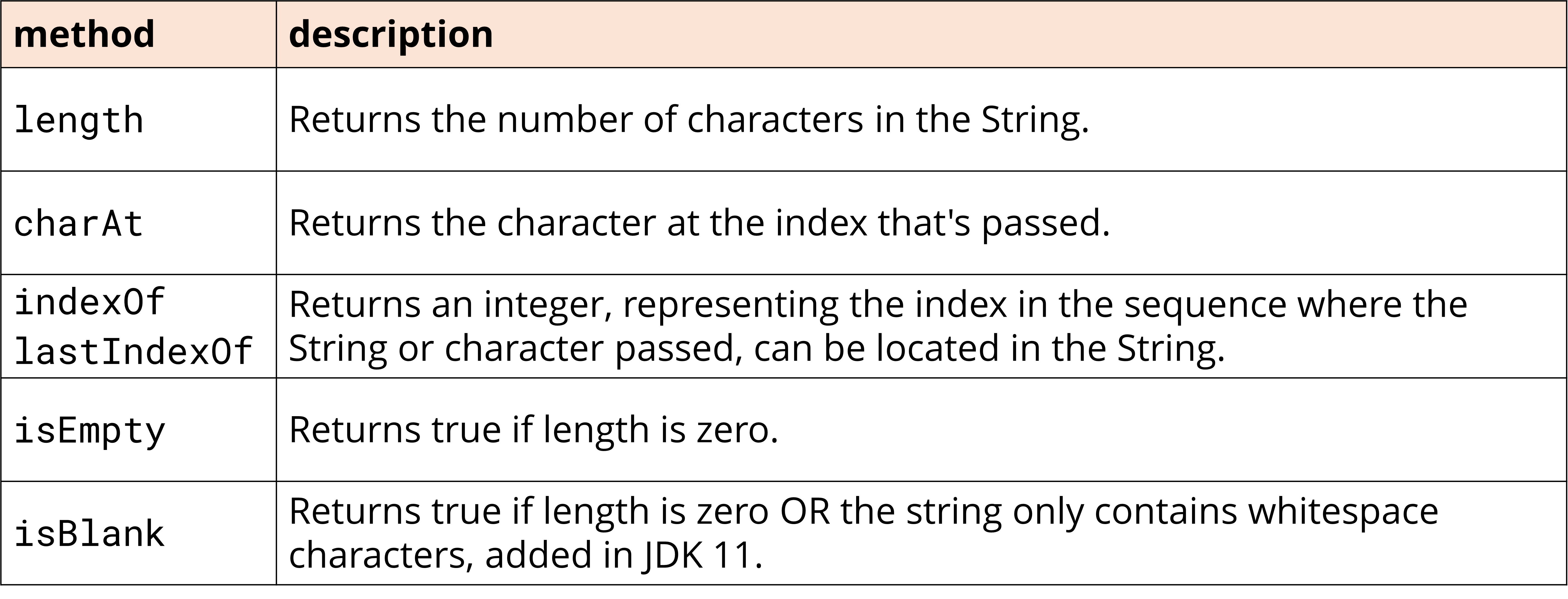
