Experiment#		Student ID	
Date		Student Name	

11. Comparator and Comparable

Aim/Objective: Analyse the practical application of the Comparator and Comparable interfaces in real-world scenarios, discussing their roles, advantages, and differences.

Description: Student will be able to understand and apply the concept of Comparator and Comparable Interfaces.

Pre-Requisites: A Strong knowledge on Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Discuss the differences between Comparator and Comparable by filling the below mentioned table.

S.no	Comparable	Comparator
1.	Used to define the natural ordering of objects. It allows a class to specify how its instances.	Used to define the natural ordering of objects. It allows for multiple ways to compare objects.
2.	A class implements the interface, & overrides the comparable() method.	A separate class implements the interface & overrides the compare() method.
3.	The method Signature is a data type of the object being compared	The method Signature is a data-type allowing comparison b/w 2 objects.
4.	Can only provide one Sorting method for the class.	Can provide multiple sorting methods, allowing for different criteria.

Experiment#		Student ID	
Date		Student Name	

2) Write a Java program that sorts a Linked List using the Comparable interface.

```
import java.util.*;
class Person implements Comparable<Person>{
   Private String name;
   Private int age;
   public Person(String name,int age) {
      this.name=name;
      this.age=age;
   }
   public String getName() {
      return name;
   }
   public int getAge(){
      return age;
   }
   public int compareTo(Person other){
      return Integer.compare(this.age,other.age);
   }
   public class LinkedListSortExample{
      Public static void main(String[] args) {
         LinkedList<Person> P = new LinkedList<>();
         P.add(new Person("Alice", 30));
         P.add(new Person("David", 20));
         System.out.println("Original List:");
      }
   }
```

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) Create a Java program that sorts a list of Movie objects by their year of release. Define the Movie class with attributes such as rating, name, and year. Implement the Comparable interface in the Movie class and override the compareTo() method to sort the movies based on their release year.

Program:

```
import java.util.*;
class Movie implements Comparable<Movie>{
   Private String name;
   Private int year;
   Private double rating;
  Public Movie(String name, int year, double rating) {
      this.name=name;
      this.year=year;
      this.rating=rating;
 }
  Public String getName() {
        return name;
 }Public int getYear() {
        return year;
 }public double getRating(){
        return rating;
 }public int compareTo(Movie other){
        return Integer.compare(this.year, other.year);
 }
```

Experiment#		Student ID	
Date		Student Name	

```
public class MovieSorter{
    public static void main(String[] args){
      List<Movie> movies = new Arraylist<>();
       movies.add(new Movie("Game changer", 2024, 9.5));
       movies.add(new Movie("Devara", 2025, 9.0));
       Collections.sort(movies);
       System.out.println("Movies sorted successfully.");
    }
}
```

Experiment#		Student ID	
Date		Student Name	

2) You are tasked with developing a system to manage employee records for a large corporation. The Employee class has attributes such as id, name, department, and salary. Different departments and teams need to sort employee records based on different criteria, such as salary, name, and department.

Implement a Java program that sorts a list of Employee objects using the Comparator interface. The program should allow sorting by multiple criteria: by salary (ascending and descending), by name (alphabetical order), and by department (alphabetical order).

Program:

```
import java.util.*;
class Employee{
    private int id;
    private String n;
    private String d;
    private double s;
    public Employee(int id, String n, String d, double s){
        this.id = id;
        this.n = n;
        this.d = d;
        this.s = s;
    }public int getId(){
        return id;
    }public String getN(){
        return n;
    }public String getD(){
        return d;
    }
```

```
public double getS() {
    return s;
} class SalaryComparator implements Comparator <Employee> {
    public int compare (Employee e1, Employee e2) {
      return Double.compare(e1.getS(), e2.getS());
    }
} public class EmployeeSorter {
    public static void main(String[] args) {
      List <Employee> e = new ArrayList <>();
      e.add(new Employee(1, "John", "IT", "50000"));
      e.add(new Employee(2, "Alice", "HR", 45000));
      e.add(new Employee(3, "Ram", "MD", 100000));
      collections.sort(e, new SalaryComparator());
      System.out.println("Salaries sorted (asc):");
      PrintEmployees(e);
    }
}
```

Experiment#		Student ID	
Date		Student Name	

✓ **Data and Results:**

O/P:

```
Salaries Sorted (asc) :
Employee{id=3, n='Ram', d='MD', S=100000}
Employee{id=1, n='John', d='IT', S=50000.0}
Employee{id=2, n='Alice', d='HR', S=45000.0}
```

✓ **Analysis and Inferences:**

These classes implement the 'Comparator' interface to define different sorting criteria.

