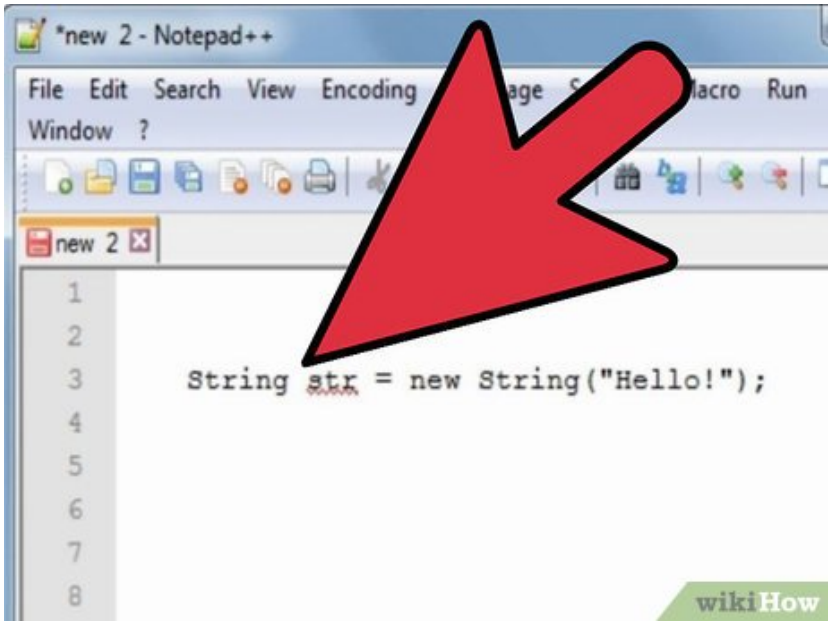
How to Manipulate Strings in Java

Strings are sequences of characters. For example, 'Hello!' is a string because it is made up of the characters 'H', 'e', 'l', 'l', 'o', and '!'. In Java, strings are objects, which means that there is a String class that has fields and...

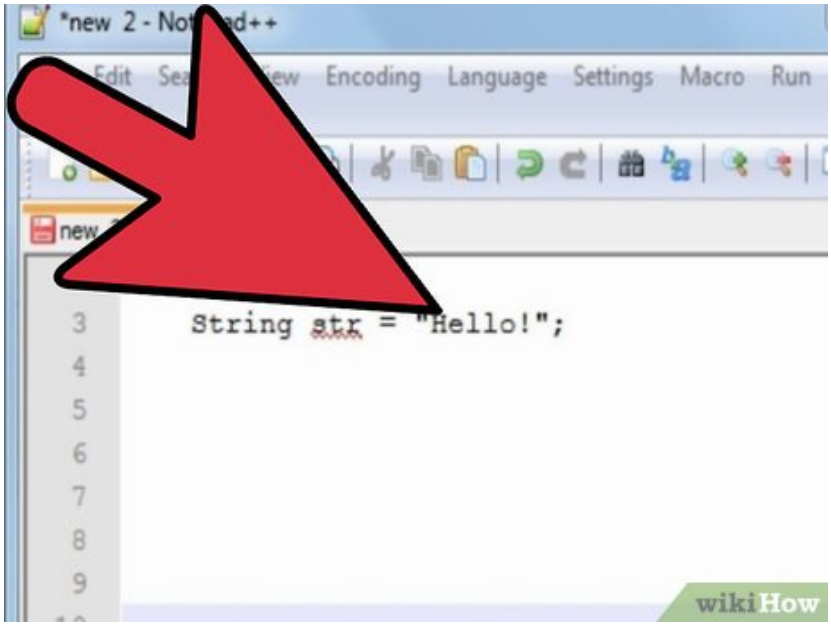
Method 1 of 5:

