**1. Overview**

In this quick article, we are going to check and discuss different techniques for removing the last character of a *String.*

**2. Using *String.substring()***

The easiest way is to use **the built-in *substring()* method** of the *String* class.

In order to remove the last character of a given *String,* we have to use two parameters: *0* as the starting index, and index of the penultimate character. We can achieve that by calling *String*‘s *length()*method and subtracting *1* from the result.

However, **this method is not null-safe** and if we use an empty string this is going to fail.

To overcome issues with null and empty strings, we can wrap the method in a helper class:

|  |  |
| --- | --- |
| 1  2  3  4  5 | public static String removeLastChar(String s) {      return (s == null || s.length() == 0)        ? null        : (s.substring(0, s.length() - 1));  } |

We can refactor the code and use Java 8:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public static String removeLastCharOptional(String s) {      return Optional.ofNullable(s)        .filter(str -> str.length() != 0)        .map(str -> str.substring(0, str.length() - 1))        .orElse(s);      } |

**3. Using *StringUtils.substring()***

Instead of reinventing the wheel, we can use ***StringUtils* class from Apache Commons Lang3** library, that offers helpful *String* operations. One of them is a **null-safe *substring()* method**, which handles exceptions.

To include *StringUtils* we have to update our *pom.xml* file:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <dependency>      <groupId>org.apache.commons</groupId>      <artifactId>commons-lang3</artifactId>      <version>3.0</version>  </dependency> |

*StringUtils.substring()* requires three parameters: a given *String,* an index of the first character (in our case it will be 0 always) and index of the penultimate character. Again, we can simply use the *length()*method and subtract *1:*

|  |  |
| --- | --- |
| 1  2  3 | String TEST\_STRING = "abcdef";    StringUtils.substring(TEST\_STRING, 0, TEST\_STRING.length() - 1); |

Yet, this operation is not null-safe again. It will work with empty *Strings* fine though.

**4. Using *StringUtils.chop()***

***StringUtils* class provides the *chop()* method that works well with all edge scenarios: empty and null Strings**.

It is very easy to use and requires only one parameter: the *String.* Its sole purpose is to remove the last character. Nothing more, nothing less:

|  |  |
| --- | --- |
| 1 | StringUtils.chop(TEST\_STRING); |

**5. Using Regular Expression**

We can also remove the last character (or any number of characters) from a *String* by making good use of regular expressions.

For example, we can use the ***replaceAll()* method of *String* class itself** – which takes two parameters: regular expression and the replacement *String*:

|  |  |
| --- | --- |
| 1 | TEST\_STRING.replaceAll(".$", ""); |

Note that, because we’re calling a method on the *String* – the operation is, of course, **not null-safe**.

Also, *replaceAll()* and regex expression can be complex at first sight. You can read more about regex [here](http://www.baeldung.com/regular-expressions-java), but to make the logic a bit more user-friendly, we can wrap it in a helper class:

|  |  |
| --- | --- |
| 1  2  3 | public static String removeLastCharRegex(String s) {      return (s == null) ? null : s.replaceAll(".$", "");  } |

Note that if a *String* ends with a newline then the above method will fail as *“.”* in regex matches any character except for line terminators.

Finally, let’s re-write **the implementation with Java 8**:

|  |  |
| --- | --- |
| 1  2  3  4  5 | public static String removeLastCharRegexOptional(String s) {      return Optional.ofNullable(s)        .map(str -> str.replaceAll(".$", ""))        .orElse(s);  } |

**6. Conclusion**

In this short article, we’ve discussed different ways of removing only the last character of a *String* – some manual, some ready out of the box.