vogella.com Tutorials Training Books Contact us

293

by Lars Vogel

Java Regex Tutorial

Lars Vogel

Version 2.2

03.06.2013

Revision History			
Revision 0.1	07.12.2007	Lars Vogel	created
Revision 0.2 - 2.2	30.12.2008 - 03.06.2013	Lars Vogel	bug fixes and enhancements

Java and Regular Expressions

This tutorial introduces the usage of regular expressions and describers their with Java. It also provides several Java regular expression examples.

Table of Contents

1. Regular Expressions

1.1. Overview

A regular expression defines a search pattern for strings. Regular expressions can be used to search, edit and manipulate text. The pattern defined by the regular expression may match one or several times or not at all for a given string.

BACK TO TOP

The abbreviation for regular expression is regex.

to the text (string) .

The pattern defined by the regex is applied on the text from left to right. Once a source character has been used in a match, it cannot be reused. For example the regex aba will match ababababa only two times (aba_aba__).

A simple example for a regular expression is a (literal) string. For example the Hello World regex will match the "Hello World" string.

. (dot) is another example for an regular expression. A dot matches any single character; it would match for example "a" or "z" or "1".

1.2. Support for regular expressions in programming languages

Regular expressions are supported by most programming languages, e.g. Java, Perl, Groovy, etc. Unfortunately each language supports regular expressions slightly different.

This tutorial describes the usage of regular expression within the Java programming language and within the Eclipse IDE.

2. Prerequisites

The following tutorial assumes that you have basic knowledge of the Java programming language.

Some of the following examples use JUnit to validate the result. You should be able to adjust them in case if you do not want to use JUnit. To learn about JUnit please see **JUnit Tutorial** .

3. Rules of writing regular expressions

The following description is an overview of available meta characters which can be used in regular expressions. This chapter is supposed to be a references for the different regex elements.

3.1. Common matching symbols

Table 1.

Regular Expression	Description
`.`	Matches any character
`^regex`	regex must match at the beginning of the line
`regex$`	Finds regex must match at the end of the line
`[abc]`	Set definition, can match the letter a or b or c
`[abc][vz]`	Set definition, can match a or b or c followed by either v or z
`[^abc]`	When a "^" appears as the first character inside [] when it negates the pattern. This can match any character except a or b or c
`[a-d1-7]`	Ranges, letter between a and d and figures from 1 to 7, will not match d1
`X|Z`	Finds X or Z
`XZ`	Finds X directly followed by Z
`$`	Checks if a line end follows

BACK TO TOP

3.2. Metacharacters

to use, e.g. \d instead of [0..9].

Table 2.

Regular Expression	Description
`\d`	Any digit, short for [0-9]
`\D`	A non-digit, short for [^0-9]
`\s`	A whitespace character, short for [\t\n\x0b\r\f]
`\S`	A non-whitespace character, for short for [^\s]
`\w`	A word character, short for [a-zA-Z_0-9]
`\W`	A non-word character [^\w]
`\S+`	Several non-whitespace characters
`\b`	Matches a word boundary. A word character is [a-zA-Z0-9_] and \b matches its bounderies.

3.3. Quantifier

A quantifier defines how often an element can occur. The symbols ?, *, + and {} define the quantity of the regular expressions

Table 3.

Regular Expression	Description	Examples
`*`	Occurs zero or more times, is short for {0,}	X* - Finds no or several letter X, .* - any character sequence
`+`	Occurs one or more times, is short for {1,}	X+ - Finds one or several letter X
`?`	Occurs no or one times, ? is short for {0,1}	X? -Finds no or exactly one letter X
`{X}`	Occurs X number of times, {} describes the order of the preceding liberal	\d{3} - Three digits, .{10} - any character sequence of length 10
`{X,Y}`	Occurs between X and Y times,	\d{1,4}- \d must occur at least once and at a maximum of four
`*?`	? after a quantifier makes it a reluctant quantifier, it tries to find the smallest match.	

3.4. Grouping and Backreference

You can group parts of your regular expression. In your pattern you group elements via round brackets, e.g. "()". This allows you to assign a repetition operator the a complete group.

In addition these groups also create a backreference to the part of the regular expression. This captures the group. A backreference stores the part of the `String` which matched the group. This allows you to use this part in the replacement.