Create a String

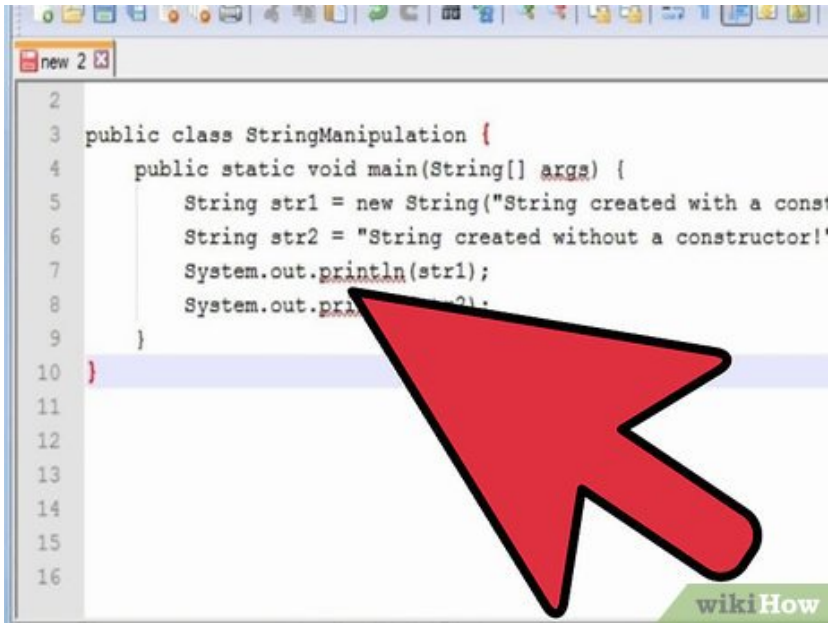
1. 

Create a string using the constructor of the String class.

```
String str = new String("Hello!");
```

2. 

Create a string by directly assigning a string.

3. 

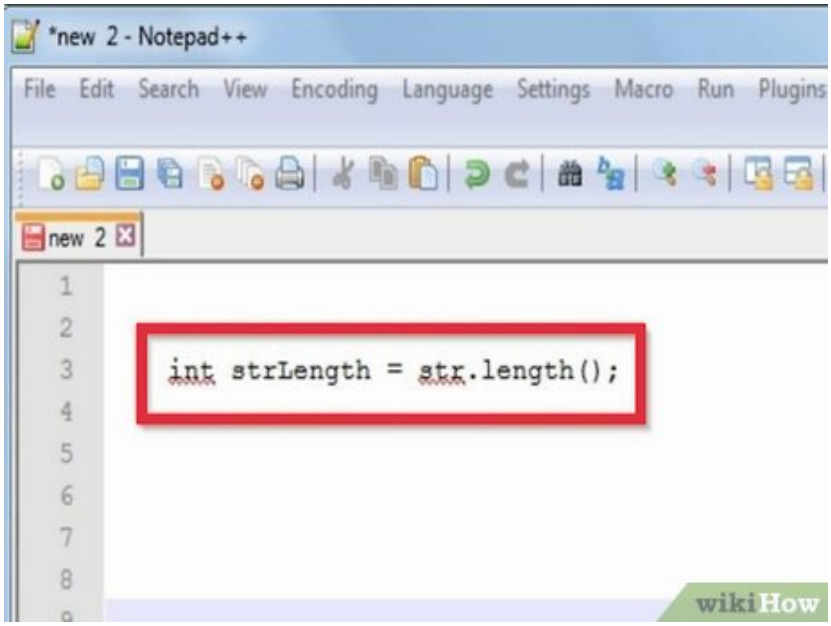
Try an example. Here is a sample program that creates a string in two different ways.

```
public class StringManipulation { public static void main(String[] args
) { String str1 = new String("String created with a constructor!");
String str2 = "String created without a constructor!"; System.out.
println(str1); System.out.println(str2); } }
```

Method 2 of 5:

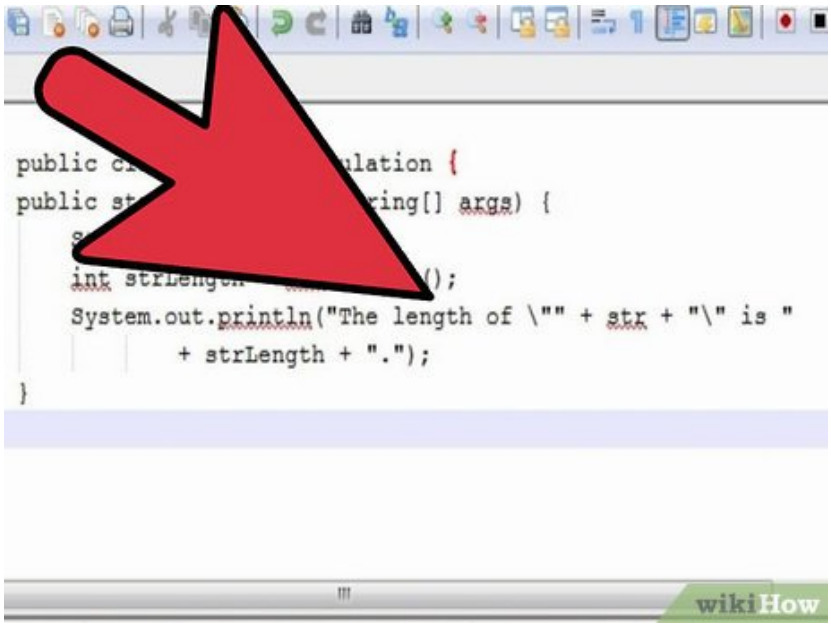
Find the Length of a String

1. **Understand what it means to find the length of a string.** The length of a string is the number of characters that the string contains. For example, the length of the string "Hello!" is 6 because it has 6 characters.