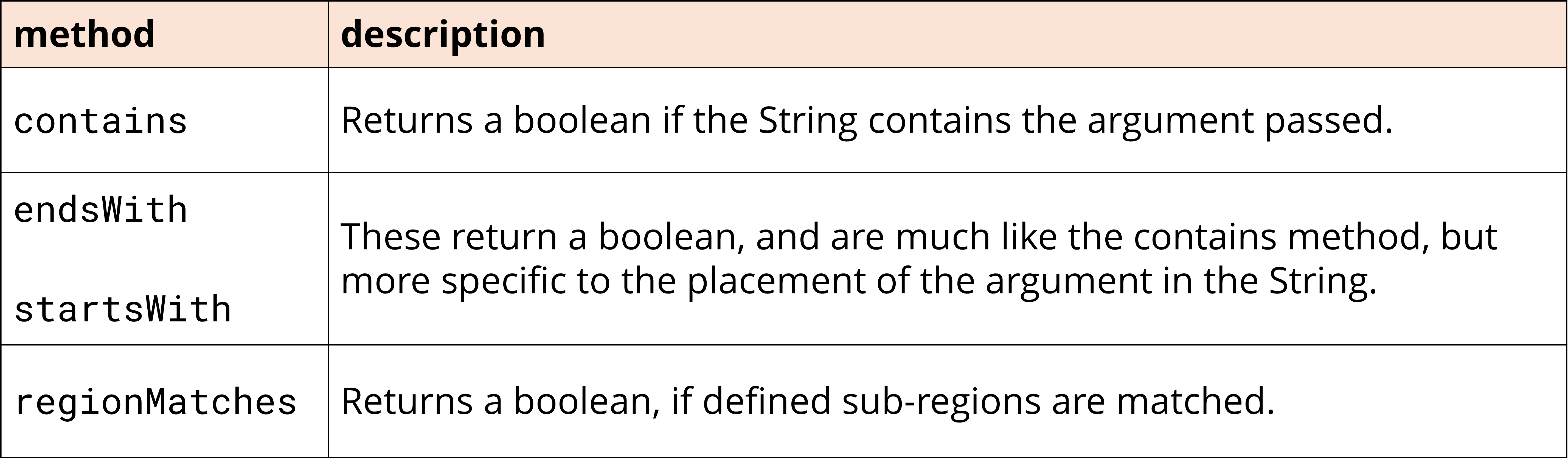
**Another Look at the String**

