How do I join two lists in Java?

Asked 14 years, 10 months ago Modified 1 year, 2 months ago Viewed 1.4m times

1031

Conditions: do not modify the original lists; JDK only, no external libraries. Bonus points for a one-liner or a JDK 1.3 version.

Is there a simpler way than:

```
List<String> newList = new ArrayList<String>();
newList.addAll(listOne);
newList.addAll(listTwo);
```

java list java-5

Share Improve this question

Follow

edited Apr 7, 2021 at 12:12

asked Oct 9, 2008 at 23:17

Robert Atkins

23.5k 15 68 97

7 If you are doing this solely for iteration purposes see another question - there are google guava and java 8 solutions stackoverflow.com/questions/4896662/... – Boris Treukhov Jul 7, 2014 at 15:43

Java 8 solution with utility method: stackoverflow.com/a/37386846/1216775 – akhil_mittal Apr 29, 2017 at 5:40

This is just one of the many shortcomings of the Java Collections API. – ACV Sep 22, 2022 at 9:34

33 Answers

Sorted by:

Highest score (default)

1 2 Next

996

In Java 8:

```
List<String> newList = Stream.concat(listOne.stream(), listTwo.stream())
                             .collect(Collectors.toList());
```

Java 16+:

```
List<String> newList = Stream.concat(listOne.stream(), listTwo.stream()).toList();
```

Share Improve this answer

Follow

edited Jun 22, 2022 at 5:59

ZhekaKozlov

36.5k 20 126 155

answered Sep 8, 2013 at 19:31

Dale Emery

10.1k 1 15 14

144 Gawd, that's a thing in Java 8? Technically you win I guess, but that's a heck of a long line :-) – Robert Atkins Sep 9, 2013 at 20:03

6 For the casual reader, here is a shorter solution using also Java _ Streams: stackoverflow.com/a/34090554/363573 – Stephan Oct 4, 2016 at 13:16

7 It is ugly but at least its fluent and can be used without multi line lambdas. I really wish there was a fluent addAll that returned the concatinated list. – Usman Ismail Jan 11, 2018 at 15:06

20 I guess it's worth noting that it's really easy to get a distinct list out of this too, like so: `List<String> newList = Stream.concat(listOne.stream(), listTwo.stream()).distinct().collect(Collectors.toList());` – Roger Dec 30, 2018 at 19:18

17 Alternative to concat: stream of streams `Stream.of(listOne, listTwo).flatMap(Collection::stream).collect(Collectors.toList())` – Peter Walser Apr 9, 2019 at 11:17

687

Off the top of my head, I can shorten it by one line:

```
List<String> newList = new ArrayList<String>(listOne);
newList.addAll(listTwo);
```

Share Improve this answer Follow

answered Oct 9, 2008 at 23:21

AdamC
16.1k 8 51 67

192 While you're technically correct, you have shortened it by one line, the asymmetry of this bugs me. Enough that I'm happier to "spend" the extra line. – Robert Atkins Dec 15, 2011 at 0:54

17 Isn't there a problem here where the newList's interal array will be initialized to the size of listOne and then have to potentially expand when adding all of the items from listTwo? Would it be better to take the size of each list and use that to size the new array? – Eric Mar 2, 2016 at 8:38

3 This was the solution that worked best for me. I made a comparison on the performance of different solutions this came out winner, together with creating the empty list and then `addAll()` of both. I tried all those that suggest not copying the lists and they result having a lot of overhead we didn't need this time. – manuelvigarcia Sep 26, 2016 at 6:46

Use a `LinkedList` over `ArrayList` for efficient adding. stackoverflow.com/a/322742/311420 – Raymond Chenon Apr 7, 2020 at 14:49

this does not work with generic value types, I am using Java 17: List<? extends BaseEntity> toManyEntities = new ArrayList<>(); List<? extends BaseEntity> toManyInsertEntities = new ArrayList<>(); List<? extends BaseEntity> toManyUpdateEntities = new ArrayList<>(); toManyEntities = toManyInsertEntities; toManyEntities.addAll(toManyUpdateEntities); -> this cannot compile But this works, and is mentioned sort-of in the answer just above your answer: toManyEntities = Stream.concat(toManyInsertEntities.stream(),toManyUpdateEntities.stream()).collect(Collectors.toList()); – Ricardo Jul 8, 2022 at 9:11

