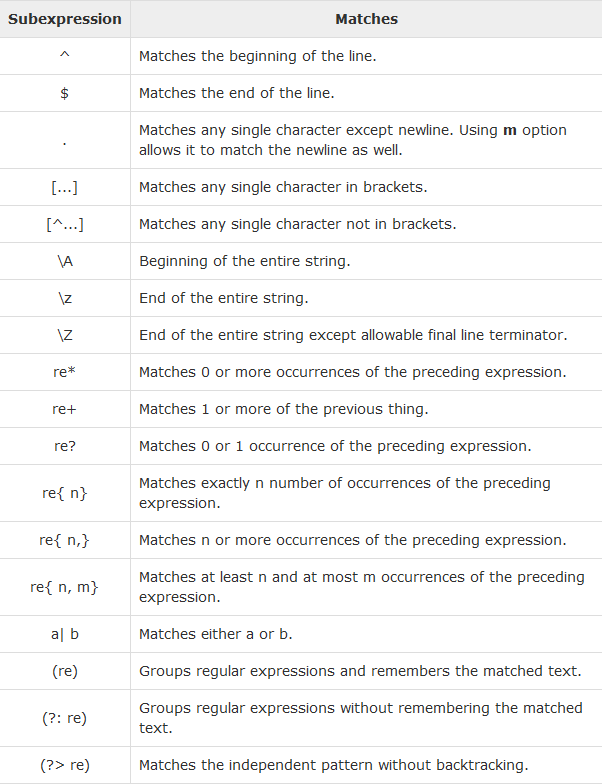
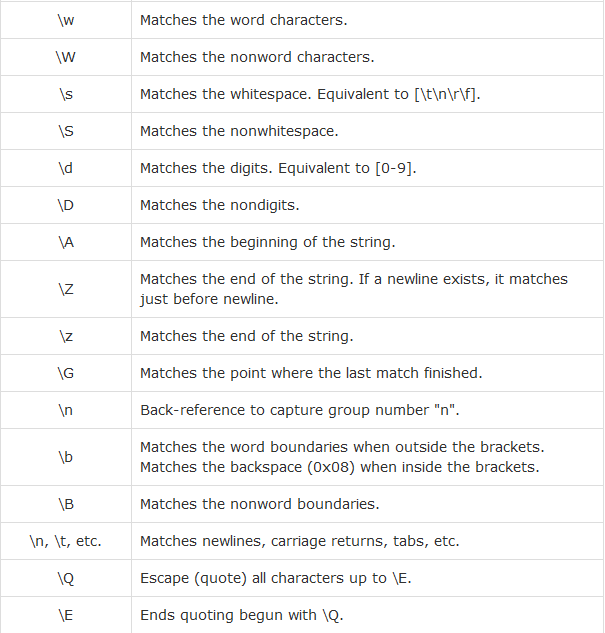
# 0. Table





# 0. Link:

http://www.tutorialspoint.com/java/java\_regular\_expressions.htm

Có tổng cộng 34 regular expression. Nào cùng tìm hiểu

<https://docs.oracle.com/javase/tutorial/essential/regex/pre_char_classes.html>

# 0. BooK:

Regular Expression trên trang của Bình giới thiệu.

Java Regular Expression mình thấy trên mạng

# 0. Exercise:

https://regexone.com/

# 0. Chú thích:

### **\r:**

\r : thay thế chuỗi sau \r ra phía trước \r

Abcdef\r123

Sẽ in ra: 123def

Abc\r123456

Sẽ in ra: 123456

### **\f:**

NguyenThien\fThuat

Sẽ in ra:



# 1. […]

***Matches any single character in brackets.***

## Code:

**package** demo.com;

**import** java.util.\*;

**public** **class** test {

**public** **static** **void** main(String[] args) {

String scanMe = "abcdefghijkl";

String delim = "[aeoiu]";

Scanner scanner = **new** Scanner(scanMe);

scanner.useDelimiter(delim);

**while** (scanner.hasNext()) {

System.***out***.println(scanner.next());

}

}

}

## Result:



Ở đây ta sử dụng bất kỳ single character trong bracket để làm split, vì thế a,e,o,i,u sẽ là các split.

