Regular Expressions

Introduction

→ If we want to represent a group of strings according to particular pattern then we should go for RegEx.

→ we can write RegEx to represent all valid Mobile number.

→ we can write RegEx to represent all valid Mail Ids.

→ The main important application areas of RegEx are:

1. To develop validation frameworks.
2. To develop Pattern matching application (CTRL+F) in window. & [GREP] in UNIX.
3. To develop translators like Assemblers, compilers, Interpreters etc.
4. To develop digital circuits
5. To develop communication protocols like TCP/IP, UDP. etc.

→ Ex.

```
import java.util.regex.*;
class RegExDemo1
{
    p s v main(str)
    {
        int count=0;
        Pattern p = Pattern.
                    compile("ab");

        Matcher m = p.matcher
                    ("abbabbba");

        while(m.find())
        {
            ++count;
            S.O.p("m.start()+"---"+
                  m.end()+"---"+
                  m.group());
        }

        S.O.p("Total Occurences:"+
              count);
    }
}
```

O/P :-

```
0 --- 2 --- ab
3 --- 5 --- ab
Total occurences : 2
```

Pattern

→ Pattern object is a compile version of Regular Expression i.e it is a java equivalent object of pattern.

→ we can create a pattern object by using compile() method of pattern class.

```
public static Pattern compile
                (String re)
```

Eg

```
Pattern p = Pattern.compile("ab");
```

Matcher

→ we can use Matcher object to check the given pattern in the target string.

→ we can create a Matcher object by using matcher() method of Pattern class.

→ `public Matcher matcher (String target)`

Eg.

```
Matcher m = p.matcher("abbabb");
```

→ Important methods of Matcher class:

① `boolean find()`

- It attempts to find next match & returns true if available.

② `int start()`

- Returns start index of match

③ `int end()`

- Returns end+1 index of match

④ ~~int~~ `String group()`

- Returns the matched pattern.

Note: Pattern & Matcher classes present in "java.util.regex" package.

- Introduced in 1.4 v

Character classes

`[abc]`
either 'a' or 'b' or 'c'

`[^abc]`
Except 'a' and 'b' and 'c'

`[a-z]`
Any lower case alphabet symbol from a to z.

`[A-Z]`
Any Uppercase alpabet symbol from A to Z

`[a-zA-Z]`
Any alphabet symbol

`[0-9]`
Any digit from 0 to 9

`[0-9a-zA-Z]`
Any alphanumeric symbol

`[^0-9a-zA-Z]`
Except alphanumerit symbol (Special character)

Ex.

```
import java.util.regex.*;
class RegExDemo2
{
    p s v main (sm)
    {
        Pattern p = Pattern.
                    compile ("x");
        Matcher m = p.matcher
                    ("a3b#k@9z");

        while (m.find())
        {
            s.o.p (m.start() + "---" +
                    m.group());
        }
    }
}
```

o/p :- [Note :- (x is any string)]

x = [abc]	x = [^abc]	x = [a-z]
0 --- a	1 --- 3	0 --- a
2 --- b	3 --- #	2 --- b
	4 --- k	4 --- k
	5 --- @	7 --- z
	6 --- 9	
	7 --- z	

x=[0-9]	x=[a-zA-Z0-9]	x=[^a-zA-Z0-9]
1---3	0---a	3---#
6---9	1---3	5---@
	2---b	
	4---k	
	6---9	
	7---Z	

Pre-defined Character classes

\s
Space character

\S
Except Space character

\d
Any digit from 0 to 9 [0-9]

\D
Except digit, any character

\w
Any word character [0-9a-zA-Z]

\W
Except word character (Special character)

.
Any character

Ex3.

```
import java.util.regex.*;
class RegExDemo3
{
    p s v main (Sr)
    {
        Pattern p = Pattern.
                    compile("x");

        Matcher m = p.matcher
                    ("a7b k@9z");

        while(m.find())
        {
            s.o.p(m.start() + "---" +
                  m.group());
        }
    }
}
```

Note: we can't use \s directly bcz if we do so compiler considers it as escape character, so we have to write double-back-slash to be considered as normal predefined character class

O/P:

x=\\s	x=\\S	
3---	@---a	5---@
	1---7	6---9
	2---b	7---Z
	4---k	

x=\\d	x=\\D	x=\\w	x=\\W
1---7	0---a	0---a	3---
6---9	2---b	1---7	5---@
	3---.	2---b	
	4---K	4---k	
	5---@	6---9	
	7---z	z---z	

x=.

