1.Employee.java

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 + "}";

}

}

2.EmployeeSortDemo

public class EmployeeSortDemo {

public static void main(String[] args) {

// Store employees in ArrayList

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);

Collections.sort(employees, (e1, e2) -> Double.compare(e2.salary, e1.salary));

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

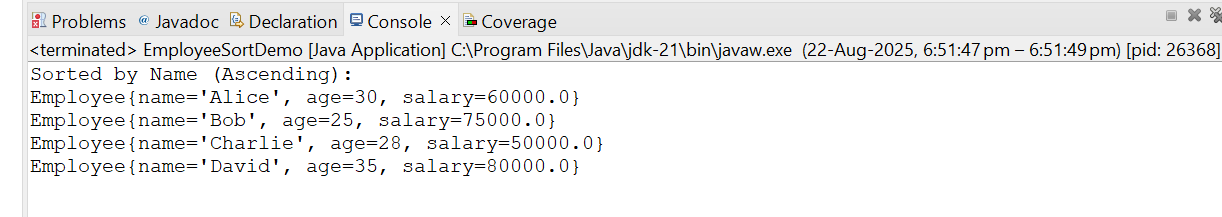
employees.forEach(System.out::println);

Collections.sort(employees, Comparator.comparing(e -> e.name));

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** for objects. It has the method:

int compare(T o1, T o2);

* Returns negative → if o1 < o2
* Returns zero → if o1 == o2
* Returns positive → if o1 > o2

**2. Difference between Comparable and Comparator?**

* **Comparable**:
  + Defined inside the class (implements Comparable<T>)
  + Has compareTo(T o) method
  + Used for **natural ordering**
  + Example: Sorting integers or strings
* **Comparator**:
  + Defined outside the class (implements Comparator<T>)
  + Has compare(T o1, T o2) method
  + Used for **custom ordering**
  + Allows **multiple sort sequences**

**3. How does Collections.sort() work?**

* It uses **TimSort** (a hybrid of merge sort and insertion sort).
* If objects implement Comparable, it calls compareTo().
* If a Comparator is given, it uses the compare() method.

**4. How do you sort objects in descending order?**

Two ways:

1. Reverse logic inside compare()
2. Collections.sort(list, (a, b) -> b.age - a.age);
3. Use built-in:
4. Collections.sort(list, Comparator.reverseOrder());

**5. What is the difference between ArrayList and LinkedList?**

* **ArrayList**:
  + Stores elements in a dynamic array
  + Fast random access (O(1))
  + Slower for insert/delete in the middle
* **LinkedList**:
  + Doubly linked nodes
  + Slower random access (O(n))
  + Faster insertion/deletion (O(1)) in middle

**6. How does compare() method work in Comparator?**

It compares two objects:

* Returns **negative** if first < second
* Returns **zero** if equal
* Returns **positive** if first > second

Example:

public int compare(Employee e1, Employee e2) {

return Double.compare(e1.salary, e2.salary);

}

**7. Can we sort a list without modifying it?**

Yes ✅.

* Make a **copy of the list** and sort the copy.

List<Employee> sortedList = new ArrayList<>(originalList);

sortedList.sort(Comparator.comparing(e -> e.salary));

**8. What is lambda expression in Java sorting?**

A **lambda expression** is a short way of writing anonymous classes.  
Example:

employees.sort((e1, e2) -> Double.compare(e2.salary, e1.salary));

This replaces writing a full Comparator class.

**9. How do you sort by multiple fields?**

Use thenComparing(). Example:

employees.sort(

Comparator.comparing(Employee::getName)

.thenComparing(Employee::getAge)

);

This sorts first by name, and if names are equal, then by age.

**10. Why is List preferred for dynamic data?**

* Unlike arrays, List can **grow or shrink dynamically**.
* Provides built-in methods like add(), remove(), sort().
* Can store duplicate elements and maintains insertion order.

Top of Form

Bottom of Form