****What is the difference between Comparable<T> and Comparator<T> :

---

* Comparable<T> and Comparator<T> both are functional interfaces.

Comparable<T>
---

* It is a predefined functional interface available in java.lang package.

* It is used to compare the content of two objects, The comparison will be done by using the following cases :

  a) First object content = another object content -> 0
  b) First Object content > another object content -> 1
  c) First Object content < another object content -> -1

```
public interface Comparable<T>
{
   public abstract int compareTo(T x);
}
```

Note : In Comparable we need to write the logic in the BLC class only because current object (this keyword) support is reqd.

Comparator<T>
---

* It is a predefined functional interface available in java.util package.
* It is also used to compare two objects by using same criteria.
* By using Comparator, We can implement Lambda Expression.

@FunctionalInterface
public interface Comparator<T>
{
  public int compare(T x, T y);
}

Comparable<T>	Comparator<T>
1) Available in java.lang package	1) Available in java.util package
2) public int compareTo(T x)	2) public int compare(T x, T y)
3) Need to modify class due to current object support.	3) Need not to modify the BLC class
4) Due to current object support, Writing Lambda is not possible	4) We can implement with Lambda
5) We can write only one sorting logic	5) We can write multiple sorting logic.
6) Provides default natural sorting order [Ascending Order OR Alphabetical order]	6) Provided user-defined sorting order
7) Arrays.sort(T[] arr).	7) Arrays.sort(T[] arr, Comparator<T> cmp)

//Program on Comparable<T> :

Q16) Sort the Employee Object based on the employee Id. [Comparable]

```
package com.ravi.basic;

import java.util.Arrays;

record Employee(Integer id, String name, Double salary) implements Comparable<Employee>
{
    @Override
    public int compareTo(Employee e2)
    {
      return Integer.compare(this.id(), e2.id());
    }
}

public class Array1x16
{
    public static void main(String[] args)
    {
        Employee []employees = new Employee[3];

        employees[0] = new Employee(333, "Scott", 1200D);
        employees[1] = new Employee(111, "Zuker", 900D);
        employees[2] = new Employee(222, "Aryan", 1000D);

        System.out.println("Original Array Data :");

        for(Employee employee : employees)
        {
            System.out.println(employee);
        }

        Arrays.sort(employees);

        System.out.println("Array Data after sorting based on the Id :");

        for(Employee employee : employees)
        {
            System.out.println(employee);
        }


    }
}
```

Limitation of Comparable<T>
---

* As we know, to work with Comparable<T>, current object (this) support is required. this keyword is available in the current class only so, the conclusion is, we need to modify the BLC class in case of Comparable<T>.

* If the BLC class is provided by another developer in the .class file format then Comparable<T> will not work here.

* By using Comparable we can write only one sorting logic.

To avoid the above said limitations, We introduced Comparator<T>

```
package com.ravi.basic;

public record Product(Integer id, String name, Double price)
{

}
 package com.ravs.oasic;

 import java.util.Arrays;
 import java.util.Comparator;

 /*
      Sort the Product Object based on the following criteria.

        a) Based on the Product Id
        b) Based on the Product Name
        c) Based on the Product Price

  */

 public class Array1x17
 {
    public static void main(String[] args)
    {
        Product []products = new Product[5];

        products[0] = new Product(444, "Mobile", 437890);
        products[1] = new Product(555, "Camera", 337890);
        products[2] = new Product(111, "Laptop", 937890);
        products[3] = new Product(333, "LED", 837890);
        products[4] = new Product(222, "Watch", 237890);

        System.out.println("Original Array Data :");

        for(Product product : products)
        {
            System.out.println(product);
        }

        System.out.println("......................");
        System.out.println("Sort the product Object based on the Id :");

        //Anonymous inner class for Comparator
        Comparator<Product> compId = new Comparator<Product>()
        {
         @Override
         public int compare(Product p1, Product p2)
         {
             return Integer.compare(p1.id(), p2.id());
         }
        };

        Arrays.sort(products, compId);

        for(Product product : products)
        {
            System.out.println(product);
        }

        System.out.println("......................");
        System.out.println("Sort the product Object based on the Name :");

       Comparator<Product> compName = (p1,p2) -> p1.name().compareTo(p2.name());

        Arrays.sort(products, compName);

        for(Product product : products)
        {
            System.out.println(product);
        }

        System.out.println("......................");
        System.out.println("Sort the product Object based on the Price :");

        Arrays.sort(products,(p1,p2)-> Double.compare(p1.price(), p2.price()));

        for(Product product : products)
        {
            System.out.println(product);
        }


    }
 }
```

Q18) Sort the Integer in Descending order.

* By default compareTo(T x) method of Integer class provides ascending order (default natural sorting order) If we want to write in descending order then we should use Comparator.

```
 package com.ravi.basic;

 import java.util.Arrays;

 // Sort the Integer array in Descending order.

 public class Array1x18
 {
    public static void main(String[] args)
    {
        Integer []arr = {12, 90,56, 34};
        Arrays.sort(arr,(i1,i2)-> i1.compareTo(i1));
        System.out.println(Arrays.toString(arr));
    }
 }
```