# 1. [^…]

**Ngược lại với […]**

# 2. re+, re\*, re?

**re+: Matches 1 or more of the previous thing.**

**re\*: Matches 0 or more occurrences of the preceding expression.**

**re?: Matches 0 or 1 occurrence of the preceding expression.**

## Code:

**package** demo.com;

**import** java.util.\*;

**public** **class** test {

**public** **static** **void** main(String[] args) {

String scanMe = "132xxxxx555";

String delim = "x";

Scanner scanner = **new** Scanner(scanMe);

scanner.useDelimiter(delim);

**while** (scanner.hasNext()) {

System.***out***.println(scanner.next());

}

}

}

## Result:



Ở đây vì ta chỉ sử dụng mỗi x làm split, vì thế nên có nhiều token được tạo ra, mà các token này là null

**Sử dụng re+: Xác định 1 hoặc nhiều x làm split**

String delim = "x+";



**Sử dụng re\*: Xác định 0 hoặc nhiều x làm split**

****

**Sử dụng re?: Xác định 0 hoặc 1 x làm split**

****

# 3. .

**Matches any single character except newline. Using m option allows it to match the newline as well.**

Nghĩa là nó match với bất kỳ ký tự nào ngoại trừ xuống dòng.

## Code:

**package** demo.com;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

String any = ".\*";

String test1 = "Hello", test2 = "29", test3 = "", test4 = "\n";

System.***out***.println(test1.matches(any));

System.***out***.println(test2.matches(any));

System.***out***.println(test3.matches(any));

System.***out***.println(test4.matches(any));

}

}

## Result:



Vì .\* là match với bất kỳ ký tự nào, với \* là 0 hoặc nhiều cho cái dấu chấm(.) nên 3 kết quả đầu đúng.

## Code2:

**package** demo.com;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

String any = ".+";

String test1 = "Hello", test2 = "29", test3 = "", test4 = "sa\ns";

System.***out***.println(test1.matches(any));

System.***out***.println(test2.matches(any));

System.***out***.println(test3.matches(any));

System.***out***.println(test4.matches(any));

}

## Result2:

Vì bây giờ là dấu +, nghĩa là 1 hoặc nhiều, nên test3 và test 4 false

# 4. \d

**Matches the digits. Equivalent to [0-9]**

Match với số 0 đến 9, tương đương [0-9]

## Code:

**package** demo.com;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

String numbersOnly = "\\d+";

String test1 = "Hello", test2 = "29", test3 = "", test4 = "a5";

System.***out***.println(test1.matches(numbersOnly));

System.***out***.println(test2.matches(numbersOnly));

System.***out***.println(test3.matches(numbersOnly));

System.***out***.println(test4.matches(numbersOnly));

}

}

## Result:

Vì \d match với số, thêm + là 1 hoặc nhiều nên chỉ có test2 là true. Nếu đổi + thành \* nữa thì cái test3 sẽ true luôn.

# 4. \D

Ngược với \d

# 5. \w

**Matches the word characters.**

Match với ký tự là word character: a-z, A-Z, 0-9.

Ở đây chấp nhận luôn các ký tự có gạch dưới, mà trong java chưa gặp nên thôi bỏ qua đi.

## Code:

**package** demo.com;

**import** java.text.DateFormat;

**import** java.text.NumberFormat;

**import** java.util.\*;

**public** **class** test {

**public** **static** **void** main(String[] args) {

String wordCharacter = "\\w\*";

String test1 = "Hello", test2 = "29", test3 = "", test4 = "\n", test5 = "#das";

System.***out***.println(test1.matches(wordCharacter));

System.***out***.println(test2.matches(wordCharacter));

System.***out***.println(test3.matches(wordCharacter));

System.***out***.println(test4.matches(wordCharacter));

System.***out***.println(test5.matches(wordCharacter));

}

}

## Result:

Ta thấy test4 và test5 chứa các ký tự k phải \w đã nói nên 2 nó ra false.