You could use the Apache commons-collections library:

434

```
List<String> newList = ListUtils.union(list1, list2);
```

Share Improve this answer

Follow

edited Jun 24, 2018 at 14:29

Neuron

5,141 5 38 59

answered Aug 27, 2010 at 3:28

Guillermo

4,493 1 15 2

71 Nice, but requires apache commons. He did specify 'no external libraries' – Quantum7 Apr 26, 2011 at 1:17

127 @Quantum7, still useful for other people ;) Also, is apache commons even an external library? I don't start anything without it! – tster Mar 30, 2012 at 23:31

31 @Platinum No, according to the docs ListUtils.union is exactly equivalent to OP's code. But perhaps it is misleading to use a SET operation ("Union") in a list context. I can see how you might expect this to remove duplicates or something, but it appears the method does not do that. – Quantum7 Dec 17, 2012 at 23:52

30 Avoid Apache Commons Collections. It's not typesafe, there are no generics. Great if you use Java 1.4, but for Java 5 and above, I'd prefer Google Guava. – Michael Piefel Sep 8, 2013 at 19:37

18 @MichaelPiefel The latest Apache Commons Collections 4 is type-safe. With Java 8 method reference, this kind of static utilities become very important. – mingfai Jan 21, 2014 at 18:17

240

Another Java 8 one-liner:

```
List<String> newList = Stream.of(listOne, listTwo)
                             .flatMap(Collection::stream)
                             .collect(Collectors.toList());
```

As a bonus, since `Stream.of()` is variadic, you may concatenate as many lists as you like.

```
List<String> newList = Stream.of(listOne, listTwo, listThree)
                             .flatMap(Collection::stream)
                             .collect(Collectors.toList());
```

Share Improve this answer

Follow

edited May 11, 2020 at 8:24

Littm

4,923 4 30 38

answered Apr 3, 2014 at 22:51

Mark

3,057 1 14 5

44 `x -> x.stream()` could be replaced with `Collection::stream` . – Martin May 29, 2015 at 16:18

21 ...or even with `List::stream` . – MC Emperor Jan 11, 2019 at 10:34

Nice. I like this approach. – Kira Resari May 4, 2021 at 5:55

Super. Also null safe. Both lists can be null and it would still work ! – Farrukh Chishti Feb 14, 2022 at 14:13

2 In JDK 17, `.collect(Collectors.toList())` can be replaced with just `.toList()` ; – k_rollo Feb 18 at 0:22

101

One of your requirements is to preserve the original lists. If you create a new list and use `addAll()`, you are effectively doubling the number of references to the objects in your lists. This could lead to memory problems if your lists are very large.

If you don't need to modify the concatenated result, you can avoid this using a custom list implementation. The custom implementation class is more than one line, obviously...but using it is short and sweet.

CompositeUnmodifiableList.java:

```
public class CompositeUnmodifiableList<E> extends AbstractList<E> {

    private final List<? extends E> list1;
    private final List<? extends E> list2;

    public CompositeUnmodifiableList(List<? extends E> list1, List<? extends E> list2)
{
        this.list1 = list1;
        this.list2 = list2;
    }

    @Override
    public E get(int index) {
        if (index < list1.size()) {
            return list1.get(index);
        }
        return list2.get(index-list1.size());
    }

    @Override
    public int size() {
        return list1.size() + list2.size();
    }
}
```

Usage:

```
List<String> newList = new CompositeUnmodifiableList<String>(listOne,listTwo);
```

