

Regular Expression in Java

Character Classes

[abc]	Either a OR b OR c
[^abc]	Except a,b, and c
[a-z]	Any Lower Case Alphabet Symbol
[A-Z]	Any Upper Case Alphabet Symbol
[a-zA-Z]	Any Alphabet Symbol
[0-9]	Any Digit from 0-9
[a-zA-Z0-9]	Any Alpha Numeric Symbol
[^a-zA-Z0-9]	Except Alpha Numeric Symbol (Special Character)
[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)

Predefined Character Classes

<code>\s</code>	Space Character [<code>\t\n\r</code>]
<code>\S</code>	Any Character Except Space [<code>^\s</code>]
<code>\d</code>	Any Digit from [0-9]
<code>\D</code>	Any Character Except Digit [<code>^0-9</code>]
<code>\w</code>	Any Word Character(Alpha Numeric Character including underscore)[<code>a-zA-Z0-9_</code>]
<code>\W</code>	Except Word Character(i.e Special Character) [<code>^\w</code>]
<code>.</code>	Any Symbol Including Special Character Also
<code>\.</code>	Period (also can be written as) [<code>.</code>]

Quantifiers(Means Quantity):

We can Use Quantifiers to Specify Number of Occurrences to Match

a	Exactly One a
a+	Atleast One a
a*	Any Number of a's including Zero Number Also
a?	Atmost One a
{m}	m Repetitions
{m,n}	m to n Repetitions
{m,}	Minimum m Repetitions

Anchors

<code>^</code>	Beginning of a line
<code>\$</code>	End of a line
<code>\b</code>	Word Bound
<code>\B</code>	Not Word Bound

Groups & Reference

(abc)	Capture Group
(a(bc))	Capture Sub-group
(.*)	Capture all
(abc def)	Matches abc or def
(?<name>abc)	Named Capturing Group
\1	Numeric Reference
(?:abc)	Non-Capturing Group
(?>abc)	Atomic Group(Don't backtrack once a match is successful)
\k<name>	To Backreference Named Group
\${name}	Group Reference in Replacement String

Modifier Flags

(?i)	Case Insensitive Mode
(?-i)	Turning off Case insensitive mode
(?m)	Multiline mode, ^ and \$ will be used to match at the start and end of each line.
(?s)	Dot-All(Single Line) Mode, line terminators (\n or \r or \r\n) are treated as literal.
(?x)	Comment & White-Spaces Mode, mode enables white whitespaces and comments in the pattern
(?u)	Unicode Aware Case Folding

All In One

abc	Letters	*	Zero or more repetitions
123	Digits	+	One or more repetitions
\d	Any Digit	?	Optional character
\D	Any Non-digit character	\s	Any Whitespace
.	Any Character	\S	Any Non-whitespace character
\.	Period	^...\$	Starts and ends
[abc]	Only a, b, or c	\b	Word Bound
[^abc]	Not a, b, nor c	\B	Not Word Bound
[a-z]	Characters a to z	(abc)	Capture Group
[0-9]	Numbers 0 to 9	(a(bc))	Capture Sub-group
\w	Any Alphanumeric character	(.*)	Capture all
\W	Any Non-alphanumeric character	(?<name>abc)	Named Capturing Group
X{m}	Repetitions(X, exactly n times)	\1	Numeric Reference
X{m,}	Repetitions(X, at least n times)	(?:abc)	Non-Capturing Group
{m,n}	X, at least n but not more than m times	(abc def)	Matches abc or def
		\k<name>	To Backreference Named Group

Write a Regular Expression to Represent All 10-digit Mobile Numbers

Rules:

- Every Number should contain Exactly 10 Digits.
- The 1st Digit should be 7 OR 8 OR 9.

[7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]
OR
[789][0-9]{9}
OR
[7-9][0-9]{9}

10 Digit OR 11 Digit

If the number contains 11 digits then the first digit should be 0

0?[7-9][0-9]{9}

10 Digit OR 11 Digit or 12 Digit

If the number contains 12 digits then the first two digit should be 91

(0|91)?[7-9][0-9]{9}

Write a Regular Expression to represent All valid mail ids:

[a-zA-Z0-9][a-zA-Z0-9_]*@[a-zA-Z0-9]+([.][a-zA-Z]+)+

Only Gmail Id

[a-zA-Z0-9][a-zA-Z0-9_]*@gmail[.]com

Assignments:

Write a program to extract number and email id from a file using regular expression

Write a program to get names of file of a directory having particular extension using regular expression

Write a Regular Expression to Represent All Valid Yava Language Identifiers:

Rules:

- The Allowed Characters are a-z, A-Z, 0-9, #, and \$.
- The Length of Identifier should be atleast 2.
- The 1st Character should be lowercase Alphabetical Symbol from a To k.
- The 2nd Character should be Digit Divisible by 3 (0, 3, 6, 9).