0---a 6---9
1---7 7---z
2---b
3---.
4---K
5---@

Quantifiers

⇒ can be used to specify no. of occurrences to match.

a
Exactly one 'a'

a+
Atleast one 'a'

a*
Any no. of 'a's including

a?
Atmost one 'a'

Ex.

```
import java.util.regex.*;
class RegExDemo4
{
    p s v main(...)
    {
        Pattern p = Pattern.
                    compile("a");

        Matcher m = p.matcher
                    ("abaaabaaaab");

        while (m.find())
        {
            s.o.p(m.start() + "---" +
                  m.group());
        }
    }
}
```

O/P :-

x=a	x=a+	x=a*	x=a?
0---a	0---a	0---a	0---a
2---a	2---aa	1---	1---
3---a	5---aaa	2---aa	2---a
5---a		4---	3---a
6---a		5---aaa	4---
7---a		8---	5---a
		9---	6---a
			7---a
			8---
			9---

Note : In RegEx searching can go upto end+1 index so 9th index is considered in above example

Pattern class split() method

⇝ we can use pattern class split() method to split the target string according to a particular pattern.

Ex.

```
import java.util.regex.*;
class RegExDemo5
{
    p s v main (Sr)
    {
        Pattern p = Pattern.
                    compile("\\s");

        String[] s = p.split("Durga slw
                              soln");

        for(String s1 : s)
        {
            s.o.p(s1);
        }
    }
}
```

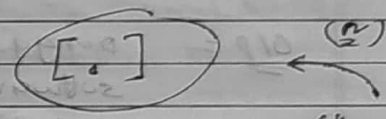
O/P : Durga
slw
soln

⇝ It WR take,

```
pattern p = Pattern.
            compile("o");
```

O/P :
Durga s
ftware s
puti
ns

⇝ Ex.




```
pattern p = Pattern.compile("\\.");

String[] s = p.split("www.durgajobs.
                      com");

for(String s1 : s)
{
    s.o.p(s1);
}
```

O/P :
www
durgajobs
com

Note : If we take (".") , It means split given string by everything so, we don't get any o/p , If we keep ("\.") compiler will give Illegal escape character, so we have to take ("\\.")

~ String class also contains split() method, to split the target string according to a particular pattern.

```
String s = "Durga software solutions";
String[] s1 = s.split(" ");

for(String s1 : s)
{
    s.o.p(s1);
}
```

O/P :- Durga
software
solutions

~ [Note :- Pattern class split() method can take target string as argument whereas - string class split() method can take pattern as argument]

✱ String Tokenizer

~ StringTokenizer is a specially designed class for tokenization activity.

~ StringTokenizer present in [java.util] package.

~ Ex.

```
StringTokenizer st = new
        StringTokenizer ("Durga software
                          solutions");

while (st.hasMoreTokens())
{
    s.o.p(st.nextToken());
}
```

O/P :- Durga
software
solutions

~ [Note :- The default regular- expression for StringTokenizer is space()]

~ Ex-

```
StringTokenizer st = new StringTokenizer
        ("19-09-2014", "-");

while (st.hasMoreTokens())
{
    s.o.p(st.nextToken());
}
```

Target string — RE/ Pattern/ Delimeter

O/P :- 19
09
2014

(*) Write a Regular Expression to represent all valid 10-digit mobile nos.

⇝ Rules :-

- Every no. should contain exactly 10 digit
- The first digit should be 6/7/8/9.

10-digit

⇝ [6789][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]

(or)

10-digit (or) 11-digit

⇝ [6-9][0-9]{9}

11-digit

⇝ 0?[6-9][0-9]{9}

10-digit or 11-digit or 12-digit

⇝ (0|91)?[6-9][0-9]{9}

↑ pipe

(*) Write a Regular Expression to represent all the valid Mail-IDs.