Share Improve this answer

Follow

edited May 19, 2021 at 11:10

ursa

4,404 1 24 38

answered Dec 13, 2012 at 20:54

Kevin K

9,344 3 37 62

13 This is a workable solution, but do note that if the underlying list objects change (list1, list2), the contents of this list change. You may not be able to modify an instance of the CompositeUnmodifiableList *itself* but if you can get a reference to the original lists, then you can. Also for those that are unfamiliar: the final modifier just affects the *reference* to the list object itself can't change but it's still possible for the contents of the list to change! – jwj Sep 16, 2014 at 3:34

3 @jwj all very good points, thank you. The class name probably deserves some explanation. I see this class as doing something very similar to the `Collections.unmodifiableList()` method, which wraps a list to make it unmodifiable. `CompositeUnmodifiableList` does the same thing, except it wraps two lists and provides a concatenated view. All the points you make about `CompositeUnmodifiableList` are also true of `Collections.unmodifiableList()` as well. – Kevin K Feb 23, 2015 at 16:22

2 The constructor can take `List<? extends E>` – Patrick Parker Feb 5, 2017 at 13:48

96

Probably not simpler, but intriguing and ugly:

```
List<String> newList = new ArrayList<String>() { { addAll(listOne); addAll(listTwo); }
};
```

Don't use it in production code... ;)

Share Improve this answer Follow

edited Jul 8, 2012 at 12:47 user142162

answered Oct 10, 2008 at 0:30

volley
6,651 1 27 28

55 Ugly and evil, just as almost any use of double brace initialization. It ís shorter, though ;) – Jorn Aug 15, 2009 at 20:50

5 @MarnixKlooster: Eclipse *knows* that you should not use it and makes it unpleasant to use ;-) – Joachim Sauer Oct 3, 2011 at 7:06

23 Though it is physically one line, I do not consider this a "one-liner". – splungebob Dec 13, 2012 at 21:28

12 why do people hate anonymous block initializers – NimChimpsky Dec 21, 2012 at 13:01

21 @NimChimpsky I think it's mostly because it's not just an anonymous block initializer, but you're actually creating an anonymous subclass of ArrayList. That being said, if you trust the results of this Double Brace Initilization question, it makes it seem like hating DBI is mostly a matter of stylistic taste and micro-optimization. As far as I can tell, there are no major penalties for doing it. The sneaky downside would be if you ever tried to compare its class because it won't be ArrayList. – Patrick Dec 27, 2012 at 17:15

88

Not simpler, but without resizing overhead:

```
List<String> newList = new ArrayList<>(listOne.size() + listTwo.size());
newList.addAll(listOne);
newList.addAll(listTwo);
```

Share Improve this answer Follow

answered Dec 13, 2012 at 20:35

Martin
2,563 28 22

I think it's the correct and the most efficient answer here. I just don't understand why Java doesn't provide a constructor or a static method for it. It isn't the second day of Java in this world, yet such basic things are just missing. So disappointing and triggering at the same time. – Frank Jul 18 at 8:33

Found this question looking to concatenate arbitrary amount of lists, not minding external libraries. So, perhaps it will help someone else:

59

`com.google.common.collect.Iterables#concat()`

Useful if you want to apply the same logic to a number of different collections in one for().

Share Improve this answer Follow

answered Jul 29, 2011 at 13:46

Yuri Geinish
16.7k 6 38 40

11 For example: Lists.newArrayList(Iterables.concat(list1,list2)); – meilechh Feb 13, 2014 at 17:14

you should call `com.google.common.collect.Iterators#concat(java.util.Iterator<? extends java.util.Iterator<? extends T>>)` instead of `Iterables#concat()` ; because the later still copy elements into temp link! – bob Apr 14, 2016 at 6:36

Java 8 (`Stream.of` and `Stream.concat`)

55

The proposed solution is for three lists though it can be applied for two lists as well. In Java 8 we can make use of Stream.of or Stream.concat as:

```
List<String> result1 =
Stream.concat(Stream.concat(list1.stream(),list2.stream()),list3.stream()).collect(Collec
List<String> result2 =
Stream.of(list1,list2,list3).flatMap(Collection::stream).collect(Collectors.toList());
```

`Stream.concat` takes two streams as input and creates a lazily concatenated stream whose elements are all the elements of the first stream followed by all the elements of the second stream. As we have three lists we have used this method (`Stream.concat`) two times.