# 5. \W

Ngược với \w

# 6. &&

Và

## Code:

**package** demo.com;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** test {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("[0-9&&[345]]");

Matcher matcher = pattern.matcher("3");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println(matcher.group());

System.***out***.println(matcher.start());

System.***out***.println(matcher.end());

}

**if** (!found) {

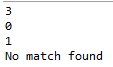
System.***out***.println("No match found");

}

}

}

## Result:



## Thử nghiệm tương tự:

Enter your regex: [0-9&&[345]]

Enter input string to search: 4

I found the text "4" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[345]]

Enter input string to search: 5

I found the text "5" starting at index 0 and ending at index 1.

Enter your regex: [0-9&&[345]]

Enter input string to search: 2

No match found.

Enter your regex: [0-9&&[345]]

Enter input string to search: 6

No match found.

# 7. \s

**Matches the whitespace. Equivalent to [\t\n\r\f].**

Match với dấu cách, \t, \n, \r, \f.

## Code:

**package** demo.com;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** test {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("\\s");

Matcher matcher = pattern.matcher(" ");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println(matcher.group());

System.***out***.println(matcher.start());

System.***out***.println(matcher.end());

}

**if** (!found) {

System.***out***.println("No match found");

}

}

}

## Result:



Kết quả tương tự với \t, \n, \r, \f

# 7. \S

Ngược lại với \s

# 8. {}

**re{n}: Matches exactly n number of occurrences of the preceding expression.**

**re{n,}: Matches n or more occurrences of the preceding expression.**

**re{n,m}: Matches at least n and at most m occurrences of the preceding expression.**

## Code:

**package** demo.com;

**public** **class** test {

**public** **static** **void** main(String[] args) {

String any = "a{1,3}";

String test1 = "aa", test2 = "aaaaaa", test3 = "", test4 = "aaa";

System.***out***.println(test1.matches(any));

System.***out***.println(test2.matches(any));

System.***out***.println(test3.matches(any));

System.***out***.println(test4.matches(any));

}

}

## Result:

Ta thấy test1 và test4 là đúng vì >=1 chữ a và <=3 chữ a.

# 9. Backreference

**Nhắc lại ký tự đã khớp, chứ không phải nhắc lại cả lớp đã khớp như ?+\***

## Code:

**package** demo.com;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** test {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("(\\d\\d)\\1");

Matcher matcher = pattern.matcher("1212");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println("group:" +matcher.group());

System.***out***.println("start:" +matcher.start());

System.***out***.println("end:" +matcher.end());

}

**if** (!found) {

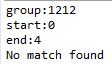
System.***out***.println("No match found");

}

}

}

## Result:



Ở đây [\\d](file:///\\d) la match với 1 số, 2 cái [\\d](file:///\\d) nên match 2 cái, ở đây match 12. [\\1](file:///\\1) là kiu lập lại 12 1 lần, nên ở đây match đúng với số 1212.

Nếu t thay bằng 1213 thì ra kết quả No match found.

# 10. ^

**Matches the beginning of the line.**

Match với phần đầu của line.

## Code:

**package** demo.com;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** test {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("^dog$");

Matcher matcher = pattern.matcher("dog");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println("group:" +matcher.group());

System.***out***.println("start:" +matcher.start());

System.***out***.println("end:" +matcher.end());

}

**if** (!found) {

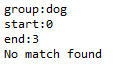
System.***out***.println("No match found");

}

}

}

## Result:



Ta thấy ở đây có $ ám chỉ phần kết thúc , nên “dog” match được.

Nếu là “ dog” là không được vì mở đầu là “ “.

Nếu là “dogabcd” là không được vì không giống kết thúc như mô tả.

# 11. $

**Matches the end of the line.**

Match vào kết thúc của line.

## Code:

**package** demo.com;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** test {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("[az]$");

Matcher matcher = pattern.matcher("sda sda saz");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println("group:" +matcher.group());

System.***out***.println("start:" +matcher.start());

System.***out***.println("end:" +matcher.end());

found = **true**;

}

**if** (!found) {

System.***out***.println("No match found");

}

}

}

## Result:



# 12. \b

**A word boundary**

Ranh giới của từ với word character, xem ví dụ để rõ hơn.

