How can I concatenate two arrays in Java?

Asked 16 years, 3 months ago Modified 11 months ago Viewed 1.4m times

1594

I need to concatenate two `String` arrays in Java.

```
void f(String[] first, String[] second) {
    String[] both = ???
}
```

Which is the easiest way to do this?

java arrays concatenation addition

Share
Improve this question
Follow

edited Apr 2, 2022 at 9:44

community wiki
7 revs, 5 users 70%
Antti Kissaniemi

4 Bytes.concat from Guava – Ben Page Mar 15, 2016 at 11:27

3 I see a lot of responses here but the question is so worded ('easiest way' ?) that it does not allow to indicate the best answer... – Artur Opalinski May 15, 2016 at 3:36

4 Dozens of answers here are copying the data into a new array because that is what was asked for - but copying data when not strictly necessary is a bad thing to do especially in Java. Instead, keep track of the indexes and use the two arrays as if they were joined. I have added a solution illustrating the technique. – Douglas Held Sep 4, 2016 at 23:34

2 The simplest is that you probably shouldn't be using arrays in the first place, you should be using ArrayLists, and your output should be an ArrayList. Once you've made these your pre-condition, the operation is built-in--first.addAll(second). The only case where this wouldn't be pretty much automatic is when your arrays are non-object types (int, long, double, ...), in that case intrinsic arrays can have a big advantage over ArrayLists--but for Strings--meh – Bill K Aug 15, 2017 at 16:27

50 The fact that a question like this currently has 50 different answers makes me wonder why Java never got a simple `array1 + array2` concatenation. – JollyJoker Nov 28, 2017 at 12:53

66 Answers

Sorted by: Highest score (default)

1293

I found a one-line solution from the good old Apache Commons Lang library.
`ArrayUtils.addAll(T[], T...)`

Code:

```
String[] both = ArrayUtils.addAll(first, second);
```

Share
Improve this answer
Follow

edited May 30, 2019 at 13:32

community wiki
8 revs, 4 users 38%
Antti Kissaniemi

208 How is it "cheating" if it answers the question? Sure, having an extra dependency is probably overkill for this specific situation, but no harm is done in calling out that it exists, especially since there's so many excellent bits of functionality in Apache Commons. – Rob Oct 12, 2008 at 15:58

40 I agree, this isn't really answering the question. High level libraries can be great, but if you want to learn an efficient way to do it, you want to look at the code the library method is using. Also, in many situations, you can't just through another library in the product on the fly. – AdamC Jun 18, 2009 at 17:09

88 I think this is a good answer. POJO solutions have also been provided, but if the OP is using Apache Commons in their program already (altogether possible considering its popularity) he may still not know this solution. Then he wouldn't be "adding a dependency for this one method," but would be making better use of an existing library. – Adam Nov 17, 2009 at 15:36

19 If you are always worried about not adding a library for a single method, no new libraries will ever get added. Given the excellent utilities present in Apache Commons, I highly recommend adding it when the very first use case arises. – Hindol Jun 25, 2015 at 8:46

11 using apache commons should never be called 'cheating' i question the sanity of developers who thing its an unnecessary dependency. – Jeryl Cook Jul 5, 2016 at 20:06

791

Here's a simple method that will concatenate two arrays and return the result:

```
public <T> T[] concatenate(T[] a, T[] b) {
    int aLen = a.length;
    int bLen = b.length;

    @SuppressWarnings("unchecked")
    T[] c = (T[]) Array.newInstance(a.getClass().getComponentType(), aLen +
bLen);
    System.arraycopy(a, 0, c, 0, aLen);
    System.arraycopy(b, 0, c, aLen, bLen);

    return c;
}
```

Note that it will not work with primitive data types, only with object types.

The following slightly more complicated version works with both object and primitive arrays. It does this by using `T` instead of `T[]` as the argument type.

It also makes it possible to concatenate arrays of two different types by picking the most general type as the component type of the result.

```
public static <T> T concatenate(T a, T b) {
    if (!a.getClass().isArray() || !b.getClass().isArray()) {
        throw new IllegalArgumentException();
    }

    Class<?> resCompType;
    Class<?> aCompType = a.getClass().getComponentType();
    Class<?> bCompType = b.getClass().getComponentType();

    if (aCompType.isAssignableFrom(bCompType)) {
        resCompType = aCompType;
    } else if (bCompType.isAssignableFrom(aCompType)) {
        resCompType = bCompType;
    } else {
        throw new IllegalArgumentException();
    }

    int aLen = Array.getLength(a);
    int bLen = Array.getLength(b);

    @SuppressWarnings("unchecked")
    T result = (T) Array.newInstance(resCompType, aLen + bLen);
    System.arraycopy(a, 0, result, 0, aLen);
    System.arraycopy(b, 0, result, aLen, bLen);

    return result;
}
```