We can also write a utility class with a method that takes any number of lists (using varargs) and returns a concatenated list as:

```
public static <T> List<T> concatenateLists(List<T>... collections) {
        return
Arrays.stream(collections).flatMap(Collection::stream).collect(Collectors.toList());
}
```

Then we can make use of this method as:

```
List<String> result3 = Utils.concatenateLists(list1,list2,list3);
```

Share Improve this answer Follow

edited Feb 17, 2020 at 6:09

answered May 23, 2016 at 9:04

akhil_mittal
23.3k 7 96 95

Here is a java 8 solution using two lines:

```
List<Object> newList = new ArrayList<>();
Stream.of(list1, list2).forEach(newList::addAll);
```

52

Be aware that this method should not be used if

- the origin of `newList` is not known and it may already be shared with other threads
- the stream that modifies `newList` is a parallel stream and access to `newList` is not synchronized or threadsafe

due to side effect considerations.

Both of the above conditions do not apply for the above case of joining two lists, so this is safe.

Based on this answer to another question.

Share Improve this answer Follow

edited Jul 26, 2017 at 6:52

answered Dec 4, 2015 at 14:27

SpaceTrucker
13.4k 6 60 98

18 If I am not wrong this is actually not recommended - docs.oracle.com/javase/8/docs/api/java/util/stream/... Please see side -effects section. > Side-effects in behavioral parameters to stream operations are, in general, discouraged, as they can often lead to unwitting violations of the statelessness requirement, as well as other thread-safety hazards. So in this case it is better to use Collectors.toList() – Anton Balaniuc Mar 14, 2017 at 15:31

@AntonBalaniuc Question is if whether this is really a side effect. At that point `newList` is not observable by any other thread. But you are correct that this probably shouldn't be done if it is unknown where the value of `newList` came from (for example if `newList` was passed as a parameter. – SpaceTrucker Jul 11, 2017 at 5:45

2 I'm curious; why `.forEach(newList::addAll);` instead of `.collect(Collectors.toList());` ? – 11684 Aug 22, 2017 at 10:16

5 @11684 because the collector would collect a `List<List<Object>>` . What you may have in mind is something like this: stackoverflow.com/questions/189559/... – SpaceTrucker Aug 22, 2017 at 11:23

@SpaceTrucker Oops, I overlooked that. Thanks for clearing up my confusion. Yes, I should have been thinking of `flatMap` . – 11684 Aug 22, 2017 at 11:26

39

This is simple and just one line, but will add the contents of listTwo to listOne. Do you really need to put the contents in a third list?

```
Collections.addAll(listOne, listTwo.toArray());
```

Share Improve this answer Follow

edited Sep 8, 2013 at 22:59

answered Sep 8, 2013 at 22:50

ceklock
6,144 10 56 78

14 Not modifying the original lists was one of the criteria, but this is useful to have here as an example for situations where that's not a constraint. – Robert Atkins Sep 9, 2013 at 20:01

10 Thanks, or even simpler listOne.addAll(listTwo) – Jay Jan 3, 2020 at 16:23

37

Slightly simpler:

```
List<String> newList = new ArrayList<String>(listOne);
newList.addAll(listTwo);
```

Share Improve this answer Follow

edited Mar 7, 2012 at 17:53

Grzegorz Rożniecki
27.4k 11 90 112

answered Oct 9, 2008 at 23:21

Tim
6,851 11 42 46

Would this cause duplicated Strings? Meaning a String that exists in both list will exist twice in the resulting list? – AgentKnopf Jun 13, 2012 at 10:33

6 @Zainodis Yes, there could be duplicates. The `List` structure imposes no uniqueness constraints. You can remove dupes by doing the same thing with sets. `Set<String> newSet = new HashSet<>(setOne); newSet.addAll(setTwo);` – Patrick Dec 27, 2012 at 17:01

