Comparable Vs Comparator

In Java, both Comparable and Comparator are interfaces that allow you to define custom ordering for objects. They are used in different scenarios and provide different levels of flexibility.

Comparable:

The Comparable interface is used to define the natural ordering of objects. When a class implements the Comparable interface, it defines a method called compareTo(), which specifies how the objects of that class should be compared to each other. This method returns an integer value that indicates the relationship between two objects. For example, a class representing a person could implement Comparable to compare based on age, name, or any other criteria.

The Comparable interface allows you to sort objects in collections that support ordering, such as arrays or collections like TreeSet or TreeMap. By implementing Comparable, you enable the objects of your class to be sorted using the Collections.sort() method or automatically ordered in data structures that rely on natural ordering.

```
public class Person implements Comparable<Person> {
    private String name;
    private int age;

    // Constructor, getters, and setters

    @Override
    public int compareTo(Person other) {
        // Compare based on age
        return Integer.compare(this.age, other.age);
    }
}
```

Comparator:

The Comparator interface is used to define custom comparison logic separate from the objects being compared. Unlike Comparable, which is implemented by the class of the objects being compared, Comparator is a separate class that implements the compare() method. It allows you to define multiple ways of comparing objects without modifying their original class or when the objects don't implement Comparable.

Comparator provides more flexibility because you can define different comparison rules for the same objects, based on different criteria. It allows you to sort objects in different ways, depending on the context or user requirements.

```
public class PersonComparator implements Comparator<Person> {
    @Override
    public int compare(Person p1, Person p2) {
        // Compare based on name
        return p1.getName().compareTo(p2.getName());
    }
}
```

You can use a Comparator with sorting methods like Collections.sort() or specific data structures that accept a Comparator, such as TreeSet or TreeMap. When using a Comparator, you pass an instance of the comparator to the sorting method or the data structure, allowing you to define the desired comparison logic.

In summary, Comparable is used to define natural ordering within the class being compared, while Comparator provides a separate class for defining custom comparison logic, allowing you to have multiple ways of comparing objects.

Comparable	Comparator
1) Comparable provides a single sorting sequence. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides multiple sorting sequences. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable affects the original class, i.e., the actual class is modified.	Comparator doesn't affect the original class, i.e., the actual class is not modified.
3) Comparable provides compareTo() method to sort elements.	Comparator provides compare() method to sort elements.
4) Comparable is present in java.lang package.	A Comparator is present in the java.util package.
5) We can sort the list elements of Comparable type by Collections.sort(List) method.	We can sort the list elements of Comparator type by Collections.sort(List, Comparator) method.

```
package com.hdfc.collections;

import java.util.Objects;

public class Player implements Comparable<Player> {
   private int rating; // 1,2,3,4,5
   private String name;
   private int age;

   public Player(int rating, String name, int age) {
      this.rating = rating;
      this.name = name;
      this.age = age;
   }

   public int getRating() {
      return rating;
   }

   public void setRating(int rating) {
      this.rating = rating;
   }

   public String getName() {
      return name;
   }

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

```
}

public int getAge() {
   return age;
}

public void setAge(int age) {
   this.age = age;
}

@Override
public String toString() {
   return "Player{" +
         "rating=" + rating +
         ", name='" + name + '\'' +
         ", age=" + age +
         '}';
}

@Override
public boolean equals(Object o) {
   if (this == o) return true;
   if (o == null || getClass() != o.getClass()) return false;

   Player player = (Player) o;

   if (rating != player.rating) return false;
   if (age != player.age) return false;
   return Objects.equals(name, player.name);
}

@Override
public int hashCode() {
   int result = rating;
   result = 31 * result + (name != null ? name.hashCode() : 0);
   result = 31 * result + age;
   return result;
}

// 0 = both are same
// -1 : less   this.rating is less than o.getRating()
//+ 1 : more  this.rating is more than o.getRating()
@Override
public int compareTo(Player o) {
   //Number, natural order sorting
    //return Integer.compare(this.rating, o.rating);  //rating

   //reverse logic
   int i = Integer.compare(this.rating, o.rating);
   return -i;

   //Double.compare, Byte.compare, Short.compare, Long.compare

   //String
   //return this.name.compareTo(o.name);
```

```
      //reverse logic
     /* if (this.rating == o.rating) {
         return 0;
      } else if (this.rating < o.rating) {
         return +1;
      } else {
         return -1;
      }*/