## Code:

**package** demo.com;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** test {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("\\bdog\\b");

Matcher matcher = pattern.matcher("I have a dog that is very cute");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println("group:" +matcher.group());

System.***out***.println("start:" +matcher.start());

System.***out***.println("end:" +matcher.end());

found = **true**;

}

**if** (!found) {

System.***out***.println("No match found");

}

}

}

## Result:



Vì chữ dog nằm giữa 2 space (không phải word character) nên ok

Nếu matcher là: "I have a doggie that is very cute" thì không được.

Nếu matcher là: "I have a ♥dog♥ that is very cute" thì được.

# 11. \B

Ngược với \b

# 13. \G

**The end of the previous match**

Phía sau phải có cái giống thì cái này nó sẽ match zô cái trước. Cái này chưa hiểu lắm

## Code:

**package** demo.com;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** test {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("\\Gdog");

Matcher matcher = pattern.matcher("dog dog");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println("group:" +matcher.group());

System.***out***.println("start:" +matcher.start());

System.***out***.println("end:" +matcher.end());

found = **true**;

}

**if** (!found) {

System.***out***.println("No match found");

}

}

}

## Result:



# 16. (.\*?)

Cái này khá đặc biệt:

. ám chỉ muốn ký tự nào cũng được except xuống dòng.

\* kéo dài cái . có có 0 hoặc nhiều hơn.

?: cái này kết hợp () người ta gọi là **Lazy.** Nghĩa là cái group này nó phải match càng ít ký tự càng tốt, cho đến khi gặp group thứ 2, vì thế, nếu có Lazy mà không có group thứ 2 thì ? đặt ở đây xem như sai cú pháp.

Ví dụ:

(.\*?)(\d+)(.\*)

Java: (.\*?)([\\d+)(.\*)](file:///\\d+)(.*))

**package** com.example;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** Example {

**private** Pattern pattern;

**private** Matcher matcher;

**private** **static** **final** String ***PATTERN*** = "(.\*?)(\\d+)(.\*)";

**public** Example() {

pattern = Pattern.*compile*(***PATTERN***);

}

**public** **void** validate(**final** String username) {

matcher = pattern.matcher(username);

**boolean** found = **false**;

**while** (matcher.find()) {

**if** (!found) {

System.***out***.println("Match found");

}

System.***out***.println("group 1: " + matcher.group(1));

System.***out***.println("group 2: " + matcher.group(2));

System.***out***.println("group 3: " + matcher.group(3));

found = **true**;

}

**if** (!found) {

System.***out***.println("No match found");

}

System.***out***.println("=============");

}

**public** **static** **void** main(String[] args) {

Example example = **new** Example();

example.validate("This order was placed for QT3000! OK?");

}

}

Result:

Match found

group 1: This order was placed for QT

group 2: 3000

group 3: ! OK?

=============

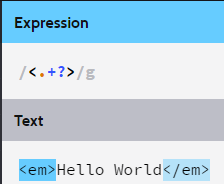
Một áp dụng rất hữu dụng là khi ta muốn loại bỏ tag trong html:

Trong project DogDao mình có làm:

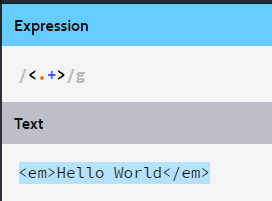
match = matcher.group().replaceAll("<.\*?>", "");

Sử dụng tool thấy khá rõ khác nhau giữa greedy và lazy:

Lazy: lưu ý là cái / và /g là global search gì đó của tool



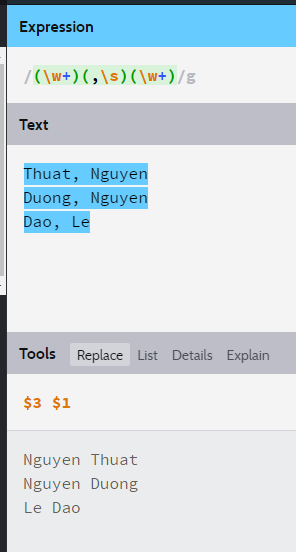
Greedy:



# 17. Group – Capture – Non-capture

## Group và Capture

Cái này rất hữu dụng khi kết hợp hàm replaceAll.



