Java Regex

The **Java Regex** or Regular Expression is an API to *define pattern for searching or manipulating strings*.

It is widely used to define constraint on strings such as password and email validation. After learning java regex tutorial, you will be able to test your own regular expressions by the Java Regex Tester Tool.

Java Regex API provides 1 interface and 3 classes in **java.util.regex** package.

java.util.regex package

It provides following classes and interface for regular expressions. The Matcher and Pattern classes are widely used in java regular expression.

1. MatchResult interface
2. Matcher class
3. Pattern class
4. PatternSyntaxException class

Matcher class

It implements **MatchResult** interface. It is a *regex engine* i.e. used to perform match operations on a character sequence.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | boolean matches() | test whether the regular expression matches the pattern. |
| 2 | boolean find() | finds the next expression that matches the pattern. |
| 3 | boolean find(int start) | finds the next expression that matches the pattern from the given start number. |
| 4 | String group() | returns the matched subsequence. |
| 5 | int start() | returns the starting index of the matched subsequence. |
| 6 | int end() | returns the ending index of the matched subsequence. |
| 7 | int groupCount() | returns the total number of the matched subsequence. |

## Pattern class

It is the *compiled version of a regular expression*. It is used to define a pattern for the regex engine.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | static Pattern compile(String regex) | compiles the given regex and return the instance of pattern. |
| 2 | Matcher matcher(CharSequence input) | creates a matcher that matches the given input with pattern. |
| 3 | static boolean matches(String regex, CharSequence input) | It works as the combination of compile and matcher methods. It compiles the regular expression and matches the given input with the pattern. |
| 4 | String[] split(CharSequence input) | splits the given input string around matches of given pattern. |
| 5 | String pattern() | returns the regex pattern. |

### Example of Java Regular Expressions

There are three ways to write the regex example in java.

1. **import** java.util.regex.\*;
2. **public** **class** RegexExample1{
3. **public** **static** **void** main(String args[]){
4. //1st way
5. Pattern p = Pattern.compile(".s");//. represents single character
6. Matcher m = p.matcher("as");
7. **boolean** b = m.matches();
9. //2nd way
10. **boolean** b2=Pattern.compile(".s").matcher("as").matches();
12. //3rd way
13. **boolean** b3 = Pattern.matches(".s", "as");
15. System.out.println(b+" "+b2+" "+b3);
16. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=RegexExample1)

#### Output

true true true

Regular Expression . Example

The . (dot) represents a single character.

1. **import** java.util.regex.\*;
2. **class** RegexExample2{
3. **public** **static** **void** main(String args[]){
4. System.out.println(Pattern.matches(".s", "as"));//true (2nd char is s)
5. System.out.println(Pattern.matches(".s", "mk"));//false (2nd char is not s)
6. System.out.println(Pattern.matches(".s", "mst"));//false (has more than 2 char)
7. System.out.println(Pattern.matches(".s", "amms"));//false (has more than 2 char)
8. System.out.println(Pattern.matches("..s", "mas"));//true (3rd char is s)
9. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=RegexExample2)

Regex Character classes

|  |  |  |
| --- | --- | --- |
| **No.** | **Character Class** | **Description** |
| 1 | [abc] | a, b, or c (simple class) |
| 2 | [^abc] | Any character except a, b, or c (negation) |
| 3 | [a-zA-Z] | a through z or A through Z, inclusive (range) |
| 4 | [a-d[m-p]] | a through d, or m through p: [a-dm-p] (union) |
| 5 | [a-z&&[def]] | d, e, or f (intersection) |
| 6 | [a-z&&[^bc]] | a through z, except for b and c: [ad-z] (subtraction) |
| 7 | [a-z&&[^m-p]] | a through z, and not m through p: [a-lq-z](subtraction) |

Regular Expression Character classes Example

1. **import** java.util.regex.\*;
2. **class** RegexExample3{
3. **public** **static** **void** main(String args[]){
4. System.out.println(Pattern.matches("[amn]", "abcd"));//false (not a or m or n)
5. System.out.println(Pattern.matches("[amn]", "a"));//true (among a or m or n)
6. System.out.println(Pattern.matches("[amn]", "ammmna"));//false (m and a comes more than once)
7. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=RegexExample3)

Regex Quantifiers

The quantifiers specify the number of occurrences of a character.

|  |  |
| --- | --- |
| **Regex** | **Description** |
| X? | X occurs once or not at all |
| X+ | X occurs once or more times |
| X\* | X occurs zero or more times |
| X{n} | X occurs n times only |
| X{n,} | X occurs n or more times |
| X{y,z} | X occurs at least y times but less than z times |

