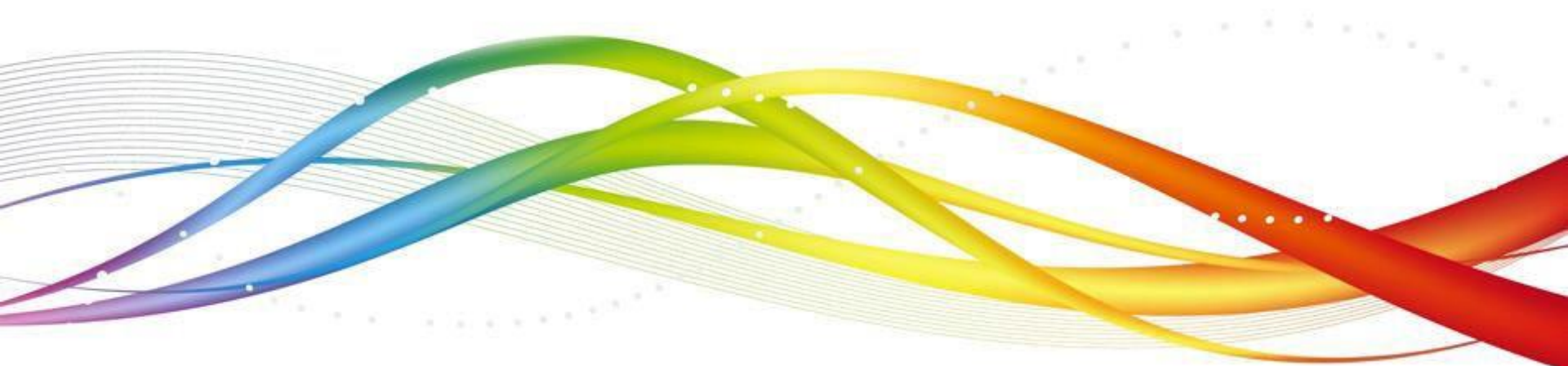


# Set



# Agenda

---

**1**

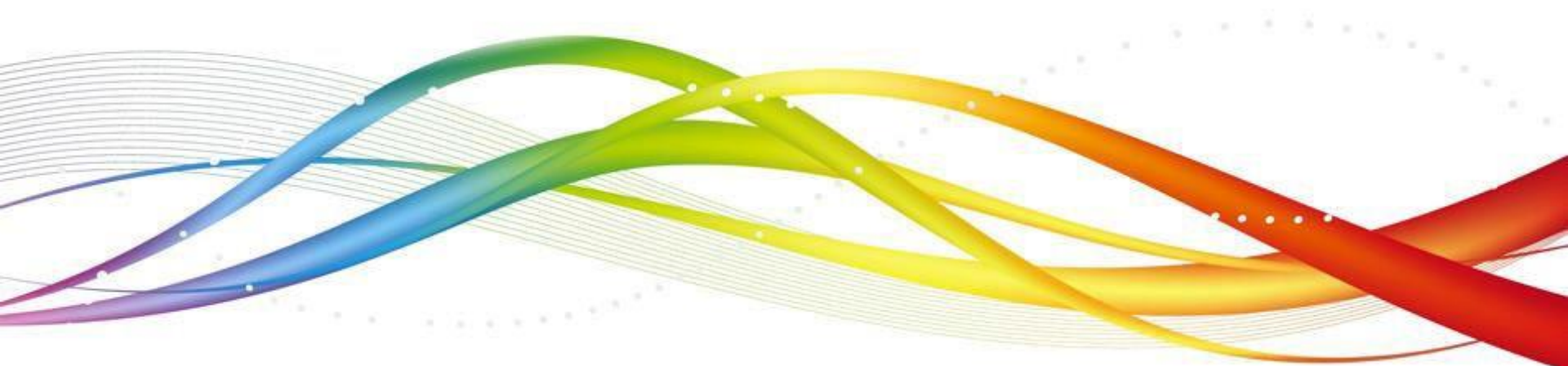
**Set**

**TreeSet**

**Comparable Interface**

**HashSet**

# Set



# Set

---

- Set interface extends from Collection interface
- Set is an unordered collection
- It doesn't allow duplicates
- If you add duplicates, it will replace the existing one
- It allows you to add only a single null value
- An iterator can be used to traverse through the list
- List interface has one legacy class called Vector, but Set doesn't have any legacy class
- Implementation classes for Set are TreeSet, HashSet and LinkedHashSet