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

⇝ only gmail IDs :-

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

(*) Write a Regular Expression to represent all Yava Language Identifiers.

⇝ Rules :-

- Allowed characters are
  - a to z
  - A to z
  - 0 to 9
  - #
  - $
- length of each identifier should be atleast two.
- The first character should be lower case alphabet symbol from a to k.

- Second character should be a digit divisible by 3. (0,3,6,9).

→ [a-k][0369][a-zA-Z0-9#$]*

⊛ WAP to check whether the given number is a valid Mobile No. or not?

→
```
import java.util.regex.*;
class RegExDemo7
{
    p s v main(String[] args)
    {
        Pattern p = Pattern.compile
                    ("[0|91]?[6-9]
                     [0-9]{9}");

        Matcher m = p.matcher(args[0]);

        if(m.find() && m.group().
                    equals(args[0]))
        {
            sop("valid mobile number");
        }
        else
        {
            sop("Invalid mobile number");
        }
    }
}
```

O/P :- java RegExDemo7 9848022338
valid mobile number

java RegExDemo7 09848098480
valid mobile number

java RegExDemo7 919293949596
valid mobile number

java RegExDemo7 19848098480
Invalid mobile number

java RegExDemo7 9292939495967
Invalid mobile number

⊛ WAP to check whether the given Mail Id is valid Mail Id or not.

→ In the above program just we have to replace mobile no. RegEx with mail Id RegEx.

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

→ O/P :- java RegExDemo8 cmistry291@gmail.com
valid Mail Id

java RegExDemo8 c_.@gmail.com
valid Mail Id

java RegExDemo7 -cm291@gmail.com
Invalid mail Id

java RegExDemo8 c.@gmail99.com
Valid mail Id

java RegExDemo8 c@912.com
Valid Mail Id

java RegExDemo8 cmgm.com
Invalid Mail Id

java RegExDemo8 cm@gmcom
Invalid Mail Id.

* WAP to read all mobile numbers present in the given input file where mobile numbers are mixed with normal text data.

```
import java.io.*;
import java.util.regex.*;

class MobileNumberExtractor
{
    p s v main(S[] ) throws Exception
    {
        PrintWriter pw = new
                PrintWriter("output.txt");
        Pattern p = Pattern.compile
            ("(0|91)?[6-9][0-9]{9}");
        BufferedReader br = new
            BufferedReader(new FileReader
                ("input.txt"));

        String line = br.readLine()

        while (line != null)
        {
            Matcher m = p.matcher(line);
            while(m.find())
            {
                pw.println(m.group());
            }

            line = br.readLine();
        }

        pw.flush();
    }
}
```

input.txt

This is ravi with mobile nuber :.
8096969696 hgdgj skjhdhsksh
sadfdsjkfhg
abpqmost durgatoc@gmail.com
This is pavan with mobile no : 9246751212
absdfjklmno
Demnefgh. ravi.kumar@yahoo.co.in
7897897890 naga.vara prasad123@
veldotmail.com

Date ____
Page ____

```
lkm
9123456789
jk* 2hckj4x2
isadjf3kj4t
9292929292
```

javac MobileExtractor.java ↵
java MobileExtractor ↵

output.txt

```
8096969696
9246751212
7897897890
9123456789
9292929292
```

Note : same thing we can do ~~with~~ to find all Mail Ids by just replacing Mobile No. RegEx by Mail Id RegEx.

(*) WAP to display all .txt filenames present in C:\durga-classes

→ import java.io.*;
import java.util.regex.*;

Date ____
Page ____

```
class FileNameExtractor
{
    p s v main (S-- ) throws Exception
    {
        int count =0;
        Pattern p = Pattern.compile
                ("[a-zA-Z0-9_$.]+[.]txt");
        File f = new File("c:\\durga-
                          classes");
        String[] s = f.list();

        for (String temp : s)
        {
            Matcher m = p.matcher(temp);
            if (m.find() &&
                    m.group().equals(temp))
            {
                count ++;
                s.o.p(temp);
            }
        }
        s.o.p (count);
    }
}
```

O/P : All file names will be displayed with count.

abc.txt
pqr.txt
:

End of Regular Expressions