



CHAPTER 27

Test Table

This is a way to fix this table

Table 27-1. Java regular expression quick reference

Syntax	Matches
Single characters	
<i>x</i>	The character <i>x</i> , as long as <i>x</i> is not a punctuation character with special meaning in the regular expression syntax.
<i>\p</i>	The punctuation character <i>p</i> .
<i>\\</i>	The backslash character.
<i>\n</i>	Newline character <i>\u000A</i> .
<i>\t</i>	Tab character <i>\u0009</i> .
<i>\r</i>	Carriage return character <i>\u000D</i> .
<i>\f</i>	Form feed character <i>\u000C</i> .
<i>\e</i>	Escape character <i>\u001B</i> .
<i>\a</i>	Bell (alert) character <i>\u0007</i> .
<i>\uxxxx</i>	Unicode character with hexadecimal code <i>xxxx</i> .
<i>\xxx</i>	Character with hexadecimal code <i>xx</i> .
<i>\On</i>	Character with octal code <i>n</i> .
<i>\Onn</i>	Character with octal code <i>nn</i> .
<i>\Onnn</i>	Character with octal code <i>nnn</i> , in which <i>nnn</i> <= 377.
<i>\cx</i>	The control character <i>^x</i> .
Character classes	

Table 27–1. Java regular expression quick reference (continued)

Syntax	Matches
[...]	One of the characters between the brackets. Characters may be specified literally, and the syntax also allows the specification of character ranges, with intersection, union and subtraction operators. See specific examples that follow.
[^...]	Any one character not between the brackets.
[a-z0-9]	Character range: a character between (inclusive) a and z or 0 and 9 .
[0-9[a-fA-F]]	Union of classes: same as [0-9a-fA-F] .
[a-z&&[aeiou]]	Intersection of classes: same as [aeiou] .
[a-z&&[^aeiou]]	Subtraction: the characters a through z except for the vowels.
.	Any character, except a line terminator. If the DOTALL flag is set, then it matches any character including line terminators.
\d	ASCII digit: [0-9] .
\D	Anything but an ASCII digit: [^\d] .
\s	ASCII whitespace: [\t\n\f\r\x0B] .
\S	Anything but ASCII whitespace: [^\s] .
\w	ASCII word character: [a-zA-Z0-9_] .
\W	Anything but ASCII word characters: [^\w] .
\p{group}	Any character in the named group. See the following group names. Many of the group names are from POSIX, which is why p is used for this character class.
\P{group}	Any character not in the named group.
\p{Lower}	ASCII lowercase letter: [a-z] .
\p{Upper}	ASCII uppercase: [A-Z] .
\p{ASCII}	Any ASCII character: [\x00-\x7f] .
\p{Alpha}	ASCII letter: [a-zA-Z] .
\p{Digit}	ASCII digit: [0-9]
\p{XDigit}	Hexadecimal digit: [0-9a-fA-F] .
\p{Alnum}	ASCII letter or digit: [\p{Alpha}\p{Digit}] .
\p{Punct}	ASCII punctuation: one of !"#\$%&'()*+,-./:;<=>?@[\\]^_`{ }~ .
\p{Graph}	Visible ASCII character: [\p{Alnum}\p{Punct}] .
\p{Print}	Visible ASCII character: same as \p{Graph} .
\p{Blank}	ASCII space or tab: [\t] .
\p{Space}	ASCII whitespace: [\t\n\f\r\x0B] .

Table 27–1. Java regular expression quick reference (continued)

Syntax	Matches
<code>\p{Cntrl}</code>	ASCII control character: <code>[\x00-\x1f\x7f]</code> .
<code>\p{category}</code>	Any character in the named Unicode category. Category names are one or two letter codes defined by the Unicode standard. One letter codes include L for letter, N for number, S for symbol, Z for separator and P for punctuation. Two-letter codes represent subcategories, such as Lu for uppercase letter, Nd for decimal digit, Sc for currency symbol, Sm for math symbol, and Zs for space separator. See <code>java.lang.Character</code> for a set of constants that correspond to these subcategories, and note that the full set of one and two-letter codes are not documented in this book.
<code>\p{block}</code>	Any character in the named Unicode block. In Java regular expressions, block names begin with “In”, followed by mixed-case capitalization of the Unicode block name, without spaces or underscores. For example: <code>\p{InOgham}</code> or <code>\p{InMathematicalOperators}</code> . See <code>java.lang.Character.UnicodeBlock</code> for a list of Unicode block names.
Sequences, alternatives, groups, and references	
<code>xy</code>	Match <i>x</i> followed by <i>y</i> .
<code>x y</code>	Match <i>x</i> or <i>y</i> .
<code>(...)</code>	Grouping. Group subexpression within parentheses into a single unit that can be used with <code>*</code> , <code>+</code> , <code>?</code> , <code> </code> , and so on. Also “capture” the characters that match this group for use later.
<code>(?:...)</code>	Grouping only. Group subexpression as with <code>()</code> , but do not capture the text that matched.
<code>\n</code>	Match the same characters that were matched when capturing group number <i>n</i> was first matched. Be careful when <i>n</i> is followed by another digit: the largest number that is a valid group number will be used.
Repetition^a	
<code>x?</code>	Zero or one occurrence of <i>x</i> ; i.e., <i>x</i> is optional.
<code>x*</code>	Zero or more occurrences of <i>x</i> .
<code>x+</code>	One or more occurrences of <i>x</i> .
<code>x{n}</code>	Exactly <i>n</i> occurrences of <i>x</i> .
<code>x{n,}</code>	<i>n</i> or more occurrences of <i>x</i> .

