**Comparable & Comparator**

***Comparable:***

**Comparable interface** is mainly used to sort the arrays (or lists) of **custom objects**.  
Lists (and arrays) of objects that implement Comparable interface can be sorted automatically by Collections.sort (and Arrays.sort). Before we see how to sort an objects of custom objects, lets see how we can sort elements of arrays and Wrapper classes that already implements Comparable.

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

public class Demo {

public static void main(String[] args) {

/\*

\* Integer class implements Comparable

\* Interface so we can use the sort method

\*/

int[] arr = {11,55,22,0,89};

Arrays.sort(arr);

System.out.print("Sorted Int Array: ");

System.out.println(Arrays.toString(arr));

/\*

\* String class implements Comparable

\* Interface so we can use the sort method

\*/

System.out.print("Sorted String Array: ");

String[] names = {"Steve", "Ajeet", "Kyle"};

Arrays.sort(names);

System.out.println(Arrays.toString(names));

/\*

\* String class implements Comparable

\* Interface so we can use the sort method

\*/

System.out.print("Sorted List: ");

List fruits = new ArrayList();

fruits.add("Orange");

fruits.add("Banana");

fruits.add("Apple");

fruits.add("Guava");

fruits.add("Grapes");

Collections.sort(fruits);

for(String s: fruits) System.out.print(s+", ");

}

}

**Output:**

Sorted Int Array: [0, 11, 22, 55, 89]

Sorted String Array: [Ajeet, Kyle, Steve]

Sorted List: Apple, Banana, Grapes, Guava, Orange,

In the above example, you have seen that how easy it is to sort the Arrays and list of objects that implements Comparable interface, you just need to call the Collections.sort (and Arrays.sort).  
**However if you want to sort the objects of custom class then you need to implement the Comparable interface in our custom class.**

This interface has only one method which is:

public abstract int compareTo(T obj)

Since this method is abstract, you must implement this method in your class if you implement the Comparable interface.

Let’s take an example to understand this better:

**Sorting Custom object by implementing Comparable interface**

As you can see I have implemented the Comparable interface in my Author class because I want to sort the objects of this class. I have written the logic of sorting in the compareTo() method, you can write logic based on the requirement. I wanted to sort the author names by last name first and if the last name is same then by first name. If you want to sort by the last name only then first line inside compareTo() method is enough.

**Author class**

public class Author implements Comparable<Author> {

String firstName;

String lastName;

String bookName;

Author(String first, String last, String book){

this.firstName = first;

this.lastName = last;

this.bookName = book;

}

@Override

/\*

\* This is where we write the logic to sort. This method sort

\* automatically by the first name in case that the last name is

\* the same.

\*/

public int compareTo(Author au){

/\*

\* Sorting by last name. compareTo should return < 0 if this(keyword)

\* is supposed to be less than au, > 0 if this is supposed to be

\* greater than object au and 0 if they are supposed to be equal.

\*/

int last = this.lastName.compareTo(au.lastName);

//Sorting by first name if last name is same d

return last == 0 ? this.firstName.compareTo(au.firstName) : last;

}

}

**Sorting class: SortAuthByNames**

import java.util.ArrayList;

import java.util.Collections;

public class SortAuthByNames{

public static void main(String args[]){

// List of objects of Author class

ArrayList<Author> al=new ArrayList<Author>();

al.add(new Author("Henry","Miller", "Tropic of Cancer"));

al.add(new Author("Nalo","Hopkinson", "Brown Girl in the Ring"));

al.add(new Author("Frank","Miller", "300"));

al.add(new Author("Deborah","Hopkinson", "Sky Boys"));

al.add(new Author("George R. R.","Martin", "Song of Ice and Fire"));

/\*

\* Sorting the list using Collections.sort() method, we

\* can use this method because we have implemented the

\* Comparable interface in our user defined class Author

\*/

Collections.sort(al);

for(Author str:al){

System.out.println(str.firstName+" "+

str.lastName+" "+"Book: "+str.bookName);

}

}

}

**Output:**

Deborah Hopkinson Book: Sky Boys

Nalo Hopkinson Book: Brown Girl in the Ring

George R. R. Martin Book: A Song of Ice and Fire

Frank Miller Book: 300

Henry Miller Book: Tropic of Cancer

Note: We should write the compareTo() method in such a way that if this( I am referring to the this keyword here) is less than the passed object then it should return negative, if greater than positive and zero if equal.

You may be wondering why I didn’t write that logic? Because first name and last name are strings, I have called the [compareTo() method of string class](https://beginnersbook.com/2013/12/java-string-compareto-method-example/), which does exactly the same.

