Experiment#		**Student ID**	
Date		**StudentName**	[@KLWKS_bot] THANOS

7.ComparatorandComparable

Aim/Objective: AnalysethepracticalapplicationoftheComparatorandComparable interfaces in real-world scenarios, discussing their roles, advantages, and differences.

Description: Studentwillbeabletounderstandandapplytheconceptof Comparatorand Comparable Interfaces.

Pre-Requisites:AStrongknowledgeonClassesandObjectsinJAVA

Tools: EclipseIDEforEnterpriseJavaandWebDevelopers

Pre-Lab:

1) DiscussthedifferencesbetweenComparatorandComparablebyfillingthebelow mentioned table.

S.no	Comparable	Comparator
1.	Defines natural ordering using compareTo().	Defines custom ordering using compare().
2.	The class must implement Comparable.	No need for class to implement Comparator.
3.	Used for default object comparison.	Used for custom or multiple comparisons.
4.	Can be used with Collections.sort() and Arrays.sort().	Can be used with Collections.sort() and Arrays.sort(), but allows custom sorting logic.

2) WriteaJavaprogramthatsortsaLinkedListusingtheComparableinterface.

```
import java.util.LinkedList;
import java.util.Collections;

class Student implements Comparable<Student> {
   String name;
   int age;

   Student(String name, int age) {
      this.name = name;
      this.age = age;
   }

   public int compareTo(Student other) {
      return name.compareTo(other.name);
   }

   public String toString() {
      return name + " (" + age + ")";
   }
}

public class LinkedListSortExample {
   public static void main(String[] args) {
      LinkedList<Student> students = new LinkedList<>() {{
         add(new Student("Alice", 22));
         add(new Student("Bob", 20));
         add(new Student("Charlie", 23));
      }};

      Collections.sort(students);
      students.forEach(System.out::println);
   }
}
```

Experiment#		**Student ID**	
Date		**StudentName**	[@KLWKS_bot] THANOS

In-Lab:

1) CreateaJavaprogramthatsortsalist ofMovieobjectsbytheir yearofrelease. Define the Movie class with attributes such as rating, name, and year. Implement the Comparable interface inthe Movie classand override the compareTo()method to sort the movies based on their release year.

   Program:

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

class Movie implements Comparable<Movie> {
   private String name;
   private double rating;
   private int year;

   public Movie(String name, double rating, int year) {
      this.name = name;
      this.rating = rating;
      this.year = year;
   }

   public int getYear() {
      return year;
   }

   @Override
   public int compareTo(Movie other) {
      return Integer.compare(this.year, other.year);
   }

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

public class MovieSorter {
    public static void main(String[] args) {
        List<Movie> movies = new ArrayList<>();
        movies.add(new Movie("Inception", 8.8, 2010));
        movies.add(new Movie("The Shawshank Redemption", 9.3, 1994));
        movies.add(new Movie("The Godfather", 9.2, 1972));
        movies.add(new Movie("Pulp Fiction", 8.9, 1994));

        Collections.sort(movies);

        for (Movie movie : movies) {
            System.out.println(movie);
        }
    }
}
```

OUTPUT