2. 

Invoke the `length()` method on the String object and store the result in an integer variable.

```
int strLength = str.length();
```

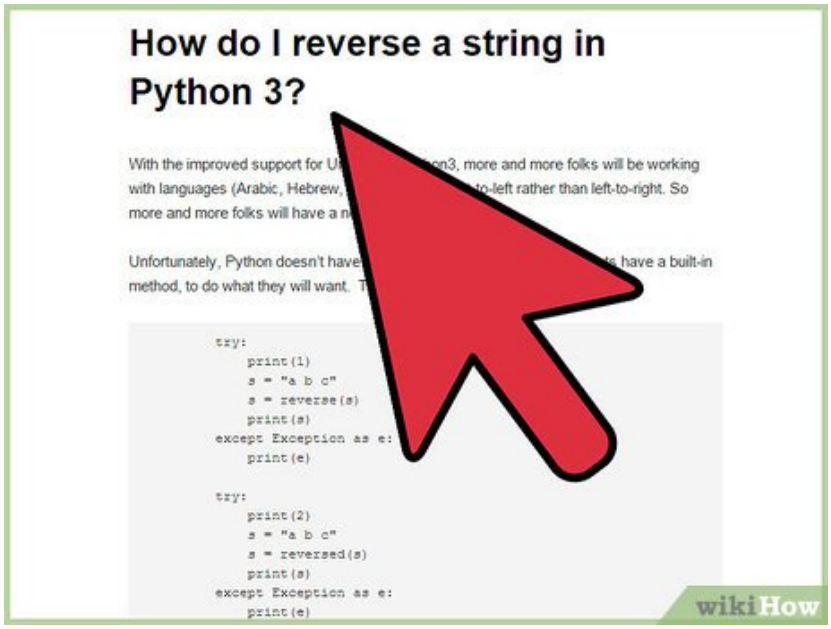
3. 

Give it a go. Here is a sample program that finds the length of a string.

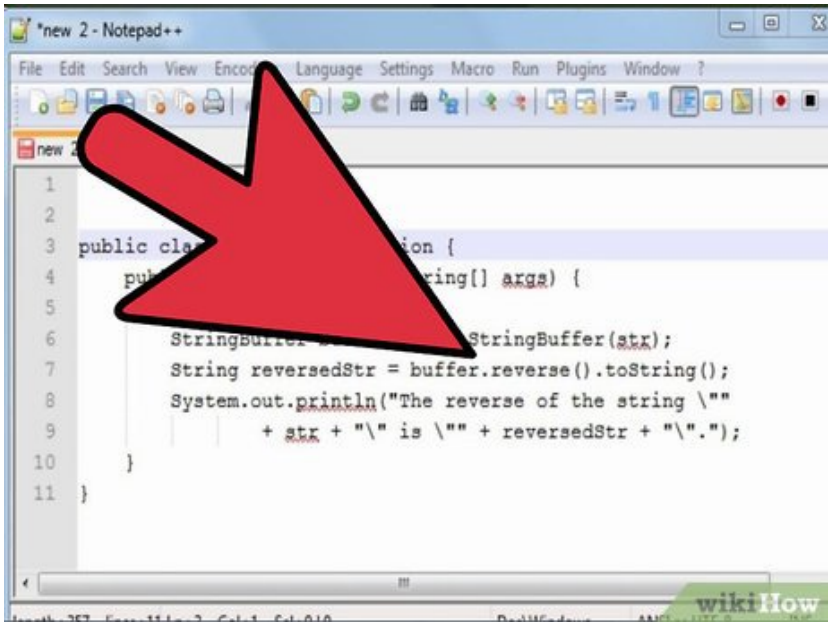
```
public class StringManipulation { public static void main(String[] args
) { String str = "Hello!"; int strLength = str.length(); System.out.
println("The length of "" + str + "" is " + strLength + "."); } }
```

Method 3 of 5:

Reverse a String

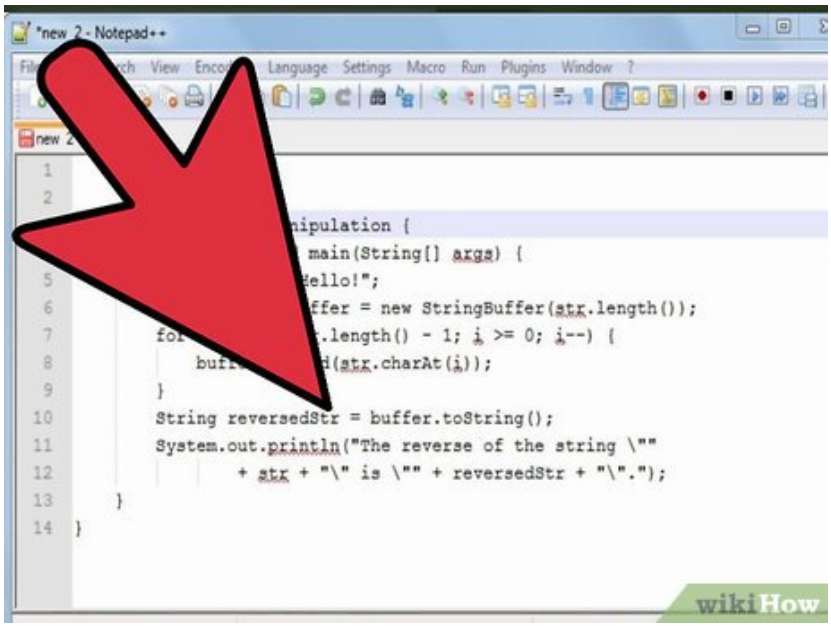
1. 

Understand what it means to reverse a string. Reversing a string means to switch the ordering of the characters in a string. For example, the reverse of the string "Hello!" is "!olleH". There are many ways to reverse a string in Java.

2. 

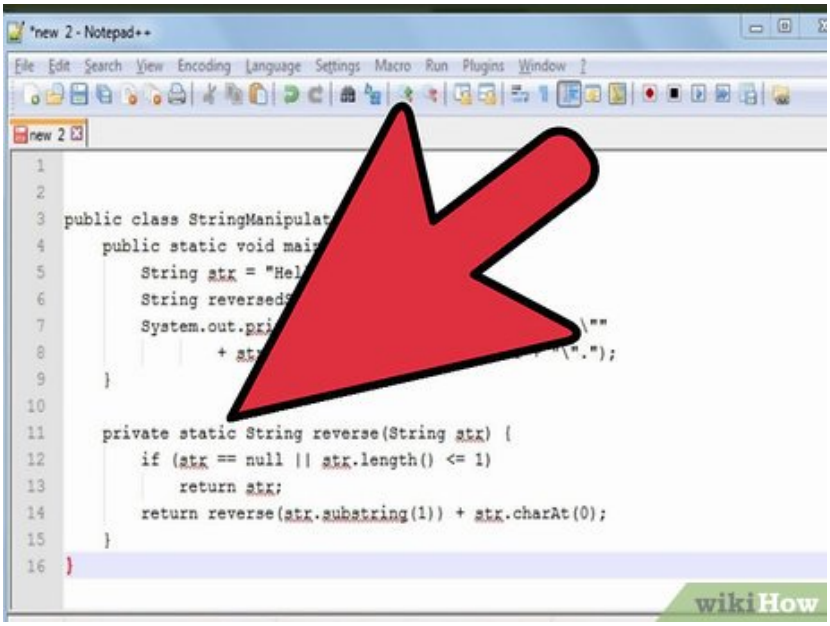
Use the reverse method of the StringBuffer class. Create a StringBuffer object that takes in the string that you want to reverse as a parameter. Use the StringBuffer's reverse() method and then retrieve the newly reversed string by using the toString() method.

```
public class StringManipulation { public static void main(String[] args
) { String str = "Hello!"; StringBuffer buffer = new StringBuffer(str);
 String reversedStr = buffer.reverse().toString(); System.out.println(
"The reverse of the string "" + str + "" is "" + reversedStr + "".");  }
 }
```

3. 