Via the $ you can refer to a group. $1 is the first group, $2 the second, etc.

Lets for example assume you want to replace all whitespace between a letter followed by a point or a comma. This would involve that the point or the comma is part of the pattern. Still it should be included in the result

BACK TO TOP

```
// Removes whitespace between a word character and . or ,
```

This example extracts the text between a title tag.

```
// Extract the text between the two title elements
pattern = "(?i)(<title.*?>)(.+?)(</title>)";
String updated = EXAMPLE_TEST.replaceAll(pattern, "$2");
```

3.5. Negative Lookahead

Negative Lookahead provide the possibility to exclude a pattern. With this you can say that a string should not be followed by another string.

Negative Lookaheads are defined via `(?!pattern)`. For example the following will match a if a is not followed by b.

```
a(?!b)
```

3.6. Backslashes in Java

The backslash is an escape character in Java Strings. e.g. backslash has a predefined meaning in Java. You have to use "\\" to define a single backslash. If you want to define "\w" then you must be using "\\w" in your regex. If you want to use backslash you as a literal you have to type \\\\ as \ is also a escape character in regular expressions.

4. Using Regular Expressions with String.matches()

4.1. Overview

`Strings` in Java have build in support for regular expressions. `Strings` have three build in methods for regular expressions, e.g. `matches()`, `split())`, `replace()`..

These methods are not optimized for performance. We will later use classes which are optimized for performance.

Table 4.

Method	Description
s.matches("regex")	Evaluates if "regex" matches s. Returns only true if the WHOLE string can be matched
s.split("regex")	Creates array with substrings of s divided at occurrence of "regex". "regex" is not included in the result.
s.replace("regex"), "replacement"	Replaces "regex" with "replacement

Create for the following example the Java project `de.vogella.regex.test`.

BACK TO TOP

```
package de.vogella.regex.test;

public class RegexTestStrings {
  public static final String EXAMPLE_TEST = "This is my small example "
      + "string which I'm going to " + "use for pattern matching.";

  public static void main(String[] args) {
    System.out.println(EXAMPLE_TEST.matches("\\w.*"));
    String[] splitString = (EXAMPLE_TEST.split("\\s+"));
    System.out.println(splitString.length);// Should be 14
    for (String string : splitString) {
      System.out.println(string);
    }
    // Replace all whitespace with tabs
}
```

4.2. Examples

The following class gives several examples for the usage of regular expressions with strings. See the comment for the purpose.

If you want to test these examples, create for the Java project `de.vogella.regex.string`.

```
package de.vogella.regex.string;

public class StringMatcher {
  // Returns true if the string matches exactly "true"
  public boolean isTrue(String s){
    return s.matches("true");
  }
  // Returns true if the string matches exactly "true" or "True"
  public boolean isTrueVersion2(String s){
    return s.matches("[tT]rue");
  }

  // Returns true if the string matches exactly "true" or "True"
  // or "yes" or "Yes"
  public boolean isTrueOrYes(String s){
    return s.matches("[tT]rue|[yY]es");
  }

  // Returns true if the string contains exactly "true"
  public boolean containsTrue(String s){
    return s.matches(".*true.*");
  }


  // Returns true if the string contains of three letters
  public boolean isThreeLetters(String s){
    return s.matches("[a-zA-Z]{3}");
    // Simpler from for
//    return s.matches("[a-Z][a-Z][a-Z]");
  }



  // Returns true if the string does not have a number at the beginning
  public boolean isNoNumberAtBeginning(String s){
    return s.matches("^[^\\d].*");
  }
  // Returns true if the string contains a arbitrary number of characters except b
  public boolean isIntersection(String s){
    return s.matches("([\\w&&[^b]])*");
  }
  // Returns true if the string contains a number less then 300
  public boolean isLessThenThreeHundret(String s){
    return s.matches("[^0-9]*[12]?[0-9]{1,2}[^0-9]*");
  }

}
```

And a small JUnit Test to validates the examples.

```
package de.vogella.regex.string;

import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

public class StringMatcherTest {
  private StringMatcher m;

  @Before
  public void setup(){
    m = new StringMatcher();
  }

  @Test
  public void testIsTrue() {
    assertTrue(m.isTrue("true"));
    assertFalse(m.isTrue("true2"));
    assertFalse(m.isTrue("True"));
  }
```

