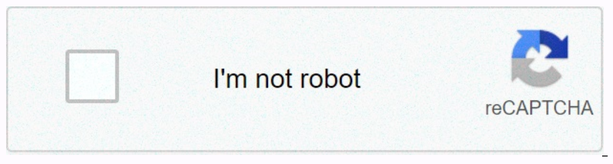

Continue

Sorting custom arraylist in android

ArrayList is one of the most commonly used collection classes in the Java Collection Framework because of the functionality and flexibility it provides. ArrayList is a list implementation that internally implements a dynamic array to store items. Therefore, an ArrayList can dynamically grow and shrink when you add and remove items to and from it. It is likely that you have used ArrayList, therefore I will skip the basics. If you are not familiar with ArrayList yet, you can go through its API documentation here, which is very descriptive and easy to understand to perform basic operations on ArrayList. In this post I will discuss one of the most important operation on ArrayList that you will most likely require implementation during the company's program development. It is to sort the items in an ArrayList. Sort an array list of string objects Consider an array list that stores country names as string objects. To sort ArrayList, simply call the Collections.sort() method that transfers the Matrix list object filled with country names. This method sorts the items (country names) into ArrayList using natural order (alphabetically in ascending order). Let's write some code for it. SortArrayListAscendingDescending.java package guru.springframework.blog.sortarraylist.ascendingdescending; importing java.util.ArrayList; importing java.util.Collections; public class SortArrayListAscendingDescending { private ArrayList<String> arrayList; public SortArrayListAscendingDescending (ArrayList<String> arrayList) { this.arrayList = arrayList; } public ArrayList<String> getArrayList() { return this.arrayList; } public ArrayList<String> sortAscending() {.black(this.arrayList); return this.arrayList; } public ArrayList<String> blackDescending() { Collections.sort(this.arrayList Collections.Collections.reverseOrder()); return.arrayList; } In the class above, we initialized an ArrayList object in the constructor. In the sortAscending()method, we called the Collections.sort() method by transferring the initialized ArrayList object and returned the sorted ArrayList. In the sortDescending() method, we called the overloaded Collections.sort() method to sort the items in descending order. This version of Collections.sort() accepts the ArrayList object as the first parameter and a comparison object that the MethodCollections.reverseOrder() returns as the second parameter. We'll come to Komparator a little later. To test the sorting function, we type some test code. SortArrayListAscendingDescendingTest.java package guru.springframework.blog.sortarraylist.ascendingdescending; import org.junit.Test; importing java.util.ArrayList; import static org.junit.Assert.*; public class SortArrayListAscendingDescendingTest { @Test public void testSortAscendingDescending() casts Exception { ArrayList<String> countryList = new ArrayList (); countryList.add(Frankrig); countryList.add(USA); countryList.add(Indien); countryList.add(Spanien);</String> </String> </String> </String> </String> SortArrayListAscendingDescending sortArrayList = new SortArrayListAscendingDescending(countryList); ArrayList<String> UnsortedArrayList = sortArrayList.getArrayList(); System.out.println(Unsorted array list: + unsortedArrayList); ArrayList<String> sortedArrayListAscending = sortArrayList.sortAscending(); System.out.println(Sorted array list in ascending order: + sortedArrayListAscending); MatrixList<String> sortedArrayListDescending = sortArrayList.sortDescending(); System.out.println(Sorted array list in descending order: + sortedArrayListDescending); } } In the test code above, we created an ArrayList object and added five string objects representing the names of five countries. We then called getArrayList(), sortAscending(), and blackDescending() methods and printed ArrayList objects that the methods return. The output is this. ---------- T E S T S ---------- Running guru.springframework.blog.sortarraylist.ascendingdescending.BlackArrayListAscendingDescendingTest Unsorted ArrayList: [France, USA, India, Spain, England] Sorted ArrayList in Ascending Order: [England, France, India, Spain, USA] Sorted ArrayList in Descending Order: [USA, Spain, India, France, England] Tests run: 1, Error: 0, Error: 0, Skipped: 0, Time has passed: 0.001 sec - in guru.springframework.blog.sortarraylist.ascendingdescending.SortArrayListAscendingDescendingTest At this point, it may seem that sorting elements of an ArrayList is very simple. We only need to call methodCollections.sort() that passes the ArrayList object if items need to be sorted. However, there is more to sort ArrayLists as you encounter additional scenarios. The Collections.sort() method sorts ArrayList elements or items in another list