Regular Expressions

Introduction to Information Retrieval
INF 141/ CS 121
Donald J. Patterson

Content adapted from www.vogella.com

Regular Expressions

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.

The abbreviation for *regular expression* is *regex*.

The process of analyzing or modifying a text with a *regex* is called: *The regular expression is applied 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 a regular expression. A dot matches any single character; it would match, for example, "a" or "z" or "1".

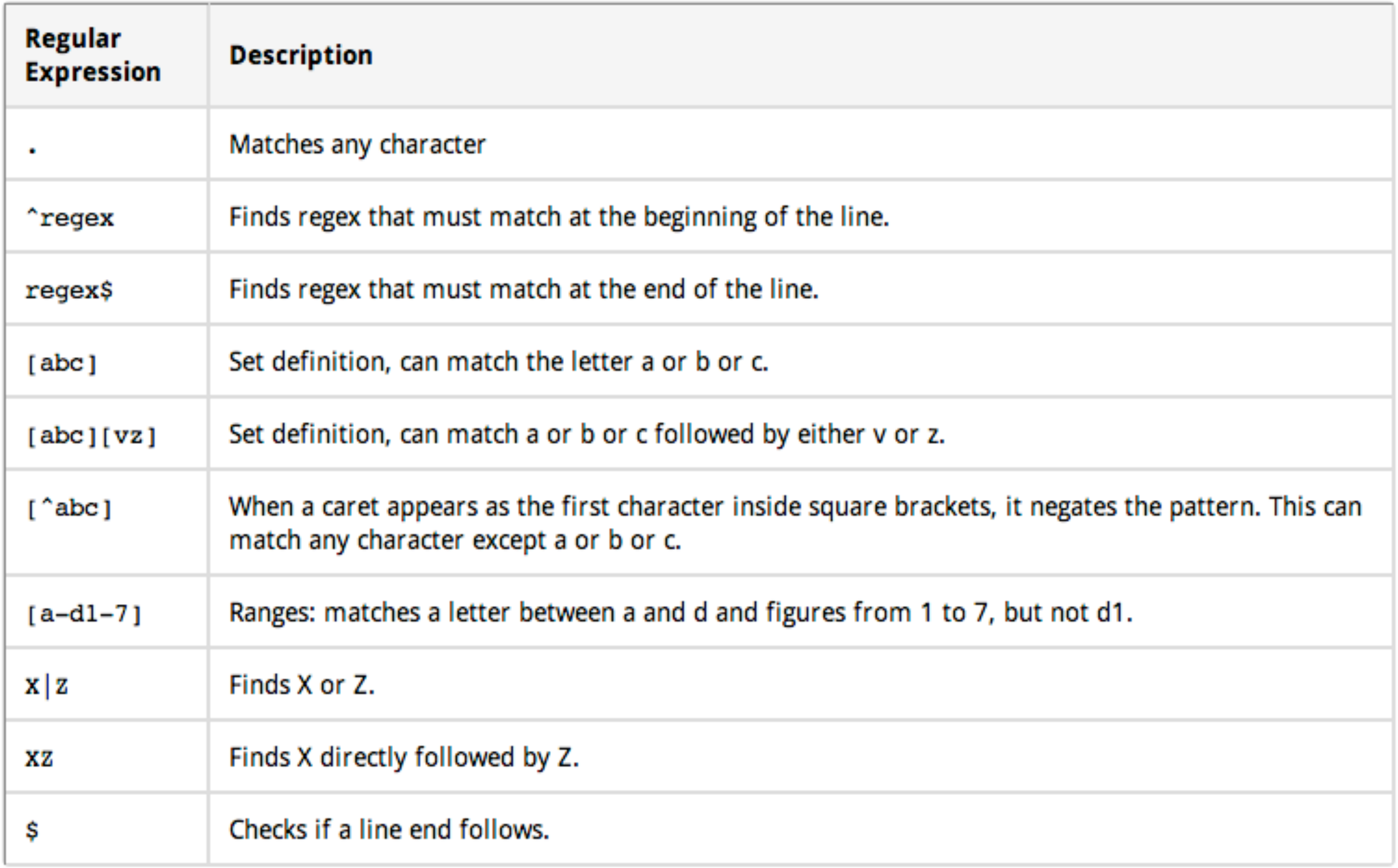
Regular Expressions

- Most programming languages support regex's
- Most have there own quirks
- These instructions are for regex's in Java / Eclipse