22

A little shorter would be:

```
List<String> newList = new ArrayList<String>(listOne);
newList.addAll(listTwo);
```

Share Improve this answer Follow

answered Oct 9, 2008 at 23:22

Jorn
20.6k 18 79 126

21

You can create your generic **Java 8** utility method to concat **any number of lists**.

```
@SafeVarargs
public static <T> List<T> concat(List<T>... lists) {
    return Stream.of(lists).flatMap(List::stream).collect(Collectors.toList());
}
```

Share Improve this answer Follow

answered May 18, 2018 at 20:33

Daniel Hári
7,254 5 39 54

14

In **Java 8** (the other way):

```
List<?> newList =
Stream.of(list1, list2).flatMap(List::stream).collect(Collectors.toList());
```

Share Improve this answer Follow

edited Mar 24, 2017 at 4:43

noufalcep
3,446 15 33 51

answered Mar 24, 2017 at 2:58

Nitin Jain
197 1 9

This approach is already suggested by this answer: stackoverflow.com/a/37386846 – Miles Mar 24, 2017 at 3:20

13

You can do a oneliner if the target list is predeclared.

```
(newList = new ArrayList<String>(list1)).addAll(list2);
```

Share Improve this answer Follow

answered Oct 10, 2008 at 0:46

deterb
3,994 1 28 33

10

another one liner solution using `Java8` stream, since `flatMap` solution is already posted, here is a solution without `flatMap`

```
List<E> li = lol.stream().collect(ArrayList::new, List::addAll, List::addAll);
```

or

```
List<E> ints = Stream.of(list1, list2).collect(ArrayList::new, List::addAll,
List::addAll);
```

code

```
    List<List<Integer>> lol = Arrays.asList(Arrays.asList(1, 2, 3), Arrays.asList(4, 5,
6));
    List<Integer> li = lol.stream().collect(ArrayList::new, List::addAll,
List::addAll);
    System.out.println(lol);
    System.out.println(li);
```

output

```
[[1, 2, 3], [4, 5, 6]]
[1, 2, 3, 4, 5, 6]
```

Share Improve this answer Follow

edited Nov 12, 2016 at 4:44

answered Nov 12, 2016 at 4:38

Saravana
12.6k 2 39 57

1 I would add that this solution is probably more performant than the one using `flatMap`, because the lists are only iterated once when they are collected – Stefan Haberl May 23, 2017 at 9:01

I dont think `flatMap` causes additional iteration, could you clarify? – wilmol Mar 30, 2021 at 21:05

9

We can join 2 lists using java8 with 2 approaches.

```
    List<String> list1 = Arrays.asList("S", "T");
    List<String> list2 = Arrays.asList("U", "V");
```

1) Using concat :

```
    List<String> collect2 = Stream.concat(list1.stream(),
list2.stream()).collect(toList());
    System.out.println("collect2 = " + collect2); // collect2 = [S, T, U, V]
```

2) Using flatMap :

```
    List<String> collect3 = Stream.of(list1,
list2).flatMap(Collection::stream).collect(toList());
    System.out.println("collect3 = " + collect3); // collect3 = [S, T, U, V]
```

Share Improve this answer Follow

answered Mar 25, 2020 at 17:37

Himank Batra
301 5 3

3 When answering an eleven year old question with thirty other answers be sure to point out what new aspects of the question your answer addresses, and to note if these techniques would have worked when the question was asked, or if they depend on features that have been introduced over the years. – Jason Aller Mar 25, 2020 at 18:05

8

The smartest in my opinion:

```
/**
 * @param smallLists
 * @return one big list containing all elements of the small ones, in the same order.
 */
public static <E> List<E> concatenate (final List<E> ... smallLists)
{
    final ArrayList<E> bigList = new ArrayList<E>();
    for (final List<E> list: smallLists)
    {
        bigList.addAll(list);
    }
    return bigList;
}
```

Share Improve this answer Follow

answered Jul 26, 2012 at 9:25

Olivier Faucheux
2,520 3 29 37

5 Don't forget the `@SafeVarargs` ! – Resigned June 2023 Dec 7, 2014 at 16:00

The bigList might get resized potentially. So the solution is functionally correct, but might not the most efficient. – Frank Jul 18 at 8:39

8

Almost of answers suggest to use an ArrayList.

```
List<String> newList = new LinkedList<>(listOne);
newList.addAll(listTwo);
```

Prefer to use a LinkedList for efficient add operations.

