What is the difference between collection and collections?

TreeMap:

Key Features:

- Sorted Order: Keys are maintained in natural order(Ascending)
- Unique Keys: Keys must be unique, if we are attempting to insert duplicate keys it will overwrite the existing value.
- Null Handling:
  - Keys: TreeMap does not allow null keys
  - Values: It allows multiple null values
- Thread-Safety:
  - TreeMap is not Synchronized.
  - For thread-safety, we can wrap it using Collections.synchronizedMap.

Commonly used Methods:

put(K key, V value)

get(Object key)

firstKey()

lastKey()

TreeMap (Java SE 21 & JDK 21)

```java
package Map;

import java.util.TreeMap;

public class TreeMapExample {
    public static void main(String[] args) {
        TreeMap<Integer,String> treeMap = new TreeMap<>();

        //Adding elements
        treeMap.put(3,"Three");
        treeMap.put(1,"One");
        treeMap.put(4,"Four");
        treeMap.put(2,"Two");

        //displaying the elements
        System.out.println("TreeMap :" + treeMap);

        //Accessing the elements
        System.out.println("Value for key 2
:"+treeMap.get(2));

        //Removing an element
        treeMap.remove(2);
        System.out.println("After removal :" + treeMap);

        //using other methods
        System.out.println("First Key: "+ treeMap.firstKey());
        System.out.println("Last Key : "+ treeMap.lastKey());
    }
}
```

To Reverse the order

```
package Map;

import java.util.Collections;
import java.util.TreeMap;

public class TreeMapExample {
    public static void main(String[] args) {
        TreeMap<Integer,String> treeMap = new
TreeMap<>(Collections.reverseOrder());

        //Adding elements
        treeMap.put(3,"Three");
        treeMap.put(1,"One");
        treeMap.put(4,"Four");
        treeMap.put(2,"Two");

        //displaying the elements
        System.out.println("TreeMap :" + treeMap);

    }
}
```

Using the Custom Comparator:

```
package Map;

import java.util.Collections;
import java.util.Comparator;
import java.util.TreeMap;

public class TreeMapExample {
    public static void main(String[] args) {
        TreeMap<Integer,String> treeMap = new
TreeMap<>(Comparator.reverseOrder());

        //Adding elements
        treeMap.put(3,"Three");
        treeMap.put(1,"One");
        treeMap.put(4,"Four");
        treeMap.put(2,"Two");

        //displaying the elements
        System.out.println("TreeMap with custom comparator :"
+ treeMap);

    }
}
```

Using Navigable Methods:

```
package Map;

import java.util.Comparator;
import java.util.TreeMap;

public class NavigableMethodsExample {
    public static void main(String[] args) {
        TreeMap<Integer,String> treeMap = new TreeMap<>();

        treeMap.put(10,"Ten");
        treeMap.put(20,"Twenty");
        treeMap.put(30,"Thirty");
        treeMap.put(40,"Fourty");

        //Finding Keys
        System.out.println("Key higher than 20 : "+
treeMap.higherKey(20));
        System.out.println("Key lower than 20 : "+
treeMap.lowerKey(20));
        System.out.println("Ceiling key of 25 : "+
treeMap.ceilingKey(25));
        System.out.println("Floor Key of 25   : "+
```

```
treeMap.floorKey(25));

        //Polling entries
        System.out.println("Poll First Entry : "+
treeMap.pollFirstEntry());
        System.out.println("Poll Last Entry  : "+
treeMap.pollLastEntry());
        System.out.println("TreeMap: "+ treeMap);
    }
}
```

Comparable:

It is an interface which is used for defining a natural ordering for objects of a class.

The Comparable interface is a part of the java.lang package and it has single method: compareTo(T o);

Comparable (Java SE 21 & JDK 21)

```
package ComparableEx;

public class Employee implements Comparable<Employee>{
    private int id;
    private  String name;
    private double salary;

    public Employee(int id, String name, double salary){
        this.id=id;
        this.name=name;
        this.salary=salary;
    }

    //Getters

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }

    //Implement compareTo method
    @Override
    public int compareTo(Employee other){
        return Double.compare(this.salary, other.salary);
    }
```

```
    //toString Method
    @Override
    public String toString(){
        return "Employee{id = "+ id + ", name="+ name+",
salary=" + salary+ "}";
    }
}
```

```
package ComparableEx;

import java.util.ArrayList;
import java.util.Collections;

public class ComparableExample {
    public static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();

        employees.add(new Employee(1,"Krishna",85000));
        employees.add(new Employee(2,"Govind",55000));
        employees.add(new Employee(3,"Gopal",75000));

        System.out.println("Before Sorting: ");
        for(Employee e : employees){
            System.out.println(e);
        }

        //Sort using Comparable
        Collections.sort(employees);

        System.out.println("\n After Sorting with salary :");
        for(Employee e : employees){
            System.out.println(e);
        }
    }
}
```

Explanation?

Comparator:

Why?

- Custom Sorting Logic
- Multiple Sort Orders
- Seperation of concern

Comparator (Java SE 21 & JDK 21)

```
package ComparatorExample;

public class Employee {
    private int id;
    private  String name;
    private double salary;

    public Employee(int id, String name, double salary){
        this.id=id;
        this.name=name;
        this.salary=salary;
    }
    //Getters

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getSalary() {
        return salary;
    }

    //toString Method
    @Override
    public String toString(){
        return "Employee{id = "+ id + ", name="+ name+",
salary=" + salary+ "}";
    }
}
```

```java
package ComparatorExample;

import java.util.Comparator;

//Sort by name
public class NameComparator implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2){
        return e1.getName().compareTo(e2.getName());
    }
}
```

```java
package ComparatorExample;
import java.util.Comparator;

public class SalaryComparator implements Comparator<Employee>
{
    @Override
    public int compare(Employee e1, Employee e2){
        return Double.compare(e1.getSalary(), e2.getSalary());
    }
}
```

```java
package ComparatorExample;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ComparatorExample {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();

        employees.add(new Employee(1,"Krishna",85000));
        employees.add(new Employee(2,"Govind",55000));
        employees.add(new Employee(3,"Gopal",75000));

        //Sort By salary
        System.out.println("Sorting by salary");
        Collections.sort(employees,new SalaryComparator());
        employees.forEach(System.out::println);

        //Sort By name
        System.out.println("Sorting by Name");
        Collections.sort(employees,new NameComparator());
        employees.forEach(System.out::println);
```

```
        }
}
```