Here is an example:

```
Assert.assertArrayEquals(new int[] { 1, 2, 3 }, concatenate(new int[] { 1, 2 },
new int[] { 3 }));
Assert.assertArrayEquals(new Number[] { 1, 2, 3f }, concatenate(new Integer[] {
1, 2 }, new Number[] { 3f }));
```

Share

Improve this answer

Follow

edited Dec 6, 2018 at 7:33

community wiki
11 revs, 9 users 41%
Lii

1 I like this suggestion since it is less dependent on the latest Java versions. In my projects I'm often stuck using older versions of Java or CLDC profiles where some of the facilities like those mentioned by Antti are not available. – kvn Feb 7, 2011 at 15:46

4 The following line will break the generic part: concatenate(new String[]{"1"},new Object[] { new Object()}) – dragon66 Jun 3, 2015 at 2:09

would be nice not to have to use the @SuppressWarnings annotation - I'll post a solution for that below. – beaudet Nov 6, 2018 at 22:57

2 ha, call me a purist but I prefer clean code that doesn't require suppression of warnings in order to remove warnings – beaudet Jun 27, 2019 at 21:18

1 There's a difference between semantic warnings and "I'm using raw memory like I would in C, so turn off your safety since I know what I'm doing" "I'm being unsafe" isn't a semantic warning suppression. It's a "I'm going lower level on purpose now, so stop whining." Feel lucky, as Swift 5 removed suppressors. Java, unlike Swift, is still made for programmers who aren't afraid of doing their own programming – Stephen J Feb 5, 2020 at 20:35

598

Using `Stream` in Java 8:

```
String[] both = Stream.concat(Arrays.stream(a), Arrays.stream(b))
                      .toArray(String[]::new);
```

Or like this, using `flatMap`:

```
String[] both = Stream.of(a, b).flatMap(Stream::of)
                      .toArray(String[]::new);
```

To do this for a generic type you have to use reflection:

```
@SuppressWarnings("unchecked")
T[] both = Stream.concat(Arrays.stream(a), Arrays.stream(b)).toArray(
    size -> (T[]) Array.newInstance(a.getClass().getComponentType(), size));
```

Share
Improve this answer
Follow

edited Dec 17, 2019 at 9:23

community wiki
5 revs, 5 users 35%
Lii

46 How efficient is this? – Ky - Oct 2, 2015 at 1:34

12 Worth a read: jaxenter.com/… tl;dr - streams could be performant or not, it depends on what you're doing with them and the constraints of the problem (isn't this always the answer? lol) – Trevor Brown Feb 25, 2016 at 17:29

6 Additionally, if *a* or *b* are arrays of primitive types, their streams will need to be `.boxed()` so they are of type `Stream` rather than e.g. `IntStream` which cannot be passed as a parameter to `Stream.concat`. – Will Hardwick-Smith Jul 16, 2016 at 9:59

31 @Will Hardwick-Smith: no, you only have to pick the right stream class, e.g. if `a` and `b` are `int[]`, use `int[] both = IntStream.concat(Arrays.stream(a), Arrays.stream(b)).toArray();` – Holger Jun 13, 2017 at 12:28

7 @Supuhstar: It is probably not as fast as `System.arrayCopy`. But not particularly slow either. You probably have to do this a *very* many times with *huge* arrays in *really* performance sensitive contexts for the execution time difference to matter. – Lii Dec 23, 2017 at 15:43

501

It's possible to write a fully generic version that can even be extended to concatenate any number of arrays. This versions require Java 6, as they use `Arrays.copyOf()`

Both versions avoid creating any intermediary `List` objects and use `System.arraycopy()` to ensure that copying large arrays is as fast as possible.

For two arrays it looks like this:

```
public static <T> T[] concat(T[] first, T[] second) {
  T[] result = Arrays.copyOf(first, first.length + second.length);
  System.arraycopy(second, 0, result, first.length, second.length);
  return result;
}
```

And for a arbitrary number of arrays (>= 1) it looks like this:

```
public static <T> T[] concatAll(T[] first, T[]... rest) {
  int totalLength = first.length;
  for (T[] array : rest) {
    totalLength += array.length;
  }
  T[] result = Arrays.copyOf(first, totalLength);
  int offset = first.length;
  for (T[] array : rest) {
    System.arraycopy(array, 0, result, offset, array.length);
    offset += array.length;
  }
  return result;
}
```

Share

Improve this answer

Follow

answered Apr 24, 2009 at 7:28

community wiki
Joachim Sauer

11 @djBO: for primitive-typed arrays you'd need to make an overload for each type: just copy the code and replace each `T` with `byte` (and lose the `<T>`). – Joachim Sauer Jun 1, 2011 at 6:01

can you plese tell me how to use <T> operator type in my class? – Johnydep Jun 27, 2011 at 2:22

6 I'd add this to the beginning, just to be defensive. if (first == null) { if (second == null) { return null; } return second; } if (second == null) { return first; } – Paul C Sep 22, 2011 at 4:18