Regular Expressions

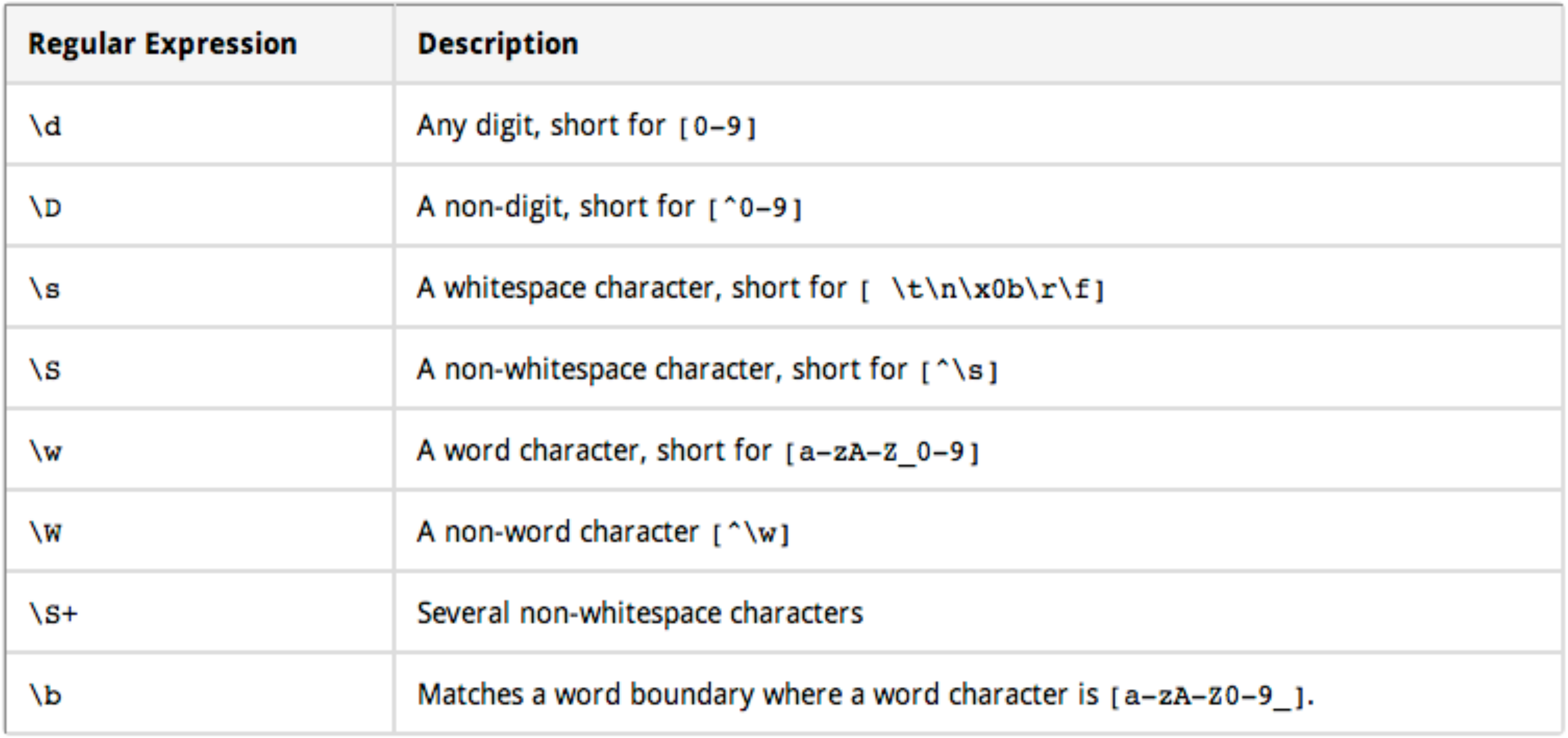
- Regular Expressions are about finding patterns in text
- Based on a line by line paradigm