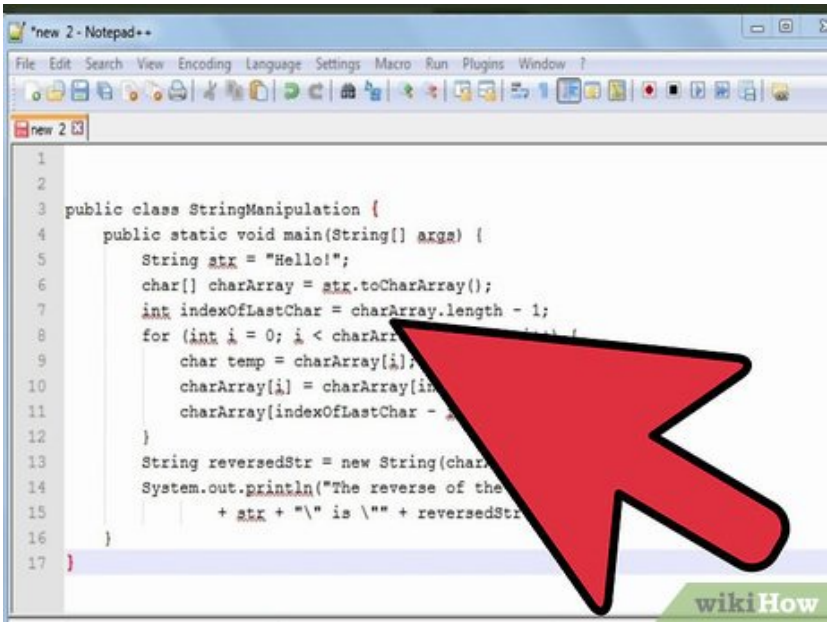
Iterate through the characters in a string in reverse, appending these characters to a StringBuffer at each iteration. Create a new StringBuffer object initialized with the length of the string that you wish to reverse as the parameter. Then use a for loop to iterate through the string, starting from the last character in the string and ending at the first character in the string. At each iteration, append the character at that index to the StringBuffer. Retrieve the newly reversed string by using the toString() method.

```
public class StringManipulation { public static void main(String[] args
) { String str = "Hello!"; StringBuffer buffer = new StringBuffer(str.
length()); for (int i = str.length() - 1; i >= 0; i--) { buffer.append(
str.charAt(i)); } String reversedStr = buffer.toString(); System.out.
println("The reverse of the string "" + str + "" is "" + reversedStr +
"".");  } }
```

4. 

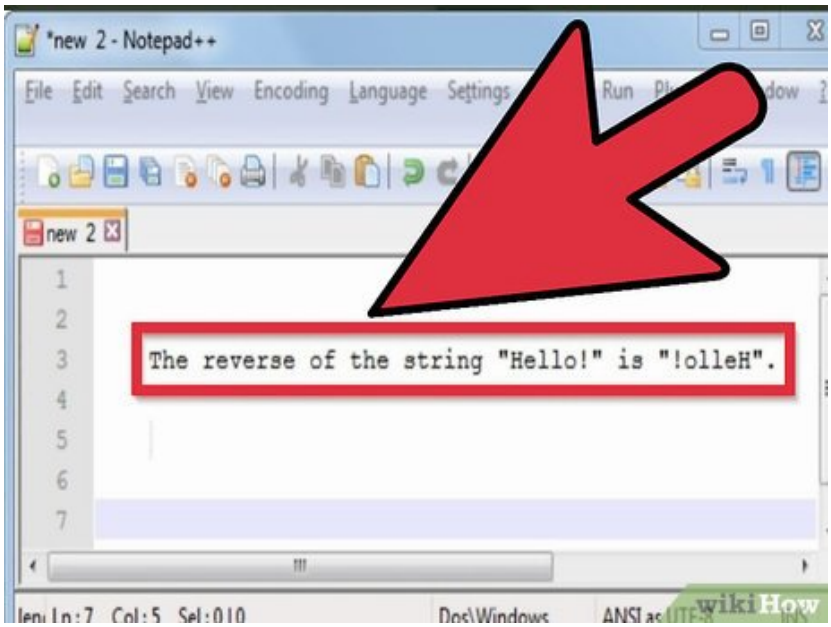
Write a recursive function to reverse the string. In the recursive function, the base case / condition is if the string is null or if the length of the string is less than or equal to none. Otherwise, the reverse() method is called again with the string minus the first character, and the first character is tacked on at the end. So if we passed in the string "Hello!", the first reverse() call after that will have the parameter "ello!".

```
public class StringManipulation { public static void main(String[] args ) { String str = "Hello!"; String reversedStr = reverse(str); System. out.println("The reverse of the string "" + str + "" is "" + reversedStr + "".""); } private static String reverse(String str) { if ( str == null || str.length() = 1) return str; return reverse(str. substring(1)) + str.charAt(0); } }
```

5. 

