

How to generate System hashCode value :

* System class has provided a predefined static and native method to find out the system hash code, It is nothing but Object class has code not the overridden hashCode value as shown below :

```
public static native int IdentityHashCode(Object obj)

//Program :
-----
package com.ravi.identity_hash_map;

public class SystemHashCode
{
    public static void main(String[] args)
    {
        String str1 = new String("Java");
        String str2 = new String("Java");

        System.out.println("Overridden Hashcode from String class :");
        System.out.println(str1.hashCode() + " : "+str2.hashCode());

        System.out.println("System HashCode value from the Object class :");
        System.out.println(System.identityHashCode(str1));
        System.out.println(System.identityHashCode(str2));
    }
}
```

IdentityHashMap<K,V> [Comparing keys based on the Memory reference]

public class IdentityHashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Clonable, Serializable.

It was introduced from JDK 1.4 onwards.

The IdentityHashMap uses == operator to compare key object.

As we know HashMap uses equals() and hashCode() method for comparing the keys based on the hashcode of the object it will search the bucket location and insert the entry there only.

So We should use IdentityHashMap where we need to compare the keys by using reference or memory address instead of logical equality.

HashMap uses hashCode of the "Object key" to find out the bucket location in Hashtable, on the other hand IdentityHashMap does not use hashCode() method actually It uses System.identityHashCode(Object o)

IdentityHashMap is more faster than HashMap in case of key Comparison.

The default capacity is 32.

It has three constructors, It does not contain loadFactor specific constructor.

1) IdentityHashMap ihm1 = new IdentityHashMap();
Will create IdentityHashMap with default capacity is 32.

2) IdentityHashMap ihm2 = new IdentityHashMap(int initialCapacity);
Will create IdentityHashMap with user specified capacity

3) IdentityHashMap ihm3 = new IdentityHashMap(Map map)

```
package com.ravi.identity_hash_map;

import java.util.HashMap;
import java.util.IdentityHashMap;

public class IdentityHashMapDemo
{
    public static void main(String[] args)
    {
        HashMap<String, Integer> map = new HashMap<>();
        map.put("Java", 1234);
        map.put(new String("Java"), 5555);

        System.out.println(map.size()+" : "+map);

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

        IdentityHashMap<String, Integer> ihm = new IdentityHashMap<>();
        ihm.put("Java", 1234);
        ihm.put(new String("Java"), 5555);

        System.out.println(ihm.size()+" : "+ihm);
    }
}
```

Methods of SortedSet<E> interface :

SortedSet<E> interface has provided the following methods :

1) public E first() : Will provide the first element.

2) public E last() : Will provide the last element.

3) public SortedSet headSet(E range) : Will provide the range of values which are less than specified range.

4) public SortedSet tailSet(E range) : Will provide the equal and greater than specified range value.

5) public SortedSet subSet(E startRange, E endRange) : Will provide range of values where startRange will inclusive and endRange will be exclusive.

```
import java.util.*;
public class SortedSetMethodDemo
{
    public static void main(String[] args)
    {
        TreeSet<Integer> times = new TreeSet<>();
        times.add(1205);
        times.add(1505);
        times.add(1545);
        times.add(1600);
        times.add(1830);
        times.add(2010);
        times.add(2100);

        SortedSet<Integer> sub = new TreeSet<>();

        sub = times.subSet(1545,2100);
        System.out.println("Using subSet() :-"+sub); // [1545, 1600, 1830, 2010]
        System.out.println(sub.first());
        System.out.println(sub.last());

        sub = times.headSet(1545);
        System.out.println("Using headSet() :-"+sub); // [1205, 1505]

        sub = times.tailSet(1545);
        System.out.println("Using tailSet() :-"+sub); // [1545 to 2100]
    }
}
```

By using above SortedSet<E> methods we can find the range of values but navigation among the element is not possible.

In order to provide navigation facility among the elements, Java software people has introduced NavigableSet(I) interface from JDK 1.6V.

NavigableSet :

It is a predefined interface available in java.util package from JDK 1.6v

It is sub interface of SorteSet<E>

It is used to navigate among the elements, Unlike SortedSet which provides range of values. Here we can navigate among the values as shown below.

```
import java.util.*;
public class NavigableSetDemo
{
    public static void main(String[] args)
    {
        NavigableSet<Integer> ns = new TreeSet<>();
        ns.add(1);
        ns.add(2);
        ns.add(3);
        ns.add(4);
        ns.add(5);
        ns.add(6);

        System.out.println("lower(3): " + ns.lower(3)); // Just below than the specified element or null

        System.out.println("floor(3): " + ns.floor(0)); // Equal less or null

        System.out.println("higher(3): " + ns.higher(3)); // Just greater than specified element or null

        System.out.println("ceiling(3): " + ns.ceiling(3)); // Equal or greater or null
    }
}
```

Methods of SortedMap<K,V> interface :

1) K firstKey() //first key

2) K lastKey() //last key

3) headMap(E keyRange) //less than the specified range

4) tailMap(E keyRange) //equal or greater than the specified range

5) subMap(E startKeyRange, E endKeyRange) //the range of key where startKey will be inclusive and endKey will be exclusive.

return type of headMap(), tailMap() and subMap() return type would be SortedMap(I)

```
import java.util.*;
public class SortedMapMethodDemo
{
    public static void main(String args[])
    {
        SortedMap<Integer, String> map = new TreeMap<>();
        map.put(100, "Amit");
        map.put(101, "Ravi");

        map.put(102, "Vijay");
        map.put(103, "Rahul");

        System.out.println("First Key: "+map.firstKey()); // 100
        System.out.println("Last Key "+map.lastKey()); // 103

        System.out.println("headMap: "+map.headMap(102)); // 100 to 101
        System.out.println("tailMap: "+map.tailMap(102)); // 102 to 103
        System.out.println("subMap: "+map.subMap(100, 102)); // 100 to 101
    }
}
```

Assignment for NavigableMap Methods :

1) ceilingEntry(K key)

2) ceilingKey(K key)

3) floorEntry(K key)

4) floorKey(K key)

5) higherEntry(K key)

6) higherKey(K key)

7) lowerEntry(K key)

8) lowerKey(K key)