Comparable and Comparator in Java Example

Comparable and Comparator in Java are very useful for sorting collection of objects. Java provides some inbuilt methods to sort primitive types array or Wrapper classes array or list. Here we will first learn how we can sort an array/list of primitive types and wrapper classes and then we will use **java.lang.Comparable**and **java.util.Comparator** interfaces to sort array/list of custom classes.

Let's see how we can sort primitive types or Object array and list with a simple program.

```
package com.journaldev.sort;


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


public class JavaObjectSorting {


    /**
     * This class shows how to sort primitive arrays,
     * Wrapper classes Object Arrays
     * @param args
     */
    public static void main(String[] args) {
        //sort primitives array like int array
        int[] intArr = {5,9,1,10};
        Arrays.sort(intArr);
```

```
        System.out.println(Arrays.toString(intArr));


        //sorting String array
        String[] strArr = {"A", "C", "B", "Z", "E"};
        Arrays.sort(strArr);
        System.out.println(Arrays.toString(strArr));


        //sorting list of objects of Wrapper classes
        List<String> strList = new ArrayList<String>();
        strList.add("A");
        strList.add("C");
        strList.add("B");
        strList.add("Z");
        strList.add("E");
        Collections.sort(strList);
        for(String str: strList) System.out.print(" "+str);
    }
}
```

Output of the above program is:

```
[1, 5, 9, 10]

[A, B, C, E, Z]

 A B C E Z
```

Now let's try to sort an array of objects.

```
package com.journaldev.sort;


public class Employee {
```

```
    private int id;
    private String name;
    private int age;
    private long salary;

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public long getSalary() {
        return salary;
    }
```

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

    @Override
    //this is overriden to print the user friendly information about the Employee
    public String toString() {
        return "[id=" + this.id + ", name=" + this.name + ", age=" + this.age + ", salary=" +
                this.salary + "]";
    }

}
```

Here is the code I used to sort the array of Employee objects.

```
//sorting object array
Employee[] empArr = new Employee[4];
```

```
empArr[0] = new Employee(10, "Mikey", 25, 10000);

empArr[1] = new Employee(20, "Arun", 29, 20000);

empArr[2] = new Employee(5, "Lisa", 35, 5000);

empArr[3] = new Employee(1, "Pankaj", 32, 50000);


//sorting employees array using Comparable interface
implementation

Arrays.sort(empArr);

System.out.println("Default Sorting of Employees
list:\n"+Arrays.toString(empArr));
```

When I tried to run this, it throws following runtime exception.

```
Exception in thread "main" java.lang.ClassCastException:
com.journaldev.sort.Employee cannot be cast to
java.lang.Comparable

      at
java.util.ComparableTimSort.countRunAndMakeAscending(Comparable
TimSort.java:290)

      at
java.util.ComparableTimSort.sort(ComparableTimSort.java:157)

      at
java.util.ComparableTimSort.sort(ComparableTimSort.java:146)

      at java.util.Arrays.sort(Arrays.java:472)
```

```
	at
com.journaldev.sort.JavaSorting.main(JavaSorting.java:41)
```

Comparable and Comparator

Java provides **Comparable** interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. Comparable interface has **compareTo(T obj)** method which is used by sorting methods, you can check any Wrapper, String or Date class to confirm this. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if “this” object is less than, equal to, or greater than the object passed as argument.

After implementing Comparable interface in Employee class, here is the resulting Employee class.

```
package com.journaldev.sort;


import java.util.Comparator;


public class Employee implements Comparable<Employee> {


    private int id;

    private String name;

    private int age;

    private long salary;
```

```
    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public long getSalary() {
        return salary;
    }

    public Employee(int id, String name, int age, int salary) {
        this.id = id;
        this.name = name;
        this.age = age;
```

```
        this.salary = salary;
    }

    @Override
    public int compareTo(Employee emp) {
        //let's sort the employee based on id in ascending
order
        //returns a negative integer, zero, or a positive
integer as this employee id
        //is less than, equal to, or greater than the specified
object.
        return (this.id - emp.id);
    }

    @Override
    //this is required to print the user friendly information
about the Employee
    public String toString() {
        return "[id=" + this.id + ", name=" + this.name + ",
age=" + this.age + ", salary=" +
                this.salary + "]";
    }
```

```
}
```

Now when we execute the above snippet for Arrays sorting of Employees and print it, here is the output.

