Comparable and Comparator in Java Example

APRIL 2, 2018 BY PANKAJ — 62 COMMENTS

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));
```

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;
    }
```

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);
```

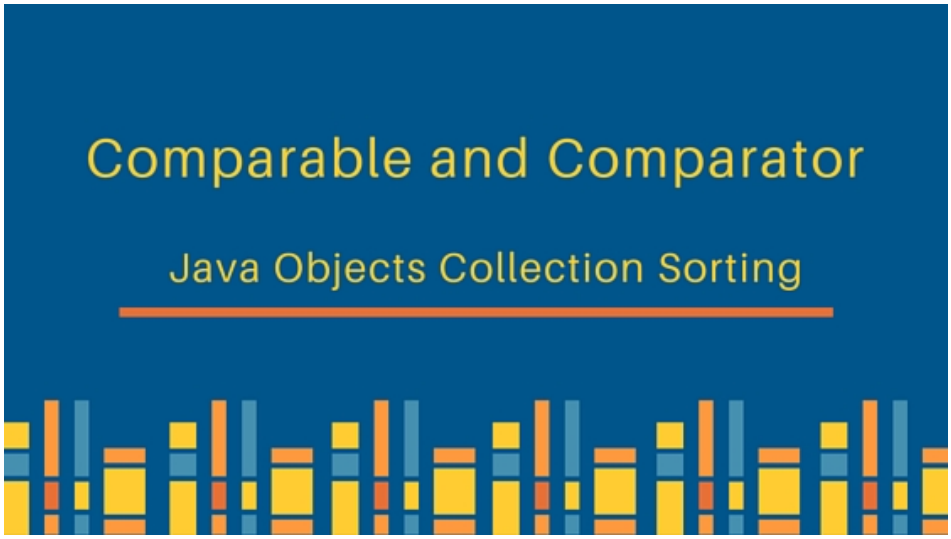
```
empArr[2] = new Employee(5, "Lisa", 35, 5000);
empArr[3] = new Employee(1, "Pankaj", 32, 50000);

//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(ComparableTimSort.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;
    }
```

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 about Java 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
     */
```

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;
```

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));
```

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.

Do you know that Collections.sort() method that takes Comparator argument follows Strategy Pattern?

FILED UNDER: JAVA

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

« Java String to String Array Example

How to sort an Array in Java »

Comments

Shradha says

MARCH 20, 2018 AT 1:59 AM

Nice one. Well explained

Reply

Pragati says

DECEMBER 4, 2017 AT 3:17 AM

Excellent explanation. Thank you.

Reply

Subham says
NOVEMBER 3, 2017 AT 5:01 PM
Nice explanation , keep posting java articles like this.
Reply

Raj says
AUGUST 30, 2017 AT 3:37 AM
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""""" .
We can't chose the field? We can chose one field, right?
Reply

Monika says
SEPTEMBER 26, 2017 AT 2:48 PM
Yes we can. This is written wrong here. We can definitely choose one field in compareTo() method.
Reply

Pankaj says
SEPTEMBER 26, 2017 AT 10:53 PM
You are not getting the point, what I meant is that once you have provided the implementation of Comparable.compareTo() method, you can't change it dynamically. For example, let's say you have Employee class and you implemented compareTo() method to sort based on ID, now you can't change it dynamically to Salary or Age. This is where Comparator is helpful, you can define multiple Comparators and use them based on your requirement. I hope you got the point now.
Reply

krish says
DECEMBER 24, 2017 AT 2:02 AM
Perhaps, You need to correct the wording. Even, I thought that we can do with only one field/property. Something like "With Comparable interface we can do only default sorting" would be helpful.
Reply

Swarna says
AUGUST 9, 2017 AT 3:36 AM
The second Point is not true. “”For using Comparable, Class needs to implement it whereas for using Compactor we don’t need to make any change in the class.”” because we can even use the comparable with out making changes(how we used compactor).
Reply