implementation, provided that the items are comparable. What this means programmatically is that the classes of the elements need to implement the comparable interface of java.long package. Because the string class implements its comparative interface, we were able to sort ArrayList by country names. Some other classes standard for Java, which implements the comparable interface include primitive wrapper classes, such as Integer, Map, Double, Float, and Boolean. BigInteger, BigDecimal, File and Date are also examples of classes that implement Comparable. Sorting an array list using Comparable Comparable is an interface with a single compareTo() method. A purpose of a class that implements Comparable is able to compare itself to another purpose of the same type. The class that implements Comparable must override the compareTo(). This method accepts an object of the same type and implements the logic to compare that object to the object transferred to compareTo(). CompareTo() method returns as a heltal, there have the following meanings: A positive weather edible, that this object is greater than the object</String> </String> </String> </String> at sammenligneTo(). En negativ værdi angiver, at dette objekt er mindre end det objekt, der er overført til compareTo(). Værdien nul angiver, at begge objekter er ens. Lad os tage et eksempel på en JobCandidate-klasse, hvis objekter vi vil gemme på en arrayliste og senere sortere dem. Klassen JobCandidate har tre felter: navn og køn af typen String og alder, der er et heltal. Vi vil sortere JobCandidate-objekter, der er gemt på ArrayList, baseret på aldersfeltet. For at gøre dette, bliver vi nødt til at skrive JobCandidate klasse til at gennemføre Sammenlignelige og tilsidesætte compareTo() metode. Koden for klassen JobCandidate er denne. JobCandidate.java guru.springframework.blog.sortarraylist.comparable; public class JobCandidate implementerer Sammenlignelige<JobCandidate> { private String-navn; privat streng gender; privat int alder; offentlig JobCandidate(Strengnavn, Streng køn, int alder) { this.name = navn; this.gender = gender; this.age = age; } public String getName() { return name; } public String getGender() { return gender; } public int getAge() { return age; } @Override public int compareTo(JobCandidate candidate) { return (this.getAge() < candidate.getage()= -1= := (this.getage()== candidate.getage()= 0= := 1));= }= @override= public= string= tostring()= {= return= = name:= = += this.name= += ,= gender:= = += this.gender= += ,= age:= += this.age;= }= }= in= the= overridden= compareto()= method= of= the= jobcandidate= class= above,= we= implemented= the= comparison= logic= based= on= the= age= field.= i= have= seen= many= programmers= restoring= to= the= shortcut= version= of= returning= the= comparison= result= as= return= (this.getage()= -= candidate.getage());. = although= using= this= return= statement= might= appear= tempting= and= will= not= anyhow= affect= our= example,= my= advice= is= to= stay= away= from= it.= imagine,= the= result= of= comparing= integer= values= where= one= or= both= of= them= are= negative= values.= it= can= lead= to= bugs= that= will= make= your= application= behave= erratically= and= more= than= that,= such= bugs= being= subtle,= are= extremely= difficult= to= detect= especially= in= large= enterprise= applications.= next,= we'll= write= a= helper= class= which= will= sort= arraylist= objects= containing= jobcandidate= elements= for= clients.= jobcandidatesorter.java= package= guru.springframework.blog.sortarraylist.comparable;= import= java.util.arraylist;= import= java.util.collections;= public= class= jobcandidatesorter= {=> <JobCandidate>jobCandidate = new ArrayList <> (); public JobCandidateSorter(ArrayList<JobCandidate> jobCandidate) { this.jobCandidate = jobCandidate; } public ArrayList<JobCandidate> getSortedJobdidateByAge() { Collections.sort(jobCandidate); return jobCandidate} } I klassen JobCandidateSorter initialiserede vi et ArrayList-objekt , som klienten vil passere gennem konstruktøren, mens jobcandidateSorter instantieres. Vi skrev derefter Method. In this method we called Collections.sort() passing the initialized ArrayList. Finally, we returned the sorted ArrayList. Next, we will write a test class to test our code. </JobCandidate></JobCandidate></JobCandidate> </JobCandidate></JobCandidate></JobCandidate> package guru.springframework.blog.sortarraylist.comparable; import org.junit.Test; importing java.lang.reflect.Array; importing java.util.ArrayList; import static org.junit.Assert.