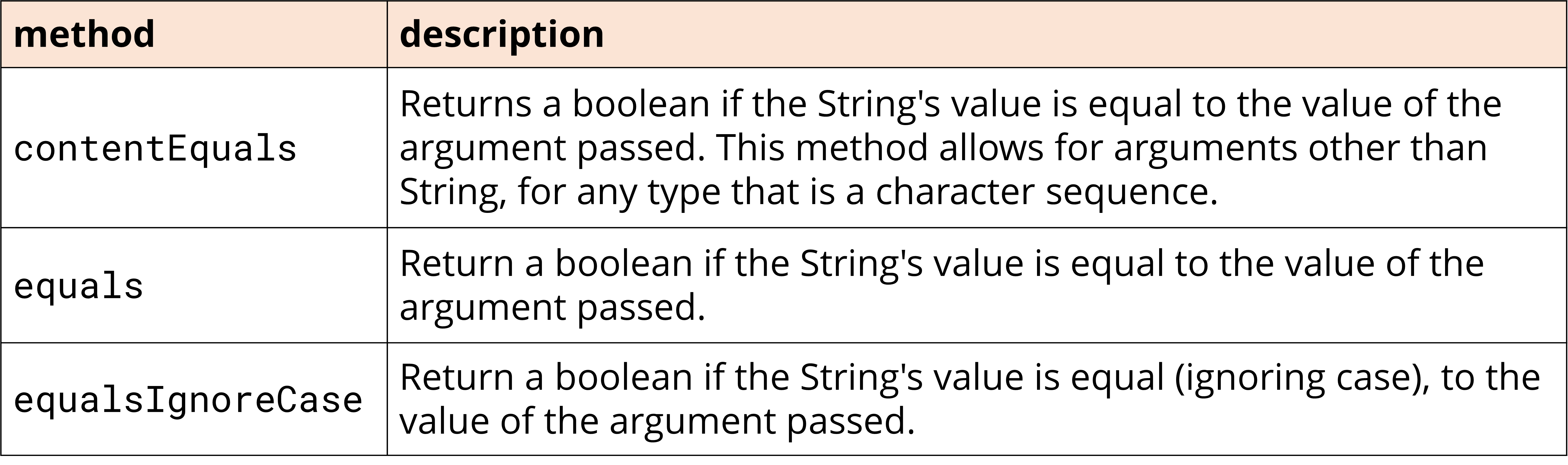
**String Inspection Methods**



**String Comparison Method:**

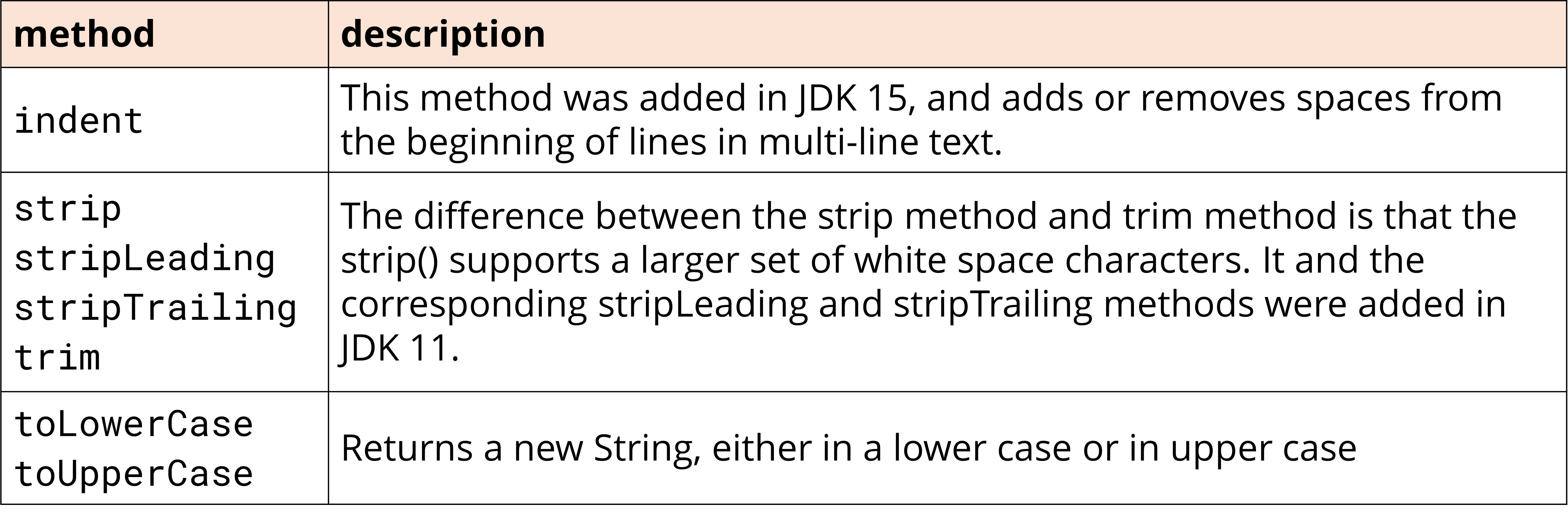


* Java have both contentEquals and equals method because the contentEquals method isn’t limited to just comparing string object. It can be used to compare StringBuilder’s value, which the equals method doesn’t support.