Monika says
SEPTEMBER 26, 2017 AT 2:55 PM
The second point is true. We cann’t do below ::
Suppose I have a class Employee as shown below –
public class Employee {
private int id;
private String name;
private Double salary;
private int age;
public Employee(int id, String name, Double salary, int age) {
super();
this.id = id;
this.name = name;
this.salary = salary;
this.age = age;
}
public Employee() {
// TODO Auto-generated constructor stub
}
}
Now I can’t have another class which implements Comparable interface for Employee.
lets suppose we define another class –
class AnotherClass implements Comparable{
@Override
public int compareTo(Employee o) {
// TODO Auto-generated method stub
//I can’t use this here. As my compareTo() method demands.
return 0;
}
}

Reply

Shweta says

JULY 16, 2017 AT 9:58 AM

Beautiful and detailed explanation. I have been trying to know why and when to use what, I got my answer here. You made it quite simple. I have been referring your latest Interview questions pdf which redirected me here.

Thanks a ton!

Reply

vishwas says

JUNE 13, 2017 AT 12:53 AM

i am not able to understand the line (this.id – emp.id) or this line o1.getId() – o2.getId(); means how here it is comparing between the values please anybody can explain me ???

Reply

kathiravan says

JUNE 30, 2017 AT 4:45 AM

Inside the compiler , it requires a value from your side which is required to determine the natural sorting.It is an understanding internally specific to the java Technology.

Reply

Shashank Sharma says

MAY 17, 2017 AT 3:48 AM

i have heard this topic a lot of times

this is my first time

didn't took me second read

i think the explanation couldn't have been simpler

Great job there

Thanks

Reply

Ajay says
APRIL 22, 2017 AT 2:12 PM
I've been trying to understand this concept from long time and I'm glad to say "Your Explanation" made so clear.
splendid .
Thank you!
Reply

Ashutosh Kumar says
APRIL 20, 2017 AT 5:06 AM
Hi Pankaj ,
In 4th point of Comparable vs Comparator I think there is some type error "Collection.sort()" .But Collection is an interface and there is no sort() method in these interface,Instead of Collection there should be Collections method.
Because Collections is an utility class and in this class there is a method
"Collections.sort()" —- This method sorts the specified collection.
Thanks
Reply

Kumar Mayank Singh says
JANUARY 28, 2017 AT 12:24 AM
Hi Pankaj,
Excellent explanation. Hats off to you!!
Just found one mistake. In the last section, Comparable vs Comparator you've written:
"For Comparator, client needs to provide the Comparator class to use in compare() method."
I think it should be:
"For Comparator, client needs to provide the Comparator class to use in sort() method."
Correct me if I'm wrong here and please continue the good work.
Reply

Pankaj says
JANUARY 30, 2017 AT 4:35 AM
What I have written is correct. Client class needs to provide which comparator to use. sort() method example uses Comparable.
Reply

shiva says
NOVEMBER 12, 2016 AT 12:14 AM
wonderful article
simple and practical
thank you
Reply

Rupesh Jha says
SEPTEMBER 14, 2016 AT 5:01 AM
Simply great article.Easy to understand.
Reply

Sajal Goel says
JULY 29, 2016 AT 1:47 AM
Thanks Pankaj !
You have simplified many complex ideas regarding collections.
Reply

Pankaj says
JULY 31, 2016 AT 7:06 AM
Thanks Sajal.
Reply

Amit says
JULY 1, 2016 AT 3:28 AM
You made a typing mistake I think, while summarizing at the end. For point 1 and point 2,
1. Comparable interface can be used to provide single way of sorting whereas Comparable interface is used to provide different ways of sorting.
2. For using Comparable, Class needs to implement it whereas for using Comparable we don't need to make any change in the class.
after the word whereas in both points, I think you mean to say “Comparator” instead of “Comparable”.
Reply

Pankaj says
JULY 1, 2016 AT 6:27 AM
Thanks for catching it, i have corrected it.
Reply

