11/04/11

Regular Expressions

→ Any group of Strings according to a particular pattern is Called "regular Expression".

ex!: ① We Can write a regular Expression to represent all valid mail-ids & By using that regular Expression we Can validate wheather the given mail-id is valid or not.

② we Can write a regular Expression to represent all valid Java identifiers.

→ the main Application areas of regular Expressions are

1. we Can implement validation mechanism.
2. we Can develope pattern matching applications.
3. we Can develope translators like Compilers, interpreters e.t.c.
4. we Can use for designing digital Circuits
5. we Can use to develope Communication protocols like TCP by IP, UDP e.t.c

Ex:

```
import java.util.regex.*;
class RegExDemo
{
 p.s.v.m(String[] args)
 {
   pattern p = Pattern.Compile("ab");
   Matcher m = p.matcher("abbbabbcbdab");
while(m.find())
{
S.o.pln(m.start() +" ---" +m.end() + "---" +m.group());
}
}
}
```

o/p:-
```
0 ---2 ...ab
4--- 6 ---ab
10 ---12 ....ab
```

Pattern class:-

→ A pattern object represents Compiled Version of regular Expression. We Can Create a pattern object by using compile() of pattern class.

```
Pattern p = Pattern.compile(String regularExpression);
```

Matcher Class!-

→ A matcher object Can be used to match character Sequence against a regular Expression. We Can Create a Matcher Object by using Matcher() of pattern class

```
Matcher m = p.matcher(String target);
```

Important methods of matcher class:-

(1) boolean find();

→ It attempts to find next match & if it is available returns True otherwise returns false.

(ii) int start();-

→ returns start index of the match

(iii) int end();

→ returns end index of the match

(iv) String group();

→ returns the matched pattern

Character Classes :-

① [a-z] → Any lower case alphabet symbol

② [A-Z] → Any upper " "

③ [a-zA-Z] → Any alphabet symbol

④ [0-9] → Any digit from 0 to 9

⑤ [abc] → either a or b or c

⑥ [^abc] → Except a or b or c.

⑦ [0-9a-zA-Z] → Any alpha numeric character.

Ex:-

```
Pattern p = Pattern.compile("x");
Matcher m = p.matcher("a3b@cu z#");
while(m.find())
{
S.o.pln(m.start() + "----" + m.group());
}
```

x = [ab]

```
0 ----a
2 ----b
```

x = [a-z]

```
0 ---a
2 ---b
4 ---c
6 ---z
```

x = [0-9]

```
1 ----3
5 ----4
```

x = [0-9a-z]

```
0 ---a      5 ----4
1 ---3      6 ---z
2 ---b
4 ---c
```

Predefined-character class :-

Space character	$\longrightarrow$ \s
[0 -9]	$\longrightarrow$ \d
[0-9a-zA-Z]	$\longrightarrow$ \w
Anycharacter	$\longrightarrow$.

Ex:.

```
Pattern p = Pattern.compile ("x");
Matcher m = p.matcher ("a3z4@ k7#");
                        0 1 2 3 4 5 6 7 8
while (m.find())
{
 S.o.pln(m.start() + "----" + m.group());
}
```

x = \\d
1 ----- 3
3 ----- 4
7 ----- 7

x = \\w
0 --- a
1 --- 3
2 --- z
3 --- 4
6 --- k
7 --- 7

x = \\s
5 -----

x = .
0 — a
1 — 3
2 — z
3 — 4
4 — @
5 —
6 — k
7 — 7
8 — #

Quantifiers:-

→ We can use Quantifiers to specify no. of characters to match

Ex:-

1) a $\longrightarrow$ Exactly one a

2) a^+ $\longrightarrow$ atleast one a

3) a^* $\longrightarrow$ Any no. of a's

4) a? $\longrightarrow$ atmost one a.

ex:-

```
Pattern p = Pattern.compile("x");
Matcher m = p.matcher("abaabaaab");
while(m.find())
{
  S.o.pln(m.start()+"-----"+m.group());
}
```

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 -----

Split method (8) :-

Pattern class Contains Split() method to Split given String according to a regular Expression.

ex:-

```
Pattern p = pattern.Compile("\\s");
String[] s = p.Split("Durga Software Solutions").
for(String s1 : s)
{
  S.o.pln(s1); // Durga
               Software
               Solutions.
}
```

Ex(2):-

```
Pattern p = pattern.compile("\\.");        "[.]"
String[] s = p.split("www.durgaJobs.com");
for (String s1 : s)
{
    S.o.pln(s1);        O/P:- www
}                             durgajobs
                              Com
```

String class split() method :-

→ String class also Contains Spilt() to Spilt the given String against a regular Expression

Ex:-

```
String s = "www.durgaJobs.com";
String[] s1 = s.split("\\.");
for ( String s2 : s1)
{
    S.o.pln(s2);          www
}                         durgajobs
                          com
```

Note!-

Pattern class Split() Can take target String as arguement where as String class Split() Can take regular Expression as arguement.

