**Formatter**

* Format of formatting String.

% [arg\_index$][flags][width][.precision]conversion- character.

* % & conversion –character are mandatory & rest things are optional.
* While representing any argument or printing any arguments then we are using this format.
* It is introduced in jdk 1.5 onwards.
* Though it’s a formatter we can format any argument from the String.

**Conversion Character:**

* If argument is byte, short, int, long then conversion character is d.
* For float and double conversion character is f.
* For boolean conversion character is b.
* For char conversion character is c.
* For String conversion character is s.

**Flags:**

* - =>left alignment.
* + => explicitly +ve sign for +ve number.
* 0 => padding with 0’s in left unoccupied places.
* , => to group a number.

Example:

**package**com.lara.FormatterNotes;

**publicclass** FormatNotes1

{

**publicstaticvoid** main(String[] args)

{

System.*out*.printf("%s", "abc");

System.*out*.printf("\n%s", "Sailendra");

System.*out*.printf("\nHello to All %s", "Sailendra Narayan Jena");

}

}

/\*

Here we are using formatter in this program. Here one changes we have done that is instead of write println() method we have used printf() method. Even after that we are passing two

argument into this printf() method, one is formatter and another one is String character.

Here is the output of this program:

abc

Sailendra

Hello to All SailendraNarayanJena

Here in this output we got to know that we are passing \n before into format sign then it will count as next line and JVM will start print from the next line onwards.

Here in third statement we got to know that if we are adding \n as well as any statement then JVM will understand the user wants to print in then next line with adding this statement with

value.

\*/

Example:

**package**com.lara.FormatterNotes;

**publicclass** FormatNotes2

{

**publicstaticvoid** main(String[] args)

{

**int**i = 10;

System.*out*.printf("i = %d", i);

}

}

/\*

Here in this program we are trying to print int value into this printf() method using formatter. So first we have declared an int data types with value.

Here is the output of this program:

i = 10

\*/

Example:

**package**com.lara.FormatterNotes;

**publicclass** FormatNotes3

{

**publicstaticvoid** main(String[] args)

{

**int** age = 10;

System.*out*.printf("Age of Kid is : %d", age);

}

}

/\*

Here we have done same thing as previous program. So here below is the output of this program:

Age of Kid is : 10

\*/

Example:

**package**com.lara.FormatterNotes;

**publicclass** FormatNotes4

{

**publicstaticvoid** main(String[] args)

{

System.*out*.printf("%d+%d=%d", 10, 10, 10+10);

}

}

/\*

Here we are changing format String value to int value by adding also.

Here is the output of this program:

10+10=20

Here printf() method taking two arguments one is String format and another is Object class type varargs.

\*/

Example:

**package**com.lara.FormatterNotes;

**publicclass** FormatNotes5

{

**publicstaticvoid** main(String[] args)

{

System.*out*.printf("Sum of %d and %d is %d", 2, 3, 2+3);

}

}

/\*

Here in this program we are trying to printing as well as adding these two int value after conversion from String value.

Here is the output of this program:

Sum of 2 and 3 is 5

\*/

Example:

**package**com.lara.FormatterNotes;

**publicclass** FormatNotes6

{

**publicstaticvoid** main(String[] args)

{

System.*out*.printf("%1$s, %2$s", "Hello", "All");

System.*out*.printf("%2$s, %1$s", "Hello", "All");

System.*out*.printf("%2$s, %1$s", "All", "All");

System.*out*.printf("%1$s, %1$s", "All", "All");

System.*out*.printf("%2$s, %2$s", "Hello", "Hello");

}

}

/\*

Here in this program we have pass two types of format in format argument part. So here %1$s is used for Hello and %2$s is used for All. So while we will write in sop statement then it will

print that way only.

Here is the below output of this program:

System.out.printf("%1$s, %2$s", "Hello", "All");

Hello, All

System.out.printf("%2$s, %1$s", "Hello", "All");

All, Hello

System.out.printf("%2$s, %1$s", "All", "All");

All, All

System.out.printf("%1$s, %1$s", "All", "All");

All, All

System.out.printf("%2$s, %2$s", "Hello", "Hello");

Hello, Hello

System.out.printf("%1$s, %2$s", "Hello", "All");

System.out.printf("%2$s, %1$s", "Hello", "All");

System.out.printf("%2$s, %1$s", "All", "All");

System.out.printf("%1$s, %1$s", "All", "All");

System.out.printf("%2$s, %2$s", "Hello", "Hello");

Hello, AllAll, HelloAll, AllAll, AllHello, Hello

Here in this program if we are adding this statement i.e %3$s then we will get an Exception i.ejava.util.MissingFormatArgumentException. Like this example:

Hello, AllAll, HelloAll, AllAll, AllHello, Hello, Exception in thread "main" java.util.MissingFormatArgumentException: Format specifier '3$s'

atjava.util.Formatter.format(Unknown Source)

atjava.io.PrintStream.format(Unknown Source)

atjava.io.PrintStream.printf(Unknown Source)

at com.lara.FormatterNotes.FormatNotes6.main(FormatNotes6.java:11)

If we will give this statement i.e %1$d in place of %1$s then also we will get java.util.IllegalFormatConversionException: d!= java.lang.String. For this example like this:

Hello, AllAll, HelloAll, AllException in thread "main" java.util.IllegalFormatConversionException: d != java.lang.String

atjava.util.Formatter$FormatSpecifier.failConversion(Unknown Source)

atjava.util.Formatter$FormatSpecifier.printInteger(Unknown Source)

atjava.util.Formatter$FormatSpecifier.print(Unknown Source)

atjava.util.Formatter.format(Unknown Source)

at java.io.PrintStream.format(Unknown Source)

atjava.io.PrintStream.printf(Unknown Source)

at com.lara.FormatterNotes.FormatNotes6.main(FormatNotes6.java:10)

From this example we got to know that we are getting if we are converting String format then we have to pass String format value in format argument section otherwise it will throw an

exceptioni.eIllegalFormatConversionException.

\*/

Example:

**package**com.lara.FormatterNotes;

**publicclass** FormatNotes7

{

**publicstaticvoid** main(String[] args)

{

// System.out.printf("(%1$d)", 100);

// System.out.printf("(%1$d), %2$s", 100, "Sailendra");

// System.out.printf("%1$d, %2$s", 100, "Sailendra");

System.*out*.printf("%1$d, %2$s", "Sailendra", 100);

}

}

/\*

Here we are passing format value argument inside a close braces and type is taking as int type and also value same int type i.e 100. So here is the output of this program:

System.out.printf("(%1$s)", 100);

(100)

Here value is coming properly

Now in other way if we are adding inside this format argument section as a String value then the output will be like this:

System.out.printf("(%1$s), %2$s", 100, "Sailendra");

(100), Sailendra

Here also we are getting proper value. If we are removing the bracket from int data type value then value will come properly.

100, Sailendra

If we will interchange value inside this argument values not in type of argument section. Then it will throw IllegalFormatConversionException. Here is the output of this program:

Exception in thread "main" java.util.IllegalFormatConversionException: d != java.lang.String

atjava.util.Formatter$FormatSpecifier.failConversion(Unknown Source)

atjava.util.Formatter$FormatSpecifier.printInteger(Unknown Source)

atjava.util.Formatter$FormatSpecifier.print(Unknown Source)

atjava.util.Formatter.format(Unknown Source)

atjava.io.PrintStream.format(Unknown Source)

atjava.io.PrintStream.printf(Unknown Source)

at com.lara.FormatterNotes.FormatNotes7.main(FormatNotes7.java:10)

\*/

Example:

**package**com.lara.FormatterNotes;

**publicclass** FormatNotes8

{

**publicstaticvoid** main(String[] args)

{

// System.out.printf("%1$5d", 100);

// System.out.printf("%1$05d", 100);

// System.out.printf("%1$-5d", 100);

// System.out.printf("%1$+10d", 100);

// System.out.printf("%1$+-10d", 100);

// System.out.printf("%1$0-10d", 100);

// System.out.printf("%1$0,10d", 100);

System.*out*.printf("%1$0,10.4d", 123.90909);

}

}

/\*

Here we have added 5 value before d format. forint value. So output of this program will be like this:

System.out.printf("%1$5d", 100);

100

Here JVM keep two space before this value 100. Because user indicate 5 before d means including 100 rest of place keep space but nit after 100 its before 100. So for this reason output is

like this above only.

System.out.printf("%1$05d", 100);

00100

Here JVM is putting 0 instead of blank space before 100. Here user mentioned like this 05d mean keep 0 in rest place before 100.

System.out.printf("%1$-5d", 100);

100

Here JVM is keeping blank space after 100 due to user mentioned as -5d mean after 100 keep blank space.

System.out.printf("%1$+10d", 100);

+100

Here JVM will keep + symbol and 6 blank space before 100 digit because here user mentioned as +10d means including d value + symbol with rest of place keep blank space.

System.out.printf("%1$+-10d", 100);

+100

Here same thing happens but little bit different is that blank space is after the value of d means 100 but + symbol will be keep before value of d. Blank space will be keep after value of d

due to - symbol.