Lưu ý là / và /g ở cái expression là của cái tool, không hiểu sao lại có 2 cái đó.

Chương trình ví dụ java:

**package** com.example;

**public** **class** ExampleGroup {

**public** **static** **void** main(String[] args) {

String str = "Thuat, Nguyen";

String str2 = str.replaceAll("(\\w+)(,\\s)(\\w+)", "$3 $1");

System.***out***.println(str2);

}

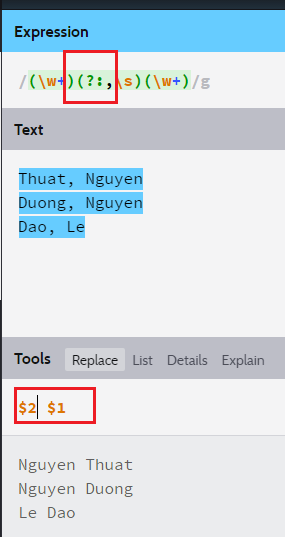
}

***Result:***



## Non-capture

Đơn giản là đặt ?: vào phần đầu tiên của group. Vì thế Group này sẽ không được đếm số.



# 14. Methods of Pattern Class

https://docs.oracle.com/javase/tutorial/essential/regex/pattern.html

## CASE\_INSENSITIVE – KHÔNG PHÂN BIỆT HOA THƯỜNG.

Không phân biệt hoa thường.

### Code:

**package** demo.com;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** test {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("dog", Pattern.***CASE\_INSENSITIVE***);

Matcher matcher = pattern.matcher("DOGdOg");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println("group:" +matcher.group());

System.***out***.println("start:" +matcher.start());

System.***out***.println("end:" +matcher.end());

found = **true**;

}

**if** (!found) {

System.***out***.println("No match found");

}

}

}

### Result:



# 15. Các regular expression phổ biến nên biết:

<http://www.mkyong.com/regular-expressions/10-java-regular-expression-examples-you-should-know/>

## 1. Username

### Pattern:

^[a-zA-Z0-9\_-]{3,15}$

### Giải thích:

^ # Start of the line

[a-z0-9\_-] # Match characters and symbols in the list, a-z,A-Z, 0-9 , underscore , hyphen

{3,15} # Length at least 3 characters and maximum length of 15

$ # End of the line

### Code:

**package** demo.com;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("^[a-zA-Z0-9\_-]{3,15}$");

Matcher matcher = pattern.matcher("\_9Nguye\_-");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println("group:" +matcher.group());

System.***out***.println("start:" +matcher.start());

System.***out***.println("end:" +matcher.end());

found = **true**;

}

**if** (!found) {

System.***out***.println("No match found");

}

}

}

### Result:



Class viết cho test nhiều case:

**package** com.example;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** Example {

**private** Pattern pattern;

**private** Matcher matcher;

**private** **static** **final** String ***PATTERN*** = "^[a-zA-Z0-9\_-]{3,15}$";

**public** Example() {

pattern = Pattern.*compile*(***PATTERN***);

}

**public** **void** validate(**final** String username) {

matcher = pattern.matcher(username);

**boolean** found = **false**;

**while** (matcher.find()) {

**if** (!found) {

System.***out***.println("Match found");

}

System.***out***.println("group:" + matcher.group());

System.***out***.println("start:" + matcher.start());

System.***out***.println("end:" + matcher.end());

found = **true**;

}

**if** (!found) {

System.***out***.println("No match found");

}

System.***out***.println("=============");

}

**public** **static** **void** main(String[] args) {

Example example = **new** Example();

example.validate("mkyong34");

example.validate("mkyong\_2002");

example.validate("mkyong-2002");

example.validate("mk3-4\_yong");

}

}

Trên mạng có class UsernameValidator khá hay:

**public** **class** UserNameValidator{

**private** Pattern pattern;

**private** Matcher matcher;

/\*

^ # Start of the line

[a-zA-Z0-9\_-] # Match characters and symbols in the list, a-z, A-Z,0-9, underscore, hyphen

{5,15} # Length at least 5 characters and maximum length of 15

$

\*/

**private** **static** **final** String ***USERNAME\_PATTERN*** = "^[a-zA-Z0-9\_-]{5,15}$";

**public** UserNameValidator(){

pattern = Pattern.*compile*(***USERNAME\_PATTERN***);

}

/\*\*

\* Validate username with regular expression

\* **@param** username username for validation

\* **@return** true valid username, false invalid username

\*/

**public** **boolean** validate(**final** String username){

matcher = pattern.matcher(username);

**return** matcher.matches();

}

}

## 2. Password

### Pattern:

((?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[@#$%]).{6,20})

Password pattern này có vấn đề vì chỉ toàn là group, mà group thì 1 String có thể có 2 group trở lên.