StringTokenizer :-

→ We can use StringTokenizer to divide the target String into Stream of Tokens according to the

→ StringTokenizer class presenting in java.util package.

Ex:-

①
```
StringTokenizer st = new StringTokenizer("Durga Software Solutions");
while (st.hasMoreTokens())
{
  S.o.pln(st.nextToken());
}
```
o/p:-
```
Durga
Software
Solutions
```

Note:- The default regular Expression is Space

②
```
StringTokenizer st = new StringTokenizer("1,00,000", ",");
while(st.hasMoreTokens())
{
  S.o.pln(st.nextToken());
}
```
o/p:-
```
1
00
000
```

o/p:-
```
1
00
000
```

Ex(2):-
Write a regular Expression to represent the Set of all valid identifiers in Java language.

Rules:- (1) The length of each identifier is atleast 2

(2) The allowed characters are a to z
A to Z
0 to 9
.
_

(3) The first character should not digit

R.E:- $\underbrace{[a-zA-Z.\_][a-zA-Z0-9.\_]}_{x}\ \underbrace{[a-zA-Z0-9.\_]^*}_{x^*}$ (or) $x.x^* = x^+$

$[a-zA-Z.\_][a-zA-Z0-9.\_]^+$

```
import java.util.regex.*;
class RegExDemo2
{
  p.s.v.m(String[] args)
  {
   Pattern p = Pattern.compile("[a-zA-Z._][a-zA-Z0-9._]+");
   Matcher m = p.matcher(args[0]);
   if(m.find()&& m.group().equals(args[0]))
   {
     S.o.pln(" valid Identifier");
   }
   else
   {
     S.o.pln(" Invalid Identifier");
   }
  }
}
```

② w.a. RE to represent all valid mobile numbers

rule:- (1) mobile No Contains 10 digits

(2) The first digit should be 7 to 9

RegEx:- [7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]

(or)

[7-9]\d{9}

③ W.a. regular Expression to represent all valid mail-ids?

rules:

(1) The Set of allowed characters in mail-id are 0 to 9, a-z, A-Z, ., -

(2) Should starts with alphabet symbol

(3) should contain atleast one symbol.

RegExp:- $[a\text{-}zA\text{-}Z][a\text{-}zA\text{-}Z0\text{-}9.\_]^{*}@[a\text{-}zA\text{-}Z0\text{-}9]^{+}([.][a\text{-}zA\text{-}Z]^{+})^{+}$

RegExp:

" @gmail[.]com

" @(gmail|yahoo|hotmail)[.]com

Ex:-

```
import java.io.*;
import java.util.regex.*;
class MobileExtractor
{
  p.s.v.m(String[] args) throws IOException
  {
    PrintWriter pw = new PrintWriter("mobile.txt");
    BufferedReader br = new BufferedReader(new FileReader("input.txt"));
String line = br.readLine();
Pattern p = Pattern.compile("[7-9][0-9]{9}");
while (line != null)
{
 Matcher m = p.matcher(line);
 while(m.find())
 {
   pw.println(m.group());
 }
  line = br.readLine();
 }
 pw.flush();
 }
}
```

P) W.a.p to Extract mail-ids from the given file where mail-ids are mixed with some raw data?

→ In the above Example replace regular Expression with the following mail-id regular Expression.

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

P) W.a.p to display all text files present in the given directory?

```
import java.io.*;
import java.util.regex.*;
class FileNameExtractor
{
```

```
Public Static void main (String[] args) throws IOException
{
  int Count = 0;
  Pattern p = Pattern.compile("[a-zA-Z0-9_]+[.]txt");
  File f = new File ("D:\\durga_classes");
  String[] s = f.list();
  for (String s1 : s)
   {
     Matcher m = p.matcher(s1);
     if(m.find() && m.group().equals(s1))
      {
        Count ++;
        S.o.pln(s1);
      }
   }
   S.o.pln (Count);
  }
}
```

P) w.a.p to delete all .bak files present in D:\\durgaclass

```
import java.io.*;
import java.util.regex.*;
class FileNamesDeleter
{
  p.s.v.m(String[] args) throws IOException
  {
    int Count = 0;
    Pattern p = Pattern.compile("[a-zA-Z0-9_.]+[.]bak");
File f = new File("D:\\durga_classes");
String[] s = f.list();
for(String s1 : s)
{
  Matcher m = p.matcher(s1);
  if(m.find() && m.group().equals(s1))
  {
    Count++;
    S.o.pln(s1);
    File f1 = new File(f, s1);
    f1.delete();
  }
}
S.o.pln(Count);
}
}
```

=x=