Regular Expressions In Java

- While developing the real time application like web-applications in the User Interface we need to perform some sort of validation based on some pattern then we have to use regular expression
  - Example the mobile number given by the user is according to the pattern or not
  - Validate mail id are according to the Pattern or Not
  - Date values based or particular mask or not dd/mmm/yy
- Reading the data from data based some specific pattern
- Finding and replacements options in ms-word …

In order to achieve regular expression then we have to specify two things
1.What to search called Pattern
2.Where to search called Matcher

To specify the Pattern and Matcher then we have to use the predefined Classes from java.util.regex package

- To create an Object For Patter class then we have to use the following method
  - java.util.regex.Pattern
    - public static Pattern compile(String pattern);

- To Create an Object For Matcher then we have to use the following method
  - Java.util.regex.Pattern
    - public Matcher matcher(String target);

Methods Of java.util.regex.Matcher Class

- public boolean find() : it will search for next Match and it will return false when no matches are existed or found
- public int start(): it will return starting index position of the each match
- public int end(): it will return ending index position+1 of the match
- public String group(): it will extract matched Pattern From target

Example:

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;
class RegEx1{
	public static void main(String args[ ]){
		Pattern p=Pattern.compile("s");
		Matcher m=p.matcher("shashis");
		while( m.find() ){
	   System.out.println("Match :"+m.group()  );
			System.out.println("Found : "+m.start() );
		}
	}
}
```

Working With Predefined Pattern

\w ----------> Alpha numerical values
\W ----------> Except A|N
\d ----------> Only digits
\D ----------> Except Digits
\s ----------> Only Spaces
\S ----------> Except Spaces
. ----------> any thing

Example:

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

class RegEx1
{
	public static void main(String args[ ])
	{
		Pattern p=Pattern.compile("\\w");
		Matcher m=p.matcher(" AB c 12 !# Hello 8 @");
		while( m.find() )	{
	   System.out.println("Match :"+m.group()  );
			System.out.println("Found : "+m.start() );
		}
	}
}
```

User defined Pattern : Whenever Predefined pattern fail to achieve the requirements then we have to defined our own patterns called User defined Pattern

```
[mno]   -----> either m or n or o
[abc]   ------> either a or b or c
[a-d]   -------> lower case a to d

[a-z]   --------> all lower case letters
[A-Z]   ----------> all upper case letters

[a-zA-Z]  -----> any alphabet
[a-zA-Z0-9] or \w ----> any a|n
[^a-zA-Z0-9] or \W ---> Except a|n

[^a-zA-Z] ---> Except Alphabets
[!&#] ----> Either ! or & or # symbols  */

import java.util.regex.Pattern;
import java.util.regex.Matcher;

class UsedefiedPattern
{
      public static void main(String args[ ])
      {
       Pattern p=Pattern.compile("[A-Z]"); //what to search
       Matcher m=p.matcher("aB !2#4 N.CE ^34 a.zn m.o ");
                                  //where to search
       while( m.find() )
            {
             System.out.println
                  ("Match : "+m.group()+" Found @ : "+m.start());
            }
      }
  }
```

Quantifiers :

- By using quantifiers we can specify the no.of. Occurrences of the match
  - "a" → Only 'a's
  - "a+" → at least one 'a' followed by N. no.of. 'a's
  - "a?" → at most one 'a' i.e 0 a's or 1'a
  - "a*" → 0 occurrences or N no.of.occurrences
  - "a{3}" → "a" for 3 types
  - "a{2,3}" → minimum '2' as and maximum 3 a's

Example:

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

class Quantifiers
{
	public static void main(String args[ ])
	{
	 Pattern p=Pattern.compile("a*"); //what to search
	 Matcher m=p.matcher("ab aac aaad aaaae");
		//where to search

	 while( m.find() )
		{
		  System.out.println
		   ("Match : "+m.group()+" Found @ : "+m.start());
	  }
	}
}
```

Example Pattern For Mail | Mobile Number | Names …

Application 1:

Pattern for Extract all the names which are starts sS

Eg: "[sS][a-zA-Z]+"

Application 2:

Pattern For Validate Mobile number

1.It must contains 10digits

2.It must be starts with 6 or 7 or 8 or 9

```
[6789][0-9][0-9][0-9]
      [0-9][0-9][0-9]
            [0-9][0-9][0-9]   //valid

" [6-9][0-9]+ "  --> Wrong Pattern
      84  --> valid
         877777777777777777777777777777 --> valid

" [6-9][0-9]* " --> Wrong Pattern
      8 --> valid
         89898 --> valid
         9897989797899999 --> valid

"[6-9][0-9]{9}"  ---> valid Pattern
```

3.Pattern For Extract all the names which are starts with any alpha total length should be 6 letters and it must ends with i or l

"[a-zA-Z]{5}[il]"

4.Patter For Extract Mobile numbers

1.total no.of.digits should be 10 or 11
2.IF it is 10 digits should be starts with 6|7|8|9
3.IF it is 11 digits then it should be starts with 0

"[6-9][0-9]{9}" ---> 10 digits
"0[6-9][0-9]{9}" --> 11 digits

"0?[6-9][0-9]{9}" --> 10 | 11 digits

5.Patter For Extract Mobile numbers

1.total no.of.digits should be 10 or 11 or 12
2.IF it is 10 digits should be starts with 6|7|8|9
3.IF it is 11 digits then it should be starts with 0
4.IF it is 12 digits then it should be starts with 91

"(91|0)?[6-9][0-9]{9}" --> 10 digits | 11 digits

Application

- Pattern For Validate Only Gmail.com

1 |----------| |----3-------|
s hashikumar.sssit @ gmail.com
[a-zA-Z0-9][a-zA-Z0-9_.]+@gmail[.]com

- Pattern For any Mail ID's

1 |----------| |---3-------|

s hashikumar.sssit @ gmail.com
yahoo.com
tv9.com
v6.net

[a-zA-Z0-9][a-zA-Z0-9_.]+@[a-z0-9]+[.][a-z]+

- Pattern For any Mail ID's

1 |----------| |---3-------|

s hashikumar.sssit @ gmail.com
yahoo.com
tv9.com
v6.net
ts.gov.in
uk.edu

[a-zA-Z0-9][a-zA-Z0-9_.]+@[a-z0-9]+[[.][a-z]+]+

Example :

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

class DateExtract
{
      public static void main(String args[ ])
      {
String students="Ramesh 12-Jan-20 sudha 12-Feb-1999  ravali 23-Dec-89 ";
Pattern p=Pattern.compile("\\d{1,2}-[a-zA-Z]{3}-\\d{2,4}");
Matcher m=p.matcher(students);

while( m.find() )
            {
               String dob=m.group();
                        System.out.println(dob);
     }
   }
}
```

Example :

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.util.Scanner;

class Validate
{
	public static void main(String args[ ])
	{
		Scanner s=new Scanner(System.in);
		System.out.println("Enter Mobile number ");
		String mobile=s.next();   // 9848022338

		Pattern p=Pattern.compile("[6-9][0-9]{9}");
		Matcher m=p.matcher(mobile);

	       if (m.find() &&  m.group().equals(mobile) )
		{
			System.out.println("Valid Mobile number ");
		}
		else
		{
			System.out.println("Sorry Invalid Mobile number ") ;
		}
	}
}
```