**HashSet**

HashSet extends AbstractSet and implements the Set Interface In Java.

Declaration of HashSet:  
class HashSet< E >  
Here, E specifies the type of objects that the set holds.

In HashSet hash table is used for storage. A hash table stores information by using a mechanism called hashing. In hashing, the information is stored in the form of (key, value). Here key is used to determine a unique value, called hashcode. The hashcode is then used as the index to retrieve the data associated with the key.  
Point to remember in HashSet:

* HashSet doesn't allow duplicates.
* The transformation of the key into its hash code is performed automatically and we never see the hash code itself.
* We can't directly index the hash table without hash code.
* HashSet does not define any additional methods other than the methods of its superclasses and interfaces.
* HashSet does not guarantee the order of its elements.

The following are the constructors defined by HahsSet.  
HashSet()  
Creats a default hash set.  
  
HashSet(Collection<? extends E> c)  
Creats a hash set by using the elements of c.  
  
HashSet(int capacity)  
Creats a hash set and initializes the capacity to capacity.(The default capacity is 16)  
  
HashSet(int capacity, float fillRatio)  
Creats a hash set and initializes both the capacity and fill ratio(also called load capacity) of the hash set from its arguments. The fill ratio must be between 0.0 and 1.0. When the number of elements is greater than the capacity, the capacity of the hash set is multiplied with the fill ratio and the hash set is expanded.

import java.util.\*;  
  
class HashSetDemo  
{  
    public static void main(String arg[])  
    {  
        List<Character> characters = new ArrayList<Character>(); //   
        characters.add('A');  
        characters.add('B');  
        characters.add('C');  
        characters.add('D');  
        characters.add('2');  
        HashSet<Character> hashcharacters = new HashSet<Character>(characters); // LINE B  
        System.out.println("Elements in hashcharacters: " + hashcharacters); // LINE C  
        System.out.print("retrieving in order: "); // LINE D  
        for (int i = 0; i < hashcharacters.size(); i++)  
        {  
            System.out.print(characters.get(i) + " "); // LINE E  
        }  
        hashcharacters.remove('2'); // LINE F  
        System.out.println();  
        System.out.print("Modified HashSet: " + hashcharacters); // LINE G      
    }  
}

**LinkedHashSet**

LinkedHashSet is a class extends HashSet Class In Java. LinkedHashSet has no no other methods other than its super class methods. LinkedHashSet guarantees the order of the elements in which they were added.

import java.util.\*;  
  
class LinkedHashSetDemo  
{  
    public static void main(String arg[])  
    {  
        LinkedHashSet<String> lhs = new LinkedHashSet<String>(); // LINE A  
        lhs.add("Anderson");  
        lhs.add("Bharath");  
        lhs.add("Cruse");  
        lhs.add("Dinesh");  
        System.out.println("Eements retrieved in the order they were inserted : ");  
        System.out.println(lhs);  
        lhs.remove("Bharath"); // LINE B  
        lhs.add("Bhuwaneshwar"); // LINE C  
        System.out.println("Set after removing Bharath and adding Bhuwaneshwar : ");  
        System.out.println(lhs);  
        Object[] names = lhs.toArray();  
        System.out.println("Displaying elements in array format : "); // LINE D  
        for (Object name : names)  
        {  
            System.out.print(name + ", ");  
        }  
        System.out.println();  
        lhs.clear(); // LINE E  
        System.out.println("Displaying empty set : ");  
        System.out.println(lhs);      
    }  
}

**TreeSet**

TreeSet is one of two sorted collections the other is TreeMap In Java - java.util.TreeMap. TreeSet extends AbstractSet and implements the Java NavigableSet Interface . It creates a collection that uses a tree for storage. Objects are stored in sorted, ascending order according to natural order. Optionally, we can change the natural order of a TreeSet by using a Comparable or Comparator interfaces.

Access and retrieval times are quite fast, which makes TreeSet an excellent choice when storing large amounts of sorted information that must be found quickly. TreeSet might not be used when our application has requirement of modification of set in terms of frequent addition of elements.  
Below shown is the hierarchy of the TreeSet.  
java.lang.Object  
  
        java.util.AbstractCollection  
  
                java.util.AbstractSet  
  
                        java.util.TreeSet

import java.util.\*;  
  
class TreeSetSample  
{  
    public static void main(String arg[])  
    {  
        TreeSet<Integer> input = new TreeSet<Integer>(); // LINE A  
        input.add(1);  
        input.add(3);  
        input.add(2);  
        input.add(6);  
        input.add(5);  
        input.add(4);  
        System.out.println(input); // LINE B  
        System.out.println("First Element in Set : " + input.first()); // LINE C  
        System.out.println("Last Element in Set : " + input.last()); // LINE D  
        System.out.println("Element higher to 4 : " + input.higher(4)); // LINE E  
        //higher returns the next higher number to the given number  
        System.out.println("Element lower to 3 : " + input.lower(3));  
        //lower returns the next lower number to the given number      
    }  
}

**toString()**

**public** **class** NmberTest {

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

Stud s1 = **new** Stud("Sam", 122);

Stud s2 = **new** Stud("Sam", 122);

Stud s3 = **new** Stud("Sam", 122);

Stud s4 = **new** Stud("Sam", 122);

Stud students[] = {s1,s2,s3,s4};

**for**(Stud student:students)

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

}

}

**class** Stud {

String name;

**int** id;

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

**this**.name = name;

**this**.id = id;

}

}

@Override

**public** String toString() {

**return** " [" + **this**.name +" , " + **this**.id + "]";

}