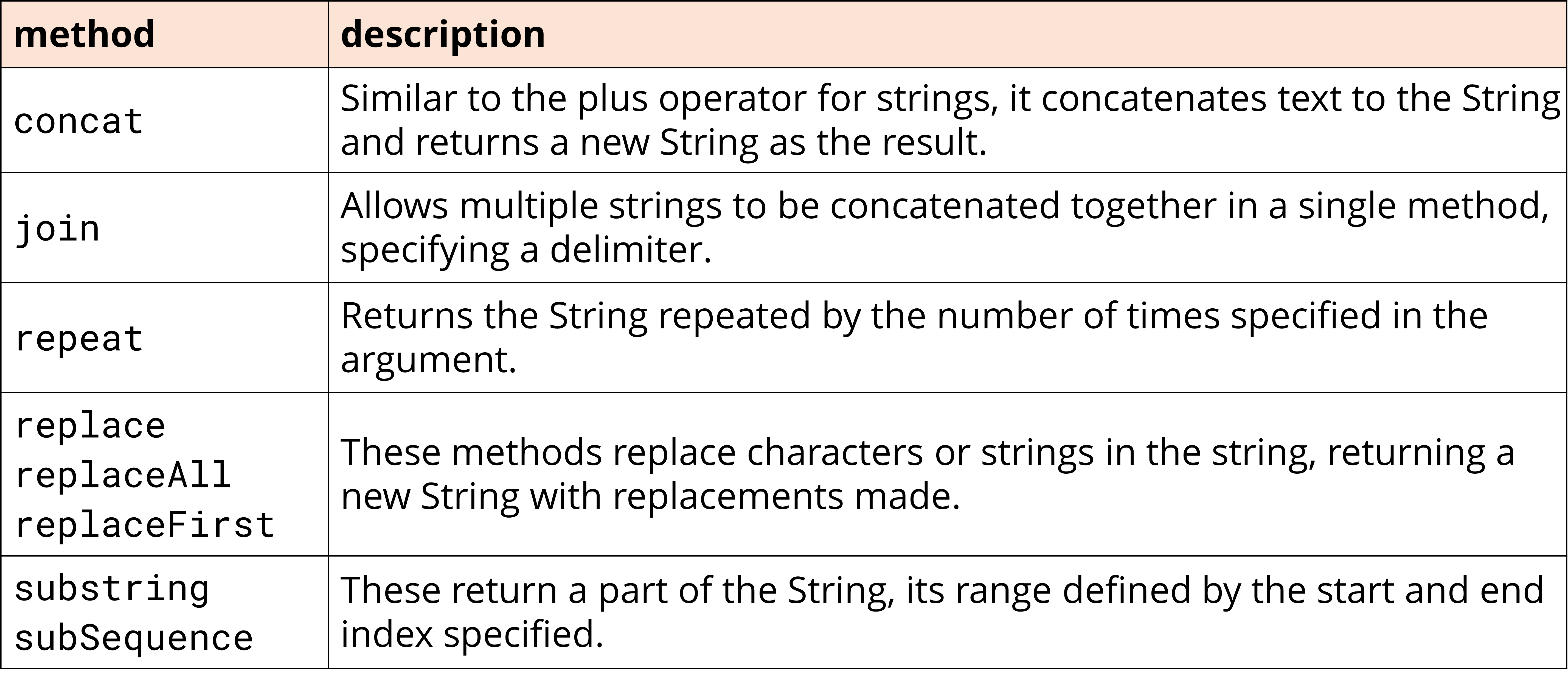


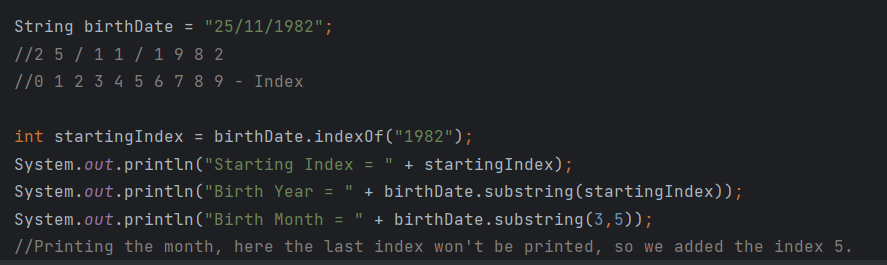
**String Manipulation Methods:**

* The first set of methods don't change the underlying meaning of the text value but perform some kind of clean up.



* The second set of string manipulation methods, transform the String value, and return a String with a different meaning, than the original String.



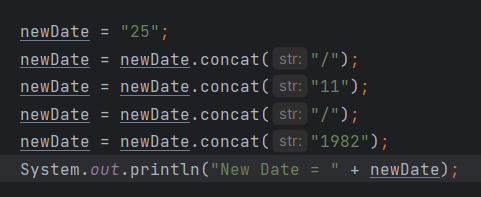
* **Substring** will have a Starting index also it has an overloaded method where we can include the Ending index.