Regular Expression Character classes and Quantifiers Example

1. **import** java.util.regex.\*;
2. **class** RegexExample4{
3. **public** **static** **void** main(String args[]){
4. System.out.println("? quantifier ....");
5. System.out.println(Pattern.matches("[amn]?", "a"));//true (a or m or n comes one time)
6. System.out.println(Pattern.matches("[amn]?", "aaa"));//false (a comes more than one time)
7. System.out.println(Pattern.matches("[amn]?", "aammmnn"));//false (a m and n comes more than one time)
8. System.out.println(Pattern.matches("[amn]?", "aazzta"));//false (a comes more than one time)
9. System.out.println(Pattern.matches("[amn]?", "am"));//false (a or m or n must come one time)
11. System.out.println("+ quantifier ....");
12. System.out.println(Pattern.matches("[amn]+", "a"));//true (a or m or n once or more times)
13. System.out.println(Pattern.matches("[amn]+", "aaa"));//true (a comes more than one time)
14. System.out.println(Pattern.matches("[amn]+", "aammmnn"));//true (a or m or n comes more than once)
15. System.out.println(Pattern.matches("[amn]+", "aazzta"));//false (z and t are not matching pattern)
17. System.out.println("\* quantifier ....");
18. System.out.println(Pattern.matches("[amn]\*", "ammmna"));//true (a or m or n may come zero or more times)
20. }}

Regex Metacharacters

The regular expression metacharacters work as a short codes.

|  |  |
| --- | --- |
| **Regex** | **Description** |
| . | Any character (may or may not match terminator) |
| \d | Any digits, short of [0-9] |
| \D | Any non-digit, short for [^0-9] |
| \s | Any whitespace character, short for [\t\n\x0B\f\r] |
| \S | Any non-whitespace character, short for [^\s] |
| \w | Any word character, short for [a-zA-Z\_0-9] |
| \W | Any non-word character, short for [^\w] |
| \b | A word boundary |
| \B | A non word boundary |

Regular Expression Metacharacters Example

1. **import** java.util.regex.\*;
2. **class** RegexExample5{
3. **public** **static** **void** main(String args[]){
4. System.out.println("metacharacters d....");\\d means digit
6. System.out.println(Pattern.matches("\\d", "abc"));//false (non-digit)
7. System.out.println(Pattern.matches("\\d", "1"));//true (digit and comes once)
8. System.out.println(Pattern.matches("\\d", "4443"));//false (digit but comes more than once)
9. System.out.println(Pattern.matches("\\d", "323abc"));//false (digit and char)
11. System.out.println("metacharacters D....");\\D means non-digit
13. System.out.println(Pattern.matches("\\D", "abc"));//false (non-digit but comes more than once)
14. System.out.println(Pattern.matches("\\D", "1"));//false (digit)
15. System.out.println(Pattern.matches("\\D", "4443"));//false (digit)
16. System.out.println(Pattern.matches("\\D", "323abc"));//false (digit and char)
17. System.out.println(Pattern.matches("\\D", "m"));//true (non-digit and comes once)
19. System.out.println("metacharacters D with quantifier....");
20. System.out.println(Pattern.matches("\\D\*", "mak"));//true (non-digit and may come 0 or more times)
22. }}

Regular Expression Question 1

1. /\*Create a regular expression that accepts alpha numeric characters only. Its
2. length must be 6 characters long only.\*/
4. **import** java.util.regex.\*;
5. **class** RegexExample6{
6. **public** **static** **void** main(String args[]){
7. System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun32"));//true
8. System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "kkvarun32"));//false (more than 6 char)
9. System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "JA2Uk2"));//true
10. System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun$2"));//false ($ is not matched)
11. }}

Regular Expression Question 2

1. /\*Create a regular expression that accepts 10 digit numeric characters
2. starting with 7, 8 or 9 only.\*/
4. **import** java.util.regex.\*;
5. **class** RegexExample7{
6. **public** **static** **void** main(String args[]){
7. System.out.println("by character classes and quantifiers ...");
8. System.out.println(Pattern.matches("[789]{1}[0-9]{9}", "9953038949"));//true
9. System.out.println(Pattern.matches("[789][0-9]{9}", "9953038949"));//true
11. System.out.println(Pattern.matches("[789][0-9]{9}", "99530389490"));//false (11 characters)
12. System.out.println(Pattern.matches("[789][0-9]{9}", "6953038949"));//false (starts from 6)
13. System.out.println(Pattern.matches("[789][0-9]{9}", "8853038949"));//true
15. System.out.println("by metacharacters ...");
16. System.out.println(Pattern.matches("[789]{1}\\d{9}", "8853038949"));//true
17. System.out.println(Pattern.matches("[789]{1}\\d{9}", "3853038949"));//false (starts from 3)
19. }}

Java Regex Finder Example

1. **import** java.util.regex.Pattern;
2. **import** java.util.Scanner;
3. **import** java.util.regex.Matcher;
4. **public** **class** RegexExample8{
5. **public** **static** **void** main(String[] args){
6. Scanner sc=**new** Scanner(System.in);
7. **while** (**true**) {
8. System.out.println("Enter regex pattern:");
9. Pattern pattern = Pattern.compile(sc.nextLine());
10. System.out.println("Enter text:");
11. Matcher matcher = pattern.matcher(sc.nextLine());
12. **boolean** found = **false**;
13. **while** (matcher.find()) {
14. System.out.println("I found the text "+matcher.group()+" starting at index "+
15. matcher.start()+" and ending at index "+matcher.end());
16. found = **true**;
17. }
18. **if**(!found){
19. System.out.println("No match found.");
20. }
21. }
22. }
23. }

Output:

Enter regex pattern: java

Enter text: this is java, do you know java

I found the text java starting at index 8 and ending at index 12

I found the text java starting at index 26 and ending at index 30