```
Default Sorting of Employees list:

[[id=1, name=Pankaj, age=32, salary=50000], [id=5, name=Lisa,
age=35, salary=5000], [id=10, name=Mikey, age=25,
salary=10000], [id=20, name=Arun, age=29, salary=20000]]
```

As you can see that Employees array is sorted by id in ascending order.

But, in most real life scenarios, we want sorting based on different parameters. For example, as a CEO, I would like to sort the employees based on Salary, an HR would like to sort them based on the age. This is the situation where we need to use **Java Comparator** interface because *Comparable.compareTo(Object o)*method implementation can sort based on one field only and we can't chose the field on which we want to sort the Object.

Java Comparator

Comparator interface *compare(Object o1, Object o2)* method need to be implemented that takes two Object argument, it should be implemented in such a way that it returns negative int if first argument is less than the second one and returns zero if they are equal and positive int if first argument is greater than second one.

Comparable and Comparator interfaces uses Generics for compile time type checking, learn more aboutJava Generics.

Here is how we can create different Comparator implementation in the Employee class.

```
/**
     * Comparator to sort employees list or array in order of
Salary
     */
    public static Comparator<Employee> SalaryComparator = new
Comparator<Employee>() {

        @Override
        public int compare(Employee e1, Employee e2) {
            return (int) (e1.getSalary() - e2.getSalary());
        }
    };

    /**
     * Comparator to sort employees list or array in order of
Age
     */
    public static Comparator<Employee> AgeComparator = new
Comparator<Employee>() {

        @Override
        public int compare(Employee e1, Employee e2) {
```

```
            return e1.getAge() - e2.getAge();
        }
    };

    /**
     * Comparator to sort employees list or array in order of
Name
     */
    public static Comparator<Employee> NameComparator = new
Comparator<Employee>() {

        @Override
        public int compare(Employee e1, Employee e2) {
            return e1.getName().compareTo(e2.getName());
        }
    };
```

All the above implementations of Comparator interface are **anonymous classes**.

We can use these comparator to pass as argument to sort function of Arrays and Collections classes.

```
//sort employees array using Comparator by Salary
Arrays.sort(empArr, Employee.SalaryComparator);
```

```
System.out.println("Employees list sorted by
Salary:\n"+Arrays.toString(empArr));


//sort employees array using Comparator by Age

Arrays.sort(empArr, Employee.AgeComparator);

System.out.println("Employees list sorted by
Age:\n"+Arrays.toString(empArr));


//sort employees array using Comparator by Name

Arrays.sort(empArr, Employee.NameComparator);

System.out.println("Employees list sorted by
Name:\n"+Arrays.toString(empArr));
```

Here is the output of the above code snippet:

```
Employees list sorted by Salary:

[[id=5, name=Lisa, age=35, salary=5000], [id=10, name=Mikey,
age=25, salary=10000], [id=20, name=Arun, age=29,
salary=20000], [id=1, name=Pankaj, age=32, salary=50000]]

Employees list sorted by Age:

[[id=10, name=Mikey, age=25, salary=10000], [id=20, name=Arun,
age=29, salary=20000], [id=1, name=Pankaj, age=32,
salary=50000], [id=5, name=Lisa, age=35, salary=5000]]

Employees list sorted by Name:
```

```
[[id=20, name=Arun, age=29, salary=20000], [id=5, name=Lisa,
age=35, salary=5000], [id=10, name=Mikey, age=25,
salary=10000], [id=1, name=Pankaj, age=32, salary=50000]]
```

So now we know that if we want to sort java object array or list, we need to implement java Comparable interface to provide default sorting and we should implement java Comparator interface to provide different ways of sorting.

We can also create separate class that implements `Comparator` interface and then use it.

Here is the final classes we have explaining **Comparable and Comparator** in Java.

```
package com.journaldev.sort;


import java.util.Comparator;


public class Employee implements Comparable<Employee> {


    private int id;

    private String name;

    private int age;

    private long salary;


    public int getId() {
```

```
        return id;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public long getSalary() {
        return salary;
    }

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

```
    @Override
    public int compareTo(Employee emp) {
        //let's sort the employee based on id in ascending
order
        //returns a negative integer, zero, or a positive
integer as this employee id
        //is less than, equal to, or greater than the specified
object.
        return (this.id - emp.id);
    }

    @Override
    //this is required to print the user friendly information
about the Employee
    public String toString() {
        return "[id=" + this.id + ", name=" + this.name + ",
age=" + this.age + ", salary=" +
                this.salary + "]";
    }

    /**
```

```
     * Comparator to sort employees list or array in order of
Salary
     */
    public static Comparator<Employee> SalaryComparator = new
Comparator<Employee>() {


        @Override
        public int compare(Employee e1, Employee e2) {
            return (int) (e1.getSalary() - e2.getSalary());
        }
    };


    /**
     * Comparator to sort employees list or array in order of
Age
     */
    public static Comparator<Employee> AgeComparator = new
Comparator<Employee>() {


        @Override
        public int compare(Employee e1, Employee e2) {
            return e1.getAge() - e2.getAge();
```

```
        }
    };

    /**
     * Comparator to sort employees list or array in order of
Name
     */
    public static Comparator<Employee> NameComparator = new
Comparator<Employee>() {

        @Override
        public int compare(Employee e1, Employee e2) {
            return e1.getName().compareTo(e2.getName());
        }
    };
}
```