BACK TO TOP

```
  public void testIsTrueVersion2() {
    assertTrue(m.isTrueVersion2("true"));
    assertFalse(m.isTrueVersion2("true2"));
    assertTrue(m.isTrueVersion2("True"));;
  }

  @Test
  public void testIsTrueOrYes() {
    assertTrue(m.isTrueOrYes("true"));
    assertTrue(m.isTrueOrYes("yes"));
    assertTrue(m.isTrueOrYes("Yes"));
    assertFalse(m.isTrueOrYes("no"));
  }

  @Test
  public void testContainsTrue() {
    assertTrue(m.containsTrue("thetruewithin"));
  }

  @Test
  public void testIsThreeLetters() {
    assertTrue(m.isThreeLetters("abc"));
    assertFalse(m.isThreeLetters("abcd"));
  }

  @Test
  public void testisNoNumberAtBeginning() {
    assertTrue(m.isNoNumberAtBeginning("abc"));
    assertFalse(m.isNoNumberAtBeginning("1abcd"));
    assertTrue(m.isNoNumberAtBeginning("a1bcd"));
    assertTrue(m.isNoNumberAtBeginning("asdfdsf"));
  }

  @Test
  public void testisIntersection() {
    assertTrue(m.isIntersection("1"));
    assertFalse(m.isIntersection("abcksdfkdskfsdfdsf"));
    assertTrue(m.isIntersection("skdskfjsmcnxmvjwque484242"));
  }


  @Test
  public void testLessThenThreeHundret() {
    assertTrue(m.isLessThenThreeHundret("288"));
    assertFalse(m.isLessThenThreeHundret("3288"));
    assertFalse(m.isLessThenThreeHundret("328 8"));
    assertTrue(m.isLessThenThreeHundret("1"));
    assertTrue(m.isLessThenThreeHundret("99"));
    assertFalse(m.isLessThenThreeHundret("300"));
  }

}
```

5. Pattern and Matcher

For advanced regular expressions the `java.util.regex.Pattern` and `java.util.regex.Matcher` classes are used.

You first create a `Pattern` object which defines the regular expression. This `Pattern` object allows you to create a `Matcher` object for a given string. This `Matcher` object then allows you to do regex operations on a `String`.

```
package de.vogella.regex.test;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegexTestPatternMatcher {
  public static final String EXAMPLE_TEST = "This is my small example string which I'm going to

  public static void main(String[] args) {
    Pattern pattern = Pattern.compile("\\w+");
    // In case you would like to ignore case sensitivity you could use this
    // statement
    // Pattern pattern = Pattern.compile("\\s+", Pattern.CASE_INSENSITIVE);
    Matcher matcher = pattern.matcher(EXAMPLE_TEST);
    // Check all occurance
    while (matcher.find()) {
      System.out.print("Start index: " + matcher.start());
      System.out.print(" End index: " + matcher.end() + " ");
```

BACK TO TOP

```
    // Now create a new pattern and matcher to replace whitespace with tabs
    Pattern replace = Pattern.compile("\\s+");
    Matcher matcher2 = replace.matcher(EXAMPLE_TEST);
    System.out.println(matcher2.replaceAll("\t"));
  }
}
```

6. Java Regex Examples

The following lists typical examples for the usage of regular expressions. I hope you find similarities to your examples.

6.1. Or

Task: Write a regular expression which matches a text line if this text line contains either the word "Joe" or the word "Jim" or both.

Create a project `de.vogella.regex.eitheror` and the following class.

```
package de.vogella.regex.eitheror;

import org.junit.Test;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

public class EitherOrCheck {
  @Test
  public void testSimpleTrue() {
    String s = "humbapumpa jim";
    assertTrue(s.matches(".*(jim|joe).*"));
    s = "humbapumpa jom";
    assertFalse(s.matches(".*(jim|joe).*"));
    s = "humbaPumpa joe";
    assertTrue(s.matches(".*(jim|joe).*"));
    s = "humbapumpa joe jim";
    assertTrue(s.matches(".*(jim|joe).*"));
  }
}
```

6.2. Phone number

Task: Write a regular expression which matches any phone number.

