Cheat Sheet: Regular Expressions
Adapted from http://www.javacodegeeks.com/2012/11/java-regular-expression-tutorial-with-examples.html.

Matching symbols

Regex	Matches
.	any one character
^abc	“abc” at the beginning of a line
abc$	“abc” at the end of a line
[abc]	“a”, “b”, or “c”
[abc][12]	“a”, “b”, or “c” followed by “1” or “2”
[^abc]	anything EXCEPT “a”, “b”, or “c”
[a-c1-5]	“a”, “b”, “c”, “1”, “2”, “3”, “4” or “5”
ab|cd	“ab” or “cd”

Metacharacters

Regex	Matches
\d	any digits (same as [0-9])
\D	any non-digit (same as [^0-9])
\s	any whitespace character (spaces, tabs, newlines, etc.)
\S	any non-whitespace character
\w	any word character (same as [a-zA-Z_0-9])
\W	any non-word character
\b	a word boundary
\B	anything not a word boundary

Quantifiers

Regex	Matches
X?	X occurring once or not at all
X*	X occurring zero or more times
X+	X occurring one or more times
X{n}	X occurring exactly n times
X{n,}	X occurring n or more times
X{n,m}	X occurring at least n times but not more than m times

Escape characters used as symbols or quantifiers with “\”, e.g., /\./ matches a period, not any one character.
Use parentheses to enclose characters parsed as strings, e.g., /(abc)/ matches “abc” but not “ab.”

Cheat Sheet: Google Refine Expression Language (GREL)

A more complete reference is available at https://github.com/OpenRefine/OpenRefine/wiki/Google-refine-expression-language.
For a complete list of GREL functions, see https://github.com/OpenRefine/OpenRefine/wiki/GREL-Functions.

Function	What it does	What it returns	Parameters	Example
value.match(/*regex*/) value.match(/*regex*/)[*index*] value.match("*string*") value.match("*string*")[*index*]	Attempts to match the regular expression *regex* or string *string* with *value*. Use /.**regex*.*/ to match a partial string.	An array (even if only one match is found). If [*index*] is present, returns the corresponding string within the array.	*regex* = regular expression to match against *index* = index of a string within the array	value = "The cat can't lay on the cot" value.match(/c.t/) → ["cat", "cot"] value.match(/c.t/)[0] → "cat"
value.contains(/*regex*/) value.contains("*string*")	Determines whether *value* contains the regular expression *regex* or the string *string*.	A Boolean (true or false).	*regex* = regular expression to search	value = "coffee and tea and chai and mate" value.contains("and") → TRUE
value.replace(*t*, *u*) If *t* is a regular expression, use /(*t*)/	Returns *value* with all occurrences of the string or regular expression *t* replaced with the string *u*.	A string.	*t* = string or regex to replace *u* = string or regex that replaces *t*	value = "coffee and tea and chai and mate" value.replace(" and ", ", ") → "coffee, tea, chai, mate"
value.trim()	Removes any leading or trailing white space *value*.	A string.	n/a	value = " coffee " value.trim() → "coffee"
value.length()	String: Returns the length of *value*. Array: Returns the number of terms in the array *value*.	A number.	n/a	value = "coffee" value.length() → 6 value = ["coffee", "tea"] value.length() → 2
value.split(*delim*) value.split(*delim*)[*index*]	Splits string *value* into an array, breaking at each instance of the string delimiter *delim*.	An array. If [*index*] is present, returns the corresponding string within the array.	*delim* = delimiter between array elements *index* = index of a string within the array	value = "coffee, tea, chai, mate" value.split(", ") → ["coffee", "tea", "chai", "mate"] value.split(", ")[-1] → "mate"
value.join(*separator*)	Joins the elements in the array *value* into a string with connector *separator*.	A string.	*separator* = the link used to join array elements into a string	value = ["coffee", "tea", "chai", "mate"] value.join(" AND ") → "coffee AND tea AND chai AND mate"

value.slice(*x*, *y*)	String: Gives each character in *value* an index as in an array, and returns the part of this array with index *x* up to but not including index *y*. Array: Returns the elements of an array from index *x* up to but not including index *y*.	String: a string. Array: an array.	*x* = index at which to start slice *y* = index before which to stop slice	value = “coffee” value.slice(1, 4) → “off” value = [“coffee”, “tea”, chai”, “mate”] value.slice(0, 2) → [“coffee”, “tea”, “chai”]
value.partition(*fragment*) value.partition(*fragment*)[*index*] value.partition(*fragment*, true) = omits *fragment* from returned array	Returns an array consisting of the part of *value* before the first occurrence of *fragment*, *fragment*, and the part of *value* after the first occurrence of *fragment*.	An array with three terms. If [*index*] is present, returns the corresponding string within the array.	*fragment* = the substring or regular expression around which *value* is partitioned *index* = index of a string within the array	value = “coffee and tea and chai” value.partition(“ and ”) → [“coffee”, “ and ”, “tea and chai”] value.partition(“ and ”)[1] → “ and ”
value.rpartition(*fragment*) value.rpartition(*fragment*)[*index*] value.rpartition(*fragment*, true) = omits *fragment* from returned array	Returns an array consisting of the part of *value* before the last occurrence of *fragment*, *fragment*, and the part of *value* after the last occurrence of *fragment*.	An array with three terms. If [*index*] is present, returns the corresponding string within the array.	*fragment* = the substring or regular expression around which *value* is partitioned *index* = index of a string within the array	value = “coffee and tea and chai” value.rpartition(“ and ”) → [“coffee and tea”, “ and ”, “chai”] value.rpartition(“ and ”)[0] → “coffee and tea”
value.reverse()	Reverses the order of the elements in the array *value*.	An array.	n/a	value = [“coffee”, “tea”, “chai”, “mate”] value.reverse() → [“mate”, “chai”, “tea”, “coffee”]
not(*booleanexp*)	Returns “TRUE” if the value of *booleanexp* is false	A Boolean (true or false).	*booleanexp* = a function or expression that returns TRUE or FALSE	value = “coffee” not(value.contains(“a”)) → TRUE