

```

4 import java.time.*;
5 public class dummt {
6
7     public static void main(String[] args) {
8         LocalDateTime datetime = LocalDateTime.of(1999,8,27,5,30,22);
9         System.out.println(datetime);
10
11         int dd = datetime.getDayOfMonth();
12         int mm = datetime.getMonthValue();
13         int yy = datetime.getYear();
14
15         System.out.printf("%d-%d-%d",dd,mm,yy);
16         System.out.println();
17         int hh = datetime.getHour();
18         int min = datetime.getMinute();
19         System.out.printf("%d ---- %d", hh,min);
20     }
21 }
22

```

REGULAR EXPRESSIONS

1. Regular Expressions
2. Pattern
3. Matcher
4. Character Classes
5. Predefined Character Classes
6. Quantifiers
7. Pattern Class Split()
8. String Class split()
9. String Tokenizer

If we want to represent a Group of strings according to a Particular Pattern, then we should go for Regular Expression.

EX1: We can write a Regular Expression to represent All Mobile Numbers and emails.

1. Validation forms
2. Pattern matches applications
3. TO develop digital circuits
4. Digital circuits

Pattern:

1. Pattern Object Represents a compiled version of regular Expression.
2. We can create a Pattern Object by using compile() of Pattern Class.

Public static Pattern Compile(String regularexpression)

Pattern p = Pattern.complie("ab");

Matcher:

1. We can matcher objects to match the given pattern in the target String.
2. Wecan Create Matcher Objects by Using matcher() of Pattern Class.
3. **Public Matcher matcher(String target);**
4. **Matcher m = p.matcher("ababbaab");**

Methods of Matcher Object:

1. **Public boolean find():**
 - a. It attempts to find the next match and returns true if its available. Otherwise returns false.
2. **Public int start()**
 - a. Returns Start index of match.
3. **Public int end()**
 - a. Returns end+1 index of match
4. **Public string group();**
 - a. Returns a Matched pattern.

Pattern and Matcher are from java.util.regex package. And 1.4v of java.

```

3 import java.util.regex.*;
4
5 public class demo {
6     public static void main(String args[])
7     {
8         Pattern p = Pattern.compile("ab");
9         // By using class name if we are call a menthod that return the
10        //same object is called Static fatory mthods.
11
12        Matcher m = p.matcher("ababbaab");
13        while(m.find())
14        {
15            System.out.println(m.start());
16        }
17    }
18 }
19
20

```

```

3 import java.util.regex.*;
4
5 public class demo {
6     public static void main(String args[])
7     {
8         Pattern p = Pattern.compile("ab");
9         // By using class name if we are call a menthod that return the
10        //same object is called Static fatory mthods.
11
12        Matcher m = p.matcher("ababbaab");
13        while(m.find())
14        {
15            System.out.println(m.start()+"----"+m.end()+"----"+m.group());
16        }
17    }
18 }
19
20

```

Console × Problems Debug Shell

<terminated> demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v2022

```

0----2----ab
2----4----ab
6----8----ab

```

Character Classes:

Character Classes:

[abc]	Either a OR b OR c
[^abc]	Except a, b and c
[a-z]	Any Lower Case Alphabet Symbol
[A-Z]	Any Upper Case Alphabet Symbol
[a-z A-Z]	Any Alphabet Symbol
[0-9]	Any Digit from 0 to 9
[a-z A-Z 0-9]	Any Alpha Numeric Symbol
[^a-z A-Z 0-9]	Except Alpha Numeric Symbol (Special Characters)

Pre - Defined Character Classes:

\s	Space Character
\S	Any Character Except Space
\d	Any Digit from [0-9]
\D	Any Character Except Digit
\w	Any Word Character [Any Alpha Numeric Character] [a-Za-z0-9]
\W	Except Word Character (Special Character)
.	Any Symbol including Special Character Also

```

2
3 import java.util.regex.*;
4
5 public class demo {
6     public static void main(String args[])
7     {
8         Pattern p = Pattern.compile("\\s");
9         String s[] = p.split("Chandu is my FRD");
10        for(String ss : s)
11            System.out.println(ss);
12
13    }
14 }
15

```

```

3 import java.util.regex.*;
4
5 public class demo {
6     public static void main(String args[])
7     {
8         Pattern p = Pattern.compile("[.]");
9         String s[] = p.split("www.chandu.com");
10        for(String ss : s)
11            System.out.println(ss);
12
13    }
14 }

```

StringTokenizer:

- Specially designed class for tokenization activities.
- Java.util package

```
StringTokenizer st = StringTokenizer("Chandu Soft ware");
```

```
while(st.hasMoreTokens())
```

```
{
```

```
    S.o.p(st.nextToken());
```

```
}
```

```

3 import java.util.regex.*;
4 import java.util.*;
5
6 public class demo {
7     public static void main(String args[])
8     {
9         StringTokenizer st = new StringTokenizer("Chandu Soft Ware");
10        while(st.hasMoreTokens())
11        {
12            System.out.println(st.nextToken());
13        }
14    }
15 }
16 }
17

```

Console × Problems Debug Shell

```

:terminated> demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_6
Chandu
Soft
Ware

```

Validate the Mobile Number By REGEX.

```

3 import java.util.regex.*;
4
5 public class demo {
6     public static void main(String args[])
7     {
8         Pattern p= Pattern.compile("[6-9][0-9]{9}");
9         String MyMobile = new String("9950264821");
10        Matcher m = p.matcher(MyMobile);
11
12        if(m.find() && m.group().equals(MyMobile))
13        {
14            System.out.println("Valid mobile Number");
15        }
16        else {
17            System.out.println("Invalid");
18        }
19    }
20 }
21 }
22 }
23

```

Console × Problems Debug Shell

```

<terminated> demo [Java Application] G:\Eclipse Setup\eclipse\plugins\org.eclipse.justj.openjdk.hot
Valid mobile Number

```

```

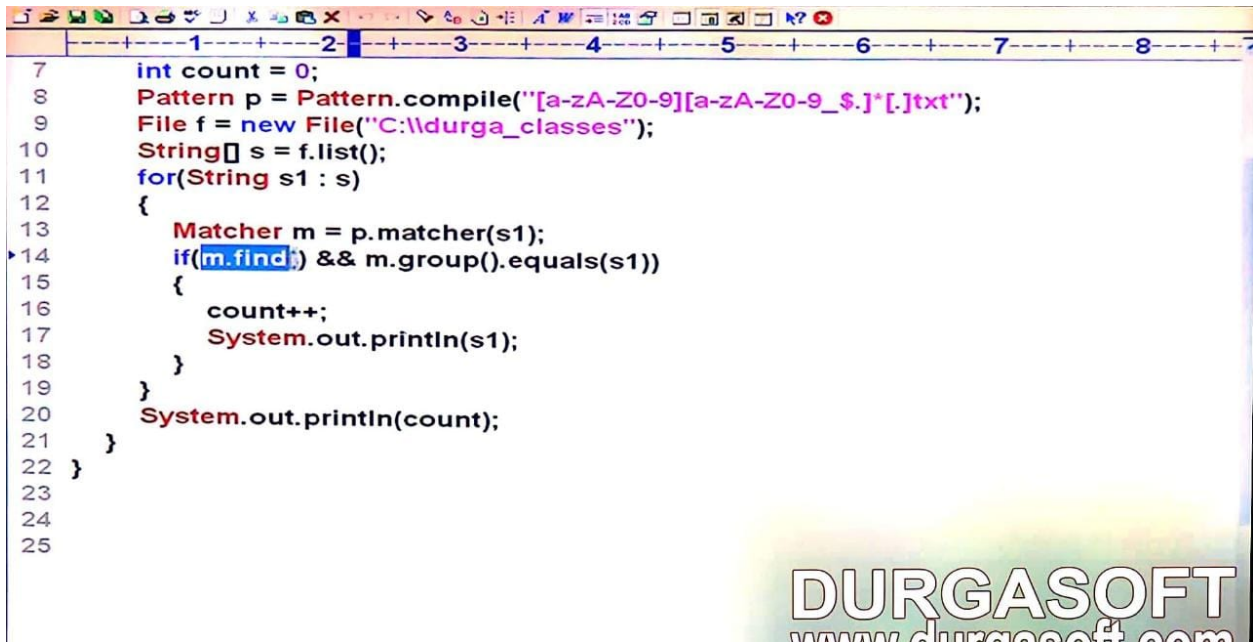
3 import java.util.regex.*;
4
5 public class demo {
6     public static void main(String args[])
7     {
8         // email regex
9         Pattern p = Pattern.compile("[a-zA-Z0-9]+@gmail[.][a-z]+");
10        Matcher m = p.matcher("7amireddyachandu77@gmail.co.in");
11
12        // Mobile Number
13        Pattern p = Pattern.compile("(0|91)?[6-9][0-9]{9}");
14        Matcher m = p.matcher("919959574853");
15
16        if(m.find())
17        {
18            System.out.println("Number Verified "+m.group());
19        }
20        else {
21            System.out.println("invalid");
22        }
23    }
24 }
25 }

