

# Regular Expressions

Brandon Krakowsky



# Regular Expressions

# Regular Expressions

- A regular expression (or **regex**) is a special sequence of characters that describes a pattern used for *searching, editing, and manipulating* text and data



About Regex: <https://www.regular-expressions.info/quickstart.html>

Java Specific Tutorials: <https://docs.oracle.com/javase/tutorial/essential/regex/>

# Regular Expressions

- A regular expression (or **regex**) is a special sequence of characters that describes a pattern used for *searching, editing, and manipulating* text and data
- For example, regular expressions are widely used to define the constraint on Strings in *password* and *email validation*



About Regex: <https://www.regular-expressions.info/quickstart.html>

Java Specific Tutorials: <https://docs.oracle.com/javase/tutorial/essential/regex/>

# Regular Expressions

- A regular expression (or **regex**) is a special sequence of characters that describes a pattern used for *searching, editing, and manipulating* text and data
- For example, regular expressions are widely used to define the constraint on Strings in *password* and *email validation*
- The most basic regular expression consists of a literal String that matches the first occurrence of that String

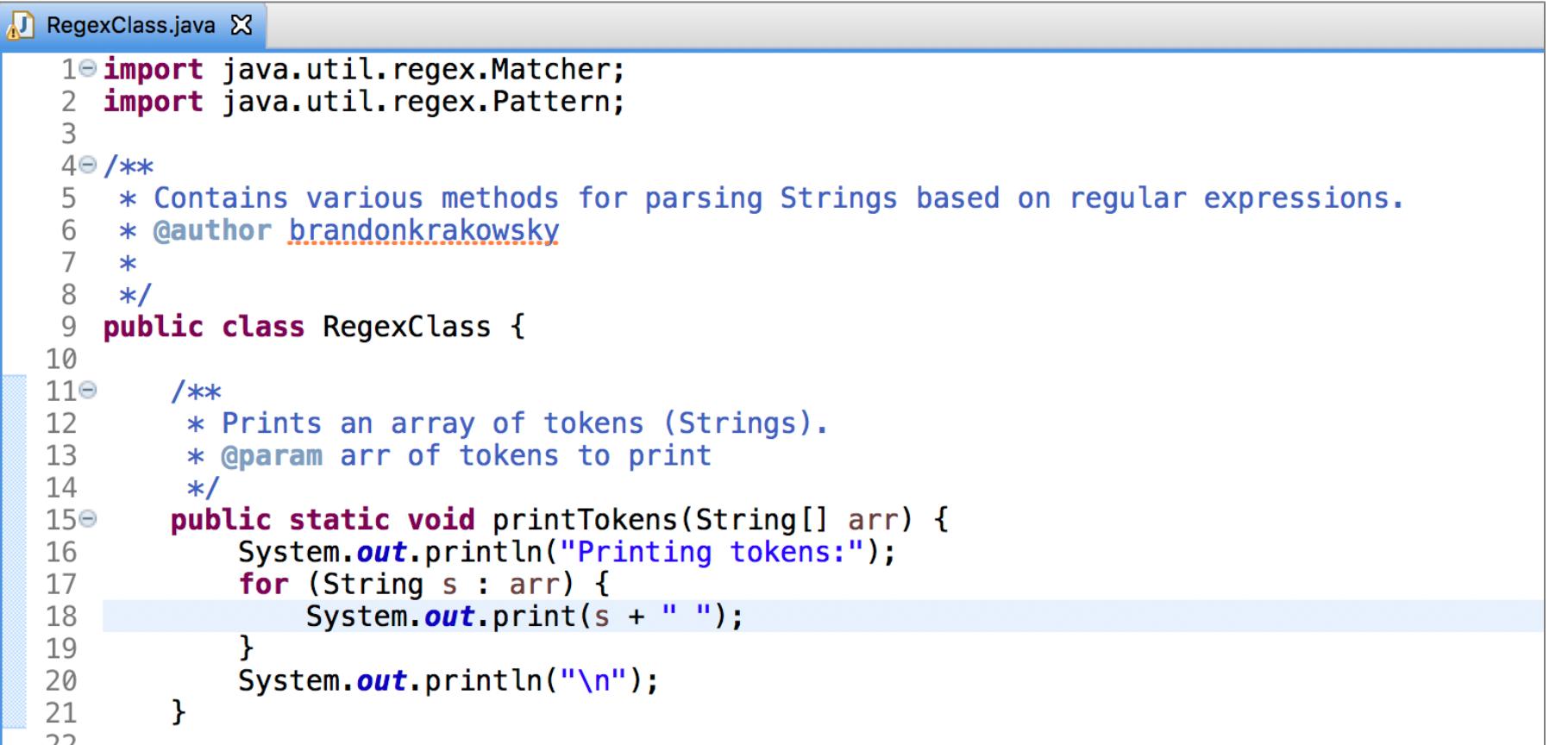


About Regex: <https://www.regular-expressions.info/quickstart.html>

Java Specific Tutorials: <https://docs.oracle.com/javase/tutorial/essential/regex/>

