Calling remove in foreach loop in Java [duplicate]

Asked 15 years, 4 months ago Modified 9 years, 1 month ago Viewed 567k times

679

This question already has answers here:
Iterating through a Collection, avoiding ConcurrentModificationException when removing objects in a loop (32 answers)
Closed 10 years ago.

In Java, is it legal to call remove on a collection when iterating through the collection using a foreach loop? For instance:

```
List<String> names = ....
for (String name : names) {
   // Do something
   names.remove(name).
}
```

As an addendum, is it legal to remove items that have not been iterated over yet? For instance,

```
//Assume that the names list as duplicate entries
List<String> names = ....
for (String name : names) {
    // Do something
    while (names.remove(name));
}
```

java loops iterator foreach

Share
Improve this question
Follow

edited Jul 30, 2009 at 20:16
James McMahon
49.5k ● 69 ● 210 ● 288

asked Jul 28, 2009 at 20:39
Michael Bobick
8,997 ● 5 ● 23 ● 13

34 Not a great plan. Just because the language tolerates it on any given run, doesn't make it a good idea. – Mark McKenna Nov 1, 2011 at 12:57

You must've caught me in a bad mood, but seems to me the answer to this question comes straight out of the foreach documentation. – CurtainDog Jun 25, 2012 at 11:47

2 stackoverflow.com/questions/223918/… – Jeevi Oct 18, 2012 at 18:29

1 As an alternative, you might consider not modifying your collection in-place but use a filtering combinator such as Guava's Iterables#filter: code.google.com/p/guava-libraries/wiki/FunctionalExplained Be aware of its lazy behavior! – thSoft Dec 17, 2012 at 15:12

Did you really intend to ask about `Collection` specifically, rather than `List` which you use in your code? If you intended to ask about `List` and not `Collection`, then please edit this question to reflect that - then this question would not be a duplicate! (One big difference of `List` vs `Collection` is that `List` includes `get` in its interface, while `Collection` does not). – cellepo Dec 19, 2018 at 22:49

11 Answers

Sorted by: Highest score (default)

1033

To safely remove from a collection while iterating over it you should use an Iterator.

For example:

```
List<String> names = ....
Iterator<String> i = names.iterator();
while (i.hasNext()) {
   String s = i.next(); // must be called before you can call i.remove()
   // Do something
   i.remove();
}
```

From the Java Documentation :

> The iterators returned by this class's iterator and listIterator methods are fail-fast: if the list is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove or add methods, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Perhaps what is unclear to many novices is the fact that iterating over a list using the for/foreach constructs implicitly creates an iterator which is necessarily inaccessible. This info can be found here

Share
Improve this answer
Follow

edited Dec 2, 2014 at 11:11
mphizi
13 ● 2

answered Jul 28, 2009 at 20:43
Mark
29.1k ● 8 ● 65 ● 93

43 Note that you must call i.next() before you can call i.remove(): docs.oracle.com/javase/6/docs/api/java/util/Iterator.html – John Mellor Mar 7, 2012 at 15:18

13 I'm curious, why is this considered safe? Is the `Iterator` acting as middle man? – James P. May 12, 2012 at 22:02

22 To quote the Javadoc for Iterator.remove() "The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling this method." The iterator is acting as a middle man to safely perform the removal but allow the iteration to continue as expected. – Mark Jun 25, 2012 at 11:22

7 but worth noting that the remove() method is OPTIONAL on an iterator and will throw exceptions if it isn't implemented for your particular collection or JVM – user1743310 Jun 12, 2013 at 17:33

1 @committedandroider From the Javadoc for ConcurrentModificationException `Note that this exception does not always indicate that an object has been concurrently modified by a different thread. If a single thread issues a sequence of method invocations that violates the contract of an object, the object may throw this exception. For example, if a thread modifies a collection directly while it is iterating over the collection with a fail-fast iterator, the iterator will throw this exception.` – Mark Mar 22, 2015 at 22:18

178

You don't want to do that. It can cause undefined behavior depending on the collection. You want to use an Iterator directly. Although the for each construct is syntactic sugar and is really using an iterator, it hides it from your code so you can't access it to call `Iterator.remove`.

> The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling this method.

Instead write your code:

```
List<String> names = ....
Iterator<String> it = names.iterator();
while (it.hasNext()) {

    String name = it.next();
    // Do something
    it.remove();
}
```

Note that the code calls `Iterator.remove`, not `List.remove`.

