1.Coding Question  
Employee.java

**package** Ex\_4;

**import** java.util.\*;

**class** Employee {

String name;

**int** age;

**double** salary;

**public** Employee(String name, **int** age, **double** salary) {

**this**.name = name;

**this**.age = age;

**this**.salary = salary;

}

@Override

**public** String toString() {

**return** "Employee{name='" + name + "', age=" + age + ", salary=" + salary + "}";

}

}

EmployeeSortDemo:

**package** Ex\_4;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.List;

**public** **class** EmployeeSortDemo {

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

List<Employee> employees = **new** ArrayList<>();

employees.add(**new** Employee("Alice", 30, 60000));

employees.add(**new** Employee("Bob", 25, 75000));

employees.add(**new** Employee("Charlie", 28, 50000));

employees.add(**new** Employee("David", 35, 80000));

System.***out***.println("Original List:");

employees.forEach(System.***out***::println);

// Step 4: Sort by salary (descending)

Collections.*sort*(employees, **new** Comparator<Employee>() {

@Override

**public** **int** compare(Employee e1, Employee e2) {

**return** Double.*compare*(e2.salary, e1.salary); // Descending

}

});

System.***out***.println("\nSorted by Salary (Descending):");

employees.forEach(System.***out***::println);

Collections.*sort*(employees, **new** Comparator<Employee>() {

@Override

**public** **int** compare(Employee e1, Employee e2) {

**return** e1.name.compareTo(e2.name); // Ascending

}

});

System.***out***.println("\nSorted by Name (Ascending):");

employees.forEach(System.***out***::println);



1. **What is a Comparator in Java?**
   * A **Comparator** is an interface in Java used to define custom sorting logic. It provides the compare(T o1, T o2) method to compare two objects.
2. **Difference between Comparable and Comparator?**
   * **Comparable**: Defines the **natural order** of objects (inside the class using compareTo).
   * **Comparator**: Defines **custom order** (outside the class using compare).
   * Comparable → single sorting sequence.
   * Comparator → multiple sorting sequences possible.
3. **How does Collections.sort() work?**
   * It internally uses a **modified Merge Sort (TimSort)** algorithm.
   * If objects implement Comparable, it uses compareTo.
   * If a Comparator is provided, it uses compare.
4. **How do you sort objects in descending order?**
   * Either reverse the compareTo() logic, or use:
   * Collections.sort(list, Comparator.reverseOrder());
5. **What is the difference between ArrayList and LinkedList?**
   * **ArrayList**: Better for **random access** (index-based retrieval), slower insert/delete in middle.
   * **LinkedList**: Better for **frequent insertions/deletions**, slower for random access.
6. **How does compare() method work in Comparator?**
   * It compares two objects and returns:
     + **negative** → if first < second
     + **zero** → if equal
     + **positive** → if first > second
7. **Can we sort a list without modifying it?**
   * Yes, by creating a **new sorted list**:
   * List<Employee> sortedList = new ArrayList<>(employees);
   * sortedList.sort(comparator);
8. **What is lambda expression in Java sorting?**
   * A shorter way to define comparators:
   * employees.sort((e1, e2) -> Double.compare(e2.salary, e1.salary));
9. **How do you sort by multiple fields?**
   * Use thenComparing:
   * employees.sort(Comparator.comparing(Employee::getName)
   * .thenComparing(Employee::getAge));
10. **Why is List preferred for dynamic data?**
    * Because unlike arrays, Lists can **grow or shrink dynamically**, provide built-in methods (add, remove, sort), and are more flexible.