Convert the string to an array of characters and then swap the first and last, second and second to last, etc. characters. First convert the string to an array of characters by using the toCharArray() method on the string. Get the index of the last character in the array, which is equal to the length of the array minus one. Then iterate through the array, swapping the i^{th} character and the indexOfLastChar - i^{th} character at each iteration. Finally, convert the character array back to a string.

```
public class StringManipulation { public static void main(String[] args
) { String str = "Hello!"; char[] charArray = str.toCharArray(); int
indexOfLastChar = charArray.length - 1; for (int i = 0; i  charArray.
length/2; i++) { char temp = charArray[i]; charArray[i] = charArray[
indexOfLastChar - i]; charArray[indexOfLastChar - i] = temp; } String
reversedStr = new String(charArray); System.out.println(
"The reverse of the string "" + str + "" is "" + reversedStr + "".");  }
 }
```

6. 

Review your output. Here is the output that results from any one of these methods for string reversal.

Method 4 of 5:

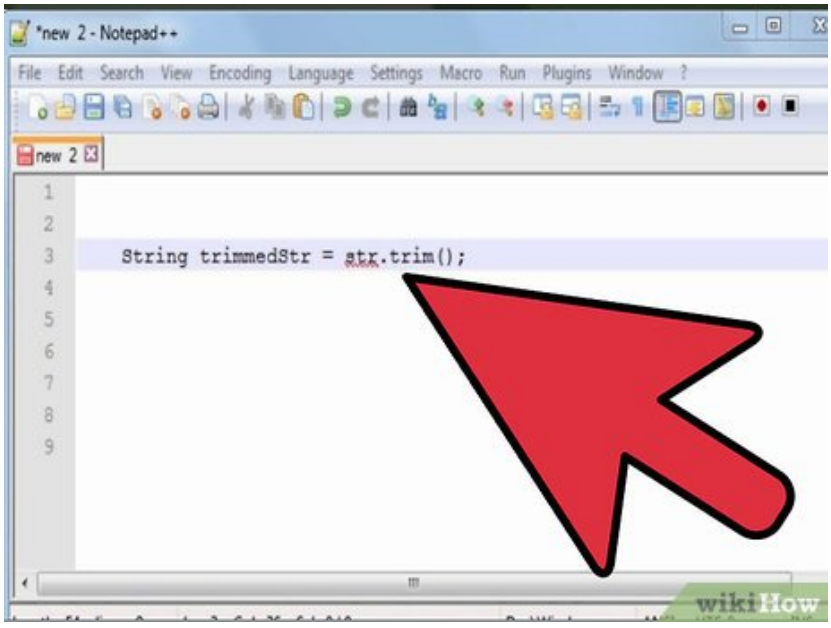
Trim White Space in a String

1. 

Understand what it means to trim white space in a string. Trimming a string in Java means to remove the leading and trailing white space in the string. For example, if you have the string "

Hello, world!

" and you want to have it say "Hello, world!" without the white space in the beginning and in the end, you can trim the string. The String class provides a method to trim() which returns a copy of the string with leading and trailing white space removed or the original string if it has no leading or trailing white space.

2. 

Use the trim() method of the String class on a String object to trim the white space. Note that the trim() method will throw an exception if the string is null. The trim() method will not change the contents

of the original string because strings in Java are immutable, which means that a string's state cannot be modified after it is created. Rather, the trim() method will return a new string that has its whitespace trimmed off.

```
String trimmedStr = str.trim();
```

3. **Try an example.** Here is a sample program that trims the white space of a string:

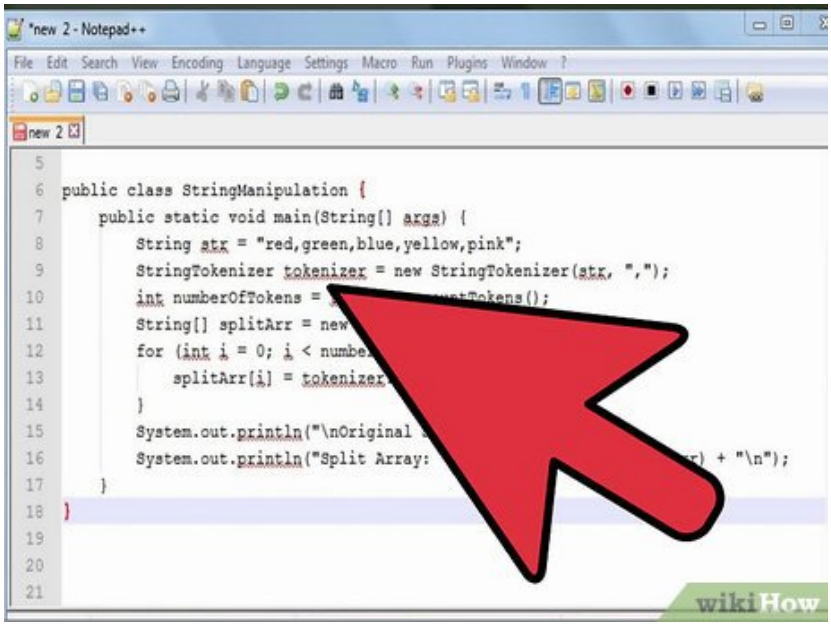
```
public class StringManipulation { public static void main(String[] args ) { String str = " Hello! "; String trimmedStr = str.trim(); System.out .println("Original String is "" + str + "."); System.out.println( "Trimmed String is "" + trimmedStr + "."); } }
```

Method 5 of 5:

Split a String

1. 

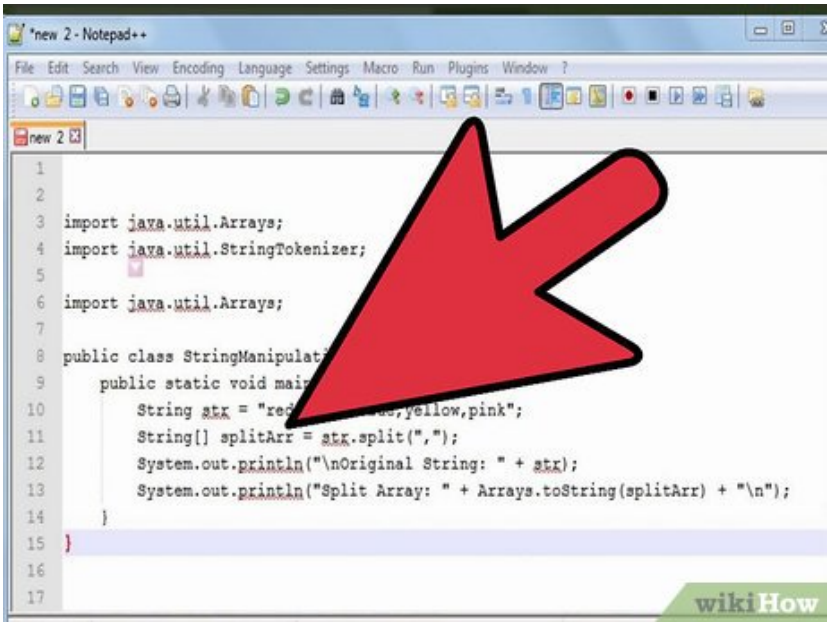
Understand what it means to split a string. Splitting a string in Java means to split a string by a certain delimiter into an array of substrings. For example, if I split the string "red,blue,green,yellow,pink" with a comma as the delimiter, I would get the array { "red", "blue", "green", "yellow", "pink" }. Here are three different ways to split a string.

2. 

Use `StringTokenizer` to tokenize the string. Import `java.util.StringTokenizer`. Then create a new instance of a `StringTokenizer` with the string to tokenize and the delimiter as parameters. If you do not enter the delimiter as a parameter, the delimiter will automatically default to white space. After you have the `StringTokenizer`, you can use the `nextToken()` method to get each token.