```
Movie{name='The Godfather', rating=9.2, year=1972}
Movie{name='The Shawshank Redemption', rating=9.3, year=1994}
Movie{name='Pulp Fiction', rating=8.9, year=1994}
Movie{name='Inception', rating=8.8, year=2010}
```

2) You are tasked with developing a system to manage employee records for a large corporation.TheEmployeeclasshasattributessuchas id,name,department,andsalary. Different departments and teams need to sort employee records based on different criteria, such as salary, name, and department.
Implement a Java programthat sorts a list ofEmployee objects using the Comparator interface. The programshould allow sorting by multiple criteria: by salary(ascending anddescending),byname(alphabeticalorder),andbydepartment(alphabeticalorder).

Program:

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

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

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

    public int getId() { return id; }
    public String getName() { return name; }
    public String getDepartment() { return department; }
    public double getSalary() { return salary; }

    @Override
    public String toString() {
        return String.format("Employee{id=%d, name='%s', department='%s', salary=%.2f}", id, name, department, salary);
    }
}

public class Main {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>(List.of(
            new Employee(1, "Alice", "HR", 60000),
            new Employee(2, "Bob", "IT", 75000),
            new Employee(3, "Charlie", "Finance", 50000),
            new Employee(4, "David", "IT", 70000),
            new Employee(5, "Eve", "HR", 80000)
        ));

        List<String> sortLabels = List.of(
            "Sort by Salary Ascending",
            "Sort by Salary Descending",
            "Sort by Name",
            "Sort by Department"
        );
        List<Comparator<Employee>> comparators = List.of(
            Comparator.comparingDouble(Employee::getSalary),
            Comparator.comparingDouble(Employee::getSalary).reversed(),
            Comparator.comparing(Employee::getName),
            Comparator.comparing(Employee::getDepartment)
        );

        for (int i = 0; i < sortLabels.size(); i++) {
            System.out.println(sortLabels.get(i) + ":");

employees.stream().sorted(comparators.get(i)).forEach(System.out::println);
            System.out.println();
        }
    }
}
```

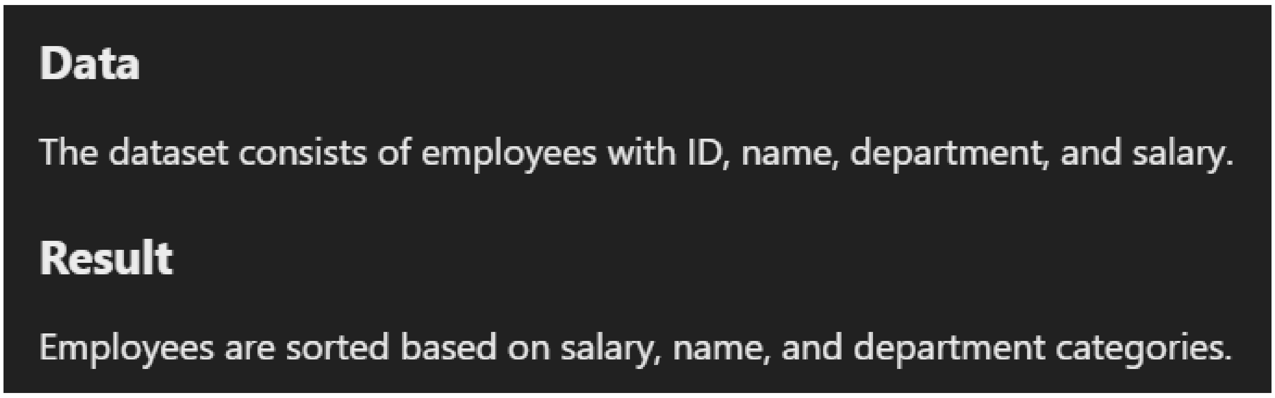
OUTPUT

```
Sort by Salary Ascending:
Employee{id=3, name='Charlie', department='Finance', salary=50000.00}
Employee{id=1, name='Alice', department='HR', salary=60000.00}
Employee{id=4, name='David', department='IT', salary=70000.00}
Employee{id=2, name='Bob', department='IT', salary=75000.00}
Employee{id=5, name='Eve', department='HR', salary=80000.00}
```

```
Sort by Salary Descending:
Employee{id=5, name='Eve', department='HR', salary=80000.00}
Employee{id=2, name='Bob', department='IT', salary=75000.00}
Employee{id=4, name='David', department='IT', salary=70000.00}
Employee{id=1, name='Alice', department='HR', salary=60000.00}
Employee{id=3, name='Charlie', department='Finance', salary=50000.00}

Sort by Name:
Employee{id=1, name='Alice', department='HR', salary=60000.00}
Employee{id=2, name='Bob', department='IT', salary=75000.00}
Employee{id=3, name='Charlie', department='Finance', salary=50000.00}
Employee{id=4, name='David', department='IT', salary=70000.00}
Employee{id=5, name='Eve', department='HR', salary=80000.00}

Sort by Department:
Employee{id=3, name='Charlie', department='Finance', salary=50000.00}
Employee{id=1, name='Alice', department='HR', salary=60000.00}
Employee{id=5, name='Eve', department='HR', salary=80000.00}
Employee{id=2, name='Bob', department='IT', salary=75000.00}
Employee{id=4, name='David', department='IT', salary=70000.00}
```

