values are allowed.(NPE)

It uses Red black tree data structure.

TreeMap implements NavigableMap and NavigableMap extends SortedMap. SortedMap extends Map interface.

TreeMap contains 4 types of Constructors :

1) TreeMap tm1 = new TreeMap(); //creates an empty TreeMap

2) TreeMap tm2 = new TreeMap(Comparator cmp); //user defined sorting logic

3) TreeMap tm3 = new TreeMap(Map m); //loose Coupling

4) TreeMap tm4 = new TreeMap(SortedMap m);

```
//Sorting based on map key
import java.util.*;
public class TreeMapDemo
{
    public static void main(String[] args)
    {
        TreeMap<Object,String> t = new TreeMap<>();
        t.put(4,"Ravi");
        t.put(7,"Aswin");
        t.put(2,"Ananya");
        t.put(1,"Dinesh");
        t.put(9,"Ravi");
        t.put(3,"Ankita");
        t.put(5,null);
        System.out.println(t);
    }
}
```

```
import java.util.*;
public class TreeMapDemo1
{
    public static void main(String args[])
    {
        TreeMap map = new TreeMap();
        map.put("one","1");
        map.put("two",null);
        map.put("three",3);
            map.put("four",true);

        displayMap(map);

    map.forEach((k, v) -> System.out.println("Key = " + k + ", Value = " + v));
    }

    public static void displayMap(TreeMap map)
    {
        Collection c = map.entrySet();   //Set<Map.Entry>

        Iterator i = c.iterator();
        i.forEachRemaining(x -> System.out.println(x));
    }
}
```

```
import java.util.*;
public class TreeMapDemo2
{
   public static void main(String[] argv)
   {
      Map<String,String> map = new TreeMap<String,String>();
      map.put("key2", "value2");
      map.put("key3", "value3");
      map.put("key1", "value1");

      System.out.println(map);

   }
}
```

```
package com.ravi.tree_map_demo;

import java.util.TreeMap;

record Product(Integer id, String name)
{

}

public class TreeMapDemo3
{
    public static void main(String[] args)
    {
        TreeMap<Product, String> map = new TreeMap<>((p1,p2)-> p1.id().compareTo(p2.id()));
        map.put(new Product(222, "Mobile"), "Hyderabad");
        map.put(new Product(111, "Laptop"), "Chennai");
        map.put(new Product(333, "Camera"), "Pune");

        System.out.println(map);

    }

}
```

Hashtable<K,V>
----
public class Hashtable<K,V> extends Dictionary<K,V> implements Map<K,V>, Cloneable, Serializable.

It is predefined class available in java.util package under Map interface from JDK 1.0.

Like Vector, Hashtable is also form the birth of java so called legacy class.

It is the sub class of Dictionary class which is an abstract class.

*The major difference between HashMap and Hashtable is, HashMap methods are not synchronized where as Hastable methods are synchronized.

HashMap can accept one null key and multiple null values where as Hashtable does not contain anything as a null(key and value both). if we try to add null then JVM will throw an exception i.e NullPointerException.

The initial default capacity of Hashtable class is 11 where as loadFactor is 0.75.

It has also same constructor as we have in HashMap.(4 constructors)

1) Hashtable hs1 = new Hashtable();
It will create the Hashtable Object with default capacity as 11 as well as load factor is 0.75

2) Hashtable hs2 = new Hashtable(int initialCapacity);
will create the Hashtable object with specified capacity

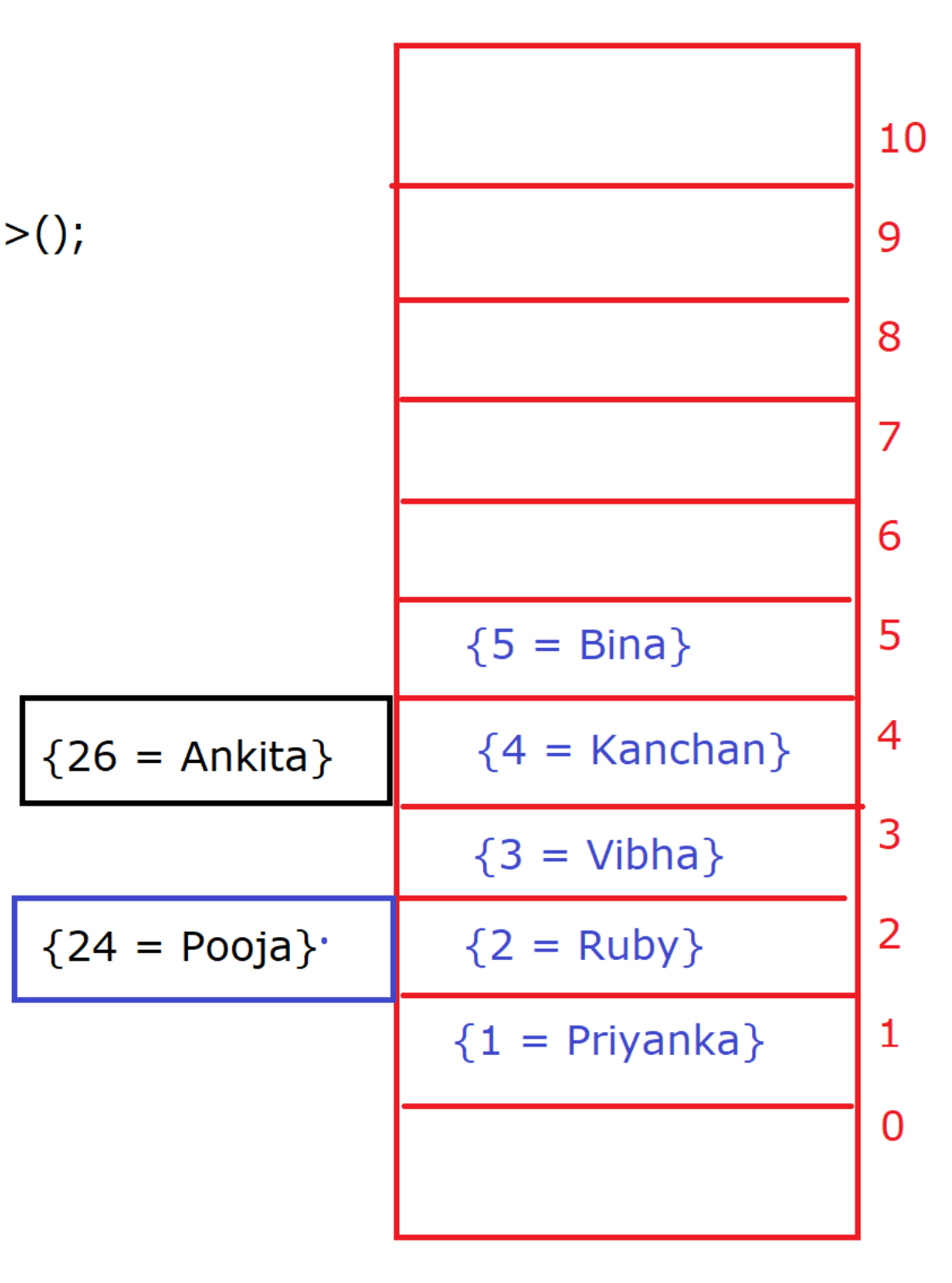
3) Hashtable hs3 = new Hashtable(int initialCapacity, float loadFactor);
we can specify our own initialCapacity and loadFactor

