* java.util.regex.Pattern
* 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 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

A typical invocation sequence is thus

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

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

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

A [matches](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#matches(java.lang.String,%20java.lang.CharSequence)) 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.

* 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.

|  |  |
| --- | --- |
| **Characters** | |
| *X* | The character *x* |
| \\ | The backslash character |
| \0*n* | The character with octal value 0*n* (0 <= *n* <= 7) |
| \0*nn* | The character with octal value 0*nn* (0 <= *n* <= 7) |
| \0*mnn* | The character with octal value 0*mnn* (0 <= *m* <= 3, 0 <= *n* <= 7) |
| \x*hh* | The character with hexadecimal value 0x*hh* |
| \u*hhhh* | The character with hexadecimal value 0x*hhhh* |
| \x*{h...h}* | The character with hexadecimal value 0x*h...h* ([Character.MIN\_CODE\_POINT](http://docs.oracle.com/javase/7/docs/api/java/lang/Character.html" \l "MIN_CODE_POINT)  <= 0x*h...h* <=  [Character.MAX\_CODE\_POINT](http://docs.oracle.com/javase/7/docs/api/java/lang/Character.html#MAX_CODE_POINT)) |
| \t | The tab character ('\u0009') |
| \n | The newline (line feed) character ('\u000A') |
| \r | The carriage-return character ('\u000D') |
| \f | The form-feed character ('\u000C') |
| \a | The alert (bell) character ('\u0007') |
| \e | The escape character ('\u001B') |
| \c*x* | The control character corresponding to *x* |
|  |  |
| **Character classes** | |
| [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) |
|  |  |
| **Predefined character classes** | |
| . | Any character (may or may not match [line terminators](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#lt)) |
| \d | A digit: [0-9] |
| \D | A non-digit: [^0-9] |
| \s | A whitespace character: [ \t\n\x0B\f\r] |
| \S | A non-whitespace character: [^\s] |
| \w | A word character: [a-zA-Z\_0-9] |
| \W | A non-word character: [^\w] |
|  |  |
| **POSIX character classes (US-ASCII only)** | |
| \p{Lower} | A lower-case alphabetic character: [a-z] |
| \p{Upper} | An upper-case alphabetic character:[A-Z] |
| \p{ASCII} | All ASCII:[\x00-\x7F] |
| \p{Alpha} | An alphabetic character:[\p{Lower}\p{Upper}] |
| \p{Digit} | A decimal digit: [0-9] |
| \p{Alnum} | An alphanumeric character:[\p{Alpha}\p{Digit}] |
| \p{Punct} | Punctuation: One of !"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~ |
| \p{Graph} | A visible character: [\p{Alnum}\p{Punct}] |
| \p{Print} | A printable character: [\p{Graph}\x20] |
| \p{Blank} | A space or a tab: [ \t] |
| \p{Cntrl} | A control character: [\x00-\x1F\x7F] |
| \p{XDigit} | A hexadecimal digit: [0-9a-fA-F] |
| \p{Space} | A whitespace character: [ \t\n\x0B\f\r] |
|  |  |
| **java.lang.Character classes (simple**[**java character type**](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#jcc)**)** | |
| \p{javaLowerCase} | Equivalent to java.lang.Character.isLowerCase() |
| \p{javaUpperCase} | Equivalent to java.lang.Character.isUpperCase() |
| \p{javaWhitespace} | Equivalent to java.lang.Character.isWhitespace() |
| \p{javaMirrored} | Equivalent to java.lang.Character.isMirrored() |
|  |  |
| **Classes for Unicode scripts, blocks, categories and binary properties** | |
| \p{IsLatin} | A Latin script character ([script](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#usc)) |
| \p{InGreek} | A character in the Greek block ([block](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#ubc)) |
| \p{Lu} | An uppercase letter ([category](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#ucc)) |
| \p{IsAlphabetic} | An alphabetic character ([binary property](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#ubpc)) |
| \p{Sc} | A currency symbol |
| \P{InGreek} | Any character except one in the Greek block (negation) |
| [\p{L}&&[^\p{Lu}]] | Any letter except an uppercase letter (subtraction) |
|  |  |
| **Boundary matchers** | |
| ^ | The beginning of a line |
| $ | The end of a line |
| \b | A word boundary |
| \B | A non-word boundary |
| \A | The beginning of the input |
| \G | The end of the previous match |
| \Z | The end of the input but for the final [terminator](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#lt), if any |
| \z | The end of the input |
|  |  |
| **Greedy quantifiers** | |
| *X*? | *X*, once or not at all |
| *X*\* | *X*, zero or more times |
| *X*+ | *X*, one or more times |
| *X*{*n*} | *X*, exactly *n* times |
| *X*{*n*,} | *X*, at least *n* times |
| *X*{*n*,*m*} | *X*, at least *n* but not more than *m* times |
|  |  |
| **Reluctant quantifiers** | |
| *X*?? | *X*, once or not at all |
| *X*\*? | *X*, zero or more times |
| *X*+? | *X*, one or more times |
| *X*{*n*}? | *X*, exactly *n* times |
| *X*{*n*,}? | *X*, at least *n* times |
| *X*{*n*,*m*}? | *X*, at least *n* but not more than *m* times |
|  |  |
| **Possessive quantifiers** | |
| *X*?+ | *X*, once or not at all |
| *X*\*+ | *X*, zero or more times |
| *X*++ | *X*, one or more times |
| *X*{*n*}+ | *X*, exactly *n* times |
| *X*{*n*,}+ | *X*, at least *n* times |
| *X*{*n*,*m*}+ | *X*, at least *n* but not more than *m* times |
|  |  |
| **Logical operators** | |
| *XY* | *X* followed by *Y* |
| *X*|*Y* | Either *X* or *Y* |
| (*X*) | X, as a [capturing group](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#cg) |
|  |  |
| **Back references** | |
| \*n* | Whatever the *n*th [capturing group](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#cg) matched |
| \*k*<*name*> | Whatever the [named-capturing group](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#groupname) "name" matched |
|  |  |
| **Quotation** | |
| \ | Nothing, but quotes the following character |
| \Q | Nothing, but quotes all characters until \E |
| \E | Nothing, but ends quoting started by \Q |
|  |  |
| **Special constructs (named-capturing and non-capturing)** | |
| (?<[name](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#groupname)>*X*) | *X*, as a named-capturing group |
| (?:*X*) | *X*, as a non-capturing group |
| (?idmsuxU-idmsuxU) | Nothing, but turns match flags [i](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html" \l "CASE_INSENSITIVE) [d](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#UNIX_LINES) [m](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#MULTILINE) [s](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#DOTALL) [u](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#UNICODE_CASE) [x](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#COMMENTS) [U](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#UNICODE_CHARACTER_CLASS) on - off |
| (?idmsux-idmsux:*X*) | *X*, as a [non-capturing group](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#cg) with the given flags [i](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html" \l "CASE_INSENSITIVE) [d](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#UNIX_LINES) [m](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#MULTILINE) [s](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#DOTALL) [u](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#UNICODE_CASE) [x](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html#COMMENTS) on - off |
| (?=*X*) | *X*, via zero-width positive lookahead |
| (?!*X*) | *X*, via zero-width negative lookahead |
| (?<=*X*) | *X*, via zero-width positive lookbehind |
| (?<!*X*) | *X*, via zero-width negative lookbehind |
| (?>*X*) | *X*, as an independent, non-capturing group |