      // natural order sorting
      //if (this.rating == o.rating) {
      //         return 0;
      //      } else if (this.rating < o.rating) {
      //         return -1;
      //      } else {
      //         return +1;
      //      }

      //return this.rating - o.getRating(); //avoid
   }
}



package com.hdfc.collections;

import java.util.*;

public class PlayerTest {

   public static void main(String[] args) {


      Player p2 = new Player(4, "A", 20);
      Player p3 = new Player(3, "F", 20);
      Player p4 = new Player(2, "G", 40);
      Player p5 = new Player(5, "X", 50);//
      Player p6 = new Player(5, "A", 60);//
      Player p1 = new Player(1, "E", 10);


      Set<Player> players =  new TreeSet<>();
      players.add(p1);
      players.add(p2);
      players.add(p3);
      players.add(p4);
      players.add(p5);
      players.add(p6);

      for (Player player: players){
         System.out.println(player);
      }
      //System.out.println(players);
```

```
        Set<Integer> number =  new TreeSet<>(); //order natural order
        number.add(4);
        number.add(9);
        number.add(2);
        //no sorting
        //System.out.println(number);

        Set<String> name =  new TreeSet<>(); //order natural order
        name.add("Sumit");
        name.add("Amit");
        name.add("Ravi");
        name.add("Mayank");
        name.add("Sourabh");

        //no sorting
       // System.out.println(name);


    }
}
```

```
package com.hdfc.collections;

import java.util.Objects;

public class Player implements Comparable<Player> {
    private int rating; // 1,2,3,4,5
    private String name;
    private int age;

    public Player(int rating, String name, int age) {
        this.rating = rating;
        this.name = name;
        this.age = age;
    }

    public int getRating() {
        return rating;
    }

    public void setRating(int rating) {
        this.rating = rating;
    }

    public String getName() {
        return name;
    }

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

```
public int getAge() {
   return age;
}

public void setAge(int age) {
   this.age = age;
}

@Override
public String toString() {
   return "Player{" +
         "rating=" + rating +
         ", name='" + name + '\'' +
         ", age=" + age +
         '}';
}

@Override
public boolean equals(Object o) {
   if (this == o) return true;
   if (o == null || getClass() != o.getClass()) return false;

   Player player = (Player) o;

   if (rating != player.rating) return false;
   if (age != player.age) return false;
   return Objects.equals(name, player.name);
}

@Override
public int hashCode() {
   int result = rating;
   result = 31 * result + (name != null ? name.hashCode() : 0);
   result = 31 * result + age;
   return result;
}

// 0 = both are same
// -1 : less   this.rating is less than o.getRating()
//+ 1 : more  this.rating is more than o.getRating()
@Override
public int compareTo(Player o) {
   //Number, natural order sorting
    //return Integer.compare(this.rating, o.rating);  //rating

   //reverse logic
   int i = Integer.compare(this.rating, o.rating);
   return -i;

   //Double.compare, Byte.compare, Short.compare, Long.compare

   //String
   //return this.name.compareTo(o.name);

   //reverse logic
```

```
      /* if (this.rating == o.rating) {
         return 0;
       } else if (this.rating < o.rating) {
         return +1;
       } else {
         return -1;
       }*/

       // natural order sorting
       //if (this.rating == o.rating) {
       //          return 0;
       //       } else if (this.rating < o.rating) {
       //          return -1;
       //       } else {
       //          return +1;
       //       }

       //return this.rating - o.getRating(); //avoid
   }
}
package com.hdfc.collections;

import java.util.*;

public class PlayerTest {

   public static void main(String[] args) {


      Player p2 = new Player(4, "A", 20);
      Player p3 = new Player(3, "F", 20);
      Player p4 = new Player(2, "G", 40);
      Player p5 = new Player(5, "X", 50);//
      Player p6 = new Player(5, "A", 60);//
      Player p1 = new Player(1, "E", 10);


      Map<Player, Player> players =  new TreeMap<>();
      players.put(p3, p3);
      players.put(p4, p4);
      players.put(p5, p5);
      players.put(p6, p6);
      players.put(p1, p1);
      players.put(p2, p2);

      for (Map.Entry<Player, Player> player: players.entrySet()){
         System.out.println("key="+player.getKey() +",  value="+player.getValue());
      }
      //System.out.println(players);


      Set<Integer> number =  new TreeSet<>(); //order natural order
      number.add(4);
      number.add(9);
      number.add(2);
```