### Giải thích:

( # Start of group

(?=.\*\d) # must contains one digit from 0-9

(?=.\*[a-z]) # must contains one lowercase characters

(?=.\*[A-Z]) # must contains one uppercase characters

(?=.\*[@#$%]) # must contains one special symbols in the list "@#$%"

. # match anything with previous condition checking

{6,20} # length at least 6 characters and maximum of 20

) # End of group

### Code:

**package** demo.com;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("((?=.\*\\d)(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[@#$%]).{6,20})");

Matcher matcher = pattern.matcher("dsd@1aE");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println("group:" +matcher.group());

System.***out***.println("start:" +matcher.start());

System.***out***.println("end:" +matcher.end());

found = **true**;

}

**if** (!found) {

System.***out***.println("No match found");

}

}

}

### Result:



## 3. Hexadecimal Color Code:

Đây là hệ màu thập lục phân

Trang này có Info khá nhiều: <http://www.color-hex.com>

Màu ký hiệu từ #000000 đến #FFFFFF (từ đen đến trắng, F cũng được mà f cũng được, F và f là giống nhau)

Có thể viết tắt #FFF, ở đây nó nghĩa là #FFFFFF, hay #121 nghĩa là #112211

### Pattern:

^#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})$

### Giải thích:

^ #start of the line

# # must constains a "#" symbols

( # start of group #1

[A-Fa-f0-9]{6} # any strings in the list, with length of 6

| # ..or

[A-Fa-f0-9]{3} # any strings in the list, with length of 3

) # end of group #1

$ #end of the line

### Match:

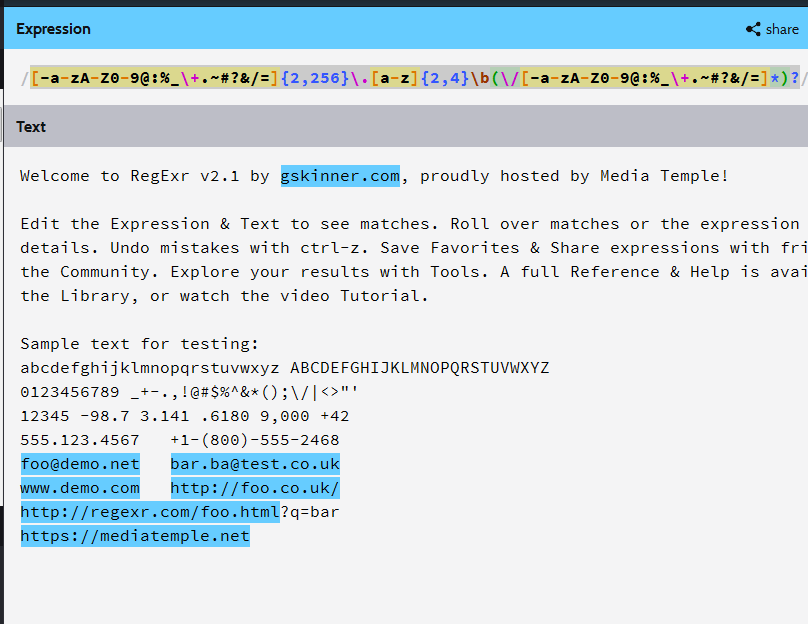
1. “#1f1f1F”, “#AFAFAF”,”#1AFFa1″,”#222fff”, “#F00”, “#F00”

### Not Match:

1. “123456” – must start with a “#” symbol  
2. “#afafah” – “h” is not allow, valid letter from “a” to “f”  
3. “#123abce” – either 6 length or 3 length  
4. “aFaE3f” – must start with a “#” symbol, either 6 length or 3 length  
5. “F00” – must start with a “#” symbol  
6. “#afaf” – either 6 length or 3 length  
7. “#F0h” – “h” is not allow, valid letter from “a” to “f”

## 4. URL Similar

[-a-zA-Z0-9@:%\_\+.~#?&/=]{2,256}\.[a-z]{2,4}\b(\/[-a-zA-Z0-9@:%\_\+.~#?&/=]\*)?