Addendum:

Even if you are removing an element that has not been iterated over yet, you still don't want to modify the collection and then use the `Iterator`. It might modify the collection in a way that is surprising and affects future operations on the `Iterator`.

Share

Improve this answer

Follow

edited Jul 28, 2009 at 20:48

answered Jul 28, 2009 at 20:43

Jared Oberhaus

14.6k ● 5 ● 58 ● 55

5 Thumbs up for the extra Note *that the code calls Iterator.remove, not List.remove". I almost missed and used list.remove – pratpor Sep 12, 2019 at 11:56

97

```
for (String name : new ArrayList<String>(names)) {
    // Do something
    names.remove(nameToRemove);
}
```

You clone the list `names` and iterate through the clone while you remove from the original list. A bit cleaner than the top answer.

Share

Improve this answer

Follow

edited Nov 22, 2015 at 3:37

Chin

20.6k ● 38 ● 112 ● 172

answered Jan 23, 2013 at 22:01

ktamlyn

4,709 ● 2 ● 31 ● 42

3 this supposed to be the first answer here... – itzhar Aug 26, 2015 at 10:45

4 Beware. For composite objects with no equals and hashCode method, it might just fail. However, marked answer safely removes it. – D3V Dec 28, 2015 at 14:54

28 Though short and clean, if performance/memory usage are an issue, it is worth to note that this solution runs in O(n²) and creates a copy of the original list, which requires memory and an operation depending on the type of the list. Using an iterator over a LinkedList you can bring complexity down to O(n). – SND Dec 15, 2017 at 7:53

2 @SND Why would it be n^2? Creating copy of the list is O(n), therefore this is O(n) as well. – FINDarkside Apr 16, 2019 at 17:43

3 @FINDarkside the O(n^2) doesn't come from creating a copy, but looping through the ArrayList (O(n)) while calling remove() on each element (also O(n)). – chakwok May 21, 2019 at 18:40

74

The java design of the "enhanced for loop" was to not expose the iterator to code, but the only way to safely remove an item is to access the iterator. So in this case you have to do it old school:

```
for(Iterator<String> i = names.iterator(); i.hasNext();) {
      String name = i.next();
      //Do Something
      i.remove();
}
```

If in the real code the enhanced for loop is really worth it, then you could add the items to a temporary collection and call removeAll on the list after the loop.

EDIT (re addendum): No, changing the list in any way outside the iterator.remove() method while iterating will cause problems. The only way around this is to use a CopyOnWriteArrayList, but that is really intended for concurrency issues.

The cheapest (in terms of lines of code) way to remove duplicates is to dump the list into a LinkedHashSet (and then back into a List if you need). This preserves insertion order while removing duplicates.

Share

Improve this answer

Follow

edited Jul 18, 2014 at 13:31

answered Jul 28, 2009 at 20:47

Yishai
91.8k ● 31 ● 192 ● 264

Although `CopyOnWriteArrayList` will avoid `ConcurrentModificationException`, there could still be other indexing problems/Exceptions (more details). – cellepo Dec 20, 2018 at 1:23

35

I didn't know about iterators, however here's what I was doing until today to remove elements from a list inside a loop:

```
List<String> names = ....
for (i=names.size()-1;i>=0;i--) {
    // Do something
    names.remove(i);
}
```

This is always working, and could be used in other languages or structs not supporting iterators.

Share

Improve this answer

Follow

edited Feb 27, 2013 at 14:53

Niko
26.7k ● 9 ● 96 ● 112

answered May 18, 2011 at 14:09

Serafeim
15.1k ● 15 ● 95 ● 136

2 Just as a side-note, that should work fine with any base-class list, but wouldn't be portable to more esoteric structures (like a self-sorting sequence, for example--in general, anywhere the ordinal for a given entry could change between list iterations). – Mark McKenna Nov 1, 2011 at 13:01

8 Note: this works precisely because you're iterating backwords over the elements, so the indexes of the other remaining elements don't change when you remove the `i` th element. If you were looping from 0 to `size()-1`, and doing the remove, you'd see the issues mentioned in other answers. Nice job, though! – Jake Toronto Sep 24, 2014 at 4:46

This is just a horrible way to do. So many things can go wrong depending on the implementation of the List (or Set, etc.). – DavidR Nov 29, 2016 at 2:21

2 Such indexed based operations should only be conducted on lists whose elements can be accessed in O(1) constant time. If the list is not randomly accessible (does not implement RandomAccess) then you should use an iterator instead, as these types of collections usually take longer to retrieve an element at a specific index position, such as a LinkedList. – TheArchon Feb 4, 2017 at 2:45

