Comparison of 2 ArrayList

Problem statement is that given two ArrayList we have to compare them. If both the list are identical then return true else return false.

We will solve this particular question in 2 different methods.

* **First**, we will compare them by searching for each element from list1 to list2 and if found remove that element from both lists. The searching is done by [contains(Object o)](http://data-structure-learning.blogspot.com/2015/05/search-element-in-arraylist.html) method which take O(n) time for each element. In all, searching for all elements in other list results in O(n2).

/\*\*

\* Take iterator and iterate through the list.

\* \*/

Iterator<Integer> iter = l1.iterator();

**while** (iter.hasNext()) {

/\*\*Take the next element from list1 into x\*/

Integer x = iter.next();

/\*\*Check if x is in list2\*/

**int** index = l2.indexOf(x);

/\*\*If not then return false\*/

**if** (index == -1) {

**return** **false**;

} **else** {

/\*\*If found then remove it from both lists.\*/

l2.remove(index);

iter.remove();

}

}

* **Second**, we will sort both the lists and then we will use equals method of [AbstractList<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-11-abstractlist.html) class to compare lists.
  + Why equals(Object o) method of AbstractList<E>? Because ArrayList<E> class extends AbstractList<E> class and hence child inherits properties of parent.
  + And ArrayList<E> does not have equals(Object o) but AbstractList<E> has it. Let us understand equals(Object o) method of AbstractList<E> class then we will move forward with our solution.
  + I have written a post regarding [Abstract<E> and its methods](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-11-abstractlist.html) are also explained. I would recommend visiting it.

Below is the explanation for equals(Object o) method.

equals(Object o)

Checks if 2 lists are same or not. More precisely both should be instance of List, both lists must have same elements & same size. Below is equals(Object o) method with comments.

/\*\*

\* This method is used to compare this list with the parameter

\* Object o for equality.

\* If both of them are list, both have same size, both have same element pairs

\* then only lists are same else return false.

\* \*/

**public** **boolean** equals(Object o) {

/\*\*

\* If o and this are same the return true

\* \*/

**if** (o == **this**)

**return** **true**;

/\*\*

\* If o is not instance of List then return false.

\* \*/

**if** (!(o **instanceof** List))

**return** **false**;

//Take listIterator() for this list.

ListIterator<E> e1 = listIterator();

/\*\*

\* Now o is instance of List so we will cast it to List<?> and get

\* listIterator() for it.

\* \*/

ListIterator<?> e2 = ((List<?>) o).listIterator();

/\*\*

\* Now we compare if both of them has next or not.

\* Reason is if one list's size is less then another

\* than this test will fail.

\* Pretty neat approach

\* \*/

**while** (e1.hasNext() && e2.hasNext()) {

E o1 = e1.next();

Object o2 = e2.next();

//Comparaision of elements of both lists

**if** (!(o1==**null** ? o2==**null** : o1.equals(o2)))

**return** **false**;

}

/\*\*

\* If any of list has next element then lists are not same

\* return false.

\* \*/

**return** !(e1.hasNext() || e2.hasNext());

}

Now below is the code for comparing 2 lists for second method.

/\*\*

\* Sort both the lists

\*/

Collections.*sort*(l1);

Collections.*sort*(l2);

**return** l1.equals(l2);

Now let us assume that list contains null elements. Then Collections.sort(..) will throw NullPointerException. To avoid this we will write a Comparator<Integer>.

/\*\*

\* If any element in List is null then Collections.sort(..)

\* throws NullPointerException.

\* So we write a Comparator<Integer>

\* \*/

**private** **static** Comparator<Integer> IntComparator(){

Comparator<Integer> intComparator=**new** Comparator<Integer>() {

@Override

**public** **int** compare(Integer o1, Integer o2) {

**if**(o1==**null**){

**return** (o2!=**null**)?0:1;

}

**else** **if**(o1!=**null**&&o2!=**null**){

**return** o1.compareTo(o2);

}

**else**{

**return** 0;

}

}

};

**return** intComparator;

}

Below is the entire code:

**package** arraylist;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.Iterator;

**import** java.util.List;

**public** **class** ArrayListEquals {

**public** **static** **void** main(String[] args) {

}

/\*\*

\* This method is used to compare two list's

\* returns true if both lists has same elements

\* else returns false

\* \*/

**public** **static** **boolean** compare(List<Integer> l1, List<Integer> l2) {

**if**(*bothNull*(l1, l2)){

**return** **true**;

}

**if**(*eitherNullOrNoSameSize*(l1,l2)){

**return** **false**;

}

/\*\*

\* Take iterator and iterate through the list.