Here is the separate class implementation of Comparator interface that will compare two Employees object first on their id and if they are same then on name.

```
package com.journaldev.sort;

import java.util.Comparator;
```

```
public class EmployeeComparatorByIdAndName implements
Comparator<Employee> {


    @Override

    public int compare(Employee o1, Employee o2) {

        int flag = o1.getId() - o2.getId();

        if(flag==0) flag =
o1.getName().compareTo(o2.getName());

        return flag;

    }


}
```

Here is the test class where we are using different ways to sort Objects in java using Comparable and Comparator.

```
package com.journaldev.sort;


import java.util.Arrays;


public class JavaObjectSorting {
```

```
    /**
     * This class shows how to sort custom objects array/list
     * implementing Comparable and Comparator interfaces
     * @param args
     */
    public static void main(String[] args) {

        //sorting custom object array
        Employee[] empArr = new Employee[4];
        empArr[0] = new Employee(10, "Mikey", 25, 10000);
        empArr[1] = new Employee(20, "Arun", 29, 20000);
        empArr[2] = new Employee(5, "Lisa", 35, 5000);
        empArr[3] = new Employee(1, "Pankaj", 32, 50000);

        //sorting employees array using Comparable interface
implementation
        Arrays.sort(empArr);
        System.out.println("Default Sorting of Employees
list:\n"+Arrays.toString(empArr));

        //sort employees array using Comparator by Salary
```

```
        Arrays.sort(empArr, Employee.SalaryComparator);

        System.out.println("Employees list sorted by
Salary:\n"+Arrays.toString(empArr));


        //sort employees array using Comparator by Age

        Arrays.sort(empArr, Employee.AgeComparator);

        System.out.println("Employees list sorted by
Age:\n"+Arrays.toString(empArr));


        //sort employees array using Comparator by Name

        Arrays.sort(empArr, Employee.NameComparator);

        System.out.println("Employees list sorted by
Name:\n"+Arrays.toString(empArr));


        //Employees list sorted by ID and then name using
Comparator class

        empArr[0] = new Employee(1, "Mikey", 25, 10000);

        Arrays.sort(empArr, new
EmployeeComparatorByIdAndName());

        System.out.println("Employees list sorted by ID and
Name:\n"+Arrays.toString(empArr));

    }
```

```
}
```

Here is the output of the above program:

```
Default Sorting of Employees list:

[[id=1, name=Pankaj, age=32, salary=50000], [id=5, name=Lisa,
age=35, salary=5000], [id=10, name=Mikey, age=25,
salary=10000], [id=20, name=Arun, age=29, salary=20000]]

Employees list sorted by Salary:

[[id=5, name=Lisa, age=35, salary=5000], [id=10, name=Mikey,
age=25, salary=10000], [id=20, name=Arun, age=29,
salary=20000], [id=1, name=Pankaj, age=32, salary=50000]]

Employees list sorted by Age:

[[id=10, name=Mikey, age=25, salary=10000], [id=20, name=Arun,
age=29, salary=20000], [id=1, name=Pankaj, age=32,
salary=50000], [id=5, name=Lisa, age=35, salary=5000]]

Employees list sorted by Name:

[[id=20, name=Arun, age=29, salary=20000], [id=5, name=Lisa,
age=35, salary=5000], [id=10, name=Mikey, age=25,
salary=10000], [id=1, name=Pankaj, age=32, salary=50000]]

Employees list sorted by ID and Name:

[[id=1, name=Mikey, age=25, salary=10000], [id=1, name=Pankaj,
age=32, salary=50000], [id=5, name=Lisa, age=35, salary=5000],
[id=10, name=Mikey, age=25, salary=10000]]
```

The **java.lang.Comparable** and **java.util.Comparator** are powerful interfaces that can be used to provide sorting objects in java.

Comparable vs Comparator

1. Comparable interface can be used to provide single way of sorting whereas Comparator interface is used to provide different ways of sorting.
2. For using Comparable, Class needs to implement it whereas for using Comparator we don’t need to make any change in the class.
3. Comparable interface is in `java.lang` package whereas Comparator interface is present in `java.util`package.
4. We don’t need to make any code changes at client side for using Comparable, `Arrays.sort()` or`Collection.sort()` methods automatically uses the `compareTo()` method of the class. For Comparator, client needs to provide the Comparator class to use in compare() method.