**Regular expression**

* 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**](file:///D:\API\java%20SE\docs\api\java\util\regex\Pattern.html) **compile(**[**String**](file:///D:\API\java%20SE\docs\api\java\lang\String.html)**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**](file:///D:\API\java%20SE\docs\api\java\util\regex\Matcher.html) **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.

1. **int start();**

* Returns the start index of the match.

1. **int end();**

* Returns the offset(equalize) after the last character matched.(or)
* Returns the end index of the matched.

1. **String group();**

* Returns the matched Pattern.

**Note:** Pattern and Matcher classes are available in **java.util.regex** package.

**Character classes:**

[abc]-------------------Either ‘a’ or ‘b’ or ‘c’

[^abc] -----------------Except ‘a’ and ‘b’ and ‘c’

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

[A-Z] --------------------Any upper case alphabet symbol

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

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

[a-zA-Z0-9] ------------Any alphanumeric 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())

{

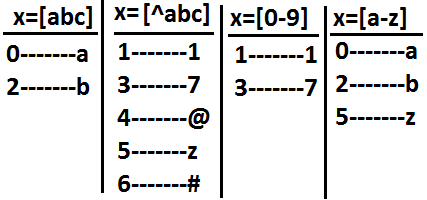
System.out.println(m.start()+"-------"+m.group());

}

}

}

**Output:**



**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.

**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())

{

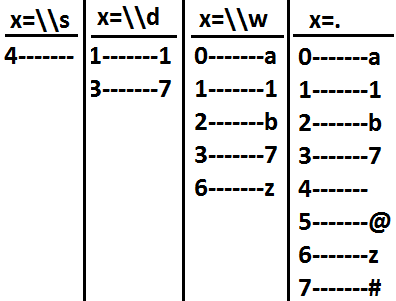
System.out.println(m.start()+"-------"+m.group());

}

}

}

**Output:**



**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())

{

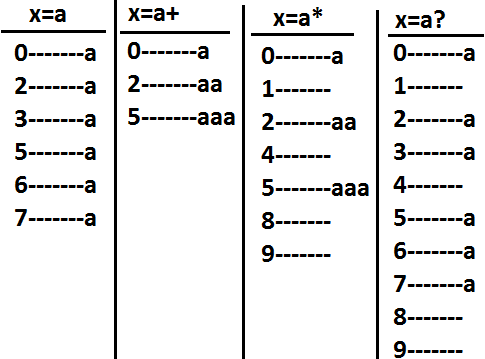
System.out.println(m.start()+"-------"+m.group());

}

}

}

**Output:**

****

**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("bhaskar software solutions");

for(String s1:s)

{

System.out.println(s1);//bhaskar

//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.durgajobs.com";

String[] s1=s.split("\\.");

for(String s2:s1)

{

System.out.println(s2);//www

//durgajobs

//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("durga software solutions");

while(st.hasMoreTokens())

{

System.out.println(st.nextToken());//durga

//software

//solutions

}

}

}

* The default regular expression for the StringTokenizer is space.

**Example 2:**

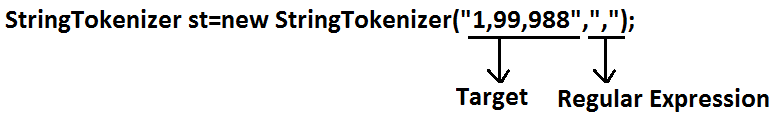
import java.util.\*;

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 yava 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 **bhaskar**

Valid identifier

E:\scjp>java RegularExpressionDemo **?bhaskar**

Invalid identifier

**Requirement:** Write a regular expression to represent all mobile numbers.

**Rules:**

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 9989308279

Valid number

E:\scjp>java RegularExpressionDemo 6989308279

Invalid number

**10 digits (or) 11 digits:**

**(0?[7-9][0-9]{9})**

**Output:**

E:\scjp>javac RegularExpressionDemo.java

E:\scjp>java RegularExpressionDemo 9989308279

Valid number

E:\scjp>java RegularExpressionDemo 09989308279

Valid number

E:\scjp>java RegularExpressionDemo 919989308279

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 9989308279

Valid number

E:\scjp>java RegularExpressionDemo 09989308279

Valid number

E:\scjp>java RegularExpressionDemo 919989308279

Valid number

E:\scjp>java RegularExpressionDemo 69989308279

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 bhaskar86.vaka@gmail.com

Valid mail id

E:\scjp>java RegularExpressionDemo 999bhaskar86.vaka@gmail.com

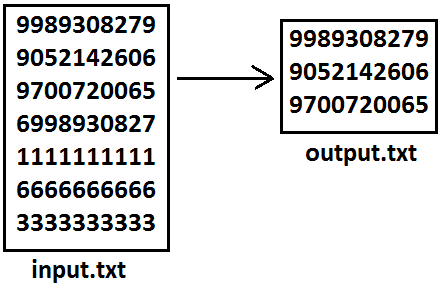
Invalid mail id

E:\scjp>java RegularExpressionDemo 999bhaskar86.vaka@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("[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