 Contact: 8087883669

```
        //no sorting
        //System.out.println(number);

        Set<String> name =  new TreeSet<>(); //order natural order
        name.add("Sumit");
        name.add("Amit");
        name.add("Ravi");
        name.add("Mayank");
        name.add("Sourabh");

        //no sorting
       // System.out.println(name);


    }
}
```

```
package com.hdfc.collections;

import java.util.Objects;


// jar
public class Player {
    private int rating;
    private String name;
    private int age;

    public Player(int rating, String name, int age) {
        this.rating = rating;
        this.name = name;
        this.age = age;
    }

    public int getRating() {
        return rating;
    }

    public void setRating(int rating) {
        this.rating = rating;
    }

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }
```

```
    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Player{" +
                "rating=" + rating +
                ", name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Player player = (Player) o;

        if (rating != player.rating) return false;
        if (age != player.age) return false;
        return Objects.equals(name, player.name);
    }

    @Override
    public int hashCode() {
        int result = rating;
        result = 31 * result + (name != null ? name.hashCode() : 0);
        result = 31 * result + age;
        return result;
    }
}
```

```
package com.hdfc.collections;

import java.util.Comparator;

public class RatingComparator implements Comparator<Player> {

    @Override
    public int compare(Player p1, Player p2) {

        //if both are same the 0
        //if p1 is less than p2 -1
        //else (p1 is more than p2) +1
        if (p1.getRating() == p2.getRating()) {
            return 0;
```

```
        } else if (p1.getRating() < p2.getRating()) {
            return -1;
        } else {
            return 1;
        }


        //return Integer.compare(p1.getRating(), p2.getRating());

        //0
        // if p1 rating is less than p2  - value
        //else p2 rating is more than p2 + value
        //avoid this kind of implemenation
        //return p1.getRating() - p2.getRating();
    }
}
```

```
package com.hdfc.collections;

import java.util.Comparator;

public class PlayerAgeComparator implements Comparator<Player> {
    @Override
    public int compare(Player o1, Player o2) {
        return Integer.compare(o1.getAge(), o2.getAge());
    }
}
```

```
package com.hdfc.collections;

import java.util.*;

public class PlayerTest {

    public static void main(String[] args) {


        Player p2 = new Player(4, "A", 20);
        Player p3 = new Player(3, "F", 20);
        Player p4 = new Player(2, "G", 40);
        Player p5 = new Player(5, "X", 50);//
        Player p6 = new Player(5, "A", 60);//
        Player p1 = new Player(1, "E", 10);


        List<Player> list = new ArrayList<>();
        list.add(p4);
        list.add(p2);
```

```
        list.add(p6);
        list.add(p1);
        list.add(p5);
        list.add(p3);


       // Collections.sort(list);//reason: no instance(s) of type variable(s) T exist so that Player conforms to Comparable<?
super T>
        Collections.sort(list, new PlayerAgeComparator());

        for (Player p: list){
           System.out.println(p);
        }









        Map<Player, Player> players = new TreeMap<>(new RatingComparator());
        players.put(p3, p3);
        players.put(p4, p4);
        players.put(p5, p5);
        players.put(p6, p6);
        players.put(p1, p1);
        players.put(p2, p2);

        for (Map.Entry<Player, Player> entry : players.entrySet()) {
           //System.out.println("key=" + entry.getKey() + ",  value=" + entry.getValue());
        }
        //System.out.println(players);

        Set<Player> playerSet = new TreeSet<>(new RatingComparator());
        playerSet.add(p3);
        playerSet.add(p4);
        playerSet.add(p5);
        playerSet.add(p6);
        playerSet.add(p1);
        playerSet.add(p2);

        for (Player set : playerSet) {
          // System.out.println(set);
        }
        //System.out.println(playerSet);




        Set<Integer> number = new TreeSet<>(); //order natural order
```

```
        number.add(4);
        number.add(9);
        number.add(2);
        //no sorting
        //System.out.println(number);

        Set<String> name = new TreeSet<>(); //order natural order
        name.add("Sumit");
        name.add("Amit");
        name.add("Ravi");
        name.add("Mayank");
        name.add("Sourabh");

        //no sorting
        // System.out.println(name);


    }
}
```