# Regex Examples

# RegexClass



```
1 import java.util.regex.Matcher;
2 import java.util.regex.Pattern;
3
4 /**
5  * Contains various methods for parsing Strings based on regular expressions.
6  * @author brandonkrakowsky
7  *
8 */
9 public class RegexClass {
10
11 /**
12  * Prints an array of tokens (Strings).
13  * @param arr of tokens to print
14  */
15 public static void printTokens(String[] arr) {
16     System.out.println("Printing tokens:");
17     for (String s : arr) {
18         System.out.print(s + " ");
19     }
20     System.out.println("\n");
21 }
22 }
```

# Split a String

```
22
23  /**
24   * Splits given string based on given regex pattern.
25   * @param str to split
26   * @param regex to match
27   * @return String array of tokens (Strings)
28  */
29  public static String[] splitString(String str, String regex) {
30      //split the given string str based on the given regex
31      return str.split(regex);
32  }
33
```

# Split a String

```
101  
162  public static void main(String[] args) {  
163  
164      String str = "the cow jumped over the moon";  
165      //split the String based on a single space  
166      String[] tokens = RegexClass.splitString(str, " ");  
167      RegexClass.printTokens(tokens);  
168  
169      //split the String based on "the"  
170      tokens = RegexClass.splitString(str, "the");  
171      RegexClass.printTokens(tokens);  
172
```

# Split a String

```
186
187     /*
188      * split the String based on various amounts of whitespace
189      * \s matches a single whitespace character
190      * \s+ matches 1 or more whitespace characters
191      */
192
193     /*
194      * \ (backslash) is a special character.
195      * if you want to use it as a literal in a regex,
196      * you need to escape it with another backslash,
197      * so we use \\s+ to match 1 or more whitespace characters
198     */
199     str = "the          cow    jumped over        the\n"
200           + "  moon";
201     tokens = RegexClass.splitString(str, "\\s+");
202     RegexClass.printTokens(tokens);
203
```

# Replace All with a Pattern

```
33
34  /**
35   * Replaces all instances of the given pattern
36   * with the given replacement in the given str.
37   * @param str to replace values in
38   * @param pattern to replace
39   * @param replace updated value
40   * @return Updated str
41   */
42  public static String replaceAllWithPattern(String str, String pattern, String replacement) {
43      //replace the given pattern with the given replacement in str
44      return str.replaceAll(pattern, replacement);
45  }
46
```

# Replace All with a Pattern

```
189  
190     //replace multiple whitespace characters with a single whitespace character  
191     String updatedStr = RegexClass.replaceAllWithPattern(str, "\\s+", " ");  
192     System.out.println("Replace whitespace: " + updatedStr);  
193     System.out.println("");  
194
```

# Get Parts of a Phone Number

```
47  /**
48  * Parses and returns various part of a phone number.
49  * @param phone number to parse
50  * @param part of phone number to return: 1 (area code), 2 (prefix) or 3 (number)
51  * @return Part of phone number
52  */
53  public static String getPhonePart(String phone, int part) {
54      if (part < 1 || part > 3) {
55          throw new IllegalArgumentException("Part must be 1, 2 or 3.");
56      }
57
58      //parenthesis() indicate groups
59      //\b matches an empty string or non-word character,
60      //at the beginning or end of pattern
61
62      //[-.\s]+ indicates a character class,
63      //matching one of several characters (with repetition): -, ., whitespace
64      String regex = "\b(\d{3})[-.\s]+(\d{3})[-.\s]+(\d{4})\b";
65
66      Pattern p = Pattern.compile(regex);
67      Matcher m = p.matcher(phone);
68
69      String phonePart = "";
70      while (m.find()) {
71          //get designated group
72          phonePart = m.group(part);
73      }
74
75      //return group
76      return phonePart;
77  }
```

# Get Parts of a Phone Number

```
189
190     //get parts of phone number
191     String areaCode = RegexClass.getPhonePart("123-982-6342", 1); //get area code
192     String prefix = RegexClass.getPhonePart("800 787 2394", 2); //get prefix
193     String number = RegexClass.getPhonePart(" 508.717.0989  ", 3); //get line number
194     System.out.println("Phone number parts: " + areaCode + " " + prefix + " " + number);
195     System.out.println("");
196
```

# Replace an Area Code

```
70
77  /**
78   * Replaces the area code in the given phone number with the given new area code.
79   * @param phone to replace area code in
80   * @param newArea for phone
81   * @return Updated phone number
82  */
83  public static String replaceAreaCode(String phone, String newArea) {
84      // [0-9] indicates a character class,
85      // matching one of several characters: 0 - 9
86      // {3} indicates a specific amount of repetition
87      return phone.replaceFirst("[0-9]{3}", newArea);
88  }
89
```