5 @djBo: what about: `ByteBuffer buffer = ByteBuffer.allocate(array1.length + array2.length); buffer.put(array1); buffer.put(array2); return buffer.array();` – Sam Goldberg Dec 2, 2011 at 15:29

25 There's a bug in this approach which becomes apparent if you invoke these functions with arrays of different component types, for example `concat(ai, ad)`, where `ai` is `Integer[]` and `ad` is `Double[]`. (In this case, the type parameter `<T>` is resolved to `<? extends Number>` by the compiler.) The array created by `Arrays.copyOf` will have the component type of the first array, i.e. `Integer` in this example. When the function is about to copy the second array, an `ArrayStoreException` will be thrown. The solution is to have an additional `Class<T> type` parameter. – T-Bull Jul 25, 2013 at 17:16

207

Or with the beloved Guava:

```
String[] both = ObjectArrays.concat(first, second, String.class);
```

Also, there are versions for primitive arrays:

- `Booleans.concat(first, second)`
- `Bytes.concat(first, second)`
- `Chars.concat(first, second)`
- `Doubles.concat(first, second)`
- `Shorts.concat(first, second)`
- `Ints.concat(first, second)`
- `Longs.concat(first, second)`
- `Floats.concat(first, second)`

Share

Improve this answer

Follow

edited Mar 27, 2017 at 9:55

community wiki
5 revs, 3 users 41%
KARASZI István

As much as I love Guava, the method from Apache Commons deals better with nullables. – Ravi Wallau Nov 1, 2013 at 19:19

7 While it is good to use libraries, it's unfortunate that the problem has been abstracted away. Therefore the underlying solution remains elusive. – KRK Owner Apr 9, 2014 at 0:15

55 Whats the problem with abstraction? Dunno what's the deal with reinventing the wheel here, if you want to learn the problem the check the source or read on it. Professional code should be using high-level libraries, much better if it's developed inside Google! – Breno Salgado Jul 15, 2014 at 20:11

1 @SébastienTromp It is the top solution for this question - ArrayUtils. – Ravi Wallau Jan 13, 2018 at 4:45

3 I don't advise to add Guava into a project only to concatenate arrays, it's big and it has a serious lack of modularity. @BrenoSalgado "much better if it's developed inside Google" I strongly disagree with that. – gouessej Aug 5, 2020 at 12:17

188

You can append the two arrays in two lines of code.

```
String[] both = Arrays.copyOf(first, first.length + second.length);
System.arraycopy(second, 0, both, first.length, second.length);
```

This is a fast and efficient solution and will work for primitive types as well as the two methods involved are overloaded.

You should avoid solutions involving ArrayLists, streams, etc as these will need to allocate temporary memory for no useful purpose.

You should avoid `for` loops for large arrays as these are not efficient. The built in methods use block-copy functions that are extremely fast.

Share

Improve this answer

Follow

edited Jan 17, 2019 at 11:03

community wiki
3 revs, 2 users 90%
rghome

17 This is one of the best solutions. 100% standard Java. Fast/efficient. Should get more upvotes! – Shebla Tsama Feb 27, 2020 at 2:25

It may be efficient, but is a bit of a mess compared to what you might expect: `both = array1 + array2` – Carson Holzheimer Sep 8, 2022 at 3:48

61

Using the Java API:

```
String[] f(String[] first, String[] second) {
    List<String> both = new ArrayList<String>(first.length + second.length);
    Collections.addAll(both, first);
    Collections.addAll(both, second);
    return both.toArray(new String[both.size()]);
}
```

Share

Improve this answer

Follow

edited Apr 21, 2012 at 16:34

community wiki
3 revs
Fabian Steeg

17 Simply, but inefficient as it make an array for ArrayList and then generate another for toArray method. But still valid as it's simple to read. – PhoneixS Mar 10, 2014 at 12:16

1 applicable for Strings and objects (as question wants), but there is no addAll method for primary types (as ints) – JRr Mar 23, 2018 at 7:20

1 As elaborated in this article, using `both.toArray(new String[0])` will be faster than `both.toArray(new String[both.size()])`, even if it contradicts our naive intuition. That's why it is so important to measure the actual performance when optimizing. Or just use the simpler construct, when the advantage of the more complicated variant can't be proven. – Holger Feb 25, 2019 at 15:19

45

A solution **100% old java** and **without** `System.arraycopy` (not available in GWT client for example):

```
static String[] concat(String[]... arrays) {
    int length = 0;
    for (String[] array : arrays) {
        length += array.length;
    }
    String[] result = new String[length];
    int pos = 0;
    for (String[] array : arrays) {
        for (String element : array) {
            result[pos] = element;
            pos++;
        }
    }
    return result;
}
```