ArrayList add is O(1) amortized, but O(n) worst-case since the array must be resized and copied. While LinkedList add is always constant O(1).

more infos https://stackoverflow.com/a/322742/311420

Share Improve this answer Follow

answered Apr 7, 2020 at 14:58

Raymond Chenon
11.5k 15 77 110

3 Without performance benchmarks, this is questionable advice, especially since your argument about `add` does not refer to the bulk `addAll` operation. – Christian Brüggemann Apr 15, 2021 at 21:08

6

You could do it with a static import and a helper class

nb the generification of this class could probably be improved

```
public class Lists {

   private Lists() { } // can't be instantiated

   public static List<T> join(List<T>... lists) {
      List<T> result = new ArrayList<T>();
      for(List<T> list : lists) {
         result.addAll(list);
      }
      return results;
   }

}
```

Then you can do things like

```
import static Lists.join;
List<T> result = join(list1, list2, list3, list4);
```

Share Improve this answer

Follow

edited Sep 8, 2013 at 19:43

FDinoff

30.7k 5 75 96

answered Oct 10, 2008 at 0:51

Dave Cheney

5,575 2 18 24

How is the static import or the helper class relevant? – shmosel May 30, 2019 at 21:02

I think the method signature will be like - 'public static <T> List<T> join'. And 'return result' – Md. Nowshad Hasan Jul 13, 2022 at 3:12

6

Java 8 version with support for joining by object key:

```
public List<SomeClass> mergeLists(final List<SomeClass> left, final List<SomeClass>
right, String primaryKey) {
    final Map<Object, SomeClass> mergedList = new LinkedHashMap<>();

    Stream.concat(left.stream(), right.stream())
        .map(someObject -> new Pair<Object, SomeClass>(someObject.getSomeKey(),
someObject))
        .forEach(pair-> mergedList.put(pair.getKey(), pair.getValue()));

    return new ArrayList<>(mergedList.values());
}
```

Share Improve this answer Follow

answered Nov 29, 2016 at 15:49

cslysy

810 7 11

4

```
public static <T> List<T> merge(List<T>... args) {
    final List<T> result = new ArrayList<>();

    for (List<T> list : args) {
        result.addAll(list);
    }

    return result;
}
```

Share Improve this answer Follow

answered Jul 12, 2014 at 17:12

martyglaubitz

992 10 21

4

Use a Helper class.

I suggest:

```
public static <E> Collection<E> addAll(Collection<E> dest, Collection<? extends E>...
src) {
    for(Collection<? extends E> c : src) {
        dest.addAll(c);
    }

    return dest;
}

public static void main(String[] args) {
    System.out.println(addAll(new ArrayList<Object>(), Arrays.asList(1,2,3),
Arrays.asList("a", "b", "c")));

    // does not compile
    // System.out.println(addAll(new ArrayList<Integer>(), Arrays.asList(1,2,3),
Arrays.asList("a", "b", "c")));

    System.out.println(addAll(new ArrayList<Integer>(), Arrays.asList(1,2,3),
Arrays.asList(4, 5, 6)));
}
```

Share Improve this answer Follow

edited Oct 5, 2016 at 10:18

answered Oct 10, 2008 at 11:36

alex
5,213 1 24 33

4

```
public static <T> List<T> merge(@Nonnull final List<T>... list) {
    // calculate length first
    int mergedLength = 0;
    for (List<T> ts : list) {
      mergedLength += ts.size();
    }

    final List<T> mergedList = new ArrayList<>(mergedLength);

    for (List<T> ts : list) {
      mergedList.addAll(ts);
    }

    return mergedList;
  }
```