Experiment#		Student ID	
Date		StudentName	[@KLWKS_bot] THANOS

✓ **Dataand Results:**



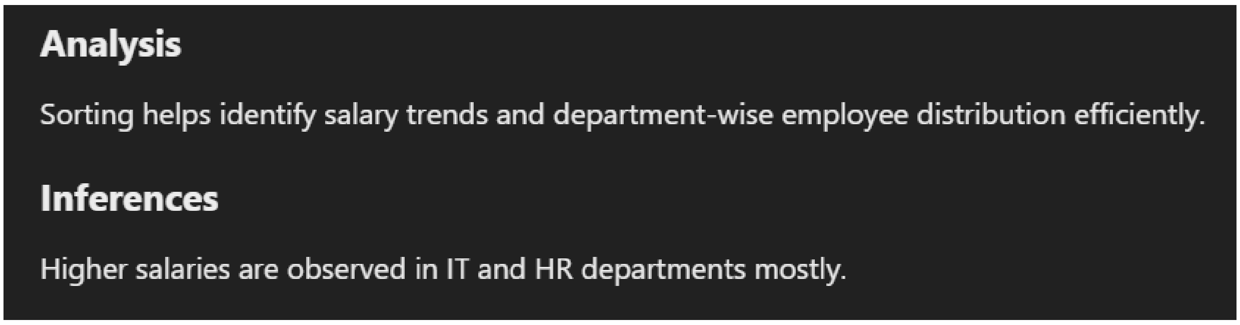
Data

The dataset consists of employees with ID, name, department, and salary.

Result

Employees are sorted based on salary, name, and department categories.

✓ **Analysisand Inferences:**



Analysis

Sorting helps identify salary trends and department-wise employee distribution efficiently.

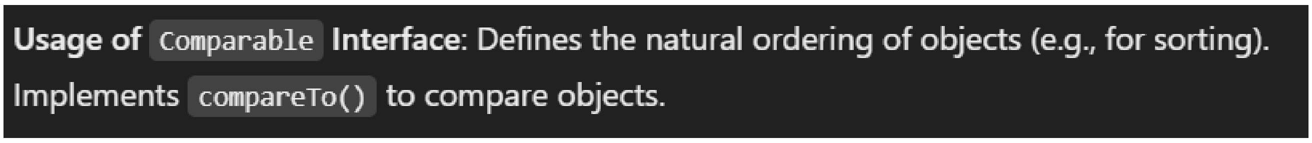
Inferences

Higher salaries are observed in IT and HR departments mostly.

Experiment#		**Student ID**	
Date		**StudentName**	[@KLWKS_bot] THANOS

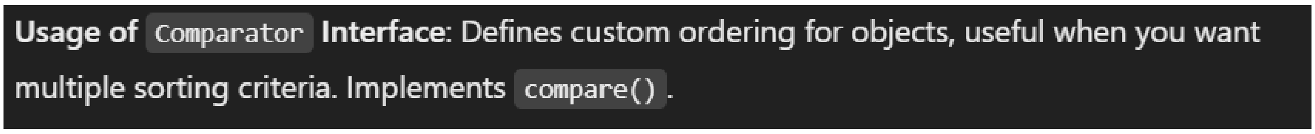
VIVA-VOCEQuestions(In-Lab):

1) Listtheusageofcomparable Interface.



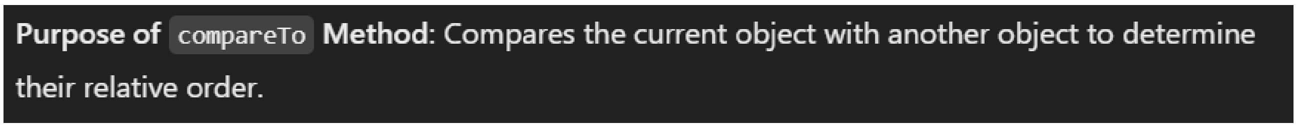
Usage of `Comparable` Interface: Defines the natural ordering of objects (e.g., for sorting). Implements `compareTo()` to compare objects.

