**from the add() method of the HashSet class following 2 method is called to identify the object uniqueness:**

**equals()**
**hashCode()**

**--if for 2 objects equals will returns true then their hashCode value should return the same value for both object.**

**basically 3 cases where we need to override equals and hashCode method inside our class.**

**1. whenever we try to make our objects logically equal.**

**2. whenever we try to add our objects inside the HashSet or LinkedHashSet class.**

**3.whenever we try to add our object as a key inside the HashMap class.**

**LinkedHashSet:-**
**----------**

**--it is the child class of HashSet class, it also does not allow duplicate, but it maintains the insertion order.**

**Note: In collection f/w all the collection classes are mutually inclusive . i.e we can convert any collection classes to any other collection class very easily.**

**Removing duplicates from the ArrayList:**
**----------**

```
package com.masai;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.LinkedHashSet;

public class Demo {

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

```
                ArrayList<Integer> al = new ArrayList<>();

                al.add(10);
                al.add(20);
                al.add(10);
                al.add(10);
                al.add(30);
                al.add(20);

                System.out.println(al);


                LinkedHashSet<Integer> lhs=new LinkedHashSet<>(al);

                System.out.println(lhs);

                al=new ArrayList<>(lhs);

                System.out.println(al);

        }
}
```

**removing duplicate from the String:**
**==========**

```
package com.masai;

import java.util.ArrayList;
import java.util.LinkedHashSet;

public class Demo {



                public static String removeDuplicateFromString(String original) {

                        char[] chr= original.toCharArray();

                        LinkedHashSet<Character> lhs=new LinkedHashSet<>();
```

```
                for(char ch:chr) {

                        lhs.add(ch);
                }

                StringBuilder result= new StringBuilder("");


                for(char ch: lhs) {
                        result.append(ch);
                }



                return result.toString();


        }



    public static void main(String[] args) {



        String s= Demo.removeDuplicateFromString("ratan");

        System.out.println(s);


    }

}
```

**Note: String class and all the wrapper classes has already overriden the equals and hashCode method internally.**

**removing duplicate from the List:**

---

**Demo.java:**
**----------**

```
package com.masai;

import java.util.ArrayList;
import java.util.LinkedHashSet;
import java.util.List;

public class Demo {



	public static List<String> removeDuplicateFromList(List<String> cities) {


		LinkedHashSet<String> lhs= new LinkedHashSet<>(cities);


		return new ArrayList<>(lhs);


	}



public static void main(String[] args) {


	List<String> cities= new ArrayList<>();

	cities.add("delhi");
	cities.add("delhi");
	cities.add("mumbai");
	cities.add("chennai");

	System.out.println(cities);

	List<String> resultList= removeDuplicateFromList(cities);

	System.out.println(resultList);

}
```

**}**

**TreeSet:**
**=======**

**--it has the nature of Collection, Set, SortedSet.**

**--In Java TreeSet class implemented by using balanced tree data structure.**

**--We use the TreeSet class, when we want to arrange our object in sorted order(natural sorting order)**

**--duplicates are not allowed.**

**--insertion order is not preserved.**

**--even a single null also not allowed, if we try to add a null value then at runtime we get a NPE.**

**example1:**

```
TreeSet<Integer> ts= new TreeSet<>();

        ts.add(10);
        ts.add(2);
        ts.add(5);
        ts.add(12);
        ts.add(6);
        ts.add(15);

        System.out.println(ts);
```

**example2:**

```
TreeSet<String> ts= new TreeSet<>();

        ts.add("delhi");
        ts.add("mumbai");
        ts.add("kolkata");
        ts.add("chennai");
```

**ts.add("chandigarh");**

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

**example3:**

**Note: if we try to add any element inside the TreeSet object, then those elements should be comparable.**

**i.e That object class should implements "java.lang.Comparable" interface. otherwise we will get a ClassCastException at runtime.**

**Note: - all the Wrapper classes and String class implements Comparable interface.**

**Comparable interface:**
**==========**

**--this interface belongs to java.lang package.**

**--by using this interface, we will specify the sorting rules/technique inside the class.**

**--this interface is having only one abstract method:**

**public int compareTo(Object obj);**

**--if we try to add any class object inside the TreeSet, then that class must implement Comparable interface and override this compareTo method , and inside this method we need to specify the sorting technique of that class object.**

**Note: in TreeSet hashCode and equals() method is not used.**

```
@Override
        public int compareTo(Object o) {

                //in this method, we need to specify the sorting rules.
```

```
		//this method internally called by the add() method of TreeSet obj,
		//when we try to add any element.

