<http://www.brilliantsheep.com/3-ways-to-reverse-a-string-in-java/>

3[**-Ways to Reverse a String in Java**](http://www.brilliantsheep.com/3-ways-to-reverse-a-string-in-java/)

Reversing a String in Java is a task that I came across in several interview experiences that I've read about. Although there are several ways to do this task, it seems that [potential] employers are especially interested in the solution involving recursion. Overall, (in case you're wondering) this task is used to get a better assessment on your understanding of the programming flow. Having said that, I will list in detail 3-ways to solve this task, from which you can pick and choose:  
  
  
**First Solution: StringBuffer**  
The easiest way to reverse a String in Java is by using an instance of the [StringBuffer](http://download.oracle.com/javase/6/docs/api/java/lang/StringBuffer.html" \t "_blank) class as it already contains a[reverse()](http://download.oracle.com/javase/6/docs/api/java/lang/StringBuffer.html#reverse%28%29) method. Thus, with this approach our reverse method will look like:

|  |  |
| --- | --- |
| 1  2  3 | **public** String reverse(String s) {  **return** **new** StringBuffer(s).reverse().toString();  } |

However, using this solution in an interview will only show how much you're familiar with the Java API which is not necessary the point of the task.  
  
  
**Second Solution: Reverse For Loop**  
You can also reverse a String by traversing it from the end in a traditional for loop. For this approach you can either use a char array which would be somewhat more efficient than by creating a large String pool as a result of continuous concatenation to a String variable. To better elaborate my point, here are both versions of this solution:

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | ***/\*\****  ***\* Reverse For Loop: Char Array***  ***\*/***  **public** String reverse(String s) {  **char**[] reverseStringArray = **new** **char**[s.length()];  **for** (**int** i = s.length() - 1, j = 0; i != -1; i--, j++) {  reverseStringArray[j] = s.charAt(i);  }  **return** **new** String(reverseStringArray);  } | |
| 1  2  3  4  5  6  7  8  9  10 | | ***/\*\****  ***\* Reverse For Loop: String Variable***  ***\*/***  **public** String reverse(String s) {  String reverseStringVariable = "";  **for** (**int** i = s.length() - 1; i != -1; i--) {  reverseStringVariable += s.charAt(i);  }  **return** reverseStringVariable;  } |

**Third Solution: Recursion**  
Finally, here's the solution that most likely your [potential] employer will like to see:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **public** String reverse(String s) {  **if** (s.length() <= 1) {  **return** s;  }  **return** reverse(s.substring(1, s.length())) + s.charAt(0);  } |

To better understand this recursive approach, let's trace for example a call to the reverse(String) method with the String "ABC" as an argument:

|  |
| --- |
| Step 1: "ABC" does NOT have a length equal to or less than 1.  Step 2: Call reverse(String) with "BC" as argument and concatenate to its return value "A".  Step 3: "BC" does NOT have a length equal to or less than 1.  Step 4: Call reverse(String) with "C" as argument and concatenate to its return value "B".  Step 5: "C" has a length of 1. Return "C" and unroll the stack. |

Now, as the stack unrolls, the following concatenation occurs: **"C" + "B" + "A"** thus we end up getting the String **"CBA"**, which is the exact reverse :)

Needless to say, I hope that you get a chance to use these solutions, and I look forward to hear about any different approaches that you've taken for solving this task.

P.S. You might also find the following post relevant regarding string reversal: [In-place String Reversal in Java](http://www.brilliantsheep.com/in-place-string-reversal-in-java/).

<http://www.brilliantsheep.com/in-place-string-reversal-in-java/>

In-[**place String Reversal in Java**](http://www.brilliantsheep.com/in-place-string-reversal-in-java/)

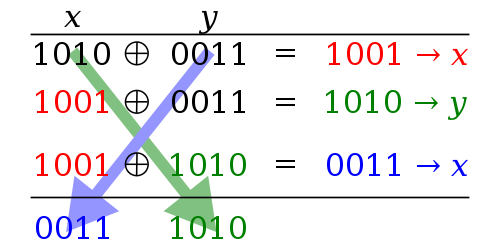
String reversal is one of the more typical assignments that is put forth on technical interviews. If designed and implemented properly, the solution should be short and easy to follow. Furthermore, by using this assignment the interviewers would be able to assess several aspects of candidate's technical skill set, such as:  
    • Has there been put any thought on the space-complexity (i.e. memory use) of the solution?  
    • What is the performance of the solution, i.e. does it run on O(n^2) complexity? Can it run on O(n) complexity?  
    • What is the candidate's approach to problem solving? etc.  
I've already included some solutions to the string reverse assignment in [3-Ways to Reverse a String in Java](http://www.brilliantsheep.com/3-ways-to-reverse-a-string-in-java/); however, what I will be covering in this post is the Java implementation of an in-place string reversal. The proposed solution assumes that we have an array representation of the string that we want to reverse; as so:

|  |
| --- |
| **public** String reverse( String string ) {    **byte**[] array = string.getBytes();  **byte** swap;    **for**( **int** i = 0, j = array.length - 1; i < array.length / 2; i++, j-- ) {  swap = array[ j ];  array[ j ] = array[ i ];  array[ i ] = swap;  }    **return** **new** String( array );  } |

As it can be seen, we use the i variable to traverse the string array from the left (i.e. beginning), and the j variable to traverse the string array from the right (i.e. end). In addition, we use a single byte variable for swapping the string characters. Because the swapping is performed up to the middle of the string, we use the array.length / 2 as an indicator for loop termination. However, it should be noted that we can also perform an in-place string reversal without using a swap variable. This solution would require the utilization of the boolean operator [XOR](http://en.wikipedia.org/wiki/XOR) (i.e. Exclusive OR), and would be implemented in the following manner:

|  |
| --- |
| **public** String reverse( String string ) {    **byte**[] array = string.getBytes();    **for**( **int** i = 0, j = array.length - 1; i < array.length / 2; i++, j-- ) {    array[ i ] ^= array[ j ];  array[ j ] ^= array[ i ];  array[ i ] ^= array[ j ];  }    **return** **new** String( array );  } |

The in-place string reversal with the XOR operator although it has a lower space-complexity, it is more cryptic to follow. That said, in order to better understand how the XOR swap algorithm works, please review the following graphical representation:



XOR Swap Algorithm

I hope that you've found the discussed solutions helpful, and I look forward to any feedback you might have :)