1. **Regular Expression**
2. **Regular Expression Character Classes.**
3. **Regular Expression Meta Characters.**
4. **Regular Expression Quantifiers**
5. **Split Method.**
6. **Regular Expression Applications.**

**REGULAR EXPRESSION**

1.What is Regular Expression?

A. a sequence of symbols and characters expressing a string or pattern to be searched for within a longer piece of text.The Regular expressio can be single character or more than one character. It consists of alphabets,digits and special characters(hypen,caret, dot, dollar,…etc).

The regular expression is used to define constraint on strings such as password / email validations.

2.what is Java Regular Expression(Java Regex)?

A. The Java Regular Expression is API which is in java.util. package.

Java.util.regex;

This api is used to define a pattern for searching or manipulating string.

The Regex API contains 3 classes and one interface.

a.Pattern class.

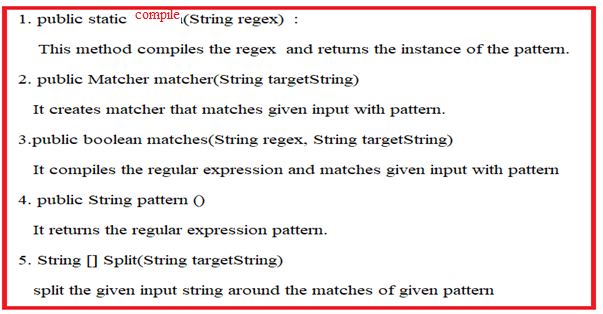
b.Matcher class.

c.PattermSyntaxException class.

d. MatchResult Interface.

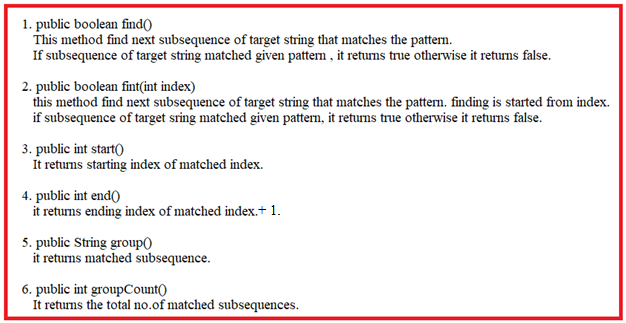
**3.Pattern Class:- A compiled representation of a regular expression**. A regular expression, specified as a string, must first be compiled into an instance of this class.

**Methods:**

****

**4.Matcher:-** It implements MatchResult interface. That provide methods which is used to perform matching operation on target string.

Methods:



**REGULAR EXPRESSION CHARACTER CLASSES**

**1. Regular Express Character classes:-**

1. [abc] -- can match a, b, or c.

2. [^ abc] -- can match any lower case alphabet except a,b and c.

3. [a-z] -- can match any lowe case alphabet.

4. [A-Z] -- can match any upper case alphabet.

5. [0-9] – can match digit from 0 ,1,… or 9.

6.[a-z A-Z] – can match any lower case and uppercase alphabet.

7.[a-z A-Z 0-9] – can match any lower case,uppercase alphabet and digit from 0 to 9.

8.[^a-z] -- can match any character except lower case alphabet.

9.[^A-Z] – can match any character except upper case alphabet.

10.[^0-9] -- can match any character except digit from 0 to 9.

11.[^ a-z A-Z 0-9] – can match ony special character.

12. you may combine the two or more regular expression character classes.

Ex: -

[^a-z A-Z 0-9][0-9][^a-z A-Z 0-9]

[a-d[m-x]] – can match from a to d and from m to x.

[a-z&&[def]]--- (intersection) d , e or f

[a-z&&[^mno]] -- can to match a to Z except m,n and o.

Example:-

import java.util.Scanner;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Sample {

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner s1=new Scanner(System.in);

String s2,s3;

s2="This is #7%suku(mar)";

System.out.println(“String is :”+s2);

boolean b=true;

while(b) {

System.out.print("Enter Regular Expression:");

s3=s1.nextLine();

Pattern p1=Pattern.compile(s3);

Matcher m1=p1.matcher(s2);

if(m1.find()) {

System.out.println(m1.start());

System.out.println(m1.end());

System.out.println(m1.group());

}

else

{

System.out.println("Wrong Pattern");

}

System.out.print("Do You want to continue?:");

b=s1.nextBoolean();