4) Hashtable hs = new Hashtable(Map c);
Interconversion of Map Collection

```
import java.util.*;
public class HashtableDemo
   {
    public static void main(String args[])
       {
        Hashtable<Integer,String> map = new Hashtable<>();
        map.put(1, "Java");
        map.put(2, "is");
        map.put(3, "best");
        map.put(4,"language");

        //map.put(5,null);

        System.out.println(map);

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

          for(Map.Entry m : map.entrySet())
            {
                System.out.println(m.getKey()+" = "+m.getValue());
            }
       }
}
```

How Hashtable works internally ?

```
Hashtable<Integer,String> map = new Hashtable<>();
     map.put(1,"Priyanka");
     map.put(2,"Ruby");
     map.put(3,"Vibha");
     map.put(4,"Kanchan");

      map.putIfAbsent(5,"Bina");
      map.putIfAbsent(24,"Pooja");
      map.putIfAbsent(26,"Ankita");

     map.putIfAbsent(1,"Sneha");
```



Top to Bottom and left to right :

Bina, Ankita........................

```
import java.util.*;
public class HashtableDemo1
{
   public static void main(String args[])
     {
    Hashtable<Integer,String> map = new Hashtable<>();
     map.put(1,"Priyanka");
     map.put(2,"Ruby");
     map.put(3,"Vibha");
     map.put(4,"Kanchan");

      map.putIfAbsent(5,"Bina");
      map.putIfAbsent(24,"Pooja");
      map.putIfAbsent(26,"Ankita");

     map.putIfAbsent(1,"Sneha");
     System.out.println("Updated Map: "+map);
 }
}
```

**WeakHashMap<K,V> :
----
public class WeakHashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>

It is a predefined class in java.util package under Map interface.It was introduced from JDK 1.2v onwards.

While working with HashMap, keys of HashMap are of strong reference type. This means the entry of map will not be deleted by the garbage collector even though the custom key is set to be null as well as Object is also not eligible for Garbage Collector.

On the other hand while working with WeakHashMap, keys of WeakHashMap are of weak reference type. This means the entry and corresponding object of a map is deleted by the garbage collector if the key value is set to be null/assigning a new object because it is of weak type.

So, HashMap dominates over Garbage Collector whereas Garbage Collector dominates over WeakHashMap.

It does not implements Cloneable and Serailizable because It is mainly used for inventory system where we need to manage the data and we can insert and delete object data frequently in the inventory.

It contains 4 types of Constructor :
----
1) WeakHashMap wm1 = new WeakHashMap();

Creates an empty WeakHashMap object with default capacity is 16 and load fator 0.75.

2) WeakHashMap wm2 = new WeakHashMap(int initialCapacity);

3) WeakHashMap wm3 = new WeakHashMap(int initialCapacity, float loadFactor);
Eg:- WeakHashMap wm = new WeakHashMap(10,0.9);

capacity - The capacity of this map is 10. Meaning, it can store 10 entries.

loadFactor - The load factor of this map is 0.9. This means whenever our hashtable is filled up by 90%, the entries are moved to a new hashtable of double the size of the original hashtable.

4) WeakHashMap wm4 = new WeakHashMap(Map m);

```
package com.ravi.weak_hash_map;

import java.util.WeakHashMap;  //e1 = null;

record Product(Integer id, String name, Double price)
{
    @Override
    public void finalize()
    {
        System.out.println("Product Object is eligible for GC");
    }
}

public class WeakHashMapDemo
{
    public static void main(String[] args)
    {
        Product p1 = new Product(111, "Laptop", 96000D);

        WeakHashMap<Product, String> map = new WeakHashMap<>();
        map.put(p1, "Hyd");

        System.out.println(map);

        p1 = null;

        System.gc();

        try
        {
            Thread.sleep(5000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }

        System.out.println(map);  //{}

    }

}
```