|  |  |
| --- | --- |
| **Classes** | **Matches** |
| \p{Lower} | A lowercase character:\p{IsLowercase} |
| \p{Upper} | An uppercase character:\p{IsUppercase} |
| \p{ASCII} | All ASCII:[\x00-\x7F] |
| \p{Alpha} | An alphabetic character:\p{IsAlphabetic} |
| \p{Digit} | A decimal digit character:p{IsDigit} |
| \p{Alnum} | An alphanumeric character:[\p{IsAlphabetic}\p{IsDigit}] |
| \p{Punct} | A punctuation character:p{IsPunctuation} |
| \p{Graph} | A visible character: [^\p{IsWhite\_Space}\p{gc=Cc}\p{gc=Cs}\p{gc=Cn}] |
| \p{Print} | A printable character: [\p{Graph}\p{Blank}&&[^\p{Cntrl}]] |
| \p{Blank} | A space or a tab: [\p{IsWhite\_Space}&&[^\p{gc=Zl}\p{gc=Zp}\x0a\x0b\x0c\x0d\x85]] |
| \p{Cntrl} | A control character: \p{gc=Cc} |
| \p{XDigit} | A hexadecimal digit: [\p{gc=Nd}\p{IsHex\_Digit}] |
| \p{Space} | A whitespace character:\p{IsWhite\_Space} |
| \d | A digit: \p{IsDigit} |
| \D | A non-digit: [^\d] |
| \s | A whitespace character: \p{IsWhite\_Space} |
| \S | A non-whitespace character: [^\s] |
| \w | A word character: [\p{Alpha}\p{gc=Mn}\p{gc=Me}\p{gc=Mc}\p{Digit}\p{gc=Pc}] |
| \W | A non-word character: [^\w] |