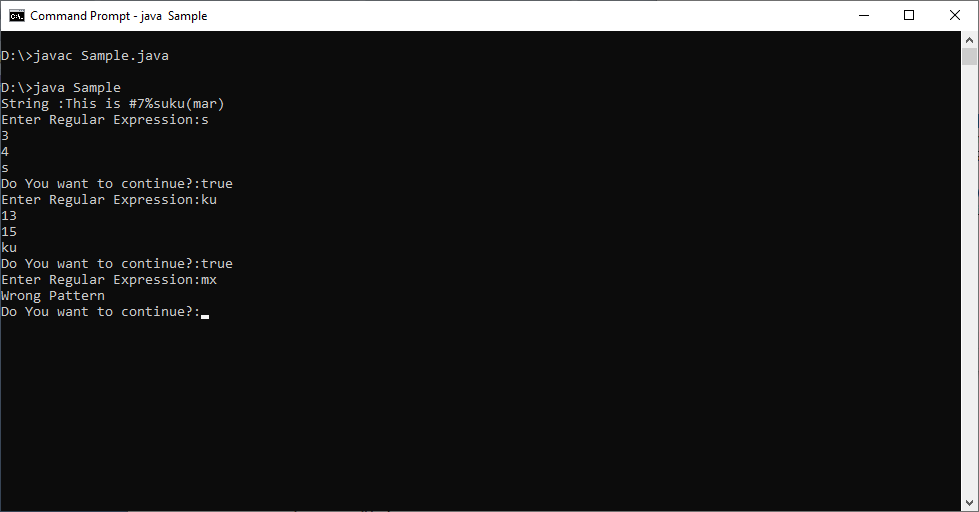
s1.nextLine();

}

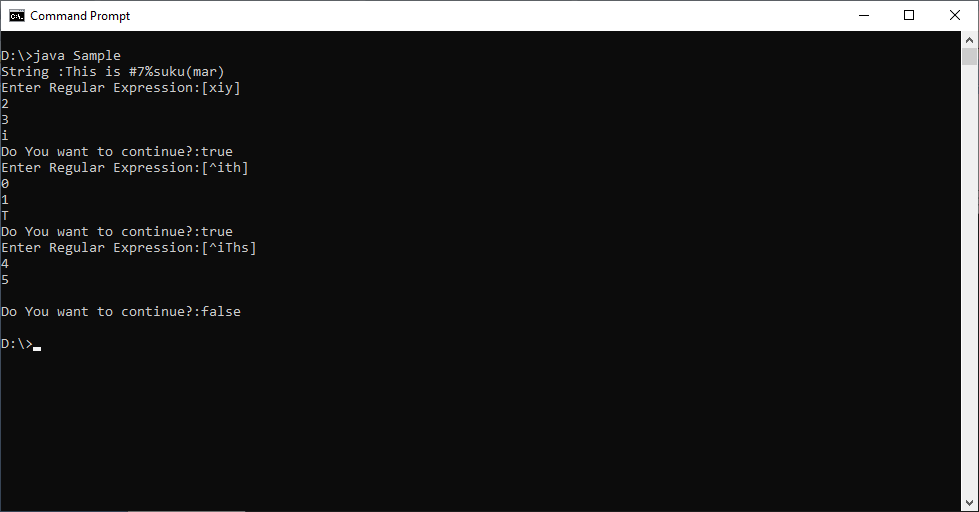
}

}

Run-1:



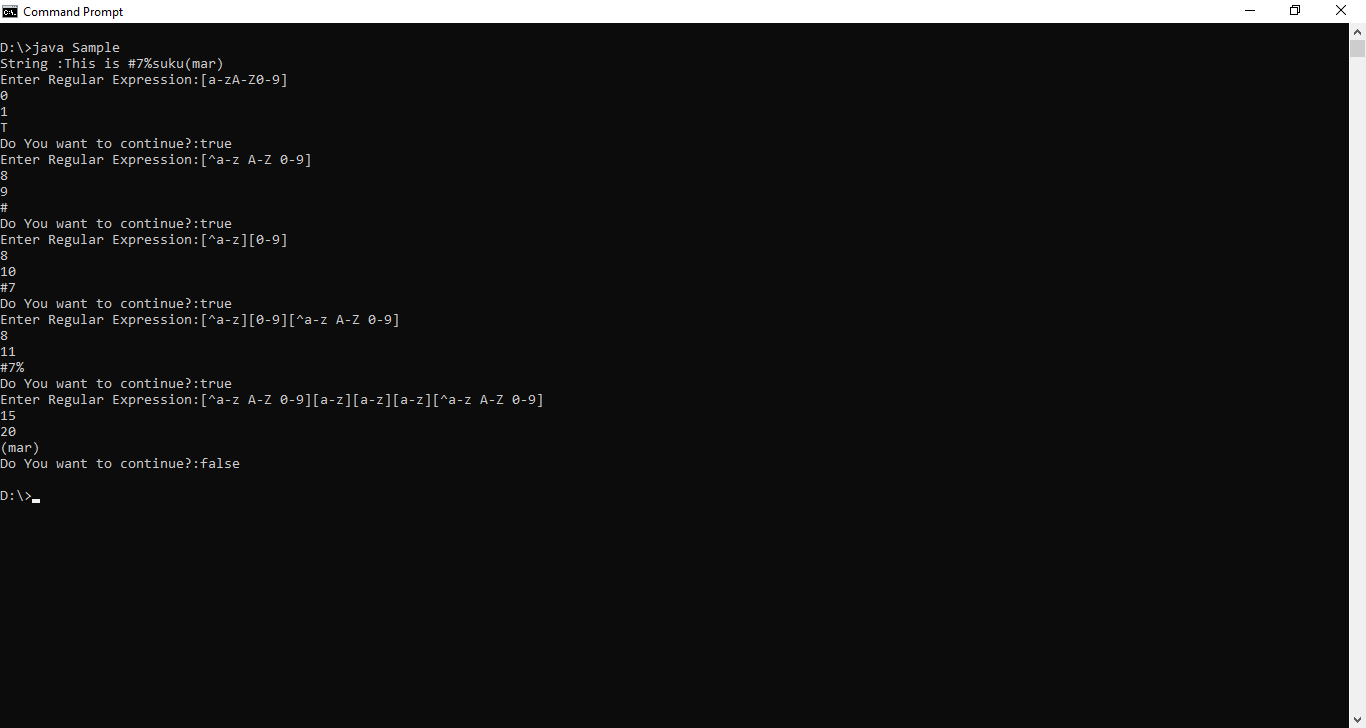
Run 2:



Run 3:



Run-4



Run 4:



**REGULAR EXPRESSION META CHARACTERS**

**1.The Regex meta characters works as shortcuts.**

1. [\\d](file:///\\d) 🡪 Match any digit.
2. [\\D](file:///\\D) 🡪 Match any non – digit.
3. [\\s](file:///\\s) 🡪 Match any white space (\t,\n,\r,\f, …etc)
4. [\\S](file:///\\S) 🡪 Match any non white space(\t,\n,\f,\r, …etc).
5. [\\w](file:///\\w) 🡪 Match Any Word character . It is short cut to [a-zA-z0-9]
6. [\\W](file:///\\W) 🡪 Match Any non-word character. It is short cut to [^a-zA-z0-9]
7. . 🡪 Match any character.

Example:-

import java.util.Scanner;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Sample {

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner s1=new Scanner(System.in);

String s2,s3;

s2="This is #7%suuuku(mar)";

System.out.println("String :"+s2);

boolean b=true;

while(b) {

System.out.print("Enter Regular Expression:");

s3=s1.nextLine();

Pattern p1=Pattern.compile(s3);

Matcher m1=p1.matcher(s2);

if(m1.find()) {

System.out.println(m1.start());

System.out.println(m1.end());

System.out.println(m1.group());

}

else

{

System.out.println("Wrong Pattern");

}

System.out.print("Do You want to continue?:");

b=s1.nextBoolean();

s1.nextLine();

}

}

}



**REGULAR EXPRESSION QUANTIFIERS**

**The quantifier specify no.of occurrences of character.**

1. X? 🡪 x occurs once or not at all.
2. X+ 🡪 x occurs once or more times.
3. X\* 🡪 x occurs zero or more times.
4. X{n} 🡪 x occurs n times only.
5. X{n,} 🡪 x occurs n or more times.
6. X{n,z} 🡪 x occurs at least n times but less than z times.

Example:-

import java.util.Scanner;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Sample {

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner s1=new Scanner(System.in);

String s2,s3;

s2="This is #7%suuuku(mar)";

System.out.println("String :"+s2);

boolean b=true;

while(b) {

System.out.print("Enter Regular Expression:");

s3=s1.nextLine();

Pattern p1=Pattern.compile(s3);

Matcher m1=p1.matcher(s2);

if(m1.find()) {

System.out.println(m1.start());

System.out.println(m1.end());

System.out.println(m1.group());

}

else

{

System.out.println("Wrong Pattern");

}

System.out.print("Do You want to continue?:");

b=s1.nextBoolean();

s1.nextLine();