		//s1.compareTo(s2);

		//if s1 is bigger than s2 it returns +1
		//if s2 is bigger than s1 it retunes -1
		//if both s1 and s2 are equal then it returns 0;
}
```

example

Student.java:

---

```
package com.masai;

public class Student implements Comparable{

	private int roll;

	private String name;

	private int marks;


	public Student() {
		// TODO Auto-generated constructor stub
	}


	public Student(int roll, String name, int marks) {
		super();
		this.roll = roll;
		this.name = name;
		this.marks = marks;
	}


	public int getRoll() {
		return roll;
```

```
}


public void setRoll(int roll) {
	this.roll = roll;
}


public String getName() {
	return name;
}


public void setName(String name) {
	this.name = name;
}


public int getMarks() {
	return marks;
}


public void setMarks(int marks) {
	this.marks = marks;
}


@Override
public String toString() {
	return "Student [roll=" + roll + ", name=" + name + ", marks=" + marks + "]";
}


@Override
public int compareTo(Object o) {


	Student s1= this;

	Student s2= (Student)o;

	if(s1.getMarks() > s2.getMarks())
		return +1;
```

```
		else if(s1.getMarks() < s2.getMarks())
			return -1;
		else
			return 0;


	}

}
```

--the above sorting logic is based on the marks..

Note: in order to check the object uniqueness
HashSet and LinkedHashSet class uses equals() and hashCode() metho
where as TreeSet class uses compareTo() method. if compareTo() returns 0.

above example using generics:
---

```
public class Student implements Comparable<Student>
{

--
--
	@Override
	public int compareTo(Student s) {

		if(this.getMarks() > s.getMarks())
			return +1;
		else if(this.getMarks() < s.getMarks())
			return -1;
		else
			return 0;


	}

}
```

**not removing duplicate:**
**==========**

```
	@Override
	public int compareTo(Student s) {

		if(this.getMarks() > s.getMarks())
			return +1;
		else
			return -1;

		//return this.getMarks() > s.getMarks() ? +1:-1;

	}
```

**example sort the student based on name:**
**----------**

```
	@Override
	public int compareTo(Student s) {

		return this.getName().compareTo(s.getName());


	}
```

**reverse order:**
**==========**

```
@Override
	public int compareTo(Student s) {

		return s.getName().compareTo(this.getName());


	}
```

**if Marks are same then sort them according to thier name:**
**==========**

```
@Override
public int compareTo(Student s) {

        if(this.getMarks() > s.getMarks())
                return +1;
        else if(this.getMarks() < s.getMarks())
                return -1;
        else
                return this.getName().compareTo(s.getName());


}
```

**java.util.Comparator(I):**
**==========**

**--by using interface, we can define the sorting technique for a class objects from outside of that class.**

**Note: if we want to define a sotring rule of a class objects inside the same class then we should use**
**java.lang.Comparable interface**

**--whereas if we want to define the sorting logic outside of that class then we need to use java.util.Comparator interface.**

**--with the help of Comparable we can define only one sorting logic, where as with the help of Comparator we can define multiple sorting logic.**

**--this Comparator interface has also only one abstract method:**

**public int compare(Object obj1, Object obj2);**

**--to use the Comparator :-**
**----------**

**step 1: here we need not polute the Student class(java bean class) by implementing Comparable interface.**

**step 2: create a seperate class by any name and implements the Comparator interface and define the sorting logic by overriding the compare(--) method.**

**ex:**

**StudentMarksComp.java:**
**----------**

```
package com.masai;

import java.util.Comparator;

public class StudentMarksComp implements Comparator{

	@Override
	public int compare(Object o1, Object o2) {

		Student s1= (Student)o1;
		Student s2= (Student)o2;


		if(s1.getMarks() > s2.getMarks())
			return +1;
		else if(s1.getMarks() < s2.getMarks())
			return -1;
		else
			return 0;



	}

}
```

**by using generics:**
**----------**

```
package com.masai;

import java.util.Comparator;
```

```
public class StudentMarksComp implements Comparator<Student>{

	@Override
	public int compare(Student s1, Student s2) {

		if(s1.getMarks() > s2.getMarks())
			return +1;
		else if(s1.getMarks() < s2.getMarks())
			return -1;
		else
			return 0;