Shuang says
MARCH 26, 2016 AT 11:22 AM
I've read Java article from different places, I found yours are the best at explaining complex ideas, you always made them easy to understand with really good example, your articles are short but express all the important ideas, thank you for post such high quality articles!
Reply

deshani says
MARCH 5, 2016 AT 10:58 PM
highly understandable work, thanks mr. Pankaj
I have a problem with print those data in a table, have any easy way to do it?
Reply

Veera says
DECEMBER 13, 2015 AT 6:50 AM
Some human come to earth to make things simple and easy □
Reply

Veera says
DECEMBER 13, 2015 AT 6:52 AM
Hey TimeStamp for Los Angeles … Why not IST
Reply

Pankaj says
DECEMBER 13, 2015 AT 9:27 AM
It's coming from server that is sitting in West Coast time zone.
Reply

Pankaj says
DECEMBER 13, 2015 AT 9:28 AM
Thanks Veera, you made my day. Subscribe to my newsletter if you have not done already, you will find more reasons to praise me. □
Reply

thanks a lot says
AUGUST 20, 2015 AT 4:29 PM
this is really helpful,thanks a lot □
Reply

Shikha Virmani says
AUGUST 2, 2015 AT 10:25 AM
Awesome article… Very simple and so informative. I love this article.
Thank you.
Reply

Chandra says
JULY 27, 2015 AT 11:45 PM
Nice explanation Sir. Performance wise which is better .. Comparable or Comparator ?
Reply

Arun@Ar says
JUNE 21, 2015 AT 11:59 AM
Very useful explanation, I was surfing through lot of pages for proper answer.
Finally I got it, Thanks for the article.
Reply

Rajesh says
MAY 27, 2015 AT 1:53 AM

Good explanation on “Comparable & Comparator”.
Reply

Anoop says
DECEMBER 18, 2014 AT 12:05 AM
With Java 8, sorting is pretty simple.
Let us sort employees according to their age in an employee list:
Collections.sort(employeeList, Comparator.comparing(Employee::getAge);
This single line does everything…
Reply

Raghav says
OCTOBER 29, 2014 AT 6:45 AM
Nice Explanation. But i have a doubt in this.
In ur blog it says tat “In a Comparable implementation can sort based on one field only and we can’t chose the field on which we want to sort the Object.”
But in this case,
@Override
public int compareTo(Employee emp) {
int flag = this.getId() – emp.getId();
if(flag==0) flag = this.getName().compareTo(emp.getName());
return flag; }
am using Comparable am overriding compareTo() to sort Employee based on both name and id. Can u explain me your context further.
Reply

Pankaj says
OCTOBER 29, 2014 AT 7:46 AM
Here we means the client application. In your code above” if client wants to sort based on employee name, its not possible. This is where Comparator is useful.
Reply

Sridhar says
OCTOBER 4, 2014 AT 10:42 PM
Excellent Explanation
Reply

madan says
SEPTEMBER 10, 2014 AT 3:17 AM
Write a program to sort the java objects in the ascending order using SET and only sort by name.
public class Test{
private String id;
private String name;
//Set and Get methods
}
Reply

Atmprakash Sharma says
AUGUST 27, 2014 AT 6:31 AM
realty nice explanation as well as very well explanation with example….great !!!
Reply

Niraj kumar singh says
AUGUST 27, 2014 AT 3:14 AM
I have a query.
How would i sort the database record after fetching the data from database using Servlet and Jsp. I want to make a column header as a link. On click of column header the all data of the table will be sorted in the basis of that column.
I want to do this sorting using coparator interface.
Reply

Pankaj says
AUGUST 27, 2014 AT 8:50 AM
Check for some javascript functions because you are looking for dynamic sorting on the page itself. I am sure you don't need any server side communication for this.
Reply

Niranjan says
JULY 19, 2014 AT 4:54 AM

Excellent! Examples
My doubt is cleared now….Clearly
I tried ….tried..but i got it now…Thank u
Reply

satya says
JULY 8, 2014 AT 4:53 AM
nice explanations
Reply

triggergirl says
JUNE 21, 2014 AT 4:37 AM
Perfect Explanation .Really it makes me to understand easily
Reply