Share
Improve this answer
Follow

edited Feb 26, 2016 at 8:06

community wiki
4 revs, 3 users 92%
francois

reworked mine for File[], but it's the same. Thanks for your solution – ShadowFlame Sep 7, 2012 at 10:49

5 Probably quite inefficient though. – Jonas Czech Apr 11, 2015 at 20:34

1 You might want to add `null` checks. And perhaps set some of your variables to `final`. – Tripp Kinetics Sep 24, 2015 at 14:31

1 @TrippKinetics `null` checks would hide NPE's rather than showing them and using final for local vars doesn't have any benefit (yet). – Maarten Bodewes Mar 24, 2019 at 13:22

3 @Maarten Bodewes I think you will find (if you benchmark it, which I have) that the for-each runs in the same time as the indexed loop on late versions of Java. The optimizer takes care of it. – rghome Mar 28, 2019 at 7:46

I've recently fought problems with excessive memory rotation. If a and/or b are known to be commonly empty, here is another adaption of silvertab's code (generified too):

35

```
private static <T> T[] concatOrReturnSame(T[] a, T[] b) {
    final int alen = a.length;
    final int blen = b.length;
    if (alen == 0) {
        return b;
    }
    if (blen == 0) {
        return a;
    }
    final T[] result = (T[]) java.lang.reflect.Array.
            newInstance(a.getClass().getComponentType(), alen + blen);
    System.arraycopy(a, 0, result, 0, alen);
    System.arraycopy(b, 0, result, alen, blen);
    return result;
}
```

Edit: A previous version of this post stated that array re-usage like this shall be clearly documented. As Maarten points out in the comments it would in general be better to just remove the if statements, thus voiding the need for having documentation. But then again, those if statements were the whole point of this particular optimization in the first place. I'll leave this answer here, but be wary!

Share

Improve this answer

Follow

edited Mar 20, 2019 at 20:45

community wiki
4 revs
volley

5 this however means that you are returning the same array and changing a value on the returned array changes the value in the same position of the input array returned. – Lorenzo Boccaccia Nov 26, 2008 at 17:55

Yes - see comment at the end of my post regarding array re-usage. The maintenance overhead imposed by this solution was worth it in our particular case, but defensive copying should probably be used in most cases. – volley Mar 17, 2009 at 14:43

Lorenzo / volley, can you explain which part in the code that cause array re-usage? I thought `System.arraycopy` copies the content of the array? – Rosdi Kasim May 13, 2010 at 2:35

4 A caller would normally expect a call to concat() to return a newly allocated array. If either a or b is null, concat() will however return one of the arrays passed into it. This re-usage is what may be unexpected. (Yep, arraycopy only does copying. The re-usage comes from returning either a or b directly.) – volley May 14, 2010 at 6:57

Code should be as much self explanatory as possible. People reading the code should not have to lookup the JavaDoc of a called function to find out that it does one thing for one particular condition and something else for another. In short: you can generally not fix design problems like these with a comment. Just leaving the two `if` statements out would be the easiest fix. – Maarten Bodewes Mar 18, 2019 at 15:25

```
ArrayList<String> both = new ArrayList(Arrays.asList(first));
both.addAll(Arrays.asList(second));
```

28

```
both.toArray(new String[0]);
```

Share
Improve this answer
Follow

edited Dec 21, 2017 at 3:37

community wiki
2 revs
nick-s

3 The answer is great but a tiny bit broken. To make it perfect you should pass to toArray() an array of the type you need. In the above example, the code should be: both.toArray(new String[0]) See: stackoverflow.com/questions/4042434/… – Ronen Rabinovici Feb 15, 2017 at 20:12

Don't know why this answer isn't rated higher... though it does seem to need the change suggested by @RonenRabinovici – drmrbrewer Dec 2, 2017 at 11:14

4 Or better, without unnecessary allocation of zero-length array: `both.toArray(new String[both.size()])` ;) – Tharok Jan 10, 2018 at 16:42

2 @Honza recommended read – Holger Feb 25, 2019 at 15:20

Hi @Honza, possible to do the same to return a primitive integer array in 3 lines? – jumping_monkey Feb 10, 2020 at 0:40

27

The Functional Java library has an array wrapper class that equips arrays with handy methods like concatenation.

```
import static fj.data.Array.array;
```

...and then

```
Array<String> both = array(first).append(array(second));
```

To get the unwrapped array back out, call

```
String[] s = both.array();
```

Share
Improve this answer
Follow

edited Jul 15, 2009 at 13:25

community wiki
2 revs
Apocalisp

20

Another way with Java8 using Stream

```
  public String[] concatString(String[] a, String[] b){
    Stream<String> streamA = Arrays.stream(a);
    Stream<String> streamB = Arrays.stream(b);
    return Stream.concat(streamA, streamB).toArray(String[]::new);
  }
```

Share

Improve this answer

Follow

answered Feb 10, 2016 at 12:48

community wiki
Vaseph

17

Here's an adaptation of silvertab's solution, with generics retrofitted:

```
static <T> T[] concat(T[] a, T[] b) {
    final int alen = a.length;
    final int blen = b.length;
    final T[] result = (T[]) java.lang.reflect.Array.
            newInstance(a.getClass().getComponentType(), alen + blen);
    System.arraycopy(a, 0, result, 0, alen);
    System.arraycopy(b, 0, result, alen, blen);
    return result;
}
```

NOTE: See Joachim's answer for a Java 6 solution. Not only does it eliminate the warning; it's also shorter, more efficient and easier to read!