\* \*/

Iterator<Integer> iter = l1.iterator();

**while** (iter.hasNext()) {

/\*\*Take the next element from list1 into x\*/

Integer x = iter.next();

/\*\*Check if x is in list2\*/

**int** index = l2.indexOf(x);

/\*\*If not then return false\*/

**if** (index == -1) {

**return** **false**;

} **else** {

/\*\*If found then remove it from both lists.\*/

l2.remove(index);

iter.remove();

}

}

**return** (l1.size() == 0 && l2.size() == 0);

}

**public** **static** **boolean** compareBySort(List<Integer> l1, List<Integer> l2) {

**if**(*bothNull*(l1, l2)){

**return** **true**;

}

**if**(*eitherNullOrNoSameSize*(l1,l2)){

**return** **false**;

}

/\*\*

\* Sort both the lists

\*/

Collections.*sort*(l1, *IntComparator*());

Collections.*sort*(l2, *IntComparator*());

**return** l1.equals(l2);

}

/\*\*

\* If any element in List is null then Collections.sort(..)

\* throws NullPointerException.

\* So we write a Comparator<Integer> to filter null elements.

\* \*/

**private** **static** Comparator<Integer> IntComparator(){

Comparator<Integer> intComparator=**new** Comparator<Integer>() {

@Override

**public** **int** compare(Integer o1, Integer o2) {

**if**(o1==**null**){

**return** (o2!=**null**)?0:1;

}

**else** **if**(o1!=**null**&&o2!=**null**){

**return** o1.compareTo(o2);

}

**else**{

**return** 0;

}

}

};

**return** intComparator;

}

**public** **static** **boolean** bothNull(List<Integer> l1, List<Integer> l2){

/\*\*

\* If both the lists are null return true

\* \*/

**if** (l1 == **null** && l2 == **null**) {

**return** **true**;

}

**return** **false**;

}

**public** **static** **boolean** eitherNullOrNoSameSize(List<Integer> l1, List<Integer> l2){

/\*\*

\* 3 conditions leads to return false.

\* First two conditions are if either lists are null then return false.

\* Third is size are not same then return false.

\* \*/

**if** ((l1 == **null** && l2 != **null**)

|| (l1 != **null** && l2 == **null**)

|| (l1.size() != l2.size())) {

**return** **true**;

}

**return** **false**;

}

}

Let us run some test cases against it.

1. If both lists are null return **true.**
2. If both lists are empty return **true.**
3. If either list is null (i.e. first list is not null or second list is not null) then return **false.**
4. Lists with same elements passes the test.
5. Lists with different elements fails the test.

Below are the test cases for both the methods written above.

**package** arraylist;

**import** java.util.ArrayList;

**import** java.util.List;

**import** junit.framework.Assert;

**import** org.junit.Test;

**public** **class** ArrayListEqualsTestCase {

/\*\*

\* Comparison for both list

\* Both lists are null

\* \*/

@Test

**public** **void** testForBothNull() {

ArrayList<Integer> l1 = **null**;

ArrayList<Integer> l2 = **null**;

Assert.*assertTrue*(ArrayListEquals.*compare*(l1, l2));

}

/\*\*

\* Comparison for either list is null

\* \*/

@Test

**public** **void** testForFirstListNull() {

ArrayList<Integer> l1 = **null**;

ArrayList<Integer> l2 = **new** ArrayList<Integer>();

Assert.*assertFalse*(ArrayListEquals.*compare*(l1, l2));

}

/\*\*

\* Comparison for either list is null

\* \*/

@Test

**public** **void** testForSecondListNull() {

ArrayList<Integer> l1 = **new** ArrayList<Integer>();

ArrayList<Integer> l2 = **null**;

Assert.*assertFalse*(ArrayListEquals.*compare*(l1, l2));

}

/\*\*

\* Test for empty lists.

\* \*/

@Test

**public** **void** testForEmptyLists(){

ArrayList<Integer> l1 = **new** ArrayList<Integer>();

ArrayList<Integer> l2 = **new** ArrayList<Integer>();

Assert.*assertTrue*(ArrayListEquals.*compare*(l1, l2));

}

/\*\*

\* Comparison that returns false.