Andy says
JUNE 17, 2014 AT 12:21 AM
Best explanation ever. . Thanks □
Reply

indu says
MAY 9, 2014 AT 12:21 AM
good nice article
Reply

indu says
MAY 9, 2014 AT 12:09 AM
Supurb!!!
I clearly understand custom objects sorting by this article
Reply

Yogesh Pal says
APRIL 30, 2014 AT 10:46 PM
write a comparator class to define customized sorting which is alphabetical order of employee name if two employee having same name consider descending order of there age.
Reply

Manash Ranjan Dakua says
FEBRUARY 23, 2014 AT 7:16 AM
Sir,your concept is pure real to focus in an core and deep point of view.We got lot of depth concept by through it thanks.
Reply

suraj says
MAY 1, 2014 AT 2:47 AM
superb!!!
Reply

prateek says
FEBRUARY 9, 2014 AT 7:59 PM
cleared concept…Amazing work SiR thnxx □
Reply

Raj says
NOVEMBER 13, 2013 AT 2:52 PM
How to sort String objects in arraylist using compartor and comparable?
Reply

Pankaj says
NOVEMBER 15, 2013 AT 8:01 AM
You will have to use the String sorting options.
Reply

kamaljit says
NOVEMBER 8, 2013 AT 10:07 AM
class A9
{
String name,state;
java.util.Date date;
int age;
A9(String name,java.util.Date date,int age)
{
this.name=name;
this.date=date;
this.age=age;
My question is that how to sort the date using comparator and what should be the format of date,
Reply

Pankaj says
NOVEMBER 11, 2013 AT 4:17 PM
You can write a comparator for sorting based on date like my example. For sorting dates, you can get the timestamp that is a long value and then compare it.
Reply

Sandy says
SEPTEMBER 24, 2013 AT 11:17 AM
This must be an easy question for you guys. But can you please provide me the explanation of the following statetment. "return e1.getName().compareTo(e2.getName())".
Below are my specific queries:
1. Do we need to override comapreTo method even if we are implementing Comaparator and want to sort on the bases of name.
2. In the above mentioned example you have provided definition to comapreTo() in which you are sorting on the basis of ID. But then while sorting according to name you are still calling the comapreTo() method. Please explain as it is very confusing.
Thanks
Reply

Pankaj says
SEPTEMBER 24, 2013 AT 7:23 PM
Hi Sandy,

1. `Employee.NameComparator` is an anonymous inner class, the example is given to showcase both Comparable and Comparator interfaces, you can use any of these based on your requirements. Read more about inner classes at https://www.journaldev.com/996/java-nested-classes-java-inner-class-static-nested-class-local-inner-class-and-anonymous-inner-class
2. `e1.getName().compareTo(e2.getName())` is simply using String class compareTo() method, it's something like "abc".compareTo("123"), it's not using Employee class compareTo() method.

I hope it will clear your confusion.

Reply

Muralidhar N says

AUGUST 10, 2013 AT 11:57 AM

Hi, nice explanation with the programs…but you didn't print the strArray in the second Sysout statement at the first program. you can change it from intArr to strArr.

Thanks ..

Reply

Pankaj says

AUGUST 10, 2013 AT 1:37 PM

yeah right, changed the program.

Thanks for spotting it.

Reply

Nagesh kaniganti says

MAY 20, 2013 AT 8:26 AM

private int id;

private String name;

private int age;

private long salar

y;

Kids kids;

like kids i have reference in kids class also i have 2 parameters age,nofkids like i need sorting salary accending and kids age desending

please any one can clarify…….

Reply

Roshan says
FEBRUARY 6, 2013 AT 1:46 PM
very nicely written.
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

CORE JAVA TUTORIAL

Java 10 Tutorials Java 9 Tutorials Java 8 Tutorials Java 7 Tutorials Core Java Basics OOPS Concepts Data Types and Operators String Manipulation Java Arrays Annotation and Enum Java Collections Java IO Operations Java Exception Handling MultiThreading and Concurrency Regular Expressions Advanced Java Concepts

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial

- Apache Axis 2
- JAX-RS
- Memcached Tutorial