Table 27–1. Java regular expression quick reference (continued)

Syntax	Matches
$x\{n,m\}$	At least n , and at most m occurrences of x .
Anchors^b	
<code>^</code>	The beginning of the input string, or if the MULTILINE flag is specified, the beginning of the string or of any new line.
<code>\$</code>	The end of the input string, or if the MULTILINE flag is specified, the end of the string or of line within the string.
<code>\b</code>	A word boundary: a position in the string between a word and a non-word character.
<code>\B</code>	A position in the string that is not a word boundary.
<code>\A</code>	The beginning of the input string. Like <code>^</code> , but never matches the beginning of a new line, regardless of what flags are set.
<code>\Z</code>	The end of the input string, ignoring any trailing line terminator.
<code>\z</code>	The end of the input string, including any line terminator.
<code>\G</code>	The end of the previous match.
<code>(?=x)</code>	A positive look-ahead assertion. Require that the following characters match x , but do not include those characters in the match.
<code>(?!x)</code>	A negative look-ahead assertion. Require that the following characters do not match the pattern x .
<code>(?<=x)</code>	A positive look-behind assertion. Require that the characters immediately before the position match x , but do not include those characters in the match. x must be a pattern with a fixed number of characters.
<code>(?<!x)</code>	A negative look-behind assertion. Require that the characters immediately before the position do not match x . x must be a pattern with a fixed number of characters.
Miscellaneous	
<code>(?>x)</code>	Match x independently of the rest of the expression, without considering whether the match causes the rest of the expression to fail to match. Useful to optimize certain complex regular expressions. A group of this form does not capture the matched text.

Table 27–1. Java regular expression quick reference (continued)

Syntax	Matches
(?onflags-offflags)	Don't match anything, but turn on the flags specified by <i>onflags</i> , and turn off the flags specified by <i>offflags</i> . These two strings are combinations in any order of the following letters and correspond to the following Pattern constants: i (CASE_INSENSITIVE), d (UNIX_LINES), m (MULTILINE), s (DOTALL), u (UNICODE_CASE), and x (COMMENTS). Flag settings specified in this way take effect at the point that they appear in the expression and persist until the end of the expression, or until the end of the parenthesized group of which they are a part, or until overridden by another flag setting expression.
(?onflags-offflags:x)	Match <i>x</i> , applying the specified flags to this subexpression only. This is a noncapturing group, such as (?:\...), with the addition of flags.
\Q	Don't match anything, but quote all subsequent pattern text until \E. All characters within such a quoted section are interpreted as literal characters to match, and none (except \E) have special meanings.
\E	Don't match anything; terminate a quote started with \Q.
#comment	If the COMMENT flag is set, pattern text between a # and the end of the line is considered a comment and is ignored.

^a These repetition characters are known as greedy quantifiers because they match as many occurrences of *x* as possible while still allowing the rest of the regular expression to match. If you want a “reluctant quantifier,” which matches as few occurrences as possible while still allowing the rest of the regular expression to match, follow the previous quantifiers with a question mark. For example, use **?* instead of ***, and *{2,}?* instead of *{2,}*. Or, if you follow a quantifier with a plus sign instead of a question mark, then you specify a “possessive quantifier,” which matches as many occurrences as possible, even if it means that the rest of the regular expression will not match. Possessive quantifiers can be useful when you are sure that they will not adversely affect the rest of the match, because they can be implemented more efficiently than regular greedy quantifiers.

^b Anchors do not match characters but instead match the zero-width positions between characters, “anchoring” the match to a position at which a specific condition holds.