Example 1.

**package** com.app.pattern;

**import** java.util.regex.Pattern;

**public** **class** TestPatternMatch {

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

System.*out*.println(*patternMatch*("[abc]", "a"));//true

System.*out*.println(*patternMatch*("[abc]", "b"));//true

System.*out*.println(*patternMatch*("[abc]", "c"));//true

System.*out*.println(*patternMatch*("[abc]", "d"));//false

System.*out*.println(*patternMatch*("[abc]", "abc"));//false

System.*out*.println(*patternMatch*("[^abc]", "a"));//false

System.*out*.println(*patternMatch*("[^abc]", "d"));//true

System.*out*.println(*patternMatch*("[abc]yz", "ayx"));//false

System.*out*.println(*patternMatch*("[^abc]yx", "dyx"));//true

}

**public** **static** **boolean** patternMatch(**final** String regex, **final** String input) {

**boolean** result = Pattern.*matches*(regex, input);

**return** result;

}

}

Example 2.

package com.test;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Test5 {

public static void main(String[] args) {

// for one character

System.out.println(patternMatch("[A-Z]","S"));// true; any character from A to Z

System.out.println(patternMatch("[ABCD]","D"));// true; any character in A,B,C,D

System.out.println(patternMatch("[^ABCD]","\*"));// true; any character not in A,B,C,D

System.out.println(patternMatch("[A-Z]","S"));// true; A to Z

System.out.println(patternMatch("[^A-Z]","("));// true; any character except A to Z

System.out.println(patternMatch("[A-Za-z]","n"));// true; either from A to Z or from a to z

System.out.println(patternMatch("[A-Z[a-z]]","d")); // true; either from A to Z or from a to z

System.out.println(patternMatch("[A-Z&&[a-z]]","d")); // false; should be in both from A to Z and form a to z

System.out.println(patternMatch("[A-Z&&[P-R]]","P")); // true; should be in both form A to Z and from P to R

System.out.println(patternMatch("[A-Z&&[^SD]]","D"));// false; A to Z except S,D

System.out.println(patternMatch("[[A-Z]&&[^R-Z]]","P"));// true; A to Z except R to Z

System.out.println(patternMatch(".","(")); // true, for any character

System.out.println(patternMatch("\\d","7")); // true; any digit from 0 to 9. => \d

System.out.println(patternMatch("[0-9]","9")); // true; any digit from 0 to 9.

System.out.println(patternMatch("\\D","1")); // false; any non digit. => \D => [^\d]

System.out.println(patternMatch("\\s"," "));// true; a white space character. => \s

System.out.println(patternMatch("\\S"," "));// false; a non white space character. =>\S => [^\S]

System.out.println(patternMatch("\\w","\_"));// true; a word character. => [a-zA-Z0-9\_]

System.out.println(patternMatch("[a-zA-Z0-9\_]","2"));// true; a word character.