Share

Improve this answer

Follow

edited May 23, 2017 at 12:02

community wiki
5 revs
volley

You can suppress the warning for this method, but other than that there isn't much you can do. Arrays and generics don't really mix. – Dan Dyer Sep 25, 2008 at 19:43

4 The unchecked warning can be eliminated if you use Arrays.copyOf(). See my answer for an implementation. – Joachim Sauer Apr 24, 2009 at 7:30

@SuppressWarnings("unchecked") – Mark Renouf Apr 26, 2009 at 19:21

16

You could try converting it into a `ArrayList` and use the `addAll` method then convert back to an array.

```
List list = new ArrayList(Arrays.asList(first));
  list.addAll(Arrays.asList(second));
  String[] both = list.toArray();
```

Share

Improve this answer

Follow

edited Mar 27, 2022 at 14:42

community wiki
2 revs, 2 users 89%
Paul

Good solution--would be better if the code was refactored to avoid arrays altogether in favor of ArrayLists, but that's outside the control of the "Answer" and up to the questioner. – Bill K Aug 15, 2017 at 16:29

I count that it requires 4 additional temporary objects to work. – rghome Jul 25, 2018 at 7:31

3 @rghome, at least it doesn't require additional library to implement such simple task – Farid Feb 3, 2020 at 12:33

13

If you use this way so you no need to import any third party class.

If you want concatenate `String`

Sample code for concate two String Array

```
public static String[] combineString(String[] first, String[] second){
        int length = first.length + second.length;
        String[] result = new String[length];
        System.arraycopy(first, 0, result, 0, first.length);
        System.arraycopy(second, 0, result, first.length, second.length);
        return result;
    }
```

If you want concatenate `Int`

Sample code for concate two Integer Array

```
public static int[] combineInt(int[] a, int[] b){
        int length = a.length + b.length;
        int[] result = new int[length];
        System.arraycopy(a, 0, result, 0, a.length);
        System.arraycopy(b, 0, result, a.length, b.length);
        return result;
    }
```

Here is Main method

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

            String [] first = {"a", "b", "c"};
            String [] second = {"d", "e"};

            String [] joined = combineString(first, second);
            System.out.println("concatenated String array : " +
Arrays.toString(joined));

            int[] array1 = {101,102,103,104};
            int[] array2 = {105,106,107,108};
            int[] concatenateInt = combineInt(array1, array2);

            System.out.println("concatenated Int array : " +
Arrays.toString(concatenateInt));

        }
    }
```

We can use this way also.

Share

Improve this answer

Follow

answered Feb 13, 2017 at 10:17

community wiki
Raj Rusia

12

Please forgive me for adding yet another version to this already long list. I looked at every answer and decided that I really wanted a version with just one parameter in the signature. I also added some argument checking to benefit from early failure with sensible info in case of unexpected input.

```
@SuppressWarnings("unchecked")
public static <T> T[] concat(T[]... inputArrays) {
  if(inputArrays.length < 2) {
    throw new IllegalArgumentException("inputArrays must contain at least 2
arrays");
  }

  for(int i = 0; i < inputArrays.length; i++) {
    if(inputArrays[i] == null) {
      throw new IllegalArgumentException("inputArrays[" + i + "] is null");
    }
  }

  int totalLength = 0;

  for(T[] array : inputArrays) {
    totalLength += array.length;
  }

  T[] result = (T[])
Array.newInstance(inputArrays[0].getClass().getComponentType(), totalLength);

  int offset = 0;

  for(T[] array : inputArrays) {
    System.arraycopy(array, 0, result, offset, array.length);

    offset += array.length;
  }

  return result;
}
```

Share

Improve this answer

Follow

edited Sep 11, 2015 at 14:02

community wiki
3 revs, 2 users 86%
Zalumon

I'd sum up the length in the same loop where you are doing your null check--but this is a really good summary of the other answers here. I believe it even handles intrinsic types like "int" without changing them to Integer objects which is really the ONLY reason to deal with them as arrays rather than just changing everything to ArrayLists. Also your method could take 2 arrays and a (...) parameter so the caller knows he needs to pass in at least two arrays before he runs it and sees the error, but that complicates the looping code.... – Bill K Aug 15, 2017 at 16:34

11

Using Java 8+ streams you can write the following function:

```
private static String[] concatArrays(final String[]... arrays) {
    return Arrays.stream(arrays)
         .flatMap(Arrays::stream)
         .toArray(String[]::new);
}
```

Share

Improve this answer

Follow

answered Oct 23, 2018 at 13:29

community wiki
keisar

9

This should be one-liner.

