Regex cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Character classes

`[abc]`	matches **a** or **b**, or **c**.
`[^abc]`	negation, matches everything except **a**, **b**, or **c**.
`[a-c]`	range, matches **a** or **b**, or **c**.
`[a-c[f-h]]`	union, matches **a**, **b**, **c**, **f**, **g**, **h**.
`[a-c&&[b-c]]`	intersection, matches **b** or **c**.
`[a-c&&[^b-c]]`	subtraction, matches **a**.

Predefined character classes

`.`	Any character.
`\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]`

Boundary matches

`^`	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.

Pattern flags

`Pattern.CASE_INSENSITIVE` - enables case-insensitive matching.
`Pattern.COMMENTS` - whitespace and comments starting with **#** are ignored until the end of a line.
`Pattern.MULTILINE` - one expression can match multiple lines.
`Pattern.UNIX_LINES` - only the '**\n**' line terminator is recognized in the behavior of **.**, **^**, and **$**.

Useful Java classes & methods

PATTERN

A pattern is a compiler representation of a regular expression.

`Pattern compile(String regex)`
Compiles the given regular expression into a pattern.

`Pattern compile(String regex, int flags)`
Compiles the given regular expression into a pattern with the given flags.

`boolean matches(String regex)`
Tells whether or not this string matches the given regular expression.

`String[] split(CharSequence input)`
Splits the given input sequence around matches of this pattern.

`String quote(String s)`
Returns a literal pattern String for the specified String.

`Predicate<String> asPredicate()`
Creates a predicate which can be used to match a string.

MATCHER

An engine that performs match operations on a character sequence by interpreting a Pattern.

`boolean matches()`
Attempts to match the entire region against the pattern.

`boolean find()`
Attempts to find the next subsequence of the input sequence that matches the pattern.

`int start()`
Returns the start index of the previous match.

`int end()`
Returns the offset after the last character matched.

Quantifiers

Greedy	Reluctant	Possessive	Description
X?	X??	X?+	*X, once or not at all.*
X*	X*?	X*+	*X, zero or more times.*
X+	X+?	X++	*X, one or more times.*
X{n}	X{n}?	X{n}+	*X, exactly n times.*
X{n,}	X{n,}?	X{n,}+	*X, at least n times.*
X{n,m}	X{n,m}?	X{n,m}+	*X, at least n but not more than m times.*

Greedy - matches the longest matching group.
Reluctant - matches the shortest group.
Possessive - longest match or bust (no backoff).

Groups & backreferences

A group is a captured subsequence of characters which may be used later in the expression with a backreference.

`(...)` - defines a group.
`\N` - refers to a matched group.

`(\d\d)` - a group of two digits.
`(\d\d)/\1` - two digits repeated twice.
`\1` - refers to the matched group.

Logical operations

XY	**X** then **Y**.
X|Y	**X** or **Y**.

BROUGHT TO YOU BY JRebel