A phone number in this example consists either out of 7 numbers in a row or out of 3 number a (white)space or a dash and then 4 numbers.

```
package de.vogella.regex.phonenumber;

import org.junit.Test;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;


public class CheckPhone {

  @Test
  public void testSimpleTrue() {
    String pattern = "\\d\\d\\d([,\\s])?\\d\\d\\d\\d";
    String s= "1233323322";
    assertFalse(s.matches(pattern));
    s = "1233323";
    assertTrue(s.matches(pattern));
    s = "123 3323";
    assertTrue(s.matches(pattern));
  }
}
```

6.3. Check for a certain number range

BACK TO TOP

The following example will check if a text contains a number with 3 digits.

```
package de.vogella.regex.numbermatch;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.junit.Test;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

public class CheckNumber {


  @Test
  public void testSimpleTrue() {
    String s= "1233";
    assertTrue(test(s));
    s= "0";
    assertFalse(test(s));
    s = "29 Kasdkf 2300 Kdsdf";
    assertTrue(test(s));
    s = "99900234";
    assertTrue(test(s));
  }




  public static boolean test (String s){
    Pattern pattern = Pattern.compile("\\d{3}");
    Matcher matcher = pattern.matcher(s);
    if (matcher.find()){
      return true;
    }
    return false;
  }

}
```

6.4. Building a link checker

The following example allows you to extract all valid links from a webpage. It does not consider links with start with "javascript:" or "mailto:".

Create a Java project called de.vogella.regex.weblinks and the following class:

```
package de.vogella.regex.weblinks;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class LinkGetter {
  private Pattern htmltag;
  private Pattern link;

  public LinkGetter() {
    htmltag = Pattern.compile("<a\\b[^>]*href=\"[^>]*>(.*?)</a>");
    link = Pattern.compile("href=\"[^>]*\">");
  }

  public List<String> getLinks(String url) {
    List<String> links = new ArrayList<String>();
    try {
      BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(new URL(url).openS
      String s;
      StringBuilder builder = new StringBuilder();
      while ((s = bufferedReader.readLine()) != null) {
        builder.append(s);
      }

      Matcher tagmatch = htmltag.matcher(builder.toString());
      while (tagmatch.find()) {
        Matcher matcher = link.matcher(tagmatch.group());
        matcher.find();
        String link = matcher.group().replaceFirst("href=\"", "")
```

BACK TO TOP

```
        if (valid(link)) {
          links.add(makeAbsolute(url, link));
        }
      }
    } catch (MalformedURLException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
    return links;
  }

  private boolean valid(String s) {
    if (s.matches("javascript:.*|mailto:.*")) {
      return false;
    }
    return true;
  }

  private String makeAbsolute(String url, String link) {
    if (link.matches("http://.*")) {
      return link;
    }
    if (link.matches("/.*") && url.matches(".*$[^/]")) {
      return url + "/" + link;
    }
    if (link.matches("[^/].*") && url.matches(".*[^/]")) {
      return url + "/" + link;
    }
    if (link.matches("/.*") && url.matches(".*[/]")) {
      return url + link;
    }
    if (link.matches("/.*") && url.matches(".*[^/]")) {
      return url + link;
    }
    throw new RuntimeException("Cannot make the link absolute. Url: " + url
        + " Link " + link);
  }
}
```

6.5. Finding duplicated words

The following regular expression matches duplicated words.

```
\b(\w+)\s+\1\b
```

\b is a word boundary and \1 references to the captured match of the first group, i.e. the first word.

The *(?!-in)\b(\w+) \1\b* finds duplicate words if they do not start with "-in".

> **Tip**
>
> Add (?s) to search across multiple lines.

7. Thank you

Please help me to support this article:

Flattr this	Donate

8. Questions and Discussion

If you find errors in this tutorial please notify me (see the top of the page). Please note that due to the high volume of feedback I receive, I cannot answer questions to your implementation. Ensure you have read the **vogella FAQ**, I also don't answer questions answered in the FAQ.

BACK TO TOP

9. Links and Literature

Website about Regular Expressions

Regular Expressions in Java

BACK TO TOP