**Java Regular Expressions:**

If we want to ***represent a group of Strings******according to a particular pattern***, we have to go for regular expression.

Examples:

We can write a regular expression, to represent all mobile numbers.

We can write a regular expression, to represent all email ids.

Note: Regular Expressions concept used in.

1. To develop **Form validations** frameworks.
2. To develop **Pattern matching applications**. eg:- ctrl f (find).
3. To develop translators’ assemblers, compliers, interpreters. [Lexical analysis phase].
4. To develop digital circuits by using more and melay.
5. To develop communications protocols (TCP/IP, UDP etc.)

**Pattern class** and **Matcher classes** are present in **java.util.regex.pkg**

* Create a Java Pattern object using pattern class compile method.
* Create a Java Matcher object using matcher method (matcher method is in Pattern class).
* Using while(m.find()) { }
* start() method returns first index of match.
* end() method returns last index of match.
* group() method returns, what is matched.

Example:

**package** cathkidston.testautomation;  
  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
**import static** java.util.regex.Pattern.*compile*;  
  
**public class** RegExp {  
  
 **public static void** main(String[] args) {  
  
 *// by using Pattern class compile() method we create a compiled version regular expression.  
 // it is java equivalent object pattern.  
 // we use matcher object to check given pattern in the target object.  
 // we use matcher() method of Pattern class.  
 // Matcher class contains boolean m.find(), int m.start() returns start index of match, int m.end() returns end index+1 match  
 // String m.group() returns which string is matched.* **int** i=0;  
 String str=**"va"**;  
 Pattern p =Pattern.*compile*(str);  
 Matcher m=p.matcher(**"Balla Srinivasa Chakravarthy"**);  
 **while** (m.find())  
 {  
 i++;  
 System.***out***.println(m.start() + **" "** + m.end() + **" "** + m.group());  
 }  
 System.***out***.println(String.*format*(**"number of times ' %s ' is is occurred "**, str) + **" "** + i);  
 }  
  
}

**Other concepts: Character classes**

Search of

[abc] means either ‘a’ , ‘b’ , ‘c’.

[^abc] means except ‘a’ , ‘b’ , ‘c’

[a-z] means any lower case alphabet symbol from a to z.

[A-Z] means any upper case alphabet symbol from a to z.

[a-z A-Z] means any lower and upper case alphabet symbol.

[0-9] means any digit from 0 to 9.

[a-z A-Z 0-9] means any alphanumeric symbols.

[^ 0-9 a-z A-Z] means except any alphanumeric symbols.

To use these Character

**package** cathkidston.testautomation;  
  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
**public class** RefExp1 {  
 **public static void** main(String[] args) {  
 {  
 Pattern p=Pattern.*compile*(**"[abc 0-9 ]"**);  
 Matcher m=p.matcher(**"45adhsddhdkllb9c9"**);  
 **while**(m.find())  
 {  
 System.***out***.println(m.start() + **" "** +m.group());  
 }  
  
 }  
  
  
 Pattern p1=Pattern.*compile*(**"[^ a-z A-Z 0-9]"**);  
 Matcher m=p1.matcher(**"aj@$£adajdjfald,,.87352##"**);  
 **while**(m.find()){  
 System.***out***.println(m.start() + **" "** +m.group());  
 }  
 }  
}

**Pre-defined Character classes:**

Search for

**\s** means space characters.

**\S** means except space characters anything includes symbols.

**\d** means any digit from 0 to 9, like [ 0 – 9]

**\D** means except digit, any character. Like [ a-z A-Z]

**\w** means any alphanumeric symbols [ 0 – 9 a-z A-Z]]

**\W** means except word characters. Means special character.

**.** means any character

Note: this pre-defined character classes character always should be used with escape character(\).

[ except for . ]

Example:

**package** cathkidston.testautomation;  
  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
**public class** RegExp2 {  
 **public static void** main(String[] args) {  
  
 Pattern p=Pattern.*compile*(**"\\s"**);  
 Matcher m=p.matcher(**"balla srinivasa @ jk157888"**);  
 **while** (m.find())  
 {  
 System.***out***.println(m.start() +**" "** + m.group());  
 }  
 }  
  
}

**Quantifiers: ( + , \*, ? )** We can use quantifies to specify number of occurrences to match.

**Example:**

**a** means exactly one ‘a’.

**a+** means atleast one ‘a’ (or) more number of ‘a’.

**a\*** means no of ‘a’ including zero number.

**a?** means atmost one ‘a’

**package** cathkidston.testautomation;  
  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
**public class** RegExp3 {  
  
 **public static void** main(String[] args) {  
 Pattern p = Pattern.*compile*(**"o"**);  
 Matcher m=p.matcher(**"goodboy"**);  
 **while**(m.find()) {  
 System.***out***.println(**"search for exactly o"**);  
 System.***out***.println(m.start()+**"..........."** + m.group());  
 }  
 Pattern p1 = Pattern.*compile*(**"o+"**);  
 Matcher m1=p1.matcher(**"goodboy"**);  
 **while**(m1.find()) {  
 System.***out***.println(**"search for atleast o more 'o'"**);  
 System.***out***.println(m1.start()+**"..........."** + m1.group());  
 }  
 Pattern p2 = Pattern.*compile*(**"o\*"**);  
 Matcher m2=p2.matcher(**"goodboy"**);  
 **while**(m2.find()) {  
 System.***out***.println(**"search for o and zeros"**);  
 System.***out***.println(m2.start()+**"..........."** + m2.group());  
 }  
 Pattern p3 = Pattern.*compile*(**"o?"**);  
 Matcher m3=p3.matcher(**"goodboy"**);  
 **while**(m3.find()) {  
 System.***out***.println(**"search for atmost o"**);  
 System.***out***.println(m3.start()+**"..........."** + m3.group());  
 }  
 }  
}

**Pattern class split() method:** splits target string.as per the given pattern.

**package** cathkidston.testautomation;  
  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
  
**public class** RexExp4 {  
 **public static void** main(String[] args) {  
  
 Pattern p=Pattern.*compile*(**"a"**);  
 String[] str=p.split(**"balla srinvasa chakravarthy"**);  
 **for**( String s:str)  
 {  
 System.***out***.println(s);  
 }  
 }  
}

**Note:** Pattern class split() method can take target as argument whereas String class split() method can take pattern as argument.

StringTokenizer class present in java.util package

It is specially designed for Tokenization activity.

StringTokenizer str=new StringTokenizer(“ balla srinivasa chakravarthy”)

while(str.hasMoreTokens()){

Sout(str.nextToken())

}

Default regular expression is space.

To divide the string according “-“.

**package** cathkidston.testautomation;  
  
**import** java.util.StringTokenizer;  
  
**public class** RegExp5 {  
  
 **public static void** main(String[] args) {  
  
 StringTokenizer str=**new** StringTokenizer(**"22-01-2019"**, **"-"**);  
 **while**(str.hasMoreTokens())  
 {  
 System.***out***.println(str.nextToken());  
 }  
 }  
}

String class, StringBuffer class and StringBuilder class are present in java.lang package.