[a-k][0369][a-zA-Z0-9#\$]*

Regular Expression to represent all Names starts with a | A

[aA][a-zA-Z]*

Regular Expression to represent all Names ends with l or L

[a-zA-Z]*[lL]

Regular Expression to represent all Names starts with a or A and ends with l or L

[aA][a-zA-Z]*[lL]

Write a Regular Expression to Represent All Valid Java Language Identifiers:

Rules:

- The Allowed Characters are a-z, A-Z, 0-9, #, and \$.
- The Length of Identifier should be atleast 2.
- The 1st Character should be lowercase Alphabetical Symbol from a To k.
- The 2nd Character should be Digit Divisible by 3 (0, 3, 6, 9).

[a-k][0369][a-zA-Z0-9#\$]*

Regular Expression to represent all Names starts with a | A

[aA][a-zA-Z]*

Regular Expression to represent all Names ends with l or L

[a-zA-Z]*[lL]

Regular Expression to represent all Names starts with a or A and ends with l or L

[aA][a-zA-Z]*[lL]

Other Examples

Exercise 1: Matching Characters

Task	Text
Match	abcdefg
Match	abcde
Match	abc

Solution 1: **[a-z]+**

Good Solution: **abc**

Another Solution : **abc[a-z]***

Exercise 2: Matching With Wildcards

Task	Text
Match	cat.
Match	896.
Match	?=+.
Skip	abc1

Solution 1: **[a-z0-9?+=+]{3}[.]**

Exercise 3: Matching Characters

Task	Text
Match	can
Match	man
Match	fan
Skip	dan
Skip	ran
Skip	pan

Solution 1: **[cmf]an**

Exercise 4: Excluding Characters

Task	Text
Match	hog
Match	dog
Skip	bog

Solution 1: **[^b]og**

Other Examples

Exercise 5: Matching Character Ranges

Task	Text
Match	Ana
Match	Bob
Match	Cpc
Skip	aax
Skip	bby
Skip	ccz

Solution 1: `[A-Z][a-z]*`

Good Solution : `[A-C][n-p][a-c]`

Exercise 6: Matching Repeated Characters

Task	Text
Match	wazzzzzup
Match	wazzzup
Skip	wazup

Solution 1: `wa[z]{2,}up`

Good solution: `waz{3,5}up`

Exercise 7: Matching Repeated Characters

Task	Text
Match	aaaabcc
Match	aabbbbc
Match	aacc
Skip	a

Solution 1: `[aa]+b*[c]+`

Solution 2: `aa+b*c+`

Good solution: `a{2,4}b{0,4}c{1,2}`

Alternatively, an even more restrictive expression would be `a{2,4}b{0,4}c{1,2}` which puts both an upper and lower bound on the number of each of the characters.

Exercise 8: Matching Optional Characters

Task	Text
Match	1 file found?
Match	2 files found?
Match	24 files found?
Skip	No files found.

Solution 1: `[0-9]+ file[s]* found?`

Good Solution: `\d+ files? found\?`

Other Examples

Exercise 9: Matching Whitespaces

Task	Text
Match	1. abc
Match	2. abc
Match	3. abc
Skip	4.abc

Solution 1: **[1-3]\.s+abc**

Good Solution: **\d\.s+abc**

Exercise 10: Matching Lines

Task	Text
Match	Mission: successful
Skip	Last Mission: unsuccessful
Skip	Next Mission: successful upon capture of target

Solution 1: **^Mission: successful\$**

Exercise 11: Matching Groups

Task	Text	Capture Groups
Capture	file_record_transcript.pdf	file_record_transcript
Capture	file_07241999.pdf	file_07241999
Skip	testfile_fake.pdf.tmp	

Solution 1: **^(file_\w+)\.pdf\$**

Good Solution: **^(file.+)\.pdf\$**

Exercise 12: Matching Nested Groups

Task	Text	Capture Groups
Capture	Jan 1987	Jan 1987 1987
Capture	May 1969	May 1969 1969
Capture	Aug 2011	Aug 2011 2011

Solution 1: **([a-zA-Z]+ (\d+))**

Good Result: **(\w+ (\d+))**

Another Result: **(\w+\.s+(\d+))**

We can alternatively use `\s+` in lieu of the space, to catch any number of whitespace between the month and year.

Other Examples

Exercise 13: Matching Nested Groups

Task	Text	Capture Groups	
Capture	1280x720	1280	720
Capture	1920x1600	1920	1600
Capture	1024x768	1024	768

Solution 1: `(\d{4})x(\d+)`

Good Solution: `(\d+)x(\d+)`

Exercise 14: Matching Conditional Text

Task	Text
Match	I love cats
Match	I love dogs
Skip	I love logs
Skip	I love cogs

Solution 1: `I love ([cb]ats*|[dh]ogs*)`

Good Solution: `I love (cats|dogs)`

<https://regexr.com/>

<https://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html#special>