StringBuilder

1. Modifying strings

In Java, strings are immutable objects. This was done to make the String class highly optimized and to allow it to be used everywhere.

However, situations often arise when programmers would find it more convenient for the `String` class to be mutable. They want a class that doesn't create a new substring every time one of its methods is called.

Well, suppose we have a very large string and we need to frequently add something to the end of it. In this case, even a collection of characters (`ArrayList<Character>`) can be more efficient than constantly recreating and concatenating String objects.

That's precisely why a String-like type that can be changed was added to the Java language. It is called `StringBuilder`.

Creating an object

To create a `StringBuilder` object based on an existing string, you need to execute a statement like:

```
StringBuilder name = new StringBuilder(string);
```

To create an empty mutable string, you need to use a statement like this:

```
StringBuilder name = new StringBuilder();
```

List of methods

The `StringBuilder` class has two dozen helpful methods. Here are the most important ones:

Method	Description
`StringBuilder append(obj)`	Converts the passed object to a string and appends it to the current string
`StringBuilder insert(int index, obj)`	Converts the passed object to a string and inserts it into the current string
`StringBuilder replace(int start, int end, String str)`	Replaces the part of the string specified by the start..end interval with the passed string
`StringBuilder deleteCharAt(int index)`	Removes the character with the specified index from the string
`StringBuilder delete(int start, int end)`	Removes characters within the specified interval from the string
`int indexOf(String str, int index)`	Searches for a substring in the current string
`int lastIndexOf(String str, int index)`	Searches for a substring in the current string, starting from the end
`char charAt(int index)`	Returns the character in the string at the passed index
`String substring(int start, int end)`	Returns the substring defined by the specified interval
`StringBuilder reverse()`	Reverses the current string.
`void setCharAt(int index, char)`	Changes the character at the specified index to the passed character
`int length()`	Returns the length of the string in characters

2. Description of methods:

Appending to a string

To add something to a mutable string (`StringBuilder`), use the `append()` method. Example:

Code	Description

`StringBuilder builder = new StringBuilder("Hi");`
`builder.append("Bye");`
`builder.append(123);`
	`Hi`
	`HiBye`
	`HiBye123`

Converting to a standard string

To convert a `StringBuilder` object to a String object, you just need to call its `toString()` method. Example

Code	Output

`StringBuilder builder = new StringBuilder("Hi");`
`builder.append(123);`
`String result = builder.toString();`
`System.out.println(result);`
	`Hi123`

How do I delete a character?

To delete a character in a mutable string, you need to use the `deleteCharAt()` method. Example:

Code	Output

`StringBuilder builder = new StringBuilder("Hello");`
`builder.deleteCharAt(2);`
`String result = builder.toString();`
`System.out.println(result);`
	`Helo`

How do I replace part of a string with another string?

For this there is the `replace(int begin, int end, String str)` method. Example:

Code	Output

`StringBuilder builder = new StringBuilder("Mellow");`
`builder.replace(2, 5, "Hello!");`
`String result = builder.toString();`
`System.out.println(result);`
	`MeHello!w`

3. Useful examples of working with strings

How do I reverse a string?

There is a special method for doing this — `reverse()`; Example:

Code	Output

`String str = "Hello";`
`StringBuilder builder = new StringBuilder(str);`
`builder.reverse();`
`String result = builder.toString();`
`System.out.println(result);`
	`olleH`

`StringBuffer` class

There is another class — StringBuffer, which is an analogue of the `StringBuilder`class, but its methods are marked with the `synchronized` modifier. It means that the `StringBuffer` object can be accessed simultaneously from multiple threads.

But it is much slower than `StringBuilder`. You may need this class when you start to actively explore multithreading in the *`Java`* quest.

Multithreading