# **Regex** Cheat Sheet

JRebel | XRebel

## 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**. |