CORE JAVA

With

SCJP / OCJP

Study Material

Chapter 13: Regular Expression

DURGA M.Tech

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students
for last 14 years across INDIA

Regular Expression

Agenda

1. Introduction.
2. The main important application areas of Regular Expression
3. Pattern class
4. Matcher class
5. Important methods of Matcher class
6. Character classes
7. Predefined character classes
8. Quantifiers
9. Pattern class split() method
10. String class split() method
11. StringTokenizer
12. Requirements:
    - Write a regular expression to represent all valid identifiers in java language
    - Write a regular expression to represent all mobile numbers
    - Write a regular expression to represent all Mail Ids
    - Write a program to extract all valid mobile numbers from a file
    - Write a program to extract all Mail IDS from the File
    - Write a program to display all .txt file names present in specific(E:\scjp) folder

Introduction

A Regular Expression is a expression which represents a group of Strings according to a particular pattern.

Example:

- We can write a Regular Expression to represent all valid mail ids.
- We can write a Regular Expression to represent all valid mobile numbers.

The main important application areas of Regular Expression are:

- To implement validation logic.
- To develop Pattern matching applications.
- To develop translators like compilers, interpreters etc.
- To develop digital circuits.
- To develop communication protocols like TCP/IP, UDP etc.

```
Example:
import java.util.regex.*;
class RegularExpressionDemo
{
      public static void main(String[] args)
      {
            int count=0;
            Pattern p=Pattern.compile("ab");
            Matcher m=p.matcher("abbbabbaba");
            while(m.find())
            {
            count++;
            System.out.println(m.start()+"------"+m.end()+"--
----"+m.group());
            }
            System.out.println("The no of occurences
:"+count);
      }
}
Output:
0------2------ab
4------6------ab
7------9------ab
The no of occurrences: 3
```

Pattern class:

- A Pattern object represents "compiled version of Regular Expression".
- We can create a Pattern object by using compile() method of Pattern class.

public static Pattern compile(String regex);

Example:

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

Note: if we refer API we will get more information about pattern class.

Matcher:

A Matcher object can be used to match character sequences against a Regular Expression.
We can create a Matcher object by using matcher() method of Pattern class.

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

Important methods of Matcher class:

1. **boolean find();**
   It attempts to find next match and returns true if it is available otherwise returns false.

2. int start();
   Returns the start index of the match.
3. int end();
   Returns the offset(equalize) after the last character matched.(or)
   Returns the "end+1" index of the matched.
4. String group();
   Returns the matched Pattern.

Note: Pattern and Matcher classes are available in java.util.regex package, and introduced in 1.4 version

Character classes:

1. [abc]----------Either 'a' or 'b' or 'c'
2. [^abc] ----------Except 'a' and 'b' and 'c'
3. [a-z] ----------Any lower case alphabet symbol
4. [A-Z] ----------Any upper case alphabet symbol
5. [a-zA-Z] ----------Any alphabet symbol
6. [0-9] ----------Any digit from 0 to 9
7. [a-zA-Z0-9] ----------Any alphanumeric character
8. [^a-zA-Z0-9] ----------Any special character

```
Example:
import java.util.regex.*;
class RegularExpressionDemo
{
	public static void main(String[] args)
	{
		Pattern p=Pattern.compile("x");
		Matcher m=p.matcher("a1b7@z#");
		while(m.find())
		{
			System.out.println(m.start()+"-------
"+m.group());
		}
	}
}
```

Output:

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

Predefined character classes:

\s----------space character
\d----------Any digit from o to 9[o-9]
\w----------Any word character[a-zA-Z0-9]
. ----------Any character including special characters.

\S----------any character except space character
\D----------any character except digit
\W----------any character except word character(special character)

```
Example:
import java.util.regex.*;
class RegularExpressionDemo
{
	public static void main(String[] args)
	{
		Pattern p=Pattern.compile("x");
		Matcher m=p.matcher("a1b7 @z#");
		while(m.find())
		{
			System.out.println(m.start()+"-------
"+m.group());
		}
	}
}
```

Output:

x=\\s	x=\\d	x=\\w	x=.
4-------	1-------1	0-------a	0-------a
	3-------7	1-------1	1-------1
		2-------b	2-------b
		3-------7	3-------7
		6-------z	4-------
			5-------@
			6-------z
			7-------#

Quantifiers:

Quantifiers can be used to specify no of characters to match.

a----------Exactly one 'a'

a+----------At least one 'a'

a*----------Any no of a's including zero number

a? ----------At most one 'a'

```
Example:
import java.util.regex.*;
class RegularExpressionDemo
{
	public static void main(String[] args)
	{
		Pattern p=Pattern.compile("x");
		Matcher m=p.matcher("abaabaaab");
		while(m.find())
		{
			System.out.println(m.start()+"-------
"+m.group());
		}
	}
}
```

Output:

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-------

Pattern class split() method:

Pattern class contains split() method to split the given string against a regular expression.

```
Example 1:
import java.util.regex.*;
class RegularExpressionDemo
{
	public static void main(String[] args)
	{
		Pattern p=Pattern.compile("\\s");
		String[] s=p.split("ashok software solutions");
		for(String s1:s)
		{
			System.out.println(s1);//ashok
					       //software
					        //solutions
		}
	}
}
Example 2:
import java.util.regex.*;
class RegularExpressionDemo
{
```

```
	public static void main(String[] args)
	{
		Pattern p=Pattern.compile("\\.");  //(or)[.]
		String[] s=p.split("www.dugrajobs.com");
		for(String s1:s)
		{
			System.out.println(s1);//www
                                       //dugrajobs
                                        //com
		}
	}
}
```

String class split() method:

String class also contains split() method to split the given string against a regular expression.

```
Example:
import java.util.regex.*;
class RegularExpressionDemo
{
	public static void main(String[] args)
	{
		String s="www.saijobs.com";
		String[] s1=s.split("\\.");
		for(String s2:s1)
		{
			System.out.println(s2);//www
                                        //saijobs
                                        //com
		}
	}
}
```

Note : String class split() method can take regular expression as argument where as pattern class split() method can take target string as the argument.

StringTokenizer:

- **This class present in java.util package.**
- **It is a specially designed class to perform string tokenization.**

```
Example 1:
import java.util.*;
class RegularExpressionDemo
{
	public static void main(String[] args)
	{
		StringTokenizer st=new StringTokenizer("sai
software solutions");
```

```
            while(st.hasMoreTokens())
            {
                  System.out.println(st.nextToken());//sai
                                                     //software
                                                     //solutions
            }
      }
}
```

The default regular expression for the StringTokenizer is space.

```
Example 2:
import java.util.*;
class RegularExpressionDemo
{
      public static void main(String[] args)
      {



         StringTokenizer st=new
StringTokenizer("1,99,988",",");
            while(st.hasMoreTokens())
            {
                  System.out.println(st.nextToken());//1
                                                     //99
                                                     //988
            }
      }
}
```

Requirement:

Write a regular expression to represent all valid identifiers in java language.

Rules:

The allowed characters are:

1. **a to z, A to Z, 0 to 9, -,#**
2. **The 1st character should be alphabet symbol only.**
3. **The length of the identifier should be at least 2.**

```
Program:
import java.util.regex.*;
class RegularExpressionDemo
{
      public static void main(String[] args)
      {
            Pattern p=Pattern.compile("[a-zA-Z][a-zA-Z0-9-
#]+"); (or)
```

```
            Pattern p=Pattern.compile("[a-zA-Z][a-zA-Z0-9-
#][a-zA-Z0-9-#]*");
            Matcher m=p.matcher(args[0]);
            if(m.find()&&m.group().equals(args[0]))
            {
                  System.out.println("valid identifier");
            }
            else
            {
                  System.out.println("invalid identifier");
            }

      }
}
Output:
E:\scjp>javac RegularExpressionDemo.java
E:\scjp>java RegularExpressionDemo ashok
Valid identifier

E:\scjp>java RegularExpressionDemo ?ashok
Invalid identifier
```

Requirement:

Write a regular expression to represent all mobile numbers.

1. **Should contain exactly 10 digits.**
2. **The 1st digit should be 7 to 9.**

```
Program:
import java.util.regex.*;
class RegularExpressionDemo
{
      public static void main(String[] args)
      {
            Pattern p=Pattern.compile("
                              [7-9][0-9][0-9][0-9][0-9][0-
9][0-9][0-9][0-9][0-9]");
            //Pattern p=Pattern.compile("[7-9][0-9]{9}");
            Matcher m=p.matcher(args[0]);
            if(m.find()&&m.group().equals(args[0]))
            {
                  System.out.println("valid number");
            }
            else
            {
                  System.out.println("invalid number");
            }
}
}
```

```
Analysis:
10 digits mobile:
[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}

Output:
E:\scjp>javac RegularExpressionDemo.java
E:\scjp>java RegularExpressionDemo 9989123456
Valid number

E:\scjp>java RegularExpressionDemo 6989654321
Invalid number
10 digits (or) 11 digits:
(0?[7-9][0-9]{9})

Output:
E:\scjp>javac RegularExpressionDemo.java

E:\scjp>java RegularExpressionDemo 9989123456
Valid number

E:\scjp>java RegularExpressionDemo 09989123456
Valid number

E:\scjp>java RegularExpressionDemo 919989123456
Invalid number
10 digits (0r) 11 digit (or) 12 digits:
(0|91)?[7-9][0-9]{9}   (or)
(91)?(0?[7-9][0-9]{9})

E:\scjp>javac RegularExpressionDemo.java
E:\scjp>java RegularExpressionDemo 9989123456
Valid number
E:\scjp>java RegularExpressionDemo 09989123456
Valid number
E:\scjp>java RegularExpressionDemo 919989123456
Valid number
E:\scjp>java RegularExpressionDemo 69989123456
Invalid number
```

Requirement:

Write a regular expression to represent all Mail Ids.