However if the things we are comparing are of other type such as int then you can write the logic like this:  
Let’s say object of Employee class is (empId, empName, empAge) and we want to sort the objects by empAge.

public int compareTo(Employee e){

if(this.empAge==e.empAge)

return 0;

else if(this.empAge>e.empAge)

return 1;

else

return -1;

}

**or**

public int compareTo(Employee e){

return this.empAge > e.empAge ? 1 : this.empAge < e.empAge ? -1 : 0;

}

***Comparator:***

In the [last tutorial](https://beginnersbook.com/2017/08/comparable-interface-in-java-with-example/), we have seen how to sort objects of a custom class using Comparable interface. By using Comparable we can sort the objects based on any data member. For example, lets say we have an Author class has data members: Author name, book name and author age, now if we want to sort the objects based on any of the data member then we can use Comparable but **what if we want to have multiple sort choices and we can sort objects based on any choice**, this can be done using Comparator interface, we can create as many Comparator as we want and then we can call Collections.sort on one or more Comparator like this:

//Sorting arraylist al by Author Age

Collections.sort(al, new AuthorAgeComparator());

//Sorting arraylist al by Book Name

Collections.sort(al, new BookNameComparator());

So how does it work? To call the Collections.sort method like this, we must first need to write these Comparator classes AuthorAgeComparator and BookNameComparator, along with Author class and the main class.

## Complete Comparator Example

**Author.java**

public class Author implements Comparable<Author> {

String firstName;

String bookName;

int auAge;

Author(String first, String book, int age){

this.firstName = first;

this.bookName = book;

this.auAge = age;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getBookName() {

return bookName;

}

public void setBookName(String bookName) {

this.bookName = bookName;

}

public int getAuAge() {

return auAge;

}

public void setAuAge(int auAge) {

this.auAge = auAge;

}

@Override

/\*

\* When we only use Comparable, this is where we write sorting

\* logic. This method is called when we implement the Comparable

\* interface in our class and call Collections.sort()

\*/

public int compareTo(Author au){

return this.firstName.compareTo(au.firstName);

}

}

**AuthorAgeComparator.java**

import java.util.\*;

class AuthorAgeComparator implements Comparator<Author>{

public int compare(Author a1,Author a2){

if(a1.auAge==a2.auAge)

return 0;

else if(a1.auAge>a2.auAge)

return 1;

else

return -1;

}

}

**BookNameComparator.java**

import java.util.\*;

public class BookNameComparator implements Comparator<Author>{

public int compare(Author a1,Author a2){

return a1.bookName.compareTo(a2.bookName);

}

}

**SortingPgm.java**

import java.util.ArrayList;

import java.util.Collections;

public class SortingPgm{

public static void main(String args[]){

// List of objects of Author class

ArrayList<Author> al=new ArrayList<Author>();

al.add(new Author("Henry", "Tropic of Cancer",  45));

al.add(new Author("Nalo", "Brown Girl in the Ring", 56));

al.add(new Author("Frank", "300", 65));

al.add(new Author("Deborah", "Sky Boys", 51));

al.add(new Author("George R. R.", "A Song of Ice and Fire", 62));

/\*

\* Sorting the list using Collections.sort() method, we

\* can use this method because we have implemented the

\* Comparable interface in our user defined class Author

\*/

System.out.println("Sorting by Author First Name:");

Collections.sort(al);

for(Author au: al){

System.out.println(au.getFirstName()+", "+au.getBookName()+", "+

au.getAuAge());

}

/\*Sorting using AuthorAgeComparator\*/

System.out.println("Sorting by Author Age:");

Collections.sort(al, new AuthorAgeComparator());

for(Author au: al){

System.out.println(au.getFirstName()+", "+au.getBookName()+", "+

au.getAuAge());

}

/\*Sorting using BookNameComparator\*/

System.out.println("Sorting by Book Name:");

Collections.sort(al, new BookNameComparator());

for(Author au: al){

System.out.println(au.getFirstName()+", "+au.getBookName()+", "+

au.getAuAge());

}

}

}

**Output:**

Sorting by Author First Name:

Deborah, Sky Boys, 51

Frank, 300, 65

George R. R., A Song of Ice and Fire, 62

Henry, Tropic of Cancer, 45

Nalo, Brown Girl in the Ring, 56

Sorting by Author Age:

Henry, Tropic of Cancer, 45

Deborah, Sky Boys, 51

Nalo, Brown Girl in the Ring, 56

George R. R., A Song of Ice and Fire, 62

Frank, 300, 65

Sorting by Book Name:

Frank, 300, 65

George R. R., A Song of Ice and Fire, 62

Nalo, Brown Girl in the Ring, 56

Deborah, Sky Boys, 51

Henry, Tropic of Cancer, 45