```
public String [] concatenate (final String array1[], final String array2[])
{
    return Stream.concat(Stream.of(array1),
Stream.of(array2)).toArray(String[]::new);
}
```

Share

Improve this answer

Follow

answered Oct 27, 2018 at 18:52

community wiki
avigaild

7

Here a possible implementation in working code of the pseudo code solution written by silvertab.

Thanks silvertab!

```
public class Array {

   public static <T> T[] concat(T[] a, T[] b, ArrayBuilderI<T> builder) {
      T[] c = builder.build(a.length + b.length);
      System.arraycopy(a, 0, c, 0, a.length);
      System.arraycopy(b, 0, c, a.length, b.length);
      return c;
```

```
    }
}
```

Following next is the builder interface.

Note: A builder is necessary because in java it is not possible to do

`new T[size]`

due to generic type erasure:

```
public interface ArrayBuilderI<T> {

   public T[] build(int size);
}
```

Here a concrete builder implementing the interface, building a `Integer` array:

```
public class IntegerArrayBuilder implements ArrayBuilderI<Integer> {

   @Override
   public Integer[] build(int size) {
      return new Integer[size];
   }
}
```

And finally the application / test:

```
@Test
public class ArrayTest {

   public void array_concatenation() {
      Integer a[] = new Integer[]{0,1};
      Integer b[] = new Integer[]{2,3};
      Integer c[] = Array.concat(a, b, new IntegerArrayBuilder());
      assertEquals(4, c.length);
      assertEquals(0, (int)c[0]);
      assertEquals(1, (int)c[1]);
      assertEquals(2, (int)c[2]);
      assertEquals(3, (int)c[3]);
   }
}
```

Share

Improve this answer

Follow

answered Jan 4, 2012 at 14:35

community wiki
hpgisler

This works, but you need to insert your own error checking.

7

```
public class StringConcatenate {

    public static void main(String[] args){

        // Create two arrays to concatenate and one array to hold both
        String[] arr1 = new String[]{"s","t","r","i","n","g"};
        String[] arr2 = new String[]{"s","t","r","i","n","g"};
        String[] arrBoth = new String[arr1.length+arr2.length];

        // Copy elements from first array into first part of new array
        for(int i = 0; i < arr1.length; i++){
            arrBoth[i] = arr1[i];
        }

        // Copy elements from second array into last part of new array
        for(int j = arr1.length;j < arrBoth.length;j++){
            arrBoth[j] = arr2[j-arr1.length];
        }

        // Print result
        for(int k = 0; k < arrBoth.length; k++){
            System.out.print(arrBoth[k]);
        }

        // Additional line to make your terminal look better at completion!
        System.out.println();
    }
}
```

It's probably not the most efficient, but it doesn't rely on anything other than Java's own API.

Share

Improve this answer

Follow

edited May 10, 2015 at 19:42

community wiki
2 revs, 2 users 72%
glue

2 +1. It would be better to replace the second `for` loop with that: `for(int j = 0; j < arr2.length; j++){arrBoth[arr1.length+j] = arr2[j];}` – bancer Oct 28, 2010 at 21:23

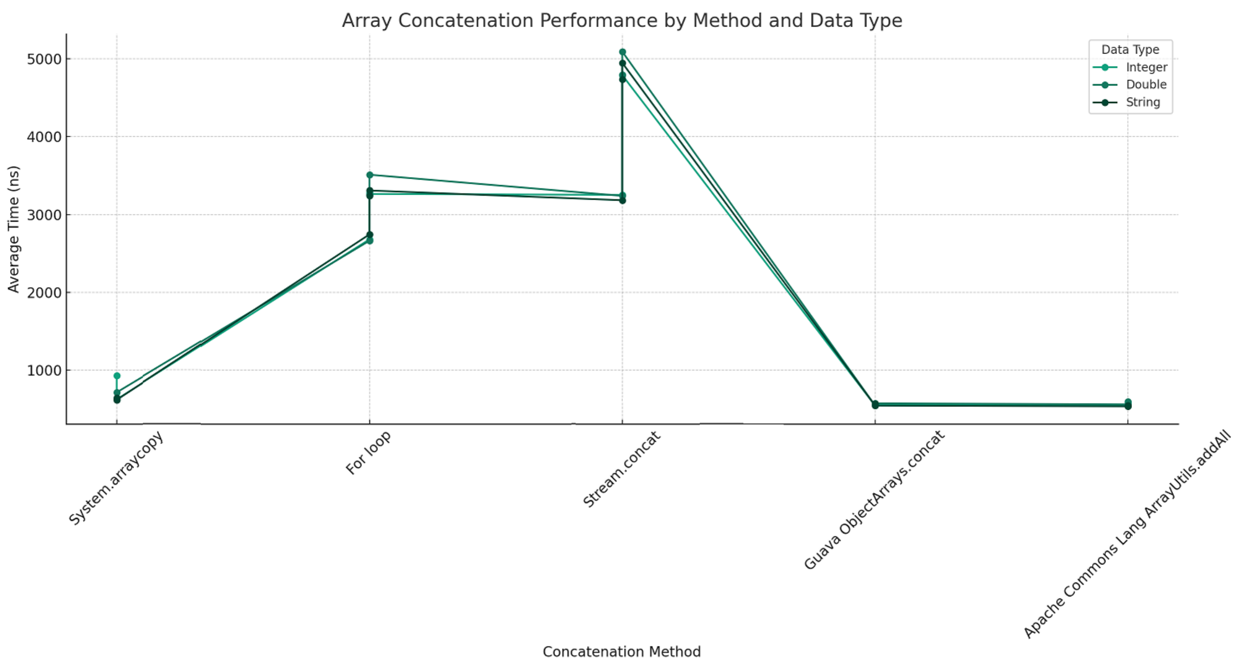
Use `String[] arrBoth = java.util.Arrays.copyOf(arr1, arr1.length + arr2.length)` to skip the first `for` loop. Saves time proportional to size of `arr1`. – John Dec 27, 2018 at 19:20

A generic static version that uses the high performing System.arraycopy without requiring a @SuppressWarnings annotation:

7