}

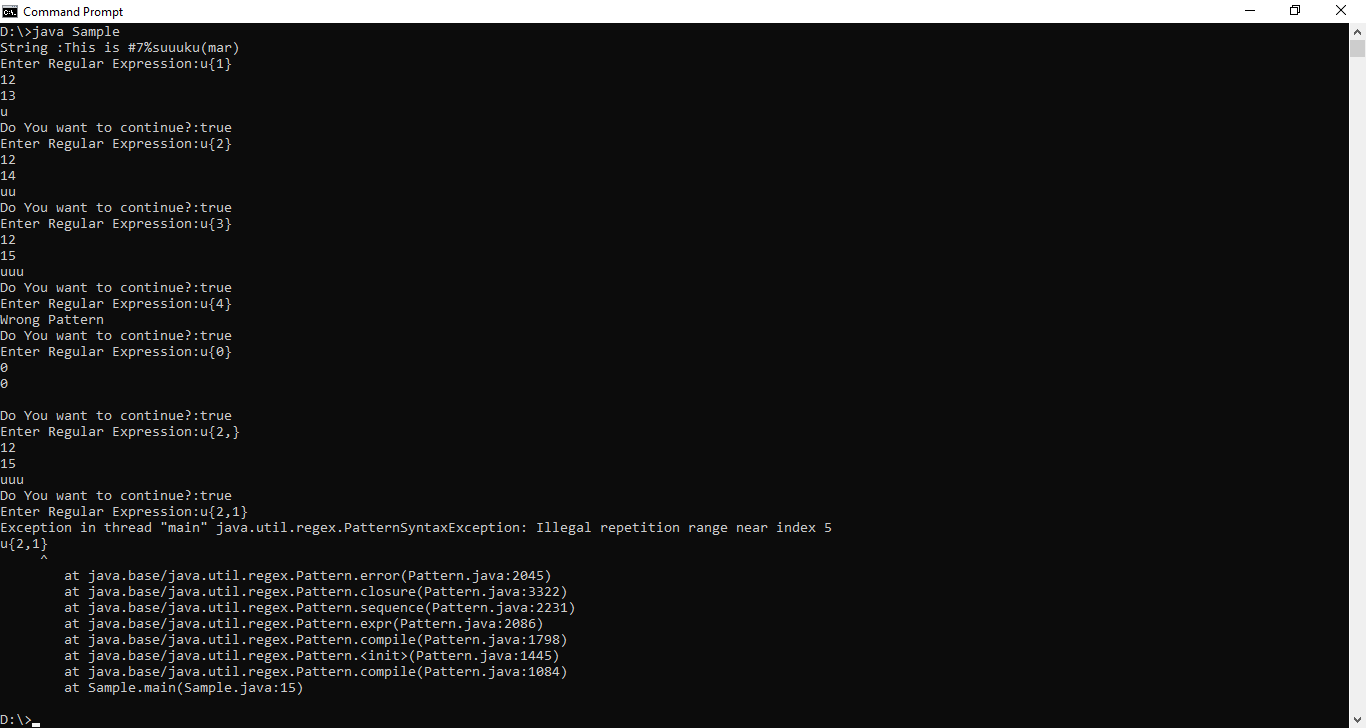
}

}

Run:1



Run :2



Run :3



**SPLIT METHOD**

String is splited using

1. Pattern.split()
2. String.split()

or

1. StringTokenizer class

**1.Pattern.split():-** The split() method of pattern class is used to split the given target string according to given pattern as tokens.

**Syntax1:**

String[] split(“target String”);

Syntax:2

String [] split(“target String”, int limit)

The limit parameter controls number of times pattern is applied on target string and therefore affects the length of resulting array.

Note:- The pattern does not appear in result tokens

Example:-

import java.util.Scanner;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Sample {

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner s1=new Scanner(System.in);

String s2;

s2="This is #7%suku(mar)";

System.out.print("Enter the pattern/Delimiter:");

String s4=s1.nextLine();

Pattern p1=Pattern.compile(s4);

String s3[]=p1.split(s2);

for(int i=0;i<s3.length;i++)

