Regular expressions

Martin Pola

Matching strings

A regular expression is a combination of a *pattern* and a *test string*.

Matching strings

A regular expression is a combination of a *pattern* and a *test string*.

The pattern is sent to a *regular expression engine* to see if the test string matches or not.

Matching strings

A regular expression is a combination of a *pattern* and a *test string*.

The pattern is sent to a *regular expression engine* to see if the test string matches or not.

Two purposes.

- Testing if a string is valid
- Extracting information from a string

Examples

Expression	Description
`[0-9]`	Any single digit
`[0-9A-ZÅÄÖ]+`	Sequence of digits and capital letters
`[A-ZÅÄÖ][a-zåäö]+`	One capital letter followed by at least one lowercase letter
`[0-9]{4}`	Four digits
`[0-9]{4,6}`	Four to six digits
`[0-9]{4,}`	At least four digits
`[0-9]{,4}`	Up to four digits
`.{3,}`	Anything, at least tree times
`[a-zåäö]?`	Zero or one lowercase letters
`[0-9]{4}-[0-9]{2}-[0-9]{2}`	Date-like information (YYYY-MM-DD)
`(Hello|Hey|Hi|([0-9]+))`	A greeting or a number

Regular expressions in Java

Java provides us with the `Pattern` and `Matcher` classes, through the `java.util.regex` package.

```
1  Pattern pattern = Pattern.compile("[0-9]+");
2  Matcher matcher = pattern.matcher("Our 123 string");
3
4  if (matcher.find()) {
5      System.out.println("There are digits.");
6  }
7
8  if (matcher.matches()) {
9      System.out.println("There are only digits.");
10 }
```

Regular expressions in Java

Java provides us with the `Pattern` and `Matcher` classes, through the `java.util.regex` package.

```
1  Pattern pattern = Pattern.compile("[0-9]+");
2  Matcher matcher = pattern.matcher("Our 123 string");
3
4  if (matcher.find()) {
5      System.out.println("There are digits.");
6  }
7
8  if (matcher.matches()) {
9      System.out.println("There are only digits.");
10 }
```

A circumflex (^) can be used to denote the start of a string, and a dollar sign ($) to denote the end of a string.

Escaping

Sometimes you want the literal character instead of its special meaning within regular expressions. For that, *escaping* is necessary.

Escaping is done with a backslash (\), just like it's done when dealing with special characters in strings (e.g. `\n`, `\t`).

Escaping

Sometimes you want the literal character instead of its special meaning within regular expressions. For that, *escaping* is necessary.

Escaping is done with a backslash (`\`), just like it's done when dealing with special characters in strings (e.g. `\n`, `\t`).

Consider this regular expression to match a Swedish phone number:
`(\+46|0046|0)[1-9][0-9]{8}`

Escaping

Sometimes you want the literal character instead of its special meaning within regular expressions. For that, *escaping* is necessary.

Escaping is done with a backslash (\), just like it's done when dealing with special characters in strings (e.g. `\n`, `\t`).

Consider this regular expression to match a Swedish phone number:
`(\+46|0046|0)[1-9][0-9]{8}`

In Java, it would correspond to:

```
1 Pattern.compile("(\\+46|0046|0)[1-9][0-9]{8}");
```

Escaping

Sometimes you want the literal character instead of its special meaning within regular expressions. For that, *escaping* is necessary.

Escaping is done with a backslash (\), just like it's done when dealing with special characters in strings (e.g. `\n`, `\t`).

Consider this regular expression to match a Swedish phone number:
`(\+46|0046|0)[1-9][0-9]{8}`

In Java, it would correspond to:

```
1 Pattern.compile("(\\+46|0046|0)[1-9][0-9]{8}");
```

Other characters to escape:
`.  ?  -  [ ]  ^$  |  ( )`