Appendix A. REGULAR EXPRESSION SEARCH

Brief Background

A regular expression consists of a character string where some characters are given special meaning with regard to pattern matching. Regular expressions have been in use from the early days of computing, and provide a powerful and efficient way to parse, interpret and search and replace text within an application.

Supported Syntax[8]

Table A.1. Characters

`x`	The character x
`\\`	The backslash character
`\0n`	The character with octal value 0n (0 <= n <= 7)
`\0nn`	The character with octal value 0nn (0 <= n <= 7)
`\0mnn`	The character with octal value 0mnn (0 <= m <= 3, 0 <= n <= 7)
`\xhh`	The character with hexadecimal value 0xhh
`\uhhhh`	The character with hexadecimal value 0xhhhh
`\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`')
`\cx`	The control character corresponding to x

Table A.2. 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)

Table A.3. Predefined character classes

`.`	Any character (may or may not match line terminators)

\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]

Table A.4. 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}]
\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]

Table A.5. Classes for Unicode blocks and categories

\p{InGreek}	A character in the Greek block (simple block)
\p{Lu}	An uppercase letter (simple category)
\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)

Table A.6. 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, if any
\z	The end of the input

Table A.7. 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

Table A.8. 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

Table A.9. 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

Table A.10. Logical operators

`XY`	X followed by Y
`X|Y`	Either X or Y
`(X)`	X, as a capturing group

Table A.11. Back references

`\n`	Whatever the nth capturing group matched

Table A.12. Quotation

`\`	Nothing, but quotes the following character
`\Q`	Nothing, but quotes all characters until `\E`
`\E`	Nothing, but ends quoting started by `\Q`

Table A.13. Special constructs (non-capturing)

`(?:X)`	X, as a non-capturing group
`(?idmsux-idmsux)`	Nothing, but turns match flags on - off
`(?idmsux-idmsux:X)`	X, as a non-capturing group with the given flags 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

Backslashes, escapes, and quoting

The backslash character ('\') serves to introduce escaped constructs, as defined in the table above, as well as to quote characters that otherwise would be interpreted as unescaped constructs. Thus the expression `\\` matches a single backslash and `\{` matches a left brace.

It is an error to use a backslash prior to any alphabetic character that does not denote an escaped construct; these are reserved for future extensions to the regular-expression language. A backslash may be used prior to a non-alphabetic character regardless of whether that character is part of an unescaped construct.

Backslashes within string literals in Java source code are interpreted as required by the Java Language Specification as either Unicode escapes or other character escapes. It is therefore necessary to double backslashes in string literals that represent regular expressions to protect them from interpretation by the Java byte code compiler. The string literal "`\b`", for example, matches a single backspace character when interpreted as a regular expression, while "`\\b`" matches a word boundary. The string literal "`\(hello\)`" is illegal and leads to a compile-time error; in order to match the string (hello) the string literal "`\\(hello\\)`" must be used.

A.1. Character Classes

Character classes may appear within other character classes, and may be composed by the union operator (implicit) and the intersection operator (`&&`). The union operator denotes a class that contains every character that is in at least one of its operand classes. The intersection operator denotes a class that contains every character that is in both of its operand classes.

The precedence of character-class operators is as follows, from highest to lowest:

1	Literal escape	`\x`
2	Grouping	`[...]`
3	Range	`a-z`
4	Union	`[a-e][i-u]`
5	Intersection	`[a-z&&[aeiou]]`

Note that the set of meta characters that is in effect inside a character class is different from the set that is outside a character class. For instance, the regular expression . loses its special meaning inside a character class, while the expression - becomes a range forming metacharacter.

---

[8] Source: http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

Prev Next
6.3.4. Tier Home A.2. Line terminators