10. Regular Expressions

Regular Expression:— Regular Expression represents a group of Strings according to a particular pattern.

Ex①: We can write a regular expression to represent all valid mobile numbers.

Ex②: We can write a regular expression to represent all valid mail id's.

→ The main important application areas of regular expression are

1. To develop validation frameworks.
2. To develop pattern matching applications (in windows, grep in UNIX).
3. To develop translators like compilers, interpreters, assemblers etc.
4. To develop digital circuits.
5. To develop communication protocol etc.

Ex: import java.util.regex.*; DEMO

```
class RegExDemo
{
  P S V m(-)
  {
    int count=0;

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

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

    while(m.find())
    {
      count++;

      S.o.p (m.start()+"..."+m.end()+"..."+m.group());
    }
    S.o.p("The no. of occurences :"+count);
  } }
}
```

olp: 0 ... 2 ... ab

3 ... 5 ... ab

5 ... 7 ... ab

The no. of occurences : 3

Pattern class :-

→ A Pattern object is compiled representation of regular expression i.e., Pattern object is Java equivalent form of regular expression.

→ We can create a Pattern object by using compile() method of Pattern class.

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

Ex: Pattern p=Pattern.compile("ab");

Matcher class :-

→ A Matcher object can be used to match the given pattern in the target string.

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


```
public Matcher matcher(String target)
```

Ex: Matcher m=p.matcher("abbaababa");

Methods of Matcher class :-

1) boolean find() : It attempts to find next match & returns true if the match is available otherwise returns false.

2) int start() : returns start index of matched pattern.

3) int end() : returns end+1 index of matched pattern.

4) String group() : returns matched pattern.

Note:- Pattern & Matcher classes are present in java.util.regex package & these classes introduced in 1.4 version.

Character classes:—

[abc] ⟶ either 'a' or 'b' or 'c'

[^abc] ⟶ except 'a' or 'b' or 'c'

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

[A-Z] ⟶ Any upper case alphabet symbol from A to Z.

[a-zA-Z] ⟶ Any alphabet symbol.

[0-9] ⟶ Any digit from 0-9.

[a-zA-Z0-9] ⟶ Any alphanumeric character.

[^a-zA-Z0-9] ⟶ Any special character.

Ex:

```
Pattern p = Pattern.compile("x");
Matcher m = p.matcher("a7b@z9#k");
while(m.find())
{
  S.o.p(m.start() + "..." + m.group());
}
```

DEMO

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

x=[^a-zA-Z0-9]
3...@
6...#

DEMO

Pre-defined character classes:—

\s → Space character.

\S → Any character except space.

\d → Any digit from 0 to 9

\D → Any character except digit.

\w → Any word character [a-z. A—Z 0-9].

\W → Special characters.

. → Any character including special characters also.

}

Ex: Pattern p = Pattern.compile("x");

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

while(m.find())

{

S.o.p(m.start() + "..." + m.end());

}

DEMO

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

Quantifiers:—

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

a → exactly one a

a+ → Atleast one a

a* → Any no. of a's including zero number also.

a? → Atmost one a.

Ex: Pattern p = Pattern.compile("a");

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

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

a=a	a=a+	a=a*	a=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...

Pattern class splits() method:—

DEMO

→ Pattern class contains splits() method to split the given String according to given Pattern (regular expression).

Ex①: Pattern p = Pattern.compile("\\s");

String[] s = p.split("Durga Software Solutions");

for(String s1 : s)
{
  S.O.p(s1); ⟶ O/P : Durga
} Software
  Solutions

(or) → "[.]"

Ex②: Pattern p = Pattern.compile("\\.");

String s1[] = p.split("www.durgasoft.com");

for(String s1 : s)
{
  S.O.p(s1); ⟶ O/P : www
} durgasoft
  com

String class split() method :—

→ String class also contains split() method to split the given target String according to a particular pattern.

Ex: String s = "www.durgasoft.com";
String[] s1 = s.split("[.]");
for (String s2 : s1)
{
S.o.p (s2); → o/p : www
durgasoft
com
}

***

→ String class split() method can take regular expression as argument, where as Pattern class split() method can take target String as argument.

StringTokenizer :—

→ It is a specially designed class for tokenization activity

→ It present in java.util package.

DEMO

Ex①: StringTokenizer st=new ST("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 character.

Ex②: StringTokenizer st=new ST("29-03-2013", "-");
while (st.hasMoreTokens()):
{
S.o.p(st.nextToken()); → o/p : 29
03
2013
}

(target string, pattern | delimeter)

Ex①: Write a regular expression to represent all valid 10 digit mobile numbers.

Rules:-

1) It should contain exactly 10 digits.
2) Should starts with 7 or 8 or 9.

Ans: [7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]

(or)

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

10-digit (or) 11-digit :-

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

10-digit (or) 11-digit (or) 12-digit :-

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

10-digit (or) 11-digit (or) 12-digit (or) 13 characters :-

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

// Write a regular expression to represent all valid mail id's.

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

// Write a regular expression to represent all valid identifiers of KAVA language.