	}



}
```

**step 3:**

**--create the above StudentMarksComp class object and pass that object to the constructor of the TreeSet class.**

**example**

```
package com.masai;

import java.util.ArrayList;
import java.util.LinkedHashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.TreeSet;

public class Demo {
```

```
	public static List<String> removeDuplicateFromList(List<String> cities) {

		LinkedHashSet<String> lhs= new LinkedHashSet<>(cities);

		return new ArrayList<>(lhs);

	}

public static void main(String[] args) {

//		StudentMarksComp mcomp = new StudentMarksComp();
//		TreeSet<Student> ts= new TreeSet<>(mcomp);

		TreeSet<Student> ts= new TreeSet<>(new StudentMarksComp());

		ts.add(new Student(10, "n1", 780)); //s1
		ts.add(new Student(12, "n2", 680)); //s2
		ts.add(new Student(13, "n5", 480));
		ts.add(new Student(14, "n4", 480));
		ts.add(new Student(15, "n3", 480));



		System.out.println(ts.size());

		System.out.println(ts);

	}

}
```

**sorting according to the name:**

---

```
package com.masai;
```

```
import java.util.Comparator;

public class StudentNameComp implements Comparator<Student>{

	@Override
	public int compare(Student s1, Student s2) {

		return s1.getName().compareTo(s2.getName());

	}

}
```

**utility or tools classes in collection f/w:**
**==========**

**1.java.util.Arrays class: it is mostly used for performing unility operation on the normal arrays**
**for examaple : sorting, searching, reversing, converting to list, printing the arrays elements. etc,**

**2.java.util.Collections class: it is mostly used to perform utility operations for the collection f/w related classes.**

**java.util.Arrays:**
**==========**

**--inside this class we have variaous static methods are there by using**
**which we can perform various operations on the normal array object.**

**ex1:**

**printing the elements from array:**
**----------**

```
		int[] arr= {12,23,34,45,55,12,22};

//		String result= Arrays.toString(arr);
//
```

```
//                  System.out.println(result);

                    System.out.println(Arrays.toString(arr));
```

**sorting an array:**
**----------**

```
                    int[] arr= {12,23,34,45,55,12,22};


                    System.out.println(Arrays.toString(arr));

                    Arrays.sort(arr);

                    System.out.println(Arrays.toString(arr));
```

**searching an element in array:**
**----------**

**--first of all our array should be sorted, otherwise we may not get correct result.**

**--if the value is found then we get index value otherwise we get the negetive value.**

**example**

```
                    int[] arr= {12,23,34,45,55,12,22};

                    Arrays.sort(arr);

                    System.out.println(Arrays.toString(arr));


                    int index= Arrays.binarySearch(arr, 24);

                    System.out.println(index);
```

**creating List in easiest manner:**
**==========**

```
List<String> cities= Arrays.asList("Delhi","Mumbai","Chennai","kolkata");

List<Integer> list= Arrays.asList(10,20,30,40,50,20,60);

List<Student> students= Arrays.asList(


new Student(10, "n1", 780),
                new Student(10, "n1", 780),
                new Student(10, "n1", 780),
                new Student(10, "n1", 780));
}
```

**Collections class:**
**==========**

**--this class also defines some of the static methods to perform some utility operations on the Collection f/w related classes.**

**Collections.sort:**
**----------**

```
List<String> cities= Arrays.asList("Delhi","Mumbai","Chennai","kolkata");



Collections.sort(cities);


System.out.println(cities);
```

**example 2:**

```
List<Student> students= Arrays.asList(
```

```
                    new Student(12, "n1", 780)
            new Student(13, "n2", 580),
        new Student(14, "n3", 480),
                    new Student(15, "n4", 980));


            Collections.sort(students, new StudentMarksComp());


            System.out.println(students);




    }
```

**getting number of occuerrence in a List:**
**==========**

```
List<Integer> list=  Arrays.asList(10,12,10,10,12,45,55);


            int result= Collections.frequency(list, 10);

            System.out.println(result);
```

**converting List to the synchronized list:**
**----------**

```
    List<Integer> list=  Arrays.asList(10,12,10,10,12,45,55);

            List<Integer> sList= Collections.synchronizedList(list);
```