2 The advantage of the above method is that you (or at least most people) don't have to google for "java iterator example" but can write it immediately, by memory. – Serafeim Oct 18, 2017 at 20:49

30

Yes you can use the for-each loop, To do that you have to maintain a separate list to hold removing items and then remove that list from names list using `removeAll()` method,

```
List<String> names = ....

// introduce a separate list to hold removing items
List<String> toRemove= new ArrayList<String>();

for (String name : names) {
   // Do something: perform conditional checks
   toRemove.add(name);
}
names.removeAll(toRemove);

// now names list holds expected values
```

Share

Improve this answer

Follow

edited Dec 11, 2010 at 13:00

answered Dec 10, 2010 at 11:52

Chathuranga Withana
882 ● 1 ● 9 ● 12

2 Why add the overhead of another list ? – Varun Mehta Jan 25, 2011 at 3:31

2 Since the `for-each` loop hides the `iterator`, you cannot call the `remove()` directly. So in order to remove items from a `for-each` loop while iterating through it, have to maintain a separate list. That list will maintain references to the items to be removed...
– Chathuranga Withana Mar 19, 2011 at 18:44

1 Its very slow. removeAll 1. runs contains() on every toRemove element 2. then runs remove() which has to find element index so need to again look through whole list. – mmatloka Jan 4, 2012 at 13:39

this isn't calling remove() in a foreach loop – user1743310 Jun 12, 2013 at 17:36

5

Make sure this is not code smell. Is it possible to reverse the logic and be 'inclusive' rather than 'exclusive'?

```
List<String> names = ....
List<String> reducedNames = ....
for (String name : names) {
   // Do something
   if (conditionToIncludeMet)
       reducedNames.add(name);
}
return reducedNames;
```

The situation that led me to this page involved old code that looped through a List using indecies to remove elements from the List. I wanted to refactor it to use the foreach style.

It looped through an entire list of elements to verify which ones the user had permission to access, and removed the ones that didn't have permission from the list.

```
List<Service> services = ...
for (int i=0; i<services.size(); i++) {
    if (!isServicePermitted(user, services.get(i)))
         services.remove(i);
}
```

To reverse this and not use the remove:

```
List<Service> services = ...
List<Service> permittedServices = ...
for (Service service:services) {
    if (isServicePermitted(user, service))
         permittedServices.add(service);
}
return permittedServices;
```

When would "remove" be preferred? One consideration is if gien a large list or expensive "add", combined with only a few removed compared to the list size. It might be more efficient to only do a few removes rather than a great many adds. But in my case the situation did not merit such an optimization.

Share Improve this answer Follow

answered Nov 14, 2011 at 16:57
bmcdonald
401 ● 7 ● 12

4

Those saying that you can't safely remove an item from a collection except through the Iterator aren't quite correct, you can do it safely using one of the concurrent collections such as ConcurrentHashMap.

Share Improve this answer Follow

answered Jul 28, 2009 at 23:25
sanity
35.7k ● 43 ● 140 ● 228

Although it will avoid `ConcurrentModificationException`, there could still be other indexing problems/Exceptions ([more details]). – cellepo Dec 20, 2018 at 1:22

1

1. Try this 2. and change the condition to "WINTER" and you will wonder:

```
public static void main(String[] args) {
  Season.add("Frühling");
  Season.add("Sommer");
  Season.add("Herbst");
  Season.add("WINTER");
  for (String s : Season) {
   if(!s.equals("Sommer")) {
    System.out.println(s);
    continue;
   }
   Season.remove("Frühling");
  }
 }
```

Share
Improve this answer
Follow

edited Aug 30, 2011 at 15:42
Grzegorz Rożniecki
28k ● 11 ● 93 ● 116

answered Jan 5, 2011 at 14:24
Carsten
19 ● 1

1

It's better to use an Iterator when you want to remove element from a list

because the source code of remove is

```
if (numMoved > 0)
    System.arraycopy(elementData, index+1, elementData, index,
             numMoved);
elementData[--size] = null;
```

so ,if you remove an element from the list, the list will be restructure ,the other element's index will be changed, this can result something that you want to happened.

Share Improve this answer Follow

answered Feb 9, 2012 at 8:51

song
19 ● 3

-5

Use

.remove() of Interator or

Use

CopyOnWriteArrayList

Share Improve this answer Follow

answered Oct 10, 2014 at 5:41

ThmHarsh
601 ● 7 ● 7