***Giải thích:***

[ **Character set.** Match any character in the set.

- **Character.** Matches a "-" character (char code 45).

a-z **Range.** Matches a character in the range "a" to "z" (char code 97 to 122).

A-Z **Range.** Matches a character in the range "A" to "Z" (char code 65 to 90).

0-9 **Range.** Matches a character in the range "0" to "9" (char code 48 to 57).

@ **Character.** Matches a "@" character (char code 64).

: **Character.** Matches a ":" character (char code 58).

% **Character.** Matches a "%" character (char code 37).

\_ **Character.** Matches a "\_" character (char code 95).

\+ **Escaped character.** Matches a "+" character (char code 43).

. **Character.** Matches a "." character (char code 46).

~ **Character.** Matches a "~" character (char code 126).

# **Character.** Matches a "#" character (char code 35).

? **Character.** Matches a "?" character (char code 63).

& **Character.** Matches a "&" character (char code 38).

/ **Character.** Matches a "/" character (char code 47).

= **Character.** Matches a "=" character (char code 61).

]

{2,256} **Quantifier.** Match between 2 and 256 of the preceding token.

\. **Escaped character.** Matches a "." character (char code 46).

[ **Character set.** Match any character in the set.

a-z **Range.** Matches a character in the range "a" to "z" (char code 97 to 122).

]

{2,4} **Quantifier.** Match between 2 and 4 of the preceding token.

\b **Word boundary.** Matches a word boundary position such as whitespace, punctuation, or the start/end of the string.

( **Capturing group #1.** Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.

\/ **Escaped character.** Matches a "/" character (char code 47).

[ **Character set.** Match any character in the set.

- **Character.** Matches a "-" character (char code 45).

a-z **Range.** Matches a character in the range "a" to "z" (char code 97 to 122).

A-Z **Range.** Matches a character in the range "A" to "Z" (char code 65 to 90).

0-9 **Range.** Matches a character in the range "0" to "9" (char code 48 to 57).

@ **Character.** Matches a "@" character (char code 64).

: **Character.** Matches a ":" character (char code 58).

% **Character.** Matches a "%" character (char code 37).

\_ **Character.** Matches a "\_" character (char code 95).

\+ **Escaped character.** Matches a "+" character (char code 43).

. **Character.** Matches a "." character (char code 46).

~ **Character.** Matches a "~" character (char code 126).

# **Character.** Matches a "#" character (char code 35).

? **Character.** Matches a "?" character (char code 63).

& **Character.** Matches a "&" character (char code 38).

/ **Character.** Matches a "/" character (char code 47).

= **Character.** Matches a "=" character (char code 61).

]

\* **Star.** Match 0 or more of the preceding token.

)

? **Optional.** Match between 0 and 1 of the preceding token.

# 16. Class Matcher trong java

## Matcher.find() – boolean

Giả sử cái String đưa vào nó match được, còn match là còn true:

**package** com.example;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** Example {

**public** **static** **void** main(String[] args) {

Pattern pattern = Pattern.*compile*("[a-z]");

Matcher matcher = pattern.matcher("\_9Nguye\_-o");

**boolean** found = **false**;

**while** (matcher.find()) {

System.***out***.println("group:" +matcher.group());

System.***out***.println("start:" +matcher.start());

System.***out***.println("end:" +matcher.end());

found = **true**;

}

**if** (!found) {

System.***out***.println("No match found");

}

}

}

Result: Với pattern và String trên sẽ match lần lượt với g,u,y,eo

group:g

start:3

end:4

group:u

start:4

end:5

group:y

start:5

end:6

group:e

start:6

end:7

group:o

start:9

end:10