Coverage Summary for Class: RegexValidator (<empty package name>)

Class	Class, %	Method, %	Line, %
RegexValidator	100% (1/ 1)	62.5% (5/ 8)	46.2% (24/ 52)

```
    1 /*
    2
  * Licensed to the Apache Software Foundation (ASF) under one or more
    3
  * contributor license agreements.  See the NOTICE file distributed with
    4
  * this work for additional information regarding copyright ownership.
    5
  * The ASF licenses this file to You under the Apache License, Version 2.0
    6
  * (the "License"); you may not use this file except in compliance with
    7   * the License.  You may obtain a copy of the License at
    8   *
    9   *      http://www.apache.org/licenses/LICENSE-2.0
   10   *
   11
  * Unless required by applicable law or agreed to in writing, software
   12   * distributed under the License is distributed on an "AS IS" BASIS,
   13
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   14
  * See the License for the specific language governing permissions and
   15   * limitations under the License.
   16   */
   17
   18
   19 import java.io.Serializable;
   20 import java.util.regex.Pattern;
   21 import java.util.regex.Matcher;
   22
   23 /**
   24
  * <b>Regular Expression</b> validation (using JDK 1.4+ regex support).
   25   * <p>
   26
  * Construct the validator either for a single regular expression or a set (array) o
   27
  * regular expressions. By default validation is <i>case sensitive</i> but construct
   28
  * are provided to allow  <i>case in-sensitive</i> validation. For example to create
   29
  * a validator which does <i>case in-sensitive</i> validation for a set of regular
   30   * expressions:
   31   * <pre>
   32   *         String[] regexs = new String[] {...};
   33
  *          RegexValidator validator = new RegexValidator(regexs, false);
   34   * </pre>
   35   * <p>
   36   * <ul>
   37   *   <li>Validate <code>true</code> or <code>false</code>:</li>
```

```
 38  *   <ul>
 39  *     <li><code>boolean valid = validator.isValid(value);</code></li>
 40  *   </ul>
 41  *   <li>Validate returning an aggregated String of the matched groups:</li>
 42  *   <ul>
 43  *     <li><code>String result = validator.validate(value);</code></li>
 44  *   </ul>
 45  *   <li>Validate returning the matched groups:</li>
 46  *   <ul>
 47  *     <li><code>String[] result = validator.match(value);</code></li>
 48  *   </ul>
 49  * </ul>
 50  * <p>
 51  * Cached instances pre-compile and re-use {@link Pattern}(s) - which according
 52  * to the {@link Pattern} API are safe to use in a multi-threaded environment.
 53  *
 54  * @version $Revision: 1227719 $ $Date: 2012-01-05 09:45:51 -0800 (Thu, 05 Jan 2012)
 55  * @since Validator 1.4
 56  */
 57 public class RegexValidator implements Serializable {
 58
 59     private static final long serialVersionUID = -8832409930574867162L;
 60
 61     private final Pattern[] patterns;
 62
 63     /**
 64      * Construct a <i>case sensitive</i> validator for a single
 65      * regular expression.
 66      *
 67      * @param regex The regular expression this validator will
 68      * validate against
 69      */
 70     public RegexValidator(String regex) {
 71         this(regex, true);
 72     }
 73
 74     /**
 75      * Construct a validator for a single regular expression
 76      * with the specified case sensitivity.
 77      *
 78      * @param regex The regular expression this validator will
 79      * validate against
 80      * @param caseSensitive when <code>true</code> matching is <i>case
 81      * sensitive</i>, otherwise matching is <i>case in-sensitive</i>
 82      */
 83     public RegexValidator(String regex, boolean caseSensitive) {
 84         this(new String[] {regex}, caseSensitive);
 85     }
 86
 87     /**
 88      * Construct a <i>case sensitive</i> validator that matches any one
```

```
 89      * of the set of regular expressions.
 90      *
 91
     * @param regexs The set of regular expressions this validator will
 92      * validate against
 93      */
 94     public RegexValidator(String[] regexs) {
             this(regexs, true);
 96     }
 97
 98     /**
 99
     * Construct a validator that matches any one of the set of regular
100      * expressions with the specified case sensitivity.
101      *
102
     * @param regexs The set of regular expressions this validator will
103      * validate against
104
     * @param caseSensitive when <code>true</code> matching is <i>case
105      * sensitive</i>, otherwise matching is <i>case in-sensitive</i>
106      */
        public RegexValidator(String[] regexs, boolean caseSensitive) {
            if (regexs == null || regexs.length == 0) {

              throw new IllegalArgumentException("Regular expressions are missing");
110           }
            patterns = new Pattern[regexs.length];
            int flags =  (caseSensitive ? 0: Pattern.CASE_INSENSITIVE);
            for (int i = 0; i < regexs.length; i++) {
                if (regexs[i] == null || regexs[i].length() == 0) {

                 throw new IllegalArgumentException("Regular expression[" + i + "] is
116               }
                  patterns[i] =  Pattern.compile(regexs[i], flags);
118           }
119     }
120
121     /**
122      * Validate a value against the set of regular expressions.
123      *
124      * @param value The value to validate.
125      * @return <code>true</code> if the value is valid
126      * otherwise <code>false</code>.
127      */
128     public boolean isValid(String value) {
            if (value == null) {
                return false;
131         }
            for (int i = 0; i < patterns.length; i++) {
                if (patterns[i].matcher(value).matches()) {
                    return true;
135             }
136         }
            return false;
138     }
139
140     /**
141      * Validate a value against the set of regular expressions
142      * returning the array of matched groups.
143      *
```

```
144         * @param value The value to validate.
145         * @return String array of the <i>groups</i> matched if
146         * valid or <code>null</code> if invalid
147         */
148        public String[] match(String value) {
               if (value == null) {
                   return null;
151            }
               for (int i = 0; i < patterns.length; i++) {
                   Matcher matcher = patterns[i].matcher(value);
                   if (matcher.matches()) {
                       int count = matcher.groupCount();
                       String[] groups = new String[count];
                       for (int j = 0; j < count; j++) {
                           groups[j] = matcher.group(j+1);
159                    }
                       return groups;
161                }
162            }
               return null;
164        }
165
166
167        /**
168         * Validate a value against the set of regular expressions
169         * returning a String value of the aggregated groups.
170         *
171         * @param value The value to validate.
172         * @return Aggregated String value comprised of the
173
    * <i>groups</i> matched if valid or <code>null</code> if invalid
174         */
175        public String validate(String value) {
               if (value == null) {
                   return null;
178            }
               for (int i = 0; i < patterns.length; i++) {
                   Matcher matcher = patterns[i].matcher(value);
                   if (matcher.matches()) {
                       int count = matcher.groupCount();
                       if (count == 1) {
                           return matcher.group(1);
185                    }
                       StringBuffer buffer = new StringBuffer();
                       for (int j = 0; j < count; j++) {
                           String component = matcher.group(j+1);
                           if (component != null) {
                               buffer.append(component);
191                        }
192                    }
                       return buffer.toString();
194                }
195            }
               return null;
197        }
198
199        /**
200         * Provide a String representation of this validator.
201         * @return A String representation of this validator
202         */
203        public String toString() {
```

```
            StringBuffer buffer = new StringBuffer();
            buffer.append("RegexValidator{");
            for (int i = 0; i < patterns.length; i++) {
                if (i > 0) {
                    buffer.append(",");
209             }
                buffer.append(patterns[i].pattern());
211         }
            buffer.append("}");
            return buffer.toString();
214     }
215
216 }
```

generated on 2017-08-11 14:35