Regular Expressions



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

Regular Expressions

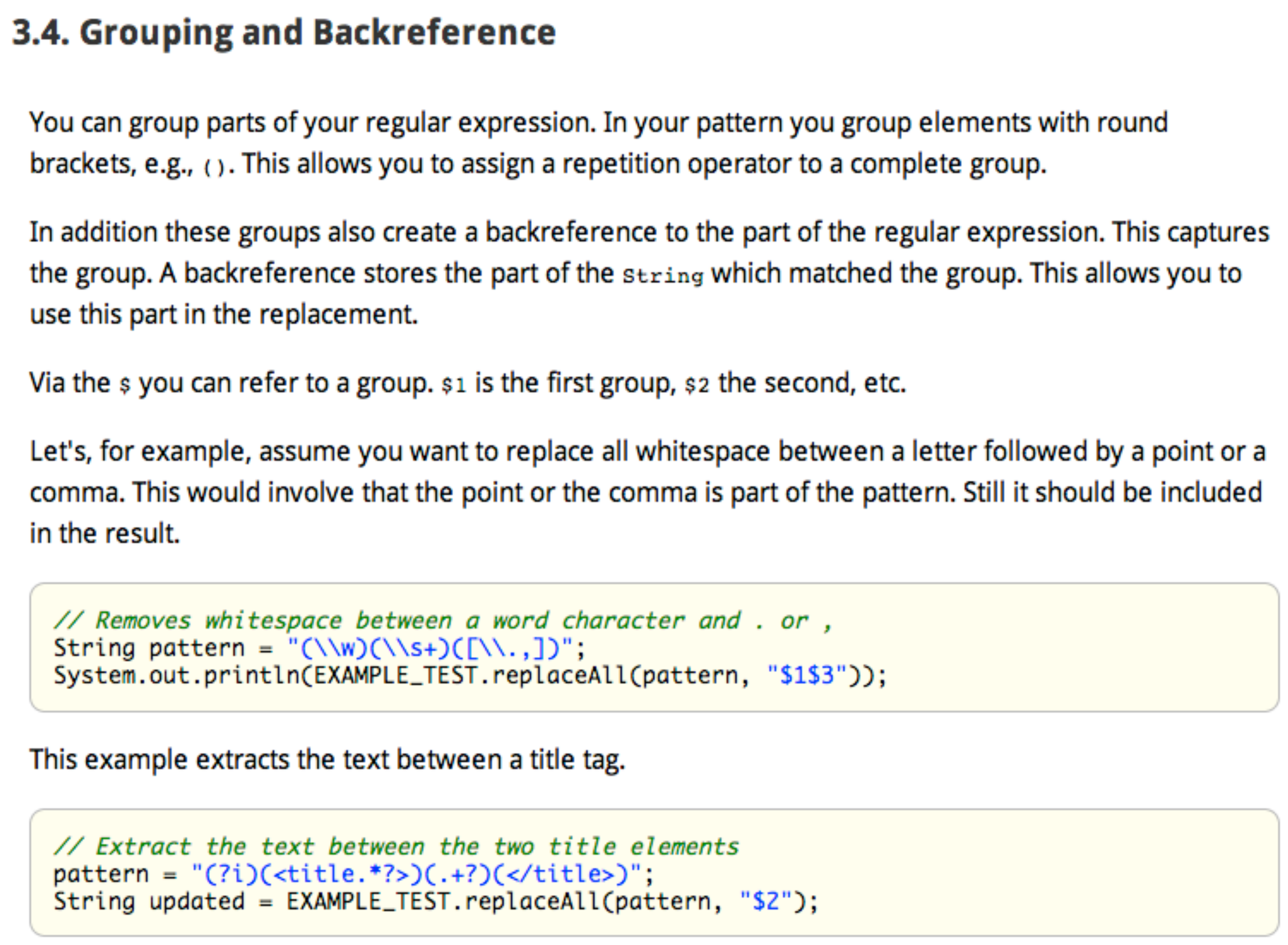


Regular Expression	Description
`\d`	Any digit, short for `[0-9]`
`\D`	A non-digit, short for `[^0-9]`
`\s`	A whitespace character, short for `[ \t\n\x0b\r\f]`
`\S`	A non-whitespace character, short for `[^\s]`
`\w`	A word character, short for `[a-zA-Z_0-9]`
`\W`	A non-word character `[^\w]`
`\S+`	Several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

Regular Expressions

Regular Expression	Description	Examples
`*`	Occurs zero or more times, is short for `{0,}`	`X*` finds no or several letter X, `.*` finds 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}` searches for three digits, `.{10}` for any character sequence of length 10.
`{X,Y}`	Occurs between X and Y times,	`\d{1,4}` means `\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.	