Sample VIVA-VOCE Questions (In-Lab):

1) List the usage of comparable Interface.

Usages:

i) Sorting Arrays & Lists

ii) Defining Natural Ordering

iii) Overriding

iv) Sorting predefined classes.

Experiment#		Student ID	
Date		Student Name	

2) List the usage of comparator interface.

Usages:

i) Sorting custom objects

ii) Chaining Comparators

iii) Anonymous Inner Classes

iv) sorting Maps.

3) What is the purpose of the compareTo method?

The 'compareTo' method in Java that to in those servers & are primarily related to comparing objects & defining their natural ordering.

4) What happens if you do not override the compareTo method when implementing Comparable?

If you do not override the compareTo method when implementing Comparable leads to default behaviour which means of the regular implementation, and sorting issues & finally leads to Inconsistent Ordering.

5) What is the difference between Comparable and Comparator?

The comparable interface is used to define a default sorting order for objects of a class.

The Comparator interface allows you to define multiple ways to compare objects, providing flexibility in sorting.

Post-Lab:

1) Develop a Java program to compare movies by their ratings using a custom Comparator implementation. Your program should follow these steps: a. Implement a class that serves as a Comparator for Movie objects, providing the comparison logic based on movie ratings. b. Instantiate the Comparator class. c. Utilize the overloaded sort () method, passing both the list of movies and the instance of the Comparator class to perform the sorting.

Sample Input:

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

Sample 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 Force Awakens
```

Program:

```
import java.util.*;


public class MovieComparatorDemo {


  static class Movie {

    private String name;

    private double rating;

    private int year;


    public Movie(double rating, String name, int year) {

      this.rating = rating;

      this.name = name;

      this.year = year;

    }


    public String getName() {

      return name;
    }

    public double getRating() {

        return rating;

    }

    public int getYear() {

        return year;

    }

    @Override

    public String toString() {

        return rating + " " + name + " " + year;

    }

}

static class RatingComparator implements Comparator<Movie> {

    @Override

    public int compare(Movie m1, Movie m2) {
        return Double.compare(m1.getRating(), m2.getRating());

    }

}



static class NameComparator implements Comparator<Movie> {

    @Override

    public int compare(Movie m1, Movie m2) {

        return m1.getName().compareTo(m2.getName());

    }

}



static class YearComparator implements Comparator<Movie> {

    @Override

    public int compare(Movie m1, Movie m2) {

        return Integer.compare(m1.getYear(), m2.getYear());

    }

}



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

```
List<Movie> movies = new ArrayList<>();

movies.add(new Movie(8.4, "Return of the Jedi", 1983));

movies.add(new Movie(8.8, "Empire Strikes Back", 1980));

movies.add(new Movie(8.3, "Force Awakens", 2015));

movies.add(new Movie(8.7, "Star Wars", 1977));



Collections.sort(movies, new RatingComparator());

System.out.println("Sorted by rating");

for (Movie movie : movies) {

   System.out.println(movie);

}



Collections.sort(movies, new NameComparator());

System.out.println("\nSorted by name");

for (Movie movie : movies) {

   System.out.println(movie);

}



Collections.sort(movies, new YearComparator());
```