A black background with white text

Description automatically generated with low confidence

* This joins a series of Strings together, with some delimiter. This is a good method to use, if you ever need to create a comma delimited String,

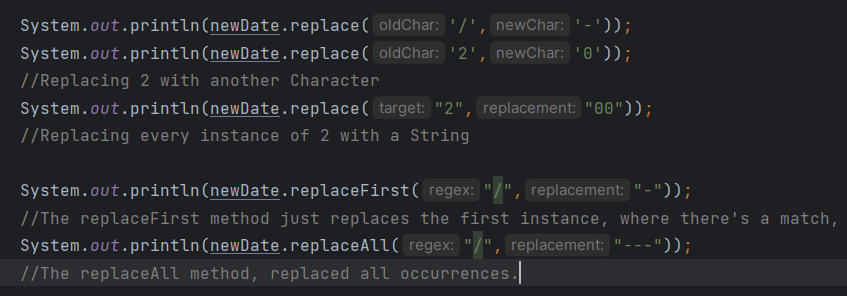


Using the **Concat** method to make the same thing.

A screenshot of a computer code

Description automatically generated with low confidence

* This coding style has a special name, and it's called method chaining.
* Each call to the concat method is still a new String object being created.
* But instead of assigning that result to a variable, we chain it to another method result.

**Replace** method:

A picture containing text, font, screenshot, design

Description automatically generated

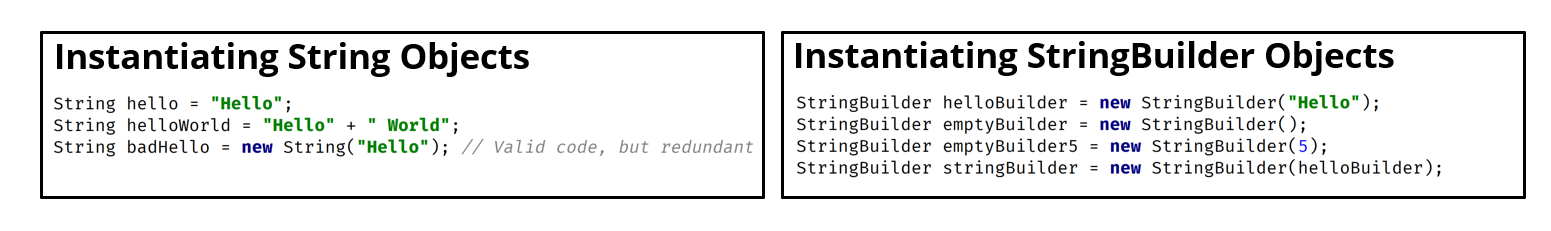
* We can use replaceFirst with basic strings, to replace the first instance. But if we want to replace all occurrences of one string with another, use the replace method, rather than the replaceAll method.

A screen shot of a computer code

Description automatically generated with low confidenceA picture containing text, screenshot, font

Description automatically generated**Repeat** method.

**The StringBuilder Class**

StringBuilder:

* There are four ways to create a new StringBuilder object, using the new keyword:
  + Pass a String.
  + Pass no arguments at all.
  + Pass an integer value.
  + Pass some other type of character sequence (like StringBuilder).
* StringBuilder's text value is mutable, but the String's isn't.
* Similar to **concat** in String, we have “**append**” in StringBuilder.
* It's important to remember to assign the result, of any String manipulation method you call on a String, to a variable.

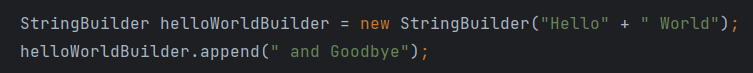
A picture containing font, text, screenshot, number

Description automatically generatedA picture containing text, screenshot, font

Description automatically generated

* Here we didn’t save we didn't assign the result of the method, the concat method, to a variable.
  + These methods don't change the internals of the existing String object.
* The String referenced by the helloWorld variable never changed, instead a new String was created by the method call.
* A picture containing text, font, line, number