2) List the usageofcomparator interface.



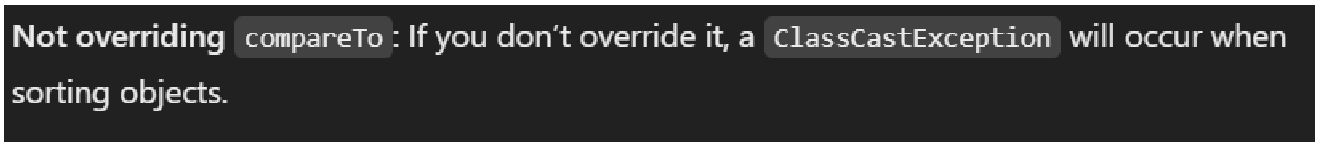
Usage of `Comparator` Interface: Defines custom ordering for objects, useful when you want multiple sorting criteria. Implements `compare()`.

3) WhatisthepurposeofthecompareTo method?



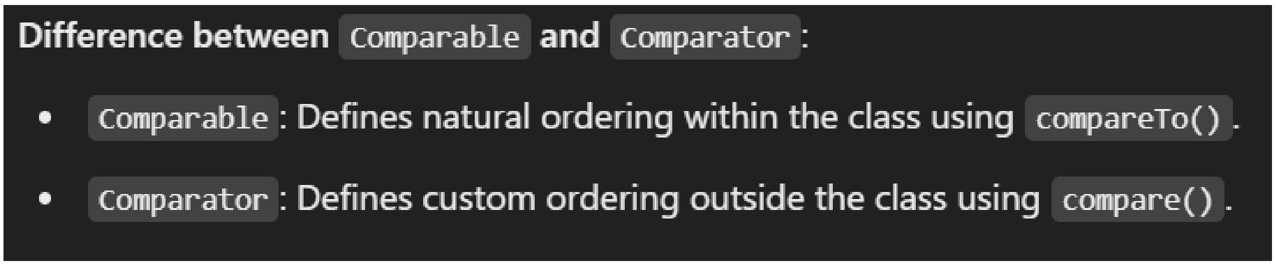
Purpose of `compareTo` Method: Compares the current object with another object to determine their relative order.

4) WhathappensifyoudonotoverridethecompareTomethodwhenimplementing Comparable?



Not overriding `compareTo`: If you don't override it, a `ClassCastException` will occur when sorting objects.

5) Whatisthedifference betweenComparableand Comparator?



Difference between `Comparable` and `Comparator`:

- `Comparable`: Defines natural ordering within the class using `compareTo()`.
- `Comparator`: Defines custom ordering outside the class using `compare()`.

CourseTitle	**AdvancedObject-OrientedProgramming**	**ACADEMICYEAR:2024-25**
CourseCode	**23CS2103R**	Page | **10**

Post-Lab:

1) DevelopaJavaprogramtocomparemoviesbytheirratingsusingacustomComparator 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.

   SampleInput:

   8.4ReturnoftheJedi1983

   8.8EmpireStrikesBack1980

   8.3ForceAwakens2015

   8.7StarWars1977

   SampleOutput:

   Sortedbyrating

   8.3ForceAwakens2015

   8.4ReturnoftheJedi1983

   8.7StarWars1977

   8.8EmpireStrikesBack1980

   Sortedbyname

   EmpireStrikesBack8.81980

   ForceAwakens8.32015

   ReturnoftheJedi8.41983

   StarWars8.71977

   Sorted by year

   19778.7StarWars

   19808.8EmpireStrikes Back

   19838.4ReturnoftheJedi

   2015ForceAwakens

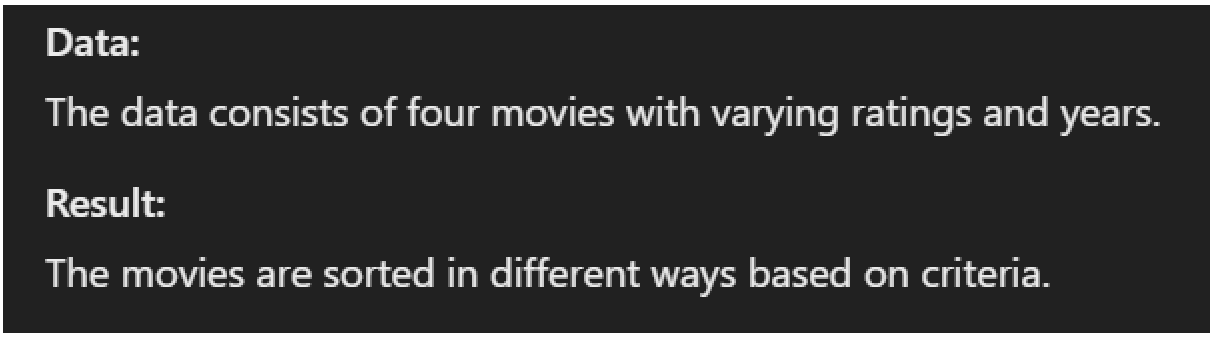
Program:

```
import java.util.*;

public class Main {

    public static void main(String[] args) {

        List<Movie> movies = Arrays.asList(

            new Movie("Return of the Jedi", 8.4, 1983),

            new Movie("Empire Strikes Back", 8.8, 1980),

            new Movie("Force Awakens", 8.3, 2015),

            new Movie("Star Wars", 8.7, 1977)

        );

        movies.sort(Comparator.comparingDouble(m -> m.rating));

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

        movies.forEach(System.out::println);



        movies.sort(Comparator.comparing(m -> m.title));

        System.out.println("Sorted by name:");

        movies.forEach(System.out::println);
movies.sort(Comparator.comparingInt(m -> m.year));
        System.out.println("Sorted by year:");
        movies.forEach(m -> System.out.println(m.year + " " + m.rating + " " +
m.title));
    }
}

class Movie {
    String title;
    double rating;
    int year;

    Movie(String title, double rating, int year) {
        this.title = title;
        this.rating = rating;
        this.year = year;
    }
```

```
@Override

   public String toString() {

      return rating + " " + title + " " + year;

   }

}
```

Experiment#		**Student ID**	
Date		**StudentName**	[@KLWKS_bot] THANOS

✓ **Dataand Results:**



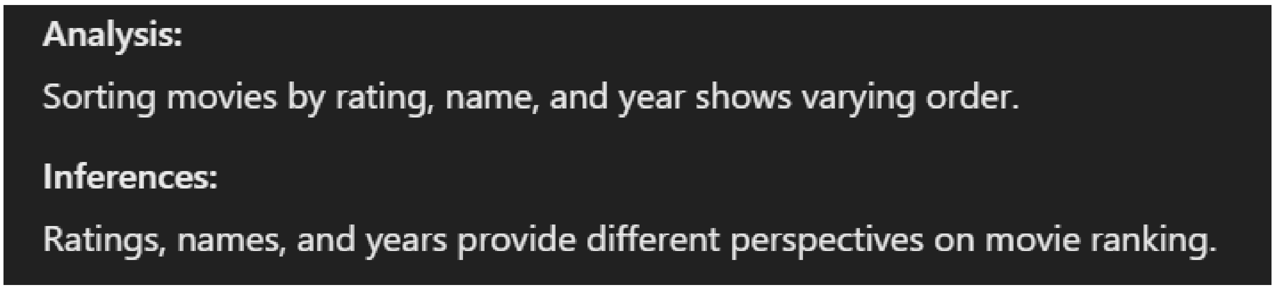
Data:

The data consists of four movies with varying ratings and years.

Result:

The movies are sorted in different ways based on criteria.

✓ **AnalysisandInferences:**



Analysis:

Sorting movies by rating, name, and year shows varying order.

Inferences:

Ratings, names, and years provide different perspectives on movie ranking.

EvaluatorRemark(ifAny):	**MarksSecured _____ outof50**
	SignatureoftheEvaluatorwithDate

EvaluatorMUSTaskViva-vocepriortosigningandposting marksforeach experiment.