Topics to cover:

1. Comparable vs Comparator in Java

1. Comparable vs Comparator

Java provides two interfaces to sort objects using data members of the class:

1.1 Comparable

1.2 Comparator

1.1 Comparable interface

A comparable object is capable of comparing itself with another object. The class itself must implement the java.lang. Comparable interface to compare its instances.

Consider a Movie class that has members like, rating, name, year. Suppose we wish to sort a list of Movies based on year of release. We can implement the Comparable interface with the Movie class, and we override the method compareTo() of Comparable interface.

Program 1:

```
// A Java program to demonstrate use of Comparable
import java.io.*;
import java.util.*;
```

```
// A class 'Movie' that implements Comparable
class Movie implements Comparable<Movie>
{
	private double rating;
	private String name;
	private int year;

	// Used to sort movies by year
	public int compareTo(Movie m)
	{
		return this.year - m.year;
	}

	// Constructor
	public Movie(String nm, double rt, int yr)
	{
		this.name = nm;
		this.rating = rt;
		this.year = yr;
	}

	// Getter methods for accessing private data
	public double getRating() { return rating; }
	public String getName() { return name; }
	public int getYear()   { return year; }
}

// Driver class
class Main
{
	public static void main(String[] args)
	{
		ArrayList<Movie> list = new ArrayList<Movie>();
		list.add(new Movie("Force Awakens", 8.3, 2015));
		list.add(new Movie("Star Wars", 8.7, 1977));
		list.add(new Movie("Empire Strikes Back", 8.8, 1980));
		list.add(new Movie("Return of the Jedi", 8.4, 1983));

		Collections.sort(list);
```

```
                System.out.println("Movies after sorting : ");
                for (Movie movie: list)
                {
                        System.out.println(movie.getName() + " " +
                                                        movie.getRating() + " " +
                                                        movie.getYear());
                }
        }
}
```

Output

```
Movies after sorting :
Star Wars 8.7 1977
Empire Strikes Back 8.8 1980
Return of the Jedi 8.4 1983
Force Awakens 8.3 2015
```

Now, suppose we want to sort movies by their rating and names as well. When we make a collection element comparable(by having it implement Comparable), we get only one chance to implement the compareTo() method. The solution is using Comparator.

1.2 Comparator interface

Unlike Comparable, Comparator is external to the element type we are comparing. It's a separate class. We create multiple separate classes (that implement Comparator) to compare by different members.
Collections class has a second sort() method and it takes Comparator. The sort() method invokes the compare() to sort objects.

To compare movies by Rating, we need to do 3 things:

- Create a class that implements Comparator (and thus the compare() method that does the work previously done by compareTo()).

- Make an instance of the Comparator class.
- Call the overloaded sort() method, giving it both the list and the instance of the class that implements Comparator.

Program 2:

```
// A Java program to demonstrate Comparator interface
import java.io.*;
import java.util.*;

// A class 'Movie' that implements Comparable
class Movie implements Comparable<Movie> {
	private double rating;
	private String name;
	private int year;

	// Used to sort movies by year
	public int compareTo(Movie m)
	{
		return this.year - m.year;
	}

	// Constructor
	public Movie(String nm, double rt, int yr)
	{
		this.name = nm;
		this.rating = rt;
		this.year = yr;
	}

	// Getter methods for accessing private data
	public double getRating() { return rating; }
	public String getName() { return name; }
	public int getYear() { return year; }
}
```

```
// Class to compare Movies by ratings
class RatingCompare implements Comparator<Movie> {
	public int compare(Movie m1, Movie m2)
	{
		if (m1.getRating() < m2.getRating())
			return -1;
		if (m1.getRating() > m2.getRating())
			return 1;
		else
			return 0;
	}
}

// Class to compare Movies by name
class NameCompare implements Comparator<Movie> {
	public int compare(Movie m1, Movie m2)
	{
		return m1.getName().compareTo(m2.getName());
	}
}

// Driver class
class Main {
	public static void main(String[] args)
	{
		ArrayList<Movie> list = new ArrayList<Movie>();
		list.add(new Movie("Force Awakens", 8.3, 2015));
		list.add(new Movie("Star Wars", 8.7, 1977));
		list.add(
			new Movie("Empire Strikes Back", 8.8, 1980));
		list.add(
			new Movie("Return of the Jedi", 8.4, 1983));

		// Sort by rating : (1) Create an object of
		// ratingCompare
		//                                (2) Call Collections.sort
		//                                (3) Print Sorted list
		System.out.println("Sorted by rating");
		RatingCompare ratingCompare = new RatingCompare();
```

```
                Collections.sort(list, ratingCompare);
                for (Movie movie : list)
                        System.out.println(movie.getRating() + " "
                                                + movie.getName() + " "
                                                + movie.getYear());

                // Call overloaded sort method with RatingCompare
                // (Same three steps as above)
                System.out.println("\nSorted by name");
                NameCompare nameCompare = new NameCompare();
                Collections.sort(list, nameCompare);
                for (Movie movie : list)
                        System.out.println(movie.getName() + " "
                                                + movie.getRating() + " "
                                                + movie.getYear());

                // Uses Comparable to sort by year
                System.out.println("\nSorted by year");
                Collections.sort(list);
                for (Movie movie : list)
                        System.out.println(movie.getYear() + " "
                                                + movie.getRating() + " "
                                                + movie.getName() + " ");
        }
}
```

Output

```
Sorted by rating
8.3 Force Awakens 2015
8.4 Return of the Jedi 1983
8.7 Star Wars 1977
8.8 Empire Strikes Back 1980

Sorted by name
Empire Strikes Back 8.8 1980
Force Awakens 8.3 2015
```

Return of the Jedi 8.4 1983
Star Wars 8.7 1977

Sorted by year
1977 8.7 Star Wars
1980 8.8 Empire Strikes Back
1983 8.4 Return of the Jedi
2015 8.3 Force Awakens

- Comparable is meant for objects with natural ordering which means the object itself must know how it is to be ordered. For example Roll Numbers of students. Whereas, Comparator interface sorting is done through a separate class.

- Logically, Comparable interface compares “this” reference with the object specified and Comparator in Java compares two different class objects provided.

- If any class implements Comparable interface in Java then collection of that object either List or Array can be sorted automatically by using Collections.sort() or Arrays.sort() method and objects will be sorted based on there natural order defined by CompareTo method.

- A basic differentiating feature is that using comparable we can use only one comparison. Whereas, we can write more than one custom comparators as you want for a given type, all using different interpretations of what sorting means. Like in the comparable example we could just sort by only one attribute, i.e., year but in the comparator, we were able to use different attributes like rating, name, and year as well.

To summarize, if sorting of objects needs to be based on natural order then use Comparable whereas if you sorting needs to be done on attributes of different objects, then use Comparator in Java.