Regular Expressions



3.4. Grouping and Backreference

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

Let's, 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.

```java
// Removes whitespace between a word character and . or ,
String pattern = "(\\w)(\\s+)([\\.,])";
System.out.println(EXAMPLE_TEST.replaceAll(pattern, "$1$3"));
```

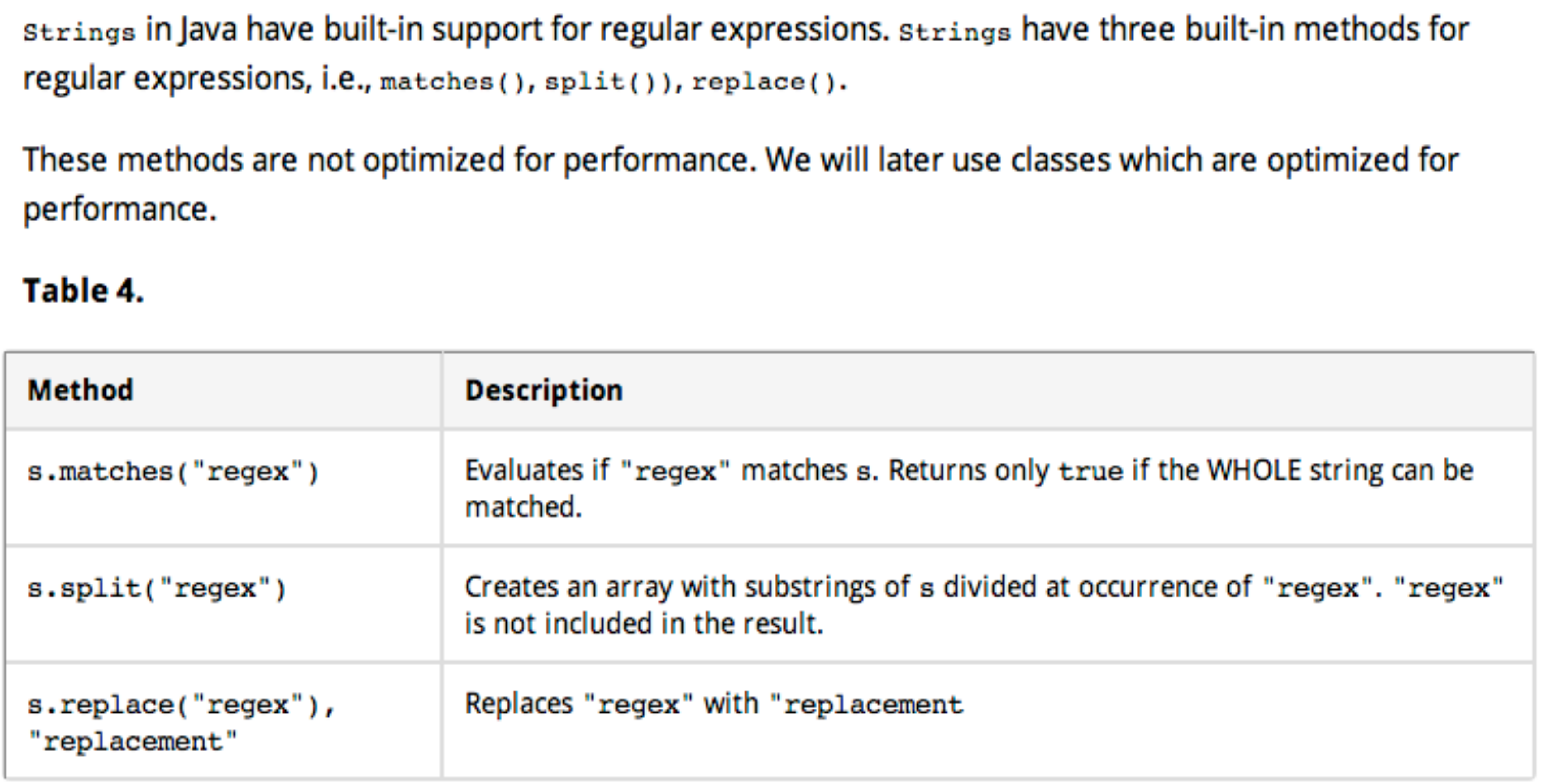
This example extracts the text between a title tag.