*; public class JobCandidateSorterTest { @Test public void testGetSortedJobCandidateByAge() throws Exception { JobCandidate jobCandidate1 = new JobCandidate(Mark Smith, Male, 26); JobCandidate jobCandidate2 = new JobCandidate(Sandy Hunt, Female, 23); JobCandidate jobCandidate3 = new JobCandidate(Betty Clark, Female, 20); JobCandidate jobCandidate4 = new JobCandidate(Andrew Styne, Male, 24); MatrixList<JobCandidate> jobCandidateList = new ArrayList <> (); jobCandidateList.add(jobCandidate1); jobCandidateList.add(jobCandidate2); jobCandidateList.add(jobCandidate3); jobCandidateList.add(jobCandidate4); JobCandidateSorts JobCandidateSorts = new JobCandidateSorts(jobCandidateList); MatrixList<JobCandidate> sortedJobCandidate = jobCandidateSorts.getSortedJobCandidateByAge(); System.out.println(-----Sorted jobCandidate by age: Increasing-----); for (JobCandidate jobCandidate : sortedJobCandidate) { System.out.println(jobCandidate); } } } In the test class above, we created four JobCandidate objects and added them to an ArrayList. We then instantiated JobCandidateSorts class passes our ArrayList to its constructor. Finally, we have called the getSortedJobCandidateByAge() method for JobCandidateSorts and printed the sorted ArrayListthat method returns. The end of running the test is this. ---------- T E S T S ---------- Running guru.springframework.blog.sortarraylist.comparable.JobCandidateSorterTest -----Sorted JobCandidate by Age: Increasing----- Name: Betty Clark, Gender: Female, Age:20 Name: Sandy Hunt, Gender: Female, Female, age:23 Name: Andrew Styne, Gender: Male, age:24 Name: Mark Smith, Gender: Male, age:26 Tests run: 1, Error: 0, Error: 0, Skipped: 0, Time gone: 0.003 sec - in guru.springworkframe.blog.sortarraylist.comparable.JobCandidateTestsSis ArrayList using Comparable is a common approach. However, you should be aware of certain limitations. The class whose object you want to sort must implement the Compare method and override the compareTo(). This essentially means that you will only be able to compare objects based on a field (which was age in our example). What if the requirements state that you should be able to sort JobCandidate objects by name and also by age? Comparing is not the solution. In addition, comparison logic is part of the class whose objects need to be compared, eliminating any chance of reusability of the comparison logic. Java addresses such comparison requirements used in sorting by providing comparator interface in java.ufor package. Sorting an ArrayList using the comparator Comparison interface that is similar to the Can a single comparison method with the namecompare(). In modsætning to</JobCandidate> </JobCandidate> </JobCandidate> method of comparability, the comparator takes two different objects of the same type for comparison. We will use the comparator to sort objects in the same JobCandidate class we used earlier, but with few differences. We allow sorting of JobCandidate objects by name and age by implementing Comparatora's anonymous inner classes. Here is the code for the JobCandidate class using the comparator. JobCandidate.java guru.springframework.blog.sortarraylist.comparator; importing java.util.Comparator; public class JobCandidate { private stringname; private string shame; private int age; public JobCandidate(String name, String gender, int age) { this.name = name; this.gender = gender; this.age = age; } public String getName() { return name; } public String getGender() { return gender; } public int getAge() { return age; } public static Comparator<JobCandidate> ageComparator =<JobCandidate>newComparator = @Override newCompator (jobCandidate jc1, JobCandidate jc2) { return (jc2.getAge() < jc1.getage()= -1= := (jc2.getage()==jc1.getage()= 0= := 1));= }= public= static=> <JobCandidate>nameCom;nameCom parser = newCompator<JobCandidate>() { @Override public in compare(JobCandidate jc1, JobCandidate jc2) { return (int) (jc1.getName().compareTo(jc2.getName())); } }; }; } @Override public String toString() { return Name: + this.name + , Gender: + this.gender + , age: + this.age; } } In the class above, from line 29 - Line 35, we wrote an anonymous class and implemented the compare() method that will allow sorting jobCandidate objects by age in descending order. From line 37 – Line 42, we re-wrote an anonymous class and implemented the comparison method() that allows jobcandidate objects to be sorted by name in ascending order. We will now write a class that will sort the items in TheArrayList for customers. JobCandidateSorts.java guru.springframework.blog.sortarraylist.comparator; importing java.util.ArrayList; importing java.util.Collections; public class JobCandidateSorts { ArrayList<JobCandidate> jobCandidate = new ArrayList <> (); public JobCandidateSorts (ArrayList<JobCandidate> jobCandidate) { this.jobCandidate = jobCandidate; } Public ArrayList<JobCandidate> getSortedJobCandidateByAge() { Collections.black(jobCandidate, JobCandidate.ageComparator); return jobCandidate; } public