Rules:-

1) The allowed characters are a to z, A to Z, 0 to 9, #, _.
2) The length of identifier should be atleast 2.
3) The first character should be lower case alphabet symbol from a to k.
4) Second character should be a digit divisible by 3.

$$[a\text{-}k][0369][a\text{-}zA\text{-}Z0\text{-}9\_\#]^{*}$$

// Write a regular expression to represent all valid names starts with 'a' and ends with 'n'. (either lowercase or uppercase).

`[aA][a-zA-Z]*[nN]`

// Write a program to check whether the given no. is a valid mobile no. or not.

```
import java.util.regex.*;
class Test
{
  p s v m(-)
  {
    Pattern p = Pattern.compile("(0|91)?[7-9][0-9]{9}");
    Matcher m = p.matcher(args[0]);
    if(m.find() && m.group().equals(args[0]))
    {
      S.o.p("Valid mobile number");
    }
    else
    {
      S.o.p("Invalid mobile number");
    }
  }
}
```

DEMO

Ex: java Test 9666666669 ✓

java Test 919292929292 ✓

java Test 9292929292292 ✗

// Write a program to check whether given mail id is valid or not.

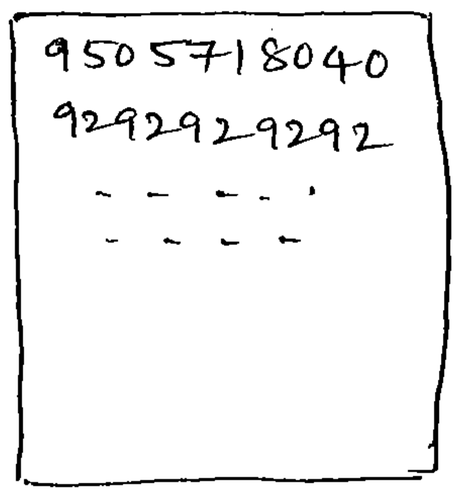
In the above program we have to replace mobile no. regular expression with mail id regular expression.

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

// Write a program to extract all mobile no's present in the given input file where mobile no's mixed with normal text data.

input.txt

```
This is Durga with mobile number:
9505718040 and mail id:
durga@gmail.com
- - - - - -
- - - - - - -
This is Shiva with mobile number:
9292929292 and mail id:
shiva@yahoo.com
- - - - - - -
- - - - - - - -
```



output.txt

```
9505718040
9292929292
- - - - -
- - - - -
```

DEMO

```
import java.io.*;
import java.util.regex.*;
class Test {
  P S V m(-) throws Exception
  {
     Pattern p= Pattern.compile ("(0|91)?[7-9][0-9]{9}");
     PrintWriter pw = new PW("output.txt");
     BufferedReader br = new BR(new FR("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();
} } pw.close();
```

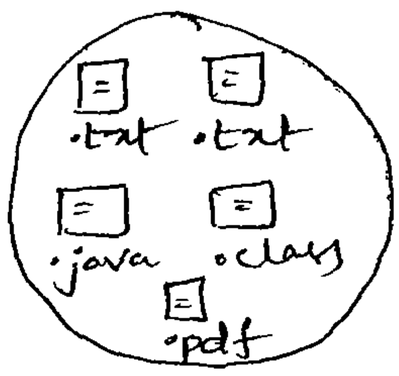
// Write a program to extract all mail id's present in the given input file.

→ In the above program, we have to replace mobile no. regular expression with mail id regular expression.

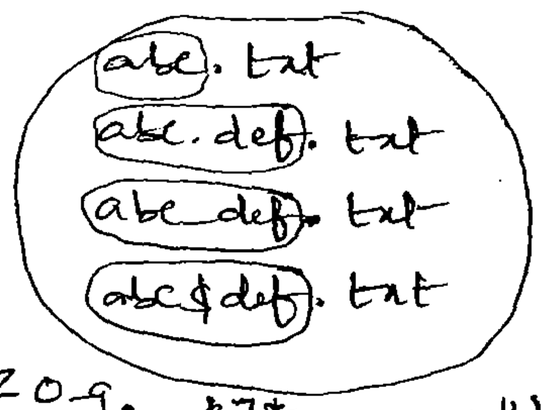
```
[a-zA-Z0-9][a-zA-Z0-9._]*@[a-zA-Z0-9]+([.][a-zA-Z]+)+
```

// Write a program to print names of all .txt file names present in c:\durga_classes.

```
[a-zA-Z0-9][a-zA-Z0-9._$]*[.]txt
```



c:\durga_classes



DEMO

```
import java.io.*;
import java.util.regex.*;
class Test
{
  p s v m() throws Exception
  {
    Pattern p=Pattern.compile("[a-zA-Z0-9][a-zA-Z0-9._$]*[.]txt");
    File f=new File("c:\\durga_classes");
    String[] s=f.list();
    for(String s1:s)
    {
      Matcher m=p.matcher(s1);
      if(m.find() && m.group().equals(s1))
      {
        S.o.p(s1);
      }
    }
  }
}
```

DEMO