Theory: String

26 minutes 0 / 5 problems solved

Skip this topic Start practicing

17248 users solved this topic. Latest completion was 17 minutes ago.

§1. The String type

`String` is a reference type consisting of characters. It is one of the most widely used types in Java. Here is an example of a string: `"Hello, Java"`. This string is a sequence of 11 characters, including one space.

This type has some features:

- **immutable type**: it's impossible to change a character in a string;
- it has methods for getting individual characters and extracting substrings;
- individual characters can be accessed by indexes, the first character has the index 0, the last one – **the length of the string – 1**;
- non-primitive type.

§2. Creating strings

A string literal is surrounded by a pair of double quotes, for instance:

```
String simpleString = "It is a simple string"; // a simple string
System.out.println(simpleString);  // it prints "It is a simple string"

String anotherString = "This is\na multiple\nstring"; // a string with escape sequences
System.out.println(anotherString); // it prints the result in several lines
```

A string can represent a long character sequence (text). A string can have one or zero characters.

```
String strangeText = "aaaaaaaaaaaassssssssssss gggggggggggggggggggg ddddddddddd qqqqqq ffff";

String emptyString = "";

String s = "s"; // a string consisting of one character
```

A string can be `null`. It means no value assigned.

```
String nullString = null; // it is null
```

Another way to create a variable of `String` is by using the keyword `new`.

```
String str = new String("my-string"); // it creates an object and assigns it to the variable
```

§3. Get the length and characters of a string

Any string has two useful methods:

- `length()` returns the number of characters in the string;
- `charAt(int index)` returns a character by its index;

Here is an example:

Current topic:

String Stage 2

Topic depends on:

✓ Characters Stage 1

✓ Calling a method Stage 2

Topic is required for:

Processing strings Stage 2

Formatted output

Final variables

Regexps in Java

Defining classes Stage 3

Table of contents:

```
String s = "Hi, all";

int len = s.length(); // the len is 7

char theFirstChar = s.charAt(0);  // 'H' has the index 0

char theFifthChar = s.charAt(4); // 'a' has the index 4

char theLastChar = s.charAt(s.length() - 1); // 'l' has the index 6
```

You can easily get a character of a string by the index, but you can't change characters because strings are immutable in Java.

§4. Useful methods of strings

The standard library of Java provides a lot of useful methods for processing strings:

- `isEmpty()` returns `true` if the string is empty, otherwise – `false`;
- `toUpperCase()` returns a new string in uppercase;
- `toLowerCase()` returns a new string in lowercase;
- `startsWith(prefix)` returns `true` if the string starts with the given string prefix, otherwise, `false`;
- `endsWith(suffix)` returns `true` if the string ends with the given string suffix, otherwise, `false`.
- `contains(...)` returns `true` if the string contains the given string or character;
- `substring(beginIndex, endIndex)` returns a substring of the string in the range: `beginIndex`, `endIndex - 1`;
- `replace(old, new)` returns a new string obtained by replacing all occurrences of `old` with `new` that can be chars or strings.
- `trim()` returns a copy of the string obtained by omitting the leading and trailing whitespace. Note that whitespace includes not only space character, but mostly everything that looks empty: tab, carriage return, newline character, etc.

See the following example to better understand these methods:

```
String text = "The simple text string";

boolean empty = text.isEmpty(); // false

String textInUpperCase = text.toUpperCase(); // "THE SIMPLE TEXT STRING"

boolean startsWith = textInUpperCase.startsWith("THE"); // true

/* replace all space characters with empty strings */

String noSpaces = textInUpperCase.replace(" ", ""); // "THESIMPLETEXTSTRING"

String textWithWhitespaces = "\t text with whitespaces   !\n  \t";

String trimmedText = textWithWhitespaces.trim(); // "text with whitespaces   !"
```

To learn more about different methods and arguments you can check out the documentation.

§5. Exceptions when processing strings

When working with strings, there can be several exceptions.

1. `NullPointerException`. If a string is `null` and you call a method of the string, it throws `NullPointerException`.

```
String s = null;
int length = s.length(); // it throws NullPointerException
```

2. `StringIndexOutOfBoundsException`. If you try to access a non-existing character by an index then this exception occurs.

```
String s = "ab";

char c = s.charAt(2); // it throws StringIndexOutOfBoundsException because indexin
g starts with 0
```

We will consider how to handle different types of exceptions later.

§6. Concatenating strings

Two strings can be concatenated using the "+" operator or the `concat` method. Both approaches lead to the same results.

```
String firstName = "John";
String lastName = "Smith";

// concatenation using the "+" operator
String fullName1 = firstName + " " + lastName; // "John Smith"

// concatenation using the concat method
String fullName2 = firstName.concat(" ").concat(lastName); // "John Smith"
```

When we concatenate two strings a new string is created (because strings are **immutable**).

Important: in the general case `str1 + str2` is not the same as `str2 + str1` because the concatenation is not a commutative operation.

§7. Appending values to a string

It's possible to add values of different types to a string. The value will be automatically converted to a string. See an example below.

```
String str = "str" + 10 + false; // the result is "str10false"
```

In the example above, the order of execution is:

1. "str" + 10 => "str10"
2. "str10" + false = "str10false"

Let's see a more complex example:

```
String shortString = "str";
int number = 100;

String result1 = shortString + number + 50; // the result is "str10050"
String result2 = number + 50 + shortString; // what is the result2?
```

The `result2` is `150str`, because, first, we calculate a sum of `number` and `50` and then `concat` it with `str`. The order of operations is important.

§8. How to compare strings correctly?

Since `String` is a reference type you shouldn't compare strings using `==` or `!=` operators. In this case, only addresses will be compared, but not actual values.

`String` has two convenient methods for comparing the equivalence of the actual content of one string with the content of another string: `equals(other)` and `equalsIgnoreCase(other)`. See an example below.

```
String first = "first";
String second = "second";

String anotherFirst = "first";
String secondInUpperCase = "SECOND";

System.out.println(first.equals(second)); // false, the strings have different values
System.out.println(first.equals(anotherFirst)); // true, the strings have the same value

System.out.println(second.equals(secondInUpperCase)); // false, the strings have different cases
System.out.println(second.equalsIgnoreCase(secondInUpperCase)); // true, it ignores cases
```

Do not forget the rules when comparing strings.

Report a typo

1640 users liked this theory. **35** didn't like it. **What about you?**

Start practicing

Comments (24) Hints (1) Useful links (0) Show discussion