Java Regex

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

It is widely used to define the constraint on strings such as password and email validation. After learning Java regex tutorial, you will be able to test your 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

The Matcher and Pattern classes provide the facility of Java regular expression. The java.util.regex package provides following classes and interfaces for regular expressions.

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



Matcher class

It implements the **MatchResult** interface. It is a *regex engine* which is 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 returns the instance of the Pattern. |
| 2 | Matcher matcher(CharSequence input) | creates a matcher that matches the given input with the 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.

**import** java.util.regex.\*;

**public** **class** RegexExample1{

**public** **static** **void** main(String args[]){

//1st way

Pattern p = Pattern.compile(".s");//. represents single character

Matcher m = p.matcher("as");

**boolean** b = m.matches();

//2nd way

**boolean** b2=Pattern.compile(".s").matcher("as").matches();

//3rd way

**boolean** b3 = Pattern.matches(".s", "as");

System.out.println(b+" "+b2+" "+b3);

}}

**import** java.util.regex.\*;

**class** RegexExample2{

**public** **static** **void** main(String args[]){

System.out.println(Pattern.matches(".s", "as"));//true (2nd char is s)

System.out.println(Pattern.matches(".s", "mk"));//false (2nd char is not s)

System.out.println(Pattern.matches(".s", "mst"));//false (has more than 2 char)

System.out.println(Pattern.matches(".s", "amms"));//false (has more than 2 char)

System.out.println(Pattern.matches("..s", "mas"));//true (3rd char is s)

}}

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

**import** java.util.regex.\*;

**class** RegexExample3{

**public** **static** **void** main(String args[]){

System.out.println(Pattern.matches("[amn]", "abcd"));//false (not a or m or n)

System.out.println(Pattern.matches("[amn]", "a"));//true (among a or m or n)

System.out.println(Pattern.matches("[amn]", "ammmna"));//false (m and a comes moze than once)

}}  **[Test it Now](https://www.javatpoint.com/opr/test.jsp?filename=RegexExample3" \t "_blank)**

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

**import** java.util.regex.\*;

**class** RegexExample4{

**public** **static** **void** main(String args[]){

System.out.println("? quantifier ....");

System.out.println(Pattern.matches("[amn]?", "a"));//true (a or m or n comes one timz)

System.out.println(Pattern.matches("[amn]?", "aaa"));//false (a comes more than one time)

System.out.println(Pattern.matches("[amn]?", "aammmnn"));//false (a m and n comes more than one time)

System.out.println(Pattern.matches("[amn]?", "aazzta"));//false (a comes more than one time)

System.out.println(Pattern.matches("[amn]?", "am"));//false (a or m or n must come one time)

System.out.println("+ quantifier ....");

System.out.println(Pattern.matches("[amn]+", "a"));//true (a or m or n once or more times)

System.out.println(Pattern.matches("[amn]+", "aaa"));//true (a comes more than one time)

System.out.println(Pattern.matches("[amn]+", "aammmnn"));//true (a or m or n comes more than once)

System.out.println(Pattern.matches("[amn]+", "aazzta"));//false (z and t are not matching pattern)

System.out.println("\* quantifier ....");

System.out.println(Pattern.matches("[amn]\*", "ammmna"));//true (a or m or n may come zero or more times)

}}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=RegexExample4)

Regex Metacharacters

The regular expression metacharacters work as shortcodes.

|  |  |
| --- | --- |
| **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

**import** java.util.regex.\*;

**class** RegexExample5{

**public** **static** **void** main(String args[]){

System.out.println("metacharacters d....");\\d means digit

System.out.println(Pattern.matches("\\d", "abc"));//false (non-digit)

System.out.println(Pattern.matches("\\d", "1"));//true (digit and comes once)

System.out.println(Pattern.matches("\\d", "4443"));//false (digit but comes more than once)

System.out.println(Pattern.matches("\\d", "323abc"));//false (digit and char)

System.out.println("metacharacters D....");\\D means non-digit

System.out.println(Pattern.matches("\\D", "abc"));//false (non-digit but comes more than once)

System.out.println(Pattern.matches("\\D", "1"));//false (digit)

System.out.println(Pattern.matches("\\D", "4443"));//false (digit)

System.out.println(Pattern.matches("\\D", "323abc"));//false (digit and char)

System.out.println(Pattern.matches("\\D", "m"));//true (non-digit and comes once)

System.out.println("metacharacters D with quantifier....");

System.out.println(Pattern.matches("\\D\*", "mak"));//true (non-digit and may come 0 or more times)

}}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=RegexExample5)

Regular Expression Question 1

/\*Create a regular expression that accepts alphanumeric characters only.

Its length must be six characters long only.\*/

**import** java.util.regex.\*;

**class** RegexExample6{

**public** **static** **void** main(String args[]){

System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun32"));//true

System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "kkvarun32"));//false (more than 6 char)

System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "JA2Uk2"));//true

System.out.println(Pattern.matches("[a-zA-Z0z9]{6}", "arun$2"));//false ($ is not matched)

}}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=RegexExample6)

Regular Expression Question 2

/\*Create a regular expression that accepts 10 digit numeric characters

 starting with 7, 8 or 9 only.\*/

**import** java.util.regex.\*;

**class** RegexExample7{

**public** **static** **void** main(String args[]){

System.out.println("by character classes and quantifiers ...");

System.out.println(Pattern.matches("[789]{1}[0-9]{9}", "9953038949"));//true

System.out.println(Pattern.matches("[789][0-9]{9}", "9953038949"));//true

System.out.println(Pattern.matches("[789][0-9]{9}", "99530389490"));//false (11 characters)

System.out.println(Pattern.matches("[789][0-9]{9}", "6953038949"));//false (starts from 6)

System.out.println(Pattern.matches("[789][0-9]{9}", "8853038949"));//true

System.out.println("by metacharacters ...");

System.out.println(Pattern.matches("[789]{1}\\d{9}", "8853038949"));//true

System.out.println(Pattern.matches("[789]{1}\\d{9}", "3853038949"));//false (starts from 3)

}}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=RegexExample7)

Java Regex Finder Example

**import** java.util.regex.Pattern;

**import** java.util.Scanner;

**import** java.util.regex.Matcher;

**public** **class** RegexExample8{

**public** **static** **void** main(String[] args){

        Scanner sc=**new** Scanner(System.in);

**while** (**true**) {

            System.out.println("Enter regex pattern:");

            Pattern pattern = Pattern.compile(sc.nextLine());

            System.out.println("Enter text:");

            Matcher matcher = pattern.matcher(sc.nextLine());

**boolean** found = **false**;

**while** (matcher.find()) {

                System.out.println("I found the text "+matcher.group()+" starting at index +

                 matcher.start()+" and ending at index "+matcher.end());

                found = **true**;

            }

**if**(!found){

                System.out.println("No match found.");

            }

        }

    }

}

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