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

Backslashes

The backslash `\` is an escape character in Java Strings. That means backslash has a predefined meaning in Java. You have to use double backslash `\\` 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 as a literal, you have to type `\\\\` as `\` is also an escape character in regular expressions.

Backslashes



`Strings` in Java have built-in support for regular expressions. `Strings` have three built-in methods for regular expressions, i.e., `matches()`, `split())`, `replace()`.

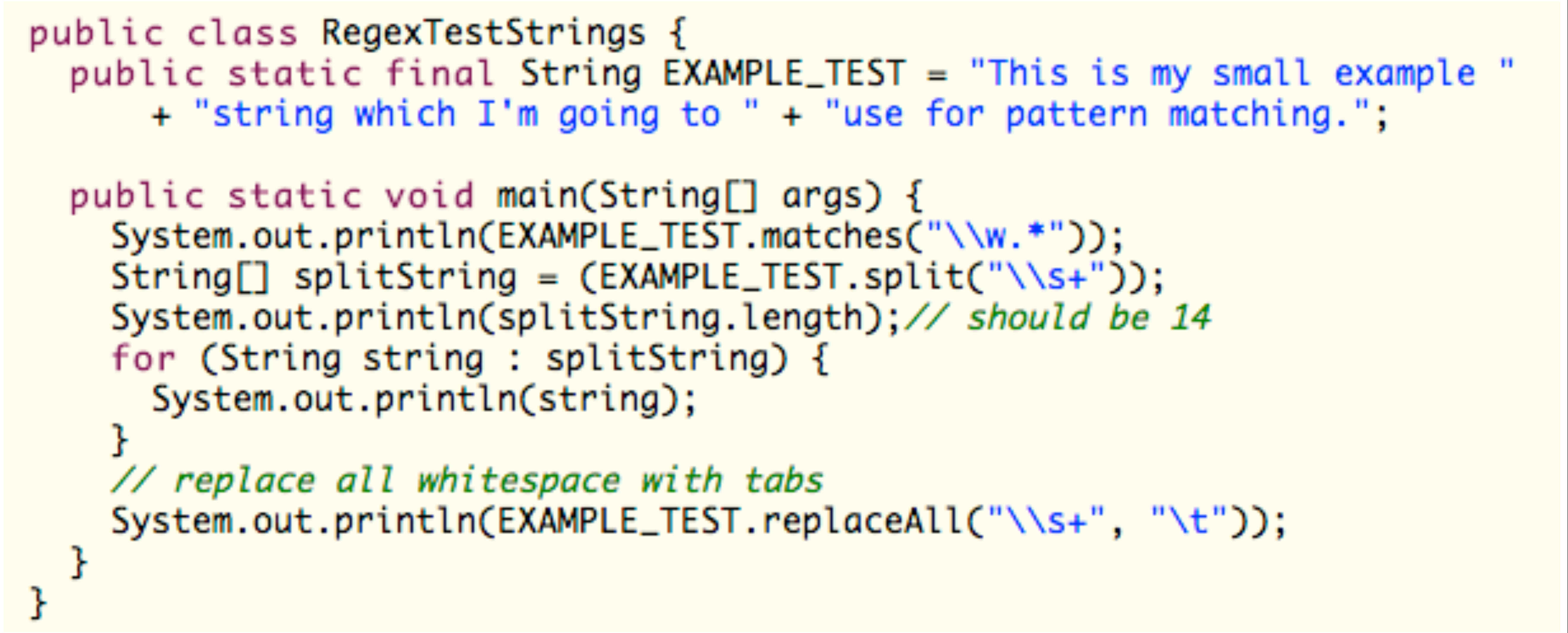
These methods are not optimized for performance. We will later use classes which are optimized for performance.

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 an 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`

www.vogella.com

Backslashes



```java
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
    System.out.println(EXAMPLE_TEST.replaceAll("\\s+", "\t"));
  }
}
```

Pattern and Matcher

```java
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 use for pattern matching.";

  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() + " ");
      System.out.println(matcher.group());
    }
    // 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"));
  }
}
```

