Quantifiers

There are the types of characters called **quantifiers**, which defines how often another character can occur in a regex pattern.

A quantifier can be written after a regular character, as well as after a special one.

In general, quantifiers are one of the most essential and important features of the regex language, since they allow a single pattern to match different strings varying in length.

The list of quantifiers

Here is a list of quantifiers to be remembered:

- + matches one or more repetitions of the preceding character;
- * matches zero or more repetitions of the preceding character;
- {n} matches exactly n repetitions of the preceding character;
- {n,m} matches at least n but not more than m repetitions of the preceding character;
- {n,} matches at least n repetitions of the preceding character;
- {0,m} matches no more than m repetitions of the preceding character.

Note, there is also another quantifier ?, which makes the preceding character optional. It is short for {0,1}. We will not consider this quantifier here, because you should already know it.

The plus quantifier

Here we demonstrate the **plus** character, which matches one or more repetitions of the preceding character:

```
String regex = "ca+b";

"cab".matches(regex); // true
"caaaaab".matches(regex); // true
"cb".matches(regex); // false because it does not have at least one repetition of 'a'
```

The star quantifier

The example below demonstrates the **star** character, which matches zero or more repetitions of the preceding character:

```
String regex = "A[0-3]*";

"A".matches(regex);  // true because the pattern matches zero or more repetitions
"A0".matches(regex); // true
"A000111222333".matches(regex); // true
```

In the following example, there is a pattern describing the string "John" located between an undefined number of undefined characters in the text:

```
String johnRegex = ".*John.*"; // it matches all strings containing the substring "John"

String textWithJohn = "My friend John is a computer programmer";

textWithJohn.matches(johnRegex); // true

String john = "John";

john.matches(johnRegex); // true

String textWithoutJohn = "My friend is a computer programmer";

textWithoutJohn.matches(johnRegex); // false
```

So, the **star** quantifier can be used to check whether a substring of a string matches a pattern.

Specifying the number of repetitions

Fortunately, there is a group of quantifiers that allow specifying the number of repetitions in **curly braces**: {n}, {n,m}, {n,}.

An important clarification: no spaces are supposed to be used inside curly braces. There can be only one or two numbers and, optionally, a comma. Putting spaces inside curly braces leads to the "deactivation" of the quantifier and, as a result, a totally different regular expression. For example, "a{1, 2}" will match only the exact string "a{1, 2}", not "a" or "aa".

Take a look at the example, where we demonstrate how to match exactly n repetitions of the preceding character using the {n} quantifier:

```
String regex = "[0-9]{4}"; // four digits

"6342".matches(regex);  // true
"9034".matches(regex);  // true

"182".matches(regex);   // false
"54312".matches(regex); // false
```

Matching from n to m repetitions is possible thanks to {n,m} quantifier. Note that the range specified in curly braces both starts and
ends **inclusively**: m encountered repetitions also count as a match.

```
String regex = "1{2,3}";

"1".matches(regex);    // false
"11".matches(regex);   // true
"111".matches(regex);  // true
"1111".matches(regex); // false
```

The last example demonstrates how to match at least n repetitions using the {n,} quantifier:

```
String regex = "ab{4,}";

"abb".matches(regex); // false, not enough 'b'
"abbbb".matches(regex); // true
"abbbbbbb".matches(regex); // true
```