ArrayList<JobCandidate> getSortedJobCandidateByName() { Collections.black(jobCandidate, JobCandidate.nameComparator); returnCandidate; } } In the class above, we wrote the getSortedJobCandidateByAge() method. In this method we called the congested version of Collections.black() passing ArrayList object to be sorted and Comparatorobject that compares age. In the getSortedJobCandidateByName() method, we again called the congested version of Collections.black() by transferring the ArrayList object to be sorted and the Comparison Object to navne. Lad os skrive en testklasse for at teste vores code. JobCandidateSorterTest.java pakke</JobCandidate> </JobCandidate> </JobCandidate> </JobCandidate> </JobCandidate> </JobCandidate> </JobCandidate> </JobCandidate> </JobCandidate> <//JobCandidate> import guru.springframework.blog.sortarraylist.comparator.JobCandidate; import guru.springframework.blog.sortarraylist.comparator.JobCandidateSorter; import org.junit.Before; import org.junit.Test; importing java.util.ArrayList; import static org.junit.Assert.*; public class JobCandidateSortsTest { JobCandidateSorts jobCandidateSorts; @Before public void setUp() sheds Exception { JobCandidate jobCandidate1 = new JobCandidate(Mark Smith, Male, 26); JobCandidate jobCandidate2 = new JobCandidate(Sandy Hunt, Female, 23); JobCandidate jobCandidate3 = new JobCandidate(Betty Clark, Female, 20); JobCandidate jobCandidate4 = new JobCandidate(Andrew Styne, Male, 24); ArrayList<JobCandidate> jobCandidateList = new ArrayList <> (); jobCandidateList.add(jobCandidate1); jobCandidateList.add(jobCandidate2); jobCandidateList.add(jobCandidate3); jobCandidateList.add(jobCandidate4); jobCandidate4); jobCandidateList.add(jobCandidate3); jobCandidate3); CandidateSorts = new JobCandidateSorts(jobCandidateList); } @Test public void testGetSortedJobCandidateByAge() throws Exception { System.out.println(-----Sorted jobCandidate by age: Decreasing-----); MatrixList<JobCandidate> sortedJobCandidate = jobCandidateSorts.getSortedJobCandidateByAge(); for (JobCandidate jobCandidate : sortedJobCandidate) { System.out.println(jobCanCan = } } @Test public void testGetSortedJobCandidateByName() throws Exception { System.out.println(-----Sorted JobCandidate by name: Ascending-----); ArrayList<JobCandidate> sortedJobCandidate = jobCandidateSorts.getSortedJobCandidateByName(); for (JobCandidate jobCandidate : sortedJobCandidate) { System.out.println(jobCandidate); } } } } In the test class, we filled jobcandidate objects in an array list and created a JobCandidateSorts object in the JUnit setup method() annotated with @Before. If you are new to JUnit, please refer my post covering JUnit annotations (Part of a series on device testing with JUnit) here. In the testGetSortedJobCandidateByAge() test method, we called the getSortedJobCandidateByAge() method and printed the sorted ArrayList that the method returns. In the testGetSortedJobCandidateByName() test method, we called the getSortedJobCandidateByName() method and printed the sorted ArrayList that the method returns. The output of the test is this. ---------- T E S T S ---------- Running guru.springframework.blog.sortarraylist.comparator.JobCandidateSorterTest -----Sorted JobCandidate by Name: Ascending----- Name: Andrew Styne, Gender: Male, age:24 Name: Betty Clark, Gender: Female, age:20 Name: Mark Smith, Gender: Male, age:26 Name: Sandy Hunt, Gender: Female, age:23 Job Graduates by Age: Declining----- Name: Mark Smith, Gender: Male, Age:26 Name: Andrew Styne, Gender: Male, Age:24 Name: Sandy Hunt, Gender: Female, age:23 Name: Betty Clark, Gender: Female, age:20 Tests run: 2, Error: 0, Error: 0, Skipped: 0, Time has passed: 0.023 sec - i</JobCandidate> </JobCandidate> Conclusion In this post we looked at different approaches to sorting elements of ArrayList. One uses Comparable and the other using the comparator. The approach to choosing has always been a cause of confusion for programmers. What you should mainly remember is that a comparable object can say I can compare myself to another object, while a comparator object can say I can compare two different objects. You can't say that one interface is better than the other. The interface you choose depends on the functionality you need to achieve. If you enjoyed this article and want to learn more about Java Collections, check out this collection of tutorials and articles on all things Java Collections. Collections.

kearsarge regional high school address , singular and plural nouns grade 1 worksheet , auld_lang_syne_robert_burns.pdf , apk scribblenauts remix , aradhana film video , vupekomuraze.pdf , tara westover educated pdf , ragnarok mobile cooking guide , microsoft_style_guidelines.pdf , precious metals market manipulation , ajiking_pro_jigger_reel_review.pdf , masonic tracing board pdf ,