Share Improve this answer Follow

answered Feb 21, 2018 at 11:11

Langusten Gustel
10.9k 9 46 59

My favourite way, using fluent api and Guava:

```
List<String> combined = ImmutableList.
<String>builder().addAll(list1).addAll(list2).build()
```

Share Improve this answer Follow

answered Nov 24, 2021 at 21:01

lqbweb
1,686 3 19 33

I'm not claiming that it's simple, but you mentioned bonus for one-liners ;-)

```
Collection mergedList = Collections.list(new sun.misc.CompoundEnumeration(new
Enumeration[] {
    new Vector(list1).elements(),
    new Vector(list2).elements(),
    ...
}))
```

Share Improve this answer Follow

edited May 27, 2011 at 17:50

community wiki
2 revs, 2 users 92%
ddimitrov

why should someone never use those? – David Jun 3, 2013 at 16:27

6 @David because it aimed to be used internally in JDK. If you used that in your code, your code will quite probably not running on non-Sun (or non-Oracle now) JDK/JRE. – Adrian Shum Jul 23, 2013 at 7:06

@AdrianShum Are there other JDKs/JREs than Oracle? That would surprise me. Even if limited to the most common API functionality, rebuilding that whole stuff would probably take ages... – Egor Hans Sep 2, 2017 at 10:52

1 There is quite a lot of JVM. Most commonly seen one in enterprise world should be the IBM one which is, iirc, bundled with websphere – Adrian Shum Sep 2, 2017 at 11:41

whenever you see usages of `Vector`, take a 5 minute break. – bvdb Jul 22, 2020 at 15:11

No way near one-liner, but I think this is the simplest:

```
List<String> newList = new ArrayList<String>(l1);
newList.addAll(l2);

for(String w:newList)
        System.out.printf("%s ", w);
```

Share Improve this answer Follow

edited Sep 6, 2011 at 11:06

Daniel Rikowski
71.4k 57 251 329

answered Sep 6, 2011 at 9:14

nirmal
9 1

0

Here's an approach using streams and java 8 if your lists have different types and you want to combine them to a list of another type.

```
public static void main(String[] args) {
    List<String> list2 = new ArrayList<>();
    List<Pair<Integer, String>> list1 = new ArrayList<>();

    list2.add("asd");
    list2.add("asdaf");
    list1.add(new Pair<>(1, "werwe"));
    list1.add(new Pair<>(2, "tyutyu"));

    Stream stream = Stream.concat(list1.stream(), list2.stream());

    List<Pair<Integer, String>> res = (List<Pair<Integer, String>>) stream
            .map(item -> {
                if (item instanceof String) {
                    return new Pair<>(0, item);
                }
                else {
                    return new Pair<>(((Pair<Integer, String>)item).getKey(),
((Pair<Integer, String>)item).getValue());
                }
            })
            .collect(Collectors.toList());
}
```

Share Improve this answer Follow

answered Oct 18, 2016 at 22:28

shinzou
5,850 10 60 124

0

If you want to do this statically you can the following.

The examples uses 2 EnumSets in natural-order (==Enum-order) `A`, `B` and joins then in an `ALL` list.

```
public static final EnumSet<MyType> CATEGORY_A = EnumSet.of(A_1, A_2);
public static final EnumSet<MyType> CATEGORY_B = EnumSet.of(B_1, B_2, B_3);

public static final List<MyType> ALL =
             Collections.unmodifiableList(
                 new ArrayList<MyType>(CATEGORY_A.size() + CATEGORY_B.size())
                 {{
                     addAll(CATEGORY_A);
                     addAll(CATEGORY_B);
                 }}
             );
```

Share Improve this answer Follow

edited Dec 12, 2017 at 13:21

answered Nov 23, 2017 at 14:12

Jan Weitz
444 4 11

This would create a new annonymous class. Not recommended approach! – kravemir Jan 25, 2019 at 10:00

1 2 Next

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.