Example: Phone Number

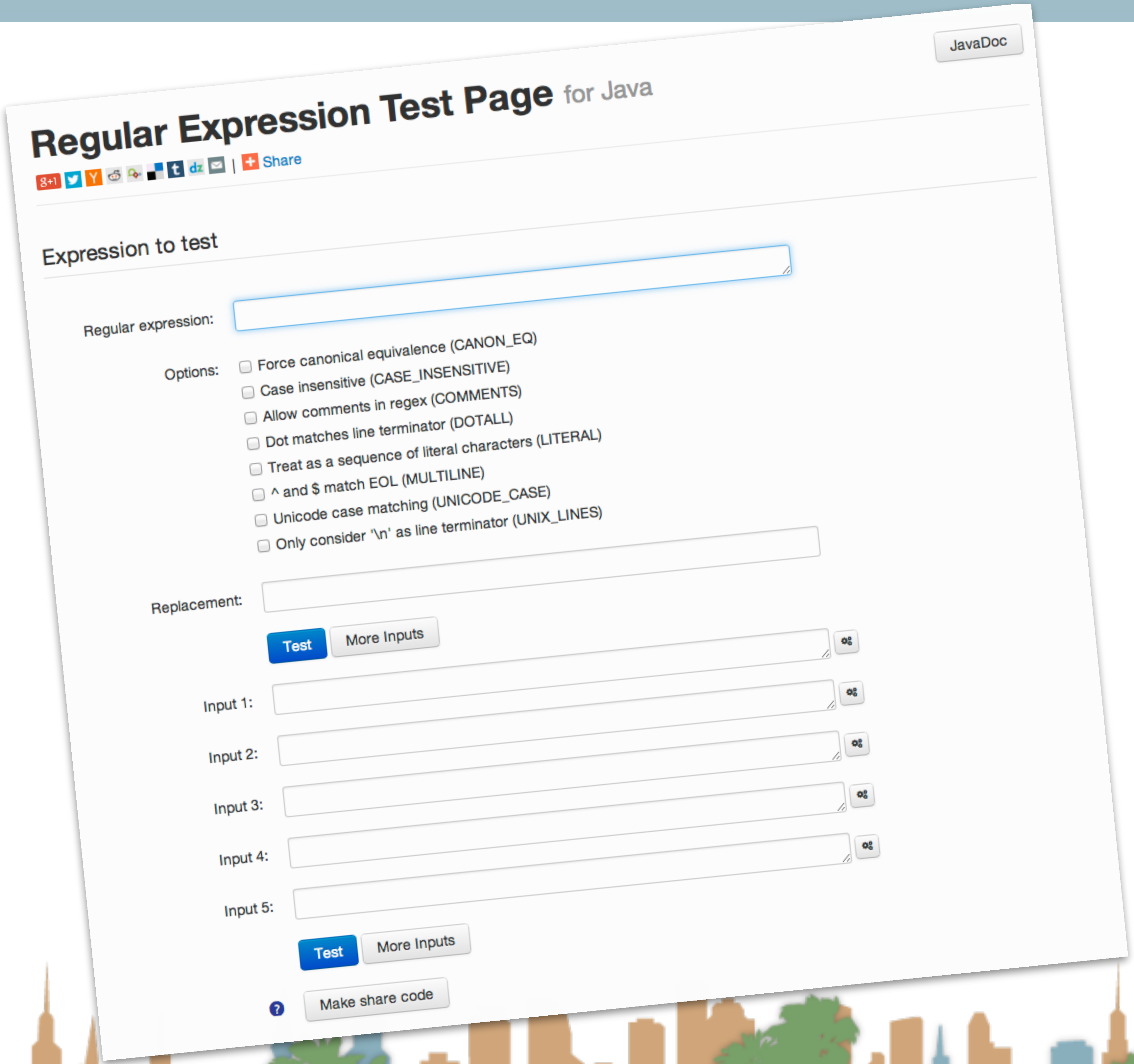
```java
import org.junit.Test;

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));
  }
}
```

Online checker


JavaDoc

Regular Expression Test Page for Java

Share

Expression to test

Regular expression:

Options:
- Force canonical equivalence (CANON_EQ)
- Case insensitive (CASE_INSENSITIVE)
- Allow comments in regex (COMMENTS)
- Dot matches line terminator (DOTALL)
- Treat as a sequence of literal characters (LITERAL)
- ^ and $ match EOL (MULTILINE)
- Unicode case matching (UNICODE_CASE)
- Only consider '\n' as line terminator (UNIX_LINES)

Replacement:

Test | More Inputs

Input 1:

Input 2:

Input 3:

Input 4:

Input 5:

Test | More Inputs

Make share code

RegexPlanet

LUCI