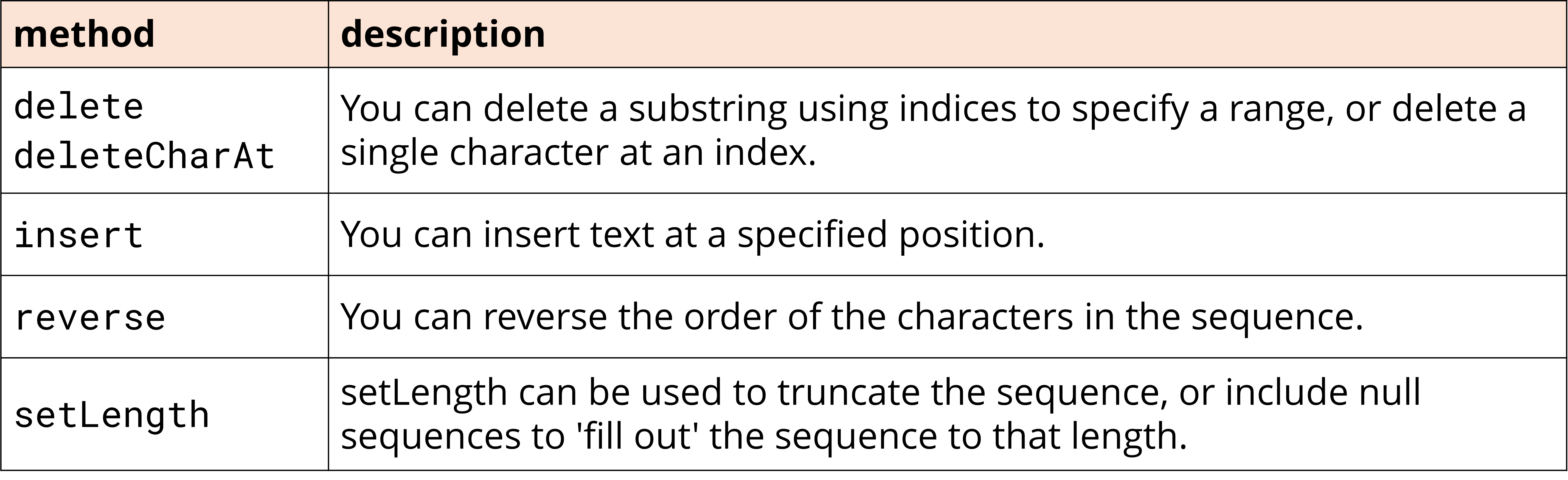
  Description automatically generated When we passed the String literal, "and Goodbye", to the concat method, this created an Object in memory for that literal, "and Goodbye".
* A black screen with white text

  Description automatically generated with low confidenceIt also created the result of the concat method, the object, the String, that has the value, "Hello World and Goodbye".

A picture containing text, font, number, line

Description automatically generated

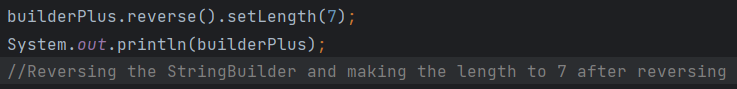
* We still have the objects in memory, that represent the String literals, that were passed to the StringBuilder, in this case Hello World.
* But we can see, that After the call to the append method, we still only have one StringBuilder object.
* The variable helloWorldBuilder, is still referencing the same object, but the value of that object changed i.e., the character sequence in the StringBuilder changed.
* We didn't have to assign the result, to another variable to access the result.
* String methods create a new object in memory and return a reference to this new object.
* StringBuilder methods return a StringBuilder reference, but it's really a self-reference.
* Unlike Strings, we can call methods on StringBuilder, without the need to assign the results, to intermediate variables, as we saw with Strings.
* StringBuilder methods return this self-reference, to support chaining methods together.
  + A StringBuilder is mutable, which means it can shrink, or grow, in size.
  + Every time a StringBuilder needs to increase capacity, the data stored in the original storage, needs to get copied over to the larger storage area.
  + The default capacity of a StringBuilder is 16
  + The new allocation size is determined by the JVM.
  + The capacity does need to adjust, as the text in our StringBuilder grows.

**Some methods unique to the StringBuilder class**

**A picture containing text, font, screenshot, typography

Description automatically generatedA screen shot of a computer

Description automatically generated with medium confidence**

**Reverse and the setLength method.**

****

* The reverse method will reverse all the characters, and then call the setLength method, with the number 7, meaning we're truncating the StringBuilder text value, to 7 characters.