```

```

3 import java.util.regex.*;
4 import java.io.*;
5
6 public class demo {
7     public static void main(String args[]) throws Exception
8     {
9         PrintWriter out = new PrintWriter("mobile.txt");
10        Pattern p = Pattern.compile("[6-9][0-9]{9}");
11        BufferedReader bf = new BufferedReader(new FileReader("data.txt"));
12        String line = bf.readLine();
13        while(line != null)
14        {
15            Matcher m = p.matcher(line);
16            while(m.find()) {
17                out.println(m.group());
18            }
19            line = bf.readLine();
20        }
21
22        out.flush();
23    }
24 }
25 }

```

A screenshot of a Java IDE window. The code is a Java program that lists files in a directory and counts those matching a specific regex pattern. The regex is '[a-zA-Z0-9][a-zA-Z0-9_\$.]*[.].txt'. The program uses Pattern.compile, File, File.listFiles, and Matcher. The output shows the count of matching files. A watermark 'DURGASOFT' and 'www.durgasoft.com' are visible in the bottom right corner of the IDE window.

```
7    int count = 0;
8    Pattern p = Pattern.compile("[a-zA-Z0-9][a-zA-Z0-9_$.]*[.].txt");
9    File f = new File("C:\\durga_classes");
10   String[] s = f.listFiles();
11   for(String s1 : s)
12   {
13       Matcher m = p.matcher(s1);
14       if(m.find()) && m.group().equals(s1)
15       {
16           count++;
17           System.out.println(s1);
18       }
19   }
20   System.out.println(count);
21 }
22 }
23
24
25
```

Enter your regex: (\d\d)\1

Enter input string to search: 1212

I found the text "1212" starting at index 0 and ending at index 4.

If you change the last two digits the match will fail:

Enter your regex: (\d\d)\1

Enter input string to search: 1234

No match found.

Boundary Construct	Description
<code>^</code>	The beginning of a line
<code>\$</code>	The end of a line
<code>\b</code>	A word boundary
<code>\B</code>	A non-word boundary
<code>\A</code>	The beginning of the input
<code>\G</code>	The end of the previous match
<code>\Z</code>	The end of the input but for the final terminator, if any
<code>\z</code>	The end of the input

The following examples demonstrate the use of boundary matchers `^` and `$`. As noted above, `^` matches the beginning of a line, and `$` matches the end.

Enter your regex: `^dog$`

Enter input string to search: `dog`

I found the text "dog" starting at index 0 and ending at index 3.

Enter your regex: `\s*dog$`

Enter input string to search: `dog`

I found the text "dog" starting at index 0 and ending at index 15.

To check if a pattern begins and ends on a word boundary (as opposed to a substring within a longer string), just use `\b` on either side; for example, `\bdog\b`

Enter your regex: `\bdog\b`

Enter input string to search: `The dog plays in the yard.`

I found the text "dog" starting at index 4 and ending at index 7.

Enter your regex: `\bdog\b`

Enter input string to search: The doggie plays in the yard.

No match found.

To match the expression on a non-word boundary, use `\B` instead:

Enter your regex: `\bdog\B`

Enter input string to search: The dog plays in the yard.

No match found.

Enter your regex: `\bdog\B`

Enter input string to search: The doggie plays in the yard.

I found the text "dog" starting at index 4 and ending at index 7.

To require the match to occur only at the end of the previous match, use `\G`:

Enter your regex: `dog`

Enter input string to search: dog dog

I found the text "dog" starting at index 0 and ending at index 3.

I found the text "dog" starting at index 4 and ending at index 7.

Enter your regex: `\Gdog`

Enter input string to search: dog dog

I found the text "dog" starting at index 0 and ending at index 3.

Here the second example finds only one match, because the second occurrence of "dog" does not start at the end of the previous match.

- `?:` Match expression but do not capture it.
- `?=` Match a suffix but exclude it from capture.
- `?!` Match if the suffix is absent.

essions 101

SHARE

ave Regex ctrl+s

R



CRE2 (PHP >=... ✓

CRE (PHP <7.3)

CMAScript (JavaSc...

REGULAR EXPRESSION

```
/ <(.*?)><(.*?)><\1><\/\2><\2>
```

TEST STRING

```
<h3><h2><h3></h2><h2>
```