Java regex Notes

Contents

[What Are Regular Expressions? 1](#_Toc82944878)

[How Are Regular Expressions Represented in This Package? 1](#_Toc82944879)

[java.util.regex package 2](#_Toc82944880)

[Matcher class 2](#_Toc82944881)

[Pattern class 2](#_Toc82944882)

[Examples of Java Regular Expressions 3](#_Toc82944883)

[Regex Character classes 4](#_Toc82944884)

[Regular Expression Character classes Example 4](#_Toc82944885)

[Regex Quantifiers 4](#_Toc82944886)

[Regular Expression Character classes and Quantifiers Example 5](#_Toc82944887)

[Regex Metacharacters 6](#_Toc82944888)

[Regular Expression Metacharacters Example 6](#_Toc82944889)

[Regular Expression Question 1 7](#_Toc82944890)

[Regular Expression Question 2 7](#_Toc82944891)

[Class Pattern 8](#_Toc82944892)

# What Are Regular Expressions?

*Regular expressions* are a way to describe a set of strings based on common characteristics shared by each string in the set. They can be used to search, edit, or manipulate text and data. You must learn a specific syntax to create regular expressions — one that goes beyond the normal syntax of the Java programming language. Regular expressions vary in complexity, but once you understand the basics of how they're constructed, you'll be able to decipher (or create) any regular expression.

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

It is widely used to define constraints on strings such as password and email validation.

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

# How Are Regular Expressions Represented in This Package?

The java.util.regex package primarily consists of three classes: [Pattern](http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html), [Matcher](http://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html), and [PatternSyntaxException](http://docs.oracle.com/javase/8/docs/api/java/util/regex/PatternSyntaxException.html).

* A Pattern object is a compiled representation of a regular expression. The Pattern class provides no public constructors. To create a pattern, you must first invoke one of its public static compile methods, which will then return a Pattern object. These methods accept a regular expression as the first argument; the first few lessons of this trail will teach you the required syntax.
* A Matcher object is the engine that interprets the pattern and performs match operations against an input string. Like the Pattern class, Matcher defines no public constructors. You obtain a Matcher object by invoking the matcher method on a Pattern object.
* A PatternSyntaxException object is an unchecked exception that indicates a syntax error in a regular expression pattern.

# java.util.regex package

This package 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

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

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean matches() | test whether the regular expression matches the pattern. |
| boolean find() | finds the next expression that matches the pattern. |
| boolean find(int start) | finds the next expression that matches the pattern from the given start number. |

## Pattern class

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

|  |  |
| --- | --- |
| **Method** | **Description** |
| static Pattern compile(String regex) | compiles the given regex and return the instance of pattern. |
| Matcher matcher(CharSequence input) | creates a matcher that matches the given input with pattern. |
| 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. |
| String[] split(CharSequence input) | splits the given input string around matches of given pattern. |
| String pattern() | returns the regex pattern. |

# Examples 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);

}

}// end RegexExample1

Output

true true true

The . (dot) represents a single character.

import java.util.regex.\*;

class  {

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)

}

}// end RegexExample2

# Regex Character classes

|  |  |
| --- | --- |
| **Character Class** | **Description** |
| [abc] | a, b, or c (simple class) |
| [^abc] | Any character except a, b, or c (negation) |
| [a-zA-Z] | a through z or A through Z, inclusive (range) |
| [a-d[m-p]] | a through d, or m through p: [a-dm-p] (union) |
| [a-z&&[def]] | d, e, or f (intersection) |
| [a-z&&[^bc]] | a through z, except for b and c: [ad-z] (subtraction) |
| [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 occurs more than once)

}

}// end 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

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 time)

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)

}

}

# 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 for [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....");

System.out.println(Pattern.matches("\\d", "abc")); \\d means digit

//false (non-digit)

System.out.println(Pattern.matches("\\d", "1"));

//true (digit and occurs once)

System.out.println(Pattern.matches("\\d", "4443"));

//false (digit but occurs more than once)

System.out.println(Pattern.matches("\\d", "323abc"));

//false (digits and chars)

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

System.out.println(Pattern.matches("\\D", "abc"));

//false (non-digit but occurs 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 (digits and chars)

System.out.println(Pattern.matches("\\D", "m"));

//true (non-digit and occurs once)

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

System.out.println(Pattern.matches("\\D\*", "mak"));

//true (non-digit and may occur 0 or more times)

}

}// end RegexExample5

## Regular Expression Question 1

/\*Create a regular expression that accepts alpha numeric characters only. Its length must be 6 characters long only.\*/

import java.util.regex.\*;

class RegexExample6{

public static void main(String args[]){

System.out.println(Patter.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-Z0-9]{6}", "arun$2"));

//false ($ is not matched)

}

}// end 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)

}

}// end RegexExample7

# Class Pattern

* + [java.lang.Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html) java.util.regex.Pattern
* Interface Implement: [Serializable](http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html)

public final class Pattern

extends [Object](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html)

implements [Serializable](http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html)

A compiled representation of a regular expression.

A regular expression, specified as a string, must first be compiled into an instance of this class. The resulting pattern can then be used to create a [Matcher](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher.html) object that can match arbitrary [character sequences](http://docs.oracle.com/javase/7/docs/api/java/lang/CharSequence.html) against the regular expression. All of the state involved in performing a match resides in the matcher, so many matchers can share the same pattern.

A typical invocation sequence is thus

Pattern p = Pattern.[compile](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#compile%28java.lang.String%29)("a\*b");

Matcher m = p.[matcher](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#matcher%28java.lang.CharSequence%29)("aaaaab");

boolean b = m.[matches](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher.html#matches%28%29)();

The [matches](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#matches%28java.lang.String,%20java.lang.CharSequence%29) method is defined by this class as a convenience for when a regular expression is used just once. This method compiles an expression and matches an input sequence against it in a single invocation. The statement

boolean b = Pattern.matches("a\*b", "aaaaab");

is equivalent to the three statements above, though for repeated matches it is less efficient since it does not allow the compiled pattern to be reused.

Instances of this class are immutable and are safe for use by multiple concurrent threads. Instances of the [Matcher](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher.html) class are not safe for such use.

http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html