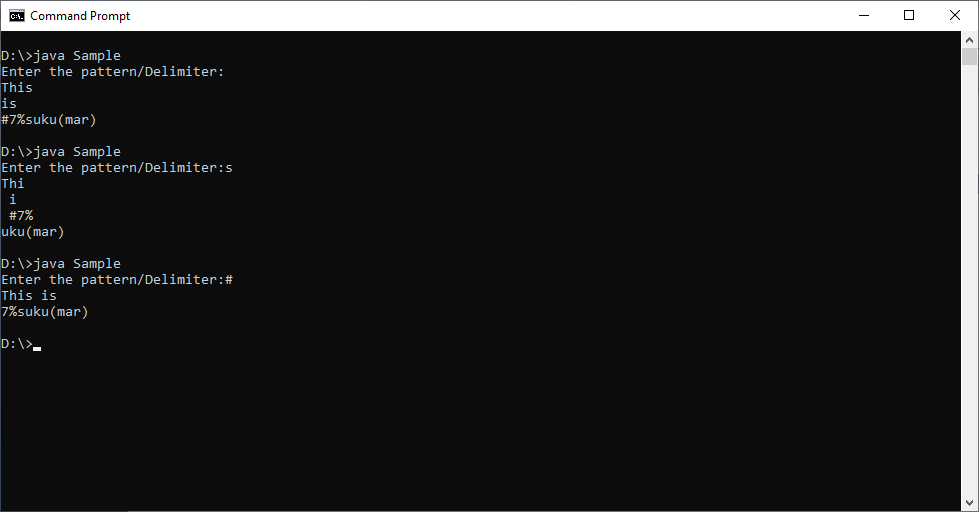
{

System.out.println(s3[i]);

}

}

}



**2.split() method of String class:- Go to String class.**

**3.StringTokanizer:-** It breaks the string into tokens.

Syntax:1

String Tokanizer(String str)

Where default delimiter is space.

Syntax:2

String Tokanizer(String str, String delimiter)

Methods:-

1. String nextToken(): It returns next token from StringTokanizer object.
2. Int countTokens():- It returns total no.of tokens .
3. Boolean hasMoreTokens():- It checks if there is more tokens available.
4. Object nextElement():- It returns next token from StringTokanizer object. Its data type is Object.

Example:-

import java.util.Scanner;

import java.util.StringTokenizer;

public class Sample {

public static void main(String[] args) {

// TODO Auto-generated method stub

StringTokenizer s3=new StringTokenizer("This is sukumar");

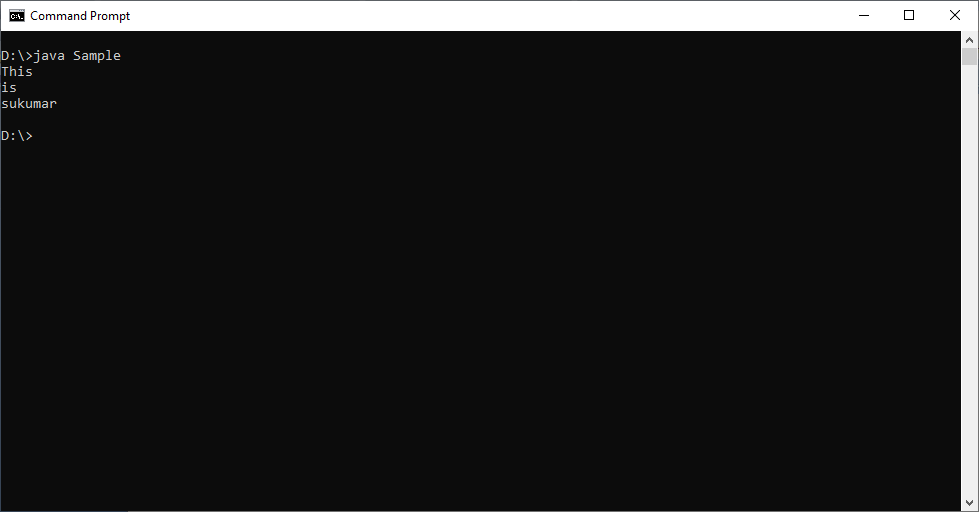
while(s3.hasMoreTokens()){

System.out.println(s3.nextToken());

}

}

}



**REGULAR EXPRESSION APPLICATION**

Example:1

Rule1 :- Each mobile number should contains 10 digits.

Rule2 :- The first digit should be 7,8 or 9.

Regular Expression: [789][0-9]{9}

Example:2

Rule1:- Each mobile number should contains either 10 or 11 digits.

Rule2:- If number have 10 digits, the first digit should be 7,8 or 9.

If number have 11 digits, the first digit should be 0.

Regular expression: 0?[789][0-9]{9}

Example:3

Rule1:- Each mobile number should contains either 10 ,11 or 12 digits.

Rule2:- If number have 10 digits, the first digit should be 7,8 or 9.

If number have 11 digits, the first digit should be 0.

If number have 12 digits, the first two digits should be 91.

Regular Expression: [0-91]?[789][0-9]{9}

Example4:

Rule1:- The first letter of email should be lower case or uppercase alphabet.

Rule2:- Remaining letters are alphabets and digits upto @ symbol.

RegularExpression: [a-zA-Z][a-zA-Z0-9\_.]\* @[a-z]{5}.com