# Replace an Area Code

```
196
197     //replace area code
198     String phone = "123-982-6342";
199     String updatedPhone = RegexClass.replaceAreaCode(phone, "888");
200     System.out.println("Updated phone: " + updatedPhone);
201     System.out.println("");
202
```

# Format a Phone Number

```
72
92    /**
93     * Formats a given phone number in the format 1234567890,
94     * to the format (123) 456-7890.
95     * @param phone to format
96     * @return Formatted phone
97     */
98    public static String formatPhone(String phone){
99        //\b matches an empty string or non-word character,
100       //at the beginning or end of pattern
101
102       //parenthesis() indicate groups
103       String regex = "\b(\d{3})(\d{3})(\d{4})\b";
104
105       Pattern p = Pattern.compile(regex);
106       Matcher m = p.matcher(phone);
107
108       String formattedPhone = "";
109       while (m.find()) {
110           formattedPhone = m.group() + " formatted as " +
111                       "(" + m.group(1) + ") " +
112                           m.group(2) + "-" +
113                           m.group(3);
114       }
115
116       return formattedPhone;
117   }
118 }
```

# Format a Phone Number

```
205  
210     //format phone number 1239826342  
211     String phone1 = "1239826342";  
212     String formattedPhone = RegexClass.formatPhone(phone1);  
213     System.out.println("Formatted phone: " + formattedPhone);  
214     System.out.println("");  
215
```

# Starts/Ends with a Number

```
100
109*     /**
110      * Returns true if str begins with a number.
111      * @param str to find number
112      * @return true if number is at beginning of string
113      */
114*     public static boolean startsWithNumeric(String str) {
115
116         //^ indicates beginning of string
117         String regex = "^\\d+";
118
119         Pattern p = Pattern.compile(regex);
120         Matcher m = p.matcher(str);
121
122         return m.find();
123     }
124
125*     /**
126      * Returns true if str ends with a number.
127      * @param str to find number
128      * @return true if number is at end of string
129      */
130*     public static boolean endsWithNumeric(String str) {
131
132         //\$ indicates end of string
133         String regex = "\\d+\$";
134
135         Pattern p = Pattern.compile(regex);
136         Matcher m = p.matcher(str);
137
138         return m.find();
139     }
140
```

# Starts/Ends with a Number

```
147     //determine if email starts with numeric characters
148     String email = "123krakowsky@gmail.com";
149     boolean startsNumeric = RegexClass.startsWithNumeric(email);
150     System.out.println(email + " startsWithNumeric: " + startsNumeric);
151
152     //determine if email ends with numeric characters
153     boolean endsNumeric = RegexClass.endsWithNumeric(email);
154     System.out.println(email + " endsWithNumeric: " + endsNumeric);
155     System.out.println();
156
```

# Get Parts of an Email Address

```
49  /**
50  * Parses and returns various part of given email address.
51  * @param email address to parse
52  * @param part of email to return: 1 (prefix) or 2 (domain)
53  * @return Part of email
54  */
55  public static String getEmailPart(String email, int part) {
56      if (part < 1 || part > 2) {
57          throw new IllegalArgumentException("Part must be 1 or 2.");
58      }
59
60      //parenthesis() indicate groups
61      //\b matches an empty string or non-word character,
62      //at the beginning or end of pattern
63
64      // [a-zA-Z0-9._%+-]+ indicates a character class,
65      // matching one of several characters (with repetition): a-z, A-Z, 0-9, ., _, %, +, -
66      // [a-zA-Z]{2,} matches an upper or lower-case letter, 2 or more times
67      String regex = "\\\b([a-zA-Z0-9._%+-]+)@[a-zA-Z0-9.-]+\\\.[a-zA-Z]{2,})\\b";
68
69      Pattern p = Pattern.compile(regex);
70      Matcher m = p.matcher(email);
71
72      String emailPart = "";
73      while (m.find()) {
74          //get designated group
75          emailPart = m.group(part);
76      }
77
78      //return group
79      return emailPart;
80  }
```

# Get Parts of an Email Address

```
18
19     //get parts of email address
20     String emailPrefix = RegexClass.getEmailPart(email, 1);
21     String emailDomain = RegexClass.getEmailPart(email, 2);
22     System.out.println("Email parts: " + emailPrefix + " " + emailDomain);
23     System.out.println("");
24
```

# Split String into Sentences & Extract Quote

```
29
30     //split text into sentences
31     //a sentence can end with a . or ! or ?
32     String text = "I'm fixing a hole where the rain gets in. " +
33             "And stops my mind from wandering! " +
34             "Where it will go?";
35     tokens = RegexClass.splitString(text, "[.!?]");
36     RegexClass.printTokens(tokens);
37
38     //extract quote from text
39     //escape double-quotes " with backslash
40     String quoteString = "\"Be yourself; everyone else is already taken.\" said Oscar Wilde";
41     String quote = RegexClass.splitString(quoteString, "\"")[1];
42     System.out.println("Oscar Wilde said: " + quote);
43     System.out.println();
44
```