```
public static <T> T[] arrayConcat(T[] a, T[] b) {
    T[] both = Arrays.copyOf(a, a.length + b.length);
    System.arraycopy(b, 0, both, a.length, b.length);
    return both;
}
```

I ran some performance tests using the different solutions suggested in the responses here.



One million iterations were performed and the average taken for the values in the table below. Tests were run on Ubuntu running on top of WSL 2. Fill Type refers to the nature of the data contained in the two 1000 element arrays that were concatenated together.

Data Type	Fill Type	System.arraycopy	For loop	Stream.concat	Guava Objec
Integer	EMPTY	928 ns	2680 ns	3252 ns	547 ns
Integer	SAME_VALUE	632 ns	3282 ns	4787 ns	545 ns
Integer	RANDOM	625 ns	3264 ns	4790 ns	557 ns
Double	EMPTY	616 ns	2663 ns	3239 ns	548 ns
Double	SAME_VALUE	632 ns	3259 ns	4793 ns	559 ns
Double	RANDOM	715 ns	3511 ns	5092 ns	575 ns
String	EMPTY	629 ns	2744 ns	3183 ns	547 ns
String	SAME_VALUE	642 ns	3242 ns	4735 ns	567 ns
String	RANDOM	619 ns	3308 ns	4948 ns	545 ns

It appears that the Google and Apache codes are quite close with System.arraycopy based approaches coming in 3rd. The *for loop* and Streamconcat approaches were far behind the other three.

Share

Improve this answer

Follow

edited Jan 19 at 4:39

community wiki
2 revs
beaudet

6

Wow! lot of complex answers here including some simple ones that depend on external dependencies. how about doing it like this:

```
String [] arg1 = new String{"a","b","c"};
String [] arg2 = new String{"x","y","z"};

ArrayList<String> temp = new ArrayList<String>();
temp.addAll(Arrays.asList(arg1));
temp.addAll(Arrays.asList(arg2));
String [] concatedArgs = temp.toArray(new String[arg1.length+arg2.length]);
```

Share

Improve this answer

Follow

answered Oct 28, 2012 at 20:23

community wiki
doles

1 ..But inefficient and slow. – Jonas Czech Apr 11, 2015 at 20:36

5

This is a converted function for a String array:

```
public String[] mergeArrays(String[] mainArray, String[] addArray) {
    String[] finalArray = new String[mainArray.length + addArray.length];
    System.arraycopy(mainArray, 0, finalArray, 0, mainArray.length);
    System.arraycopy(addArray, 0, finalArray, mainArray.length,
addArray.length);

    return finalArray;
}
```

Share

Improve this answer

Follow

answered Jun 11, 2011 at 19:59

community wiki
Oritm

5

How about simply

```
public static class Array {

    public static <T> T[] concat(T[]... arrays) {
        ArrayList<T> al = new ArrayList<T>();
        for (T[] one : arrays)
```

```
            Collections.addAll(al, one);
        return (T[]) al.toArray(arrays[0].clone());
    }
}
```

And just do `Array.concat(arr1, arr2)`. As long as `arr1` and `arr2` are of the same type, this will give you another array of the same type containing both arrays.

Share

Improve this answer

Follow

edited May 10, 2015 at 19:30

community wiki
3 revs, 2 users 81%
Ephraim

1 For performance reasons, I would pre-compute the ArrayList's final size because ArrayList, by definition, allocates a new array and copies its elements every time the current array is full. Otherwise I would go straight for LinkedList which does not suffer such problem
– usr-local-ΕΨΗΕΛΩΝ Aug 20, 2015 at 8:07

5

A simple variation allowing the joining of more than one array:

```
public static String[] join(String[]...arrays) {

    final List<String> output = new ArrayList<String>();

    for(String[] array : arrays) {
        output.addAll(Arrays.asList(array));
    }

    return output.toArray(new String[output.size()]);
}
```

Share

Improve this answer

Follow

edited May 10, 2015 at 19:38

community wiki
2 revs, 2 users 78%
Damo

4

```
public String[] concat(String[]... arrays)
{
    int length = 0;
    for (String[] array : arrays) {
        length += array.length;
    }
    String[] result = new String[length];
    int destPos = 0;
    for (String[] array : arrays) {
        System.arraycopy(array, 0, result, destPos, array.length);
        destPos += array.length;
    }
    return result;
}
```

Share

Improve this answer

Follow

answered Dec 29, 2010 at 7:29

community wiki
Sujay

4

Here's my slightly improved version of Joachim Sauer's concatAll. It can work on Java 5 or 6, using Java 6's System.arraycopy if it's available at runtime. This method (IMHO) is perfect for Android, as it work on Android <9 (which doesn't have System.arraycopy) but will use the faster method if possible.

```
  public static <T> T[] concatAll(T[] first, T[]... rest) {
    int totalLength = first.length;
    for (T[] array : rest) {
      totalLength += array.length;
    }
    T[] result;
    try {
      Method arraysCopyOf = Arrays.class.getMethod("copyOf", Object[].class, 
int.class);
      result = (T[]) arraysCopyOf.invoke(null, first, totalLength);
    } catch (Exception e){
      //Java 6 / Android >= 9 way didn't work, so use the "traditional" 
approach
      result = (T[]) 
java.lang.reflect.Array.newInstance(first.getClass().getComponentType(), 
totalLength);
      System.arraycopy(first, 0, result, 0, first.length);
    }
    int offset = first.length;
    for (T[] array : rest) {
      System.arraycopy(array, 0, result, offset, array.length);
      offset += array.length;
    }
    return result;
  }
```

Share

Improve this answer

Follow

answered Jun 9, 2011 at 15:48

community wiki
candrews

1 Good general idea, but to anyone implementing: I'd prefer a copyOf and non-copyOf versions than one that does both by way of reflection. – rektide Sep 12, 2011 at 14:36

4

Another way to think about the question. To concatenate two or more arrays, one have to do is to list all elements of each arrays, and then build a new array. This sounds like create a `List<T>` and then calls `toArray` on it. Some other answers uses `ArrayList`, and that's fine. But how about implement our own? It is not hard:

```java
private static <T> T[] addAll(final T[] f, final T...o){
    return new AbstractList<T>(){

        @Override
        public T get(int i) {
            return i>=f.length ? o[i - f.length] : f[i];
        }

        @Override
        public int size() {
            return f.length + o.length;
        }

    }.toArray(f);
}
```

I believe the above is equivalent to solutions that uses `System.arraycopy`. However I think this one has its own beauty.

Share
Improve this answer
Follow

answered Feb 27, 2013 at 3:03

community wiki
Earth Engine

4

How about :

```java
public String[] combineArray (String[] ... strings) {
    List<String> tmpList = new ArrayList<String>();
    for (int i = 0; i < strings.length; i++)
        tmpList.addAll(Arrays.asList(strings[i]));
    return tmpList.toArray(new String[tmpList.size()]);
}
```

Share
Improve this answer
Follow

answered Feb 11, 2015 at 23:35

community wiki
clément francomme

4

This is probably the only generic and type-safe way:

```java
public class ArrayConcatenator<T> {
    private final IntFunction<T[]> generator;

    private ArrayConcatenator(IntFunction<T[]> generator) {
        this.generator = generator;
    }

    public static <T> ArrayConcatenator<T> concat(IntFunction<T[]> generator) {
        return new ArrayConcatenator<>(generator);
    }

    public T[] apply(T[] array1, T[] array2) {
        T[] array = generator.apply(array1.length + array2.length);
        System.arraycopy(array1, 0, array, 0, array1.length);
        System.arraycopy(array2, 0, array, array1.length, array2.length);
        return array;
    }
}
```

And the usage is quite concise:

```
Integer[] array1 = { 1, 2, 3 };
Double[] array2 = { 4.0, 5.0, 6.0 };
Number[] array = concat(Number[]::new).apply(array1, array2);
```

(requires static import)

Invalid array types are rejected:

```
concat(String[]::new).apply(array1, array2); // error
concat(Integer[]::new).apply(array1, array2); // error
```

Share
Improve this answer
Follow

answered Mar 24, 2019 at 4:22

community wiki
ZhekaKozlov