Set was introduced JDK 1.2 Version
Set is the Child Interface to Collection Interface
It is **not index based**, it uses Hashcode
Set is **Unordered** but **LinkedHashSet** is **Ordered** and **TreeSet** is sorted by **Natural Order**
No Sorting in set, but SortedSet, NavigableSet and TreeSet are following Sorting order

Collection(I)→ Set(I)→ HashSet(C) →LinkedHashSet(C)
Collection(I)→ Set(I)→ SortedSet(I) →NavigableSet(I)→ TreeSet(C)

	HashSet	**LinkedHashSet**	**TreeSet**
Insertion Order	No Insertion Order	Insertion Order	Natural Sorting Order
null	Only Once	Only Once	Not Allowed
Data Structure	HashTable	Hash Table and Linked List	TreeMap
Version	JDK 1.2	JDK 4.0	JDK 2.0
Implements	public class **HashSet<E>** extends AbstractSet<E> implements Set<E>, **java.lang.Cloneable**, **java.io.Serializable**	public class LinkedHashSet<E> extends **HashSet<E>** implements Set<E>, **java.lang.Cloneable, java.io.Serializable**	public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, **Cloneable, Serializable**
Duplicates	**No**	**No**	**No**

HashSet	LinkedHashSet
JDK 1.2V	JDK 1.4V
No Indexing	No Indexing
Duplicates Not allowed	Duplicates Not allowed
No Insertion/Unordered elements	Insertion/Ordered elements
Hetrogenious Elements	Hetrogenious Elements
null allowed only once	null allowed only once
In HashSet we don’t have any method to get the specific index elements	In LinkedHashSet we don’t have any method to get the specific index elements
DataStrutcure is HashTable	DataStrutcure is HashTable and LinkedList
Not Synchronized	Not Synchronized
Set does not provide anything like listIterator. It simply return Iterator in java.	Set does not provide anything like listIterator. It simply return Iterator in java.

```java
//Unordered, No Duplicates, Null Allowed Once
HashSet<String> set = new HashSet<>();
set.add("NameOne");
set.add("NameTwo");
set.add("NameThree");
set.add("NameFour");
set.add("NameFive");
set.add("NameOne");
set.add("NameTwo");
set.add(null);
set.add(null);
System.out.println(set); // [NameOne, null, NameFive, NameFour, NameTwo, NameThree]
```

```java
// Insertion Order, No Duplicates, Null Allowed Once
LinkedHashSet<String> set = new LinkedHashSet<>();
set.add("NameOne");
set.add("NameTwo");
set.add("NameThree");
set.add("NameFour");
set.add("NameFive");
set.add("NameOne");
set.add("NameTwo");
set.add(null);
set.add(null);
System.out.println(set); // [NameOne, NameTwo, NameThree, NameFour, NameFive, null]
```

```
// Natural Sorting, No Duplicates, Null Not Allowed
Set<String> set = new TreeSet<String>();
set.add("A");
set.add("D");
set.add("E");
set.add("B");
set.add("C");
set.add("D");
// set.add(null); // java.lang.NullPointerException
System.out.println(set); // [A, B, C, D, E]
```

```java
//Natural Sorting , No Duplicates, Null Not Allowed
SortedSet<String> set = new TreeSet<String>();
set.add("C");
set.add("D");
set.add("a");
set.add("A");
set.add("B");
set.add("E");
set.add("F");
set.add("B");
set.add("C");
System.out.println(set); //[A, B, C, D, E, F, a]

System.out.println(set.first()); //A
System.out.println(set.last()); // a
System.out.println(set.headSet("C")); //[A, B] // less than C
System.out.println(set.tailSet("C")); //[C, D, E, F, a] // greater than
System.out.println(set.subSet("D", "a")); //[D, E, F] // from and to elements
```

```
NavigableSet<Integer> set = new TreeSet<Integer>();
set.add(65);
set.add(67);
set.add(66);
set.add(68);
set.add(69);
set.add(70);
set.add(68);
set.add(85);
System.out.println(set); // [65, 66, 67, 68, 69, 70, 85]

System.out.println(set.descendingSet()); // [85, 70, 69, 68, 67, 66, 65]

System.out.println(set.ceiling(68)); // 68 //Greater Than Equal to
System.out.println(set.ceiling(90)); // null  //Greater Than Equal to
System.out.println(set.ceiling(85)); // 85  //Greater Than Equal to

System.out.println(set.higher(66)); // 67 //Greater Than
System.out.println(set.higher(90)); // null //Greater Than

System.out.println(set.floor(85)); // 85 //Greater Than, Less Than or Equal to
System.out.println(set.floor(70)); // 70
System.out.println(set.floor(90)); // 85
System.out.println(set.lower(70)); // 69 //Less than 70
System.out.println(set.lower(60)); // null //Less than 60

System.out.println(set.pollFirst()); // 65
System.out.println(set.pollLast()); // 85
```

```java
//String Class public int compareTo(String anotherString)
int i1;
ArrayList<String> list1 = new ArrayList<String>();
list1.add("A");
i1 = list1.get(0).compareTo("A");
System.out.println(i1); // 0

int i2;
ArrayList<String> list2 = new ArrayList<String>();
list2.add("B");
i2 = list2.get(0).compareTo("A");
System.out.println(i2); // 1 // B is Greater than A

int i3;
ArrayList<String> list3 = new ArrayList<String>();
list3.add("A");
i3 = list3.get(0).compareTo("B");
System.out.println(i3); // -1 // A is Less than B
```

```
//Print Ordered List, Sorting Order and Reverse the Order
ArrayList<String> al = new ArrayList<String>();
al.add("A");
al.add("C");
al.add("B");
al.add("D");
System.out.println(al); // [A, C, B, D] //Ordered List

al.sort(null);
System.out.println(al); // [A, B, C, D] //Sorting Order

Collections.reverse(al);
System.out.println(al); // [D, C, B, A] //Reverse the Order
```