System.out.println(patternMatch("\\W","2"));// false; a non word character. => [^\w]

System.out.println(patternMatch("[^\\w]","\*"));// true; a non word character.

System.out.println(patternMatch("[()]","+"));// false;

System.out.println(patternMatch("[()]","("));// true;

System.out.println(patternMatch("[()]",")"));// true;

System.out.println(patternMatch("[(]","("));// true;

System.out.println(patternMatch("[\*]","\*"));// true;

//System.out.println(patternMatch("(","("));// Exception in thread "main" java.util.regex.PatternSyntaxException: Unclosed group near index 1 (

// match for more than two characters

System.out.println(patternMatch("ACver","adas"));// false; match exact string as it is

System.out.println(patternMatch("ACver","ACver"));// true;

System.out.println(patternMatch("[A-Z]&&[a-z]","A&&b"));// true;

System.out.println(patternMatch("SUM[A-Z]","SUMX")); // true;

System.out.println(patternMatch("SUM[(][A-Z][0-9]:[A-Z][0-9][)]","SUM(A1)"));//false;

System.out.println(patternMatch("SUM[(][A-Z][0-9]:[A-Z][0-9][)]","SUM(A1:D1)"));//true;

System.out.println(patternMatch("SUM[(][A-Z][0-9][0-9]?:[A-Z][0-9][0-9]?[)]","SUM(A2:D23)"));//true;

System.*out*.println(*patternMatch*("\\p{Lower}","a"));//true;

System.*out*.println(*patternMatch*("\\p{Alnum}","1"));//true; \p{Alnum} => [\p{Alpha}\p{Digit}]

System.*out*.println(*patternMatch*("[\\p{Alpha}\\p{Digit}]","h"));//true;

System.*out*.println(*patternMatch*("\\p{Alpha}\*\\p{Digit}+\\p{Alpha}\*","sdsfg3fsdf"));//true; alpha numeric with one must digit.

System.*out*.println(*patternMatch*("\\p{Alpha}\*\\p{Digit}+\\p{Alpha}\*","sdsfgfsdf"));//false;

System.*out*.println(*patternMatch*("\\p{Alpha}\*\\p{Digit}+\\p{Alpha}\*\\p{Digit}+\\p{Alpha}\*","sdsfgfsdf12"));//true; alpha numeric with two must digit.

System.*out*.println(*patternMatch*("\\p{Alpha}\*\\p{Digit}+\\p{Alpha}\*\\p{Digit}+\\p{Alpha}\*","3sds1fgfsdf2"));//false; alpha numeric with two must digit.

}

public static boolean patternMatch(final String regex, final String input) {

boolean result = Pattern.matches(regex, input);

return result;

}

*// returns true if the string matches exactly "true" or "True"*

*// or "yes" or "Yes"*

**public** **boolean** isTrueOrYes(String s){

**return** s.matches("[tT]rue|[yY]es");

}

*// returns true if the string contains exactly "true"*

**public** **boolean** containsTrue(String s){

**return** s.matches(".\*true.\*");

}

*// returns true if the string contains of three letters*

**public** **boolean** isThreeLetters(String s){

**return** s.matches("[a-zA-Z]{3}");

*// simpler from for*

*// return s.matches("[a-Z][a-Z][a-Z]");*

}

*// returns true if the string does not have a number at the beginning*

**public** **boolean** isNoNumberAtBeginning(String s){

**return** s.matches("^[^\\d].\*");

}

*// returns true if the string contains a arbitrary number of characters except b*

**public** **boolean** isIntersection(String s){

**return** s.matches("([\\w&&[^b]])\*");

}

*// returns true if the string contains a number less then 300*

**public** **boolean** isLessThenThreeHundred(String s){

**return** s.matches("[^0-9]\*[12]?[0-9]{1,2}[^0-9]\*");

}

}