\* \*/

@Test

**public** **void** compareFails() {

List<Integer> l1 = **new** ArrayList<Integer>();

l1.add(1);

l1.add(2);

l1.add(20);

l1.add(12);

l1.add(2);

List<Integer> l2 = **new** ArrayList<Integer>();

l2.add(2);

l2.add(20);

l2.add(1);

l2.add(12);

l2.add(21);

Assert.*assertFalse*(ArrayListEquals.*compare*(l1, l2));

}

/\*\*

\* Comparison that returns true.

\* \*/

@Test

**public** **void** comparePasses() {

List<Integer> l1 = **new** ArrayList<Integer>();

l1.add(1);

l1.add(2);

l1.add(20);

l1.add(12);

l1.add(2);

List<Integer> l2 = **new** ArrayList<Integer>();

l2.add(2);

l2.add(20);

l2.add(1);

l2.add(12);

l2.add(2);

Assert.*assertTrue*(ArrayListEquals.*compare*(l1, l2));

}

// ------------------

/\*\*

\* Comparison for both list

\* Both lists are null

\* \*/

@Test

**public** **void** testForBothNullBySort() {

ArrayList<Integer> l1 = **null**;

ArrayList<Integer> l2 = **null**;

Assert.*assertTrue*(ArrayListEquals.*compareBySort*(l1, l2));

}

/\*\*

\* Comparison for either list is null

\* \*/

@Test

**public** **void** testForFirstListNullBySort() {

ArrayList<Integer> l1 = **null**;

ArrayList<Integer> l2 = **new** ArrayList<Integer>();

Assert.*assertFalse*(ArrayListEquals.*compareBySort*(l1, l2));

}

/\*\*

\* Comparison for either list is null

\* \*/

@Test

**public** **void** testForSecondListNullBySort() {

ArrayList<Integer> l1 = **new** ArrayList<Integer>();

ArrayList<Integer> l2 = **null**;

Assert.*assertFalse*(ArrayListEquals.*compareBySort*(l1, l2));

}

/\*\*

\* Test for empty lists.

\* \*/

@Test

**public** **void** testForEmptyListsBySort(){

ArrayList<Integer> l1 = **new** ArrayList<Integer>();

ArrayList<Integer> l2 = **new** ArrayList<Integer>();

Assert.*assertTrue*(ArrayListEquals.*compareBySort*(l1, l2));

}

/\*\*

\* Comparison that returns false.

\* \*/

@Test

**public** **void** compareFailsBySort() {

List<Integer> l1 = **new** ArrayList<Integer>();

l1.add(1);

l1.add(2);

l1.add(20);

l1.add(12);

l1.add(2);

List<Integer> l2 = **new** ArrayList<Integer>();

l2.add(2);

l2.add(20);

l2.add(1);

l2.add(12);

l2.add(21);

Assert.*assertFalse*(ArrayListEquals.*compareBySort*(l1, l2));

}

/\*\*

\* Comparison that returns true.

\* \*/

@Test

**public** **void** comparePassesBySort() {

List<Integer> l1 = **new** ArrayList<Integer>();

l1.add(1);

l1.add(2);

l1.add(20);

l1.add(12);

l1.add(2);

List<Integer> l2 = **new** ArrayList<Integer>();

l2.add(2);

l2.add(20);

l2.add(1);

l2.add(12);

l2.add(2);

Assert.*assertTrue*(ArrayListEquals.*compareBySort*(l1, l2));

}

}

Test Cases Execution

That’s all on comparison of two lists.

Do check other blogs such as [iterating over List<E>(6 ways)](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-6iterating-over.html), [iterating over Map<K, V>(5 ways)](http://data-structure-learning.blogspot.com/2015/05/different-ways-to-traverse-map.html), [Differences](http://data-structure-learning.blogspot.com/2015/05/difference-between-arraylist-and_28.html) and [similarities](http://data-structure-learning.blogspot.com/2015/05/similarities-between-arraylist-and.html) between LinkedList<E> and ArrayList<E> and [when](http://data-structure-learning.blogspot.com/2015/05/when-to-use-arraylist-and-linkedlist.html) to use them.

I am also writing on [Collection Interview Questions](http://data-structure-learning.blogspot.com/p/java-collections_26.html).