Strings

Strings - Methods

Strings are **Objects**, so they have **Methods**.

String name = “Simon”;

String methods

Method	Use
name.**substring(int start, int end)**	returns a copy of the string between the two indices excluding the end
name.**substring(int start)**	returns a copy of the string starting at the index, up until the end
name.**equals(Object another)**	return true if the strings have identical contents
name.**length()**	returns the number of characters in str
name.**compareTo(String another)**	for less than/ greater than / equal comparison
name.**charAt(int index)**	return the character at the index position of the String

More about class String:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html

Immutability

String methods do **not** alter the existing String, they create **new** ones.

```
String name = “simon”;

String newName = name.toUpperCase();

System.out.println(name);          - - - - - - - - - - ->       simon

System.out.println(newName);       - - - - - - - - - - ->       SIMON
```

Immutability

Strings are **immutable** which means they cannot change once they are created.

The only way to **change** the value of name is to **re-assign** it:

```
String name = “simon”;

System.out.println(name);          - - - - - - - - - - ->       simon

name = “SIMON”;

System.out.println(name);          - - - - - - - - - - ->       SIMON
```

Concatenation

Strings can be **concatenated** with one another to create a **new String** value.

Concatenate: Add 2 string values together.

```
String firstName = “Simon”;

String lastName = “Smith”;

String fullName = firstName + lastName;

System.out.println(fullName);        - - - - - - - - - ->        SimonSmith
```

Concatenation

```
String firstName = “Simon ”;

String lastName = “Smith”;

String fullName = firstName + lastName;

System.out.println(fullName);          - - - - - - - - - ->          Simon Smith
```

Concatenation - Shortcut

The shortcut **+=** works on String values:

```
String name = “Simon ”;

name += “Smith”;

System.out.println(name);        - - - - - - - - - ->        Simon Smith
```

Concatenating Primitives

Primitive Types can be concatenated with String objects:

```
String name = “Simon”;

int age = 8;

System.out.println(name + “  is “ + age);  - - - - - - - - - ->    Simon is 8
```

The primitive type is converted to String, this is called **implicit conversion.**

Concatenating Primitives

Implicit conversion can be tricky.

What do you think the outcome of this program is?

```
int currentAge = 20;

int age = 10;

System.out.println(“In ten years, I will be: “ + currentAge + age);
```

```
In ten years, I will be: 2010
```

Concatenating Primitives

How can we make it work without using an extra variable?

```
int currentAge = 20;

int age = 10;


 System.out.println(“In ten years, I will be: “ + (currentAge + age));

 In ten years, I will be: 30
```

String dilemma

Some characters cannot be used directly because they hold a specific meaning in Java.

Example:

Include quotes in a line of code, will cause an error:

String str = ”She said, “Hello!”” - - - - - -> “Hello!” is interpreted as a String value

- - - - - -> The compiler thinks Hello! Is a variable name

Escape Sequences

We can include “” in our code by writing a **escape sequence \”**

Example:

String str = “She said, ****”Hello!****””;

System.out.println(str); - - - - - -> She said, “Hello!”

Some useful escape sequences

Escape Sequences allow us to include special characters and actions in String objects.

Escape Sequence	**Function**	**Output**
\”	“ \” Allow for quotations\” “	“Allow for quotations”
\\	“Includes a backslash\\”	Includes a backslash\
\n	This creates \na line break	This creates a line break
\t	“This adds a \ttab space”	This adds a tab space

Let’s practice

Classwork on GitHub