Regular Expressions:
—--------------------
Regular Expression is an expression which represents a group of String according to a particular pattern.

EX:
We can write a Regular Expression to represent all Valid Email Ids.
We can write a Regular Expression to represent all Valid Mobile Numbers.

In Java applications we are able to utilize the Regular Expressions in the following locations.

1. To perform Data Validations.
2. To develop Pattern Matching applications.
3. To Prepare Compilers , JVMs, Servers,...
4. To prepare Digital Circuits
5. To prepare Protocols like Http, Ftp,....

   —----
   —-----

To implement Regular Expression applications Java has provided a predefined library in java.util.regex .

To prepare Regular Expressions in Java applications we have to use the following steps.

1. Create Pattern object:
Pattern is an object representing a compiled version of the Expression.
To create a Pattern object we have to use the following method from Pattern class.

```
public static Pattern compile(String regex)
EX: Pattern p = Pattern.compile(“ab”);
```

2. Create Matcher Object:
Matcher object is able to match a string against the RegularExpression which is available in the Pattern.

To create a Matcher object we will use the following method from the Pattern class.

```
public Matcher matcher(String target)
Matcher m = p.matcher(“abababababa”);
```

3. Find the Matches and their location in the String:
To find the matches and their locations in the String we have to use the following methods from Matcher class.

public boolean find():
It will find the next match , if it exists then it will return true , if no matchy is identified then it will return false value.

public int start():
It will return the start index of the match.

public int end():
It will return the end index of the match.

public String group():
It will return the matched String.

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("ab");
Matcher matcher = pattern.matcher("ababababab");
int count = 0;
while (matcher.find()){
System.out.println(matcher.group()+"----->"+matcher.start()+"------
->"+matcher.end());
count = count + 1;
}
System.out.println("No of occurences : "+count);
}
}
```

```
ab----->0------->2
ab----->2------->4
ab----->4------->6
ab----->6------->8
ab----->8------->10
No of occurences   : 5
```

```
Character Classes:
-------------------
To prepare Regular Expressions we will use character classes like below.

[abc] ------> Either ‘a’ or ‘b’ or ‘c’.
[^abc] -----> Except ‘a’, ‘b’ and ‘c’.
[a-z] ------> All Lower case letters.
[A-Z] ------> All Upper case letters.
[a-zA-Z] ---> All Lower case letters and all Uppercase letters
[0-9] ------> All digits from 0 to 9
[a-zA-Z0-9]-> All Alphanumeric characters
[^a-zA-Z0-9]-> Any Special character
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("[abc]");
Matcher matcher = pattern.matcher("a1b7@z#");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
a ----> 0
b ----> 2
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("[^abc]");
Matcher matcher = pattern.matcher("a1b7@z#");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
1 ----> 1
7 ----> 3
@ ----> 4
z ----> 5
# ----> 6
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("[a-z]");
Matcher matcher = pattern.matcher("a1b7@z#");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
a ----> 0
b ----> 2
z ----> 5
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("[0-9]");
Matcher matcher = pattern.matcher("a1b7@z#");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
1 ----> 1
7 ----> 3
```

In Regular Expressions we are able to use the following predefined character classes.

```
\s ---------> Space character
\d ---------> ANy digit from 0-9 that is [0-9]
\w ---------> Any word character[a-zA-Z0-9]
. ----------> Any character including space characters.

\S ---------> Any character except Space character
\D ---------> Any Character except digits.
\W ---------> Any character except word characters that are special symbols.
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("\\s");
Matcher matcher = pattern.matcher("a1b7 @z#");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
  ----> 4
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("\\d");
Matcher matcher = pattern.matcher("a1b7 @z#");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
1 ----> 1
```

```
7 ----> 3
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("\\w");
Matcher matcher = pattern.matcher("a1b7 @z#");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
a ----> 0
1 ----> 1
b ----> 2
7 ----> 3
z ----> 6
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile(".");
Matcher matcher = pattern.matcher("a1b7 @z#");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
a ----> 0
1 ----> 1
b ----> 2
7 ----> 3
  ----> 4
@ ----> 5
z ----> 6
```

```
# ----> 7
```

Quantifiers:

Quantifiers can be used to specify the number of characters to match.

```
a ---------> Exactly one ‘a’.
a+ --------> At least one ‘a’. One or more
a* --------> Any number of a’s including 0.
a? --------> Atmost one ‘a’.
```

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("a");
Matcher matcher = pattern.matcher("abaabaab");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
a ----> 0
a ----> 2
a ----> 3
a ----> 5
a ----> 6
```

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("a+");
Matcher matcher = pattern.matcher("abaabaaab");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
a ----> 0
aa ----> 2
aaa ----> 5
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("a*");
Matcher matcher = pattern.matcher("abaabaaab");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
a ----> 0
 ----> 1
aa ----> 2
 ----> 4
aaa ----> 5
 ----> 8
 ----> 9
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("a?");
Matcher matcher = pattern.matcher("abaabaaab");
while(matcher.find()){
System.out.println(matcher.group()+" ----> "+matcher.start());
}
}
}
```

```
a ----> 0
```

```
 ----> 1
a ----> 2
a ----> 3
 ----> 4
a ----> 5
a ----> 6
a ----> 7
 ----> 8
 ----> 9
```

split() method in Pattern Class:

It will split a String into the number of pieces.
EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("\\s");
String[] strAttay = pattern.split("Durga Software Solutions");
for(String str: strAttay){
System.out.println(str);
}
}
}
```

```
Durga
Software
Solutions
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("\\.");
String[] strAttay = pattern.split("www.durgasoft.com");
for(String str: strAttay){
System.out.println(str);
}
}
}
```

```
www
durgasoft
com
```

EX:

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

public class Main {
public static void main(String[] args) {
StringTokenizer stringTokenizer = new StringTokenizer("1,99,988",
",");
while (stringTokenizer.hasMoreTokens()){
System.out.println(stringTokenizer.nextToken());
}
}
}
```

```
1
99
988
```

Q) Write a Regular Expression to represent all valid identifiers in java language.

1. a to z, A to Z, 0 to 9
2. The first character should be an Alphabet symbol only.
3. The length of the Identifier should be atleast 2.

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("[a-zA-Z][a-zA-Z][a-zA-Z0-9]*");
String data = "abc";
Matcher matcher = pattern.matcher(data);
if(matcher.find() && matcher.group().equals(data)){
System.out.println("Valid Identifier");
}else{
System.out.println("Invalid Identifier");
}
}
}
```

```
Valid Identifier
```

EX:

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("[a-zA-Z][a-zA-Z][a-zA-Z0-9]*");
String data = "9eno";
Matcher matcher = pattern.matcher(data);
if(matcher.find() && matcher.group().equals(data)){
System.out.println("Valid Identifier");
}else{
System.out.println("Invalid Identifier");
}
}
}
```

```
Invalid Identifier
```

Q) Write a Regular Expression to represent all mobile numbers?

1. It must have exactly 10 digits.
2. 1 st digit must be from 7 to 9.

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

public class Main {
public static void main(String[] args) {
Pattern pattern = Pattern.compile("[7-9]\\d{9}");// [7-9][0-9]{9}
String data = "9977665544";
Matcher matcher = pattern.matcher(data);
if(matcher.find() && matcher.group().equals(data)){
System.out.println("Valid Mobile Number");
}else{
System.out.println("Invalid Mobile Number");
}
}
}
```

Valid Mobile Number

Q)Write a Regular Expression to represent all Email Ids:
—---

Good Evening Guys, Today no class , please attend tomorrow…