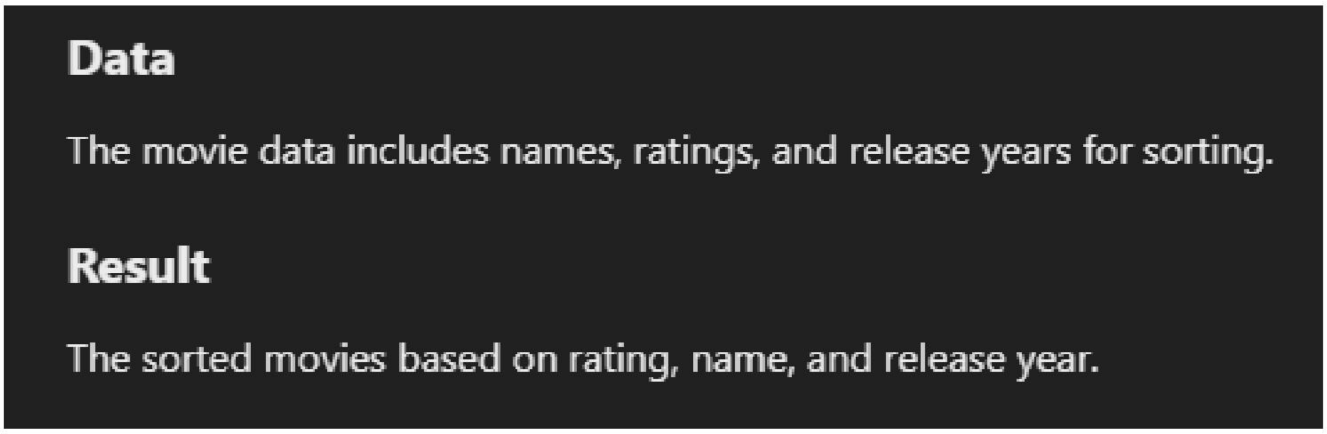
```
System.out.println("\nSorted by year");
        for (Movie movie : movies) {
            System.out.println(movie);
        }
    }
}
```

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

8.8 Empire Strikes Back 1980

8.3 Force Awakens 2015

8.4 Return of the Jedi 1983
```

```
8.7 Star Wars 1977



Sorted by year

8.7 Star Wars 1977

8.8 Empire Strikes Back 1980

8.4 Return of the Jedi 1983

8.3 Force Awakens 2015
```

✓ **Data and Results:**



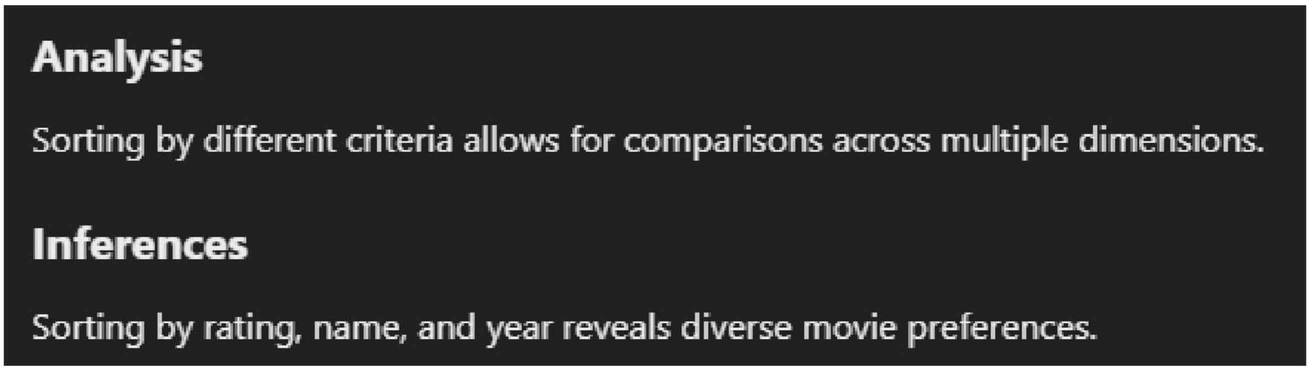
Data

The movie data includes names, ratings, and release years for sorting.

Result

The sorted movies based on rating, name, and release year.

✓ **Analysis and Inferences:**



Analysis

Sorting by different criteria allows for comparisons across multiple dimensions.

Inferences

Sorting by rating, name, and year reveals diverse movie preferences.

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.