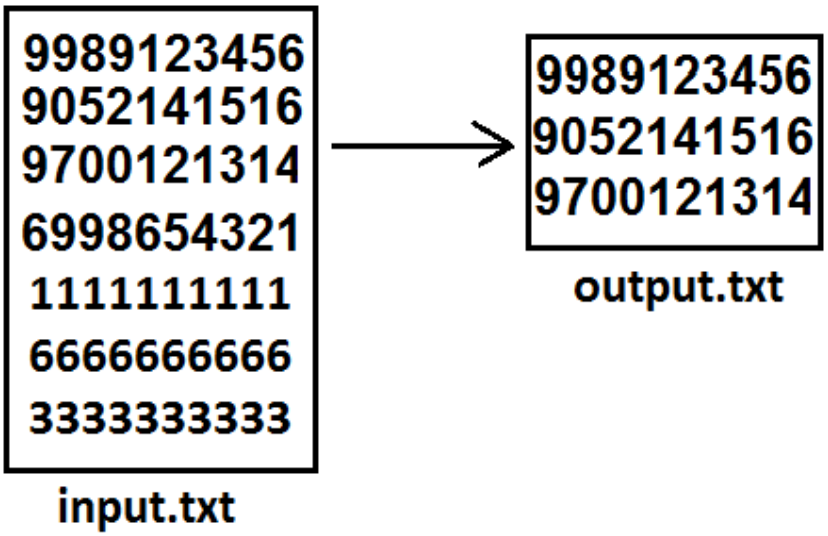
```
Program:
import java.util.regex.*;
class RegularExpressionDemo
{
      public static void main(String[] args)
      {
            Pattern p=Pattern.compile("
```

```
                                  [a-zA-Z][a-zA-Z0-9-.]*@[a-zA-Z0-
9]+([.][a-zA-Z]+)+");
            Matcher m=p.matcher(args[0]);
            if(m.find()&&m.group().equals(args[0]))
            {
                  System.out.println("valid mail id");
            }
            else
            {
                  System.out.println("invalid mail id");
            }
      }
}
Output:
E:\scjp>javac RegularExpressionDemo.java
E:\scjp>java RegularExpressionDemo sunmicrosystem@gmail.com
Valid mail id
E:\scjp>java RegularExpressionDemo 999sunmicrosystem@gmail.com
Invalid mail id
E:\scjp>java RegularExpressionDemo 999sunmicrosystem@gmail.co9
Invalid mail id
```

Requirement:

Write a program to extract all valid mobile numbers from a file.

Diagram:




```
Program:
import java.util.regex.*;
import java.io.*;
class RegularExpressionDemo
{
      public static void main(String[] args)throws IOException
```

```
	{
		PrintWriter out=new PrintWriter("output.txt");
		BufferedReader br=new BufferedReader(new
FileReader("input.txt"));
		Pattern p=Pattern.compile("(0|91)?[7-9][0-
9]{9}");
		String line=br.readLine();
		while(line!=null)
		{
			Matcher m=p.matcher(line);
			while(m.find())
			{
				out.println(m.group());
			}
			line=br.readLine();
		}
		out.flush();
	}
}
```

Requirement:

Write a program to extract all Mail IDS from the File.

Note: In the above program replace mobile number regular expression with MAIL ID regular expression.

Requirement:

Write a program to display all .txt file names present in E:\scjp folder.

```
Program:
import java.util.regex.*;
import java.io.*;
class RegularExpressionDemo
{
	public static void main(String[] args)throws IOException
	{
		int count=0;
		Pattern p=Pattern.compile("[a-zA-Z0-9-
$.]+[.]txt");
		File f=new File("E:\\scjp");
		String[] s=f.list();
		for(String s1:s)
		{
			Matcher m=p.matcher(s1);
			if(m.find()&&m.group().equals(s1))
			{
				count++;
				System.out.println(s1);
			}
		}
```

```
                System.out.println(count);
        }
}
Output:
input.txt
output.txt
outut.txt
3
```

Write a program to check whether the given mailid is valid or not.

In the above program we have to replace mobile number regular expression with mailid regular expression

Write a regular expressions to represent valid Gmail mail id's :

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

Write a regular expressions to represent all Java language identifiers :

Rules :

- **The length of the identifier should be atleast two.**
- **The allowed characters are**
- **a-z**
- **A-Z**
- **0-9**
- **#**
- **$**
-
-
- **The first character should be lower case alphabet symbol k-z , and second character should be a digit divisible by 3**

[k-z][0369][a-zA-Z0-9#$]*

Write a regular expressions to represent all names starts with 'a'

[aA][a-zA-Z]*

To represent all names starts with 'A' ends with 'K'

[aA][a-zA-Z]*[kK]