# Regexps in Java

A **regular expression** is a sequence of characters that specifies a set of strings and that is used to search, edit, and manipulate text.

**Simple matching**
First of all, we can create a regular expression by means of a String . Take a look at the following example:
String aleRegex = "ale"; // the "ale" regex

**Java, String data type has built-in support for regular expressions.**

Strings have a special method called matches that takes a regular expression pattern as its argument and checks whether the string matches this pattern.

**Keep in mind that the method returns true only when the *whole* string matches the regexp, otherwise, it returns false**

**The pattern defined by the regex is applied to the text from left to right.**

## In the example below, we try to match aleRegex and different strings:

```
"ale".matches(aleRegex);  // true
"pale".matches(aleRegex); // false, "pale" string has an additional character
"ALE".matches(aleRegex);  // false, uppercase letters don't match lowercase
```

You can see that the string "pale" is not matched by our regex pattern. **The reason is that Java regex implementation checks whether the *whole* string can be fit into the regex pattern, not just some substring. In this regard, Java differs from many other programming languages.**

```
String helloRegex = "Hello, World";

"Hello, World".matches(helloRegex); // true
"Hello, world".matches(helloRegex); // false
"Hello,World".matches(helloRegex); // false
```

**As you remember, the real power of regular expressions lies in special characters that help you define a pattern matching several different strings at once.**

**The dot character and the question mark**
The dot . matches any single character including letters, digits, spaces, and so on. The only character it is unable to match with is the newline character \n.

```
String learnRegex = "Learn.Regex";

"Learn Regex".matches(learnRegex); // true
"Learn.Regex".matches(learnRegex); // true
"LearnRegex".matches(learnRegex); // false
"Learn, Regex".matches(learnRegex); // false
"Learn\nRegex".matches(learnRegex); // false
```

As you remember, the question mark ? is a special character that means “the preceding character or nothing". Words with slightly different spelling in American and British English serve as a traditional example of this character's application:

```
String pattern = "behaviou?r";

"behaviour".matches(pattern); // true
"behavior".matches(pattern);  // true
```

Now let's combine the dot character . and the question mark ? in one regex pattern. This combination (.?) basically means "there can be any single character or no character at all".

```
String pattern = "..?";

"I".matches(pattern);  // true
"am".matches(pattern); // true
"".matches(pattern);   // false
```

**The tricky escape character**

Right now you're probably wondering what we should do if we want to use the dot . or the ? as a regular punctuation mark and not as a special symbol within the regex pattern?

Well, in this case, we should protect our special symbol by putting the backslash \ before it.

The backslash is called an escape character because it helps symbols to "escape" their working duties.

When you want to use the backslash \ itself in its literal meaning, you need to escape it as well! This way, a double backslash \\ in your regex means a single backslash in the matching string.

However, it gets more complicated when you implement such patterns in your Java program.

The backslash \ works as an escape character not only for regular expressions but for String literals as well. So, in fact, we have to use an additional backslash to escape the one we need in the regular expression, just like this:

```
String endRegex = "The End\\.";

"The End.".matches(endRegex); // true
"The End?".matches(endRegex); // false
```

For instance, the regular expression for any five-character sequence that ends with a dot looks like this:

```
String pattern = ".....\\.";

"a1b2c.".matches(pattern); // true
"Wrong.".matches(pattern); // true
"Hello!".matches(pattern); // false
```

**Here's the explanation: first, two backslashes \\ are needed for regex language to match a single backslash \ in the string. Second, for the regex to look this way, Java requires each of these two backslashes to be escaped with another backslash.**