# TreeSet

---

- No duplicates allowed
- Iterates in sorted order
- Sorted Collection
  - By default elements will be in ascending order
- Not synchronized
  - If more than one thread wants to access it at the same time then it must be synchronized externally

```
TreeSet<String> t1 = new TreeSet<String>();  
t1.add("One");  
t1.add("Two");  
t1.add("Three");
```

TreeSet implements the Set interface, backed by a TreeMap instance. This class guarantees that the sorted set will be in ascending element order, sorted according to the *natural order* of the elements, or by the comparator provided at set creation time, depending on which constructor is used.

# Example

---

```
class treeset{
public static void main(String args[]){
TreeSet<String> t1 = new TreeSet<String>();    //create a
Treeset object
t1.add("One");
t1.add("Two");
t1.add("Three");
t1.add("Four");
t1.add("Five");
System.out.println("Contents of treeset");
Iterator it1 = t1.iterator();    //obtaining iterator object
    while(it1.hasNext()){        // to iterate thru collection.
        Object o1=it1.next();
        System.out.print(o1+"\t");    } } }
```

# Comparable Interface

---

- This interface can be used to order the user defined objects
- It is found in java.lang package
- The method found in Comparable interface is  
**public int compareTo(Object obj)**
- This method will compare the current object with the previous object

# Example using Comparable interface

---

**Sorting Students based on their marks:**

**Student.java**

```
class Student implements Comparable{  
    int rollno;  
    String name;  
    int marks;  
    Student(int rollno,String name,int marks){  
        this.rollno=rollno;  
        this.name=name;  
        this.marks=marks;  
    }  
}
```



# Example using Comparable interface

---

```
public int compareTo(Object obj){  
    Student student=(Student)obj;  
    if(marks==student.marks)  
        return 0;  
    else if(marks>student.marks)  
        return 1;  
    else  
        return -1;  
}  
  
}
```

# Example using Comparable interface

---

## Test.java

```
import java.util.*;
import java.io.*;
class Test{
    public static void main(String args[]){
        TreeSet treeset=new TreeSet();
        treeset.add(new Student(1,"Ajay",66));
        treeset.add(new Student(2,"Abhi",96));
        treeset.add(new Student(3,"Sanjai",45));
```

# Example using Comparable interface

## Test.java

```
Iterator itr=treeset.iterator();  
while(itr.hasNext()){  
    Student st=(Student)itr.next();  
    System.out.println(st.rollno+":"+st.name+":"+st.age);  
    } } }
```

## O/P:

3:Sanjai:45

1:Ajay:66

2:Abhi:96

# The HashSet Class

---

- No duplicates allowed
- A HashSet is an unsorted, unordered Set
- Can be used when you want a collection with no duplicates and you don't care about the order when you iterate through it
- Uses HashTable for storage

```
Set<Integer> s = new HashSet<Integer>();  
s.add(1);  
s.add(2);  
s.add(3);
```

Remember that Sets are used when you don't want any duplicates in your collection. If you attempt to add an element to a set that already exists in the set, the duplicate element will not be added, and the `add()` method will return `false`. Remember, HashSets tend to be very fast because they use hashcodes.

# Example

---

```
import java.util.*;
class eg {
public static void main(String[] args) {
Set<Integer> s = new HashSet<Integer>();
s.add(1);
s.add(2);
s.add(3);
s.add(4);
s.add(5);
for(Integer i : s)
System.out.print(i + " "); }}
```

O/P: 2 4 1 3 5

Note: The order of the objects printed are not predictable

# Quiz

---

1. Set allows atmost one null element
  - a. True
  - b. False
  
2. Which implementation of Set should we use if we want the iterator to retrieve the objects in the order we have inserted?
  - a. TreeSet
  - b. HashSet
  - c. LinkedHashSet
  
3. If we need to store user defined objects in a TreeSet, which interface should the corresponding class implement?

# Summary

---

- Set
- TreeSet
- Comparable Interface
- HashSet