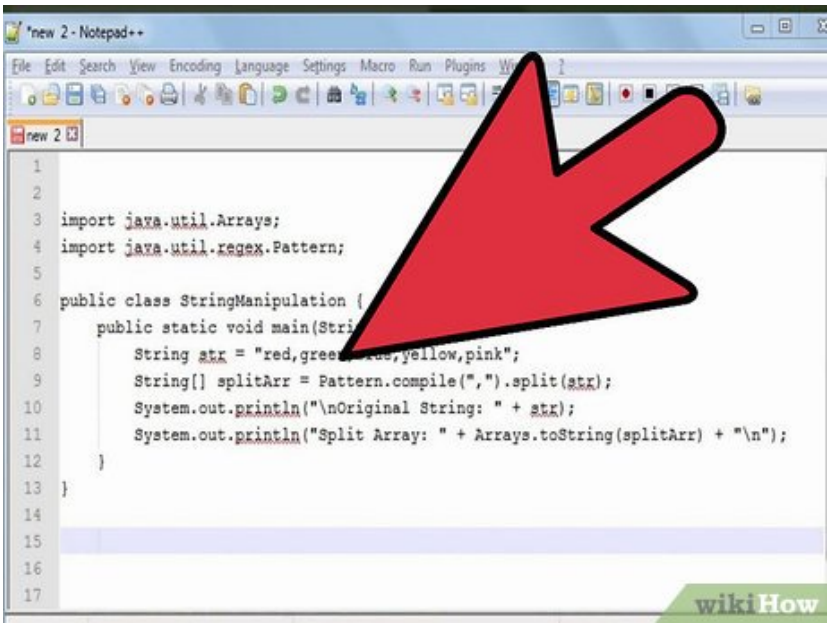
```
import java.util.Arrays; import java.util.StringTokenizer; public class
 StringManipulation { public static void main(String[] args) { String
str = "red,green,blue,yellow,pink"; StringTokenizer tokenizer = new
StringTokenizer(str, ","); int numberOfTokens = tokenizer.countTokens
(); String[] splitArr = new String[numberOfTokens]; for (int i = 0; i
numberOfTokens; i++) { splitArr[i] = tokenizer.nextToken(); } System.
out.println("nOriginal String: " + str); System.out.println(
"Split Array: " + Arrays.toString(splitArr) + "n"); } }
```

1. Before Java 1.4, the `StringTokenizer` class was used to split strings in Java. But now, the use of `StringTokenizer` is discouraged and the use of the `split()` method in the `String` class or the use of the `java.util.regex` package is encouraged.

3. 

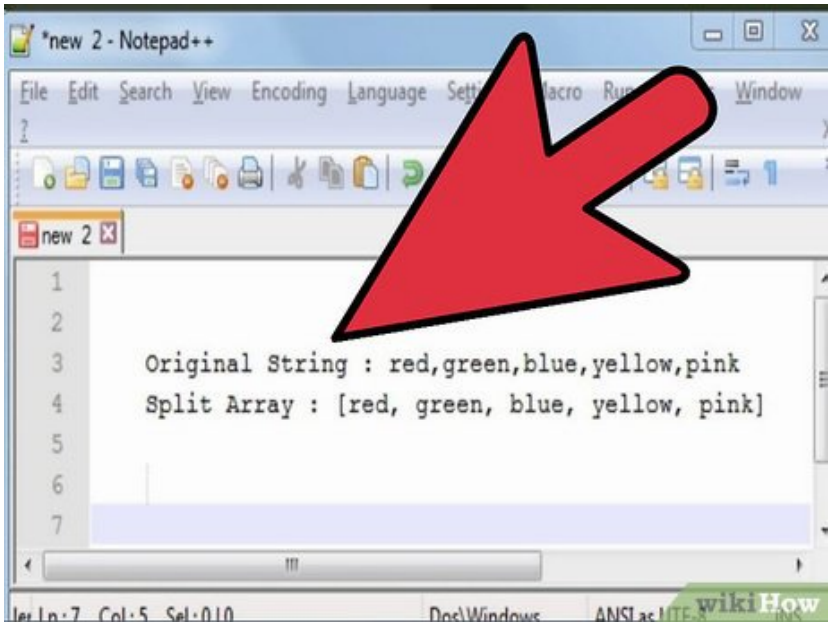
Use the `String` class's `split()` method. The `split()` method will take in the delimiter as a param and return an array of sub-strings that are the same as the tokens from the `StringTokenizer`.

```
import java.util.Arrays; public class StringManipulation { public
static void main(String[] args) { String str =
"red,green,blue,yellow,pink"; String[] splitArr = str.split(",");
System.out.println("nOriginal String: " + str); System.out.println(
"Split Array: " + Arrays.toString(splitArr) + "n"); } }
```

4. 

Use regular expressions to split the string. Import `java.util.regex.Pattern`. Use the `compile()` method of the `Pattern` class to set the delimiter and then give the `split()` method the string that you want to split. The `Pattern` will return an array of substrings.

```
import java.util.Arrays; import java.util.regex.Pattern; public class
StringManipulation { public static void main(String[] args) { String
str = "red,green,blue,yellow,pink"; String[] splitArr = Pattern.compile
(",").split(str); System.out.println("nOriginal String: " + str);
System.out.println("Split Array: " + Arrays.toString(splitArr) + "n");
} }
```

5. 

Review your output. Here is the output that results from any one of these methods for splitting strings.