System.out.printf("%1$0-10d", 100);

Exception in thread "main" java.util.IllegalFormatFlagsException: Flags = '-0'

at java.util.Formatter$FormatSpecifier.checkNumeric(Formatter.java:2979)

at java.util.Formatter$FormatSpecifier.checkInteger(Formatter.java:2934)

atjava.util.Formatter$FormatSpecifier.<init>(Formatter.java:2684)

atjava.util.Formatter.parse(Formatter.java:2528)

atjava.util.Formatter.format(Formatter.java:2469)

atjava.io.PrintStream.format(PrintStream.java:970)

atjava.io.PrintStream.printf(PrintStream.java:871)

at com.lara.FormatterNotes.FormatNotes8.main(FormatNotes8.java:12)

System.out.printf("%1$0,10d", 100);

0000000100

System.out.printf("%1$0,10.4d", 123.90909);

Exception in thread "main" java.util.IllegalFormatPrecisionException: 4

at java.util.Formatter$FormatSpecifier.checkInteger(Formatter.java:2936)

atjava.util.Formatter$FormatSpecifier.<init>(Formatter.java:2684)

atjava.util.Formatter.parse(Formatter.java:2528)

atjava.util.Formatter.format(Formatter.java:2469)

atjava.io.PrintStream.format(PrintStream.java:970)

atjava.io.PrintStream.printf(PrintStream.java:871)

at com.lara.FormatterNotes.FormatNotes8.main(FormatNotes8.java:14)

\*/

Date is the concrete class. Its available inside the java.util.\* package. We can create the object of this Date class. Similarly Calendar class is also one class through which we can call the Calendar of the system or server Calendar also. We can customized this both class like Date and Calendar. Here we can’t create the object of Calendar class because this class is the abstract class.

Here DateFormat&NumberFormat is an abstract class but Locale class is the concrete class.

Here is the class name with package name list below for these classes.

java.util.Date

java.util.Calendar

java.text.DateFormat

java.text.NumberFormat

java.util.Locale

Example:

**package**com.lara.FormatterNotes;

**import**java.util.Date;

**publicclass** FormatNotes9

{

**publicstaticvoid** main(String[] args)

{

Date date = **new**Date();

System.*out*.println(date);

}

}

/\*

Here in this program JVM will call the System Calendar and get the date and time from this Calendar.

Here is the output of this program:

Wed Sep 09 11:22:46 IST 2015

From this program we got to know that toString() method got override inside this class Date.

\*/

Example:

**package**com.lara.FormatterNotes;

**import**java.util.Date;

**publicclass** FormatNotes10

{

**publicstaticvoid** main(String[] args)

{

Date date1 = **new**Date();

Date date2 = **new**Date();

System.*out*.println("date1.equals(date2): "+date1.equals(date2));

}

}

/\*

Here in this program we are checking the equality between two class Date objects. Checking whether both are equal or not.

Here is the output of this program:

date1.equals(date2): true

From this program we got to know that equals() and hashCode() method also got override inside this class Date.

\*/

Example:

**package**com.lara.FormatterNotes;

**import**java.util.Date;

**publicclass** FormatNotes11

{

**publicstaticvoid** main(String[] args)

{

// Date date = new Date(0);

// System.out.println("Date is : "+date);

// Date date = new Date(1000);

// System.out.println("Date is : "+date);

// Date date = new Date(60\*1000);

// System.out.println("Date is : "+date);

// Date date = new Date(60\*60\*1000);

// System.out.println("Date is: "+date);

Date date = **new**Date(24\*60\*60\*1000);

System.*out*.println("Date is: "+date);

}

}

/\*

Here in this program we are passing 0 inside this Date class constructor. Here 0 is the base value of all languages. So here 0 also base value of Date class also.

Date date = new Date(0);

System.out.println("Date is : "+date);

Date is :ThuJan 01 05:30:00 IST 1970

\* If we are passing 1000 value into Date class Constructor then it will shows 1 second from the System with date and time with second. Like this below:

Date date = new Date(1000);

System.out.println("Date is : "+date);

Date is :ThuJan 01 05:30:01 IST 1970

\* If we are passing 60\*1000 value into this Date Constructor then it will reset second value as 00 whenever we will check the Date and Time from the System. Here is the below output of

program.

Date date = new Date(60\*1000);

System.out.println("Date is : "+date);

Date is :ThuJan 01 05:31:00 IST 1970

\* If we are passing 60\*60\*1000 into Date class constructor then we will get the value as below:

Date date = new Date(60\*60\*1000);

System.out.println("Date is: "+date);

Date is: ThuJan 01 06:30:00 IST 1970

Here whatever value is coming the actual value. Its showing its default value and time. So we are getting the default date and time also.

\* If we are passing 24\*60\*60\*1000 into Date class constructor then we are getting complete details about date and time as default value not actual date and time of the System.

Here is the output of this program:

Date date = new Date(24\*60\*60\*1000);

System.out.println("Date is: "+date);

Date is: Fri Jan 02 05:30:00 IST 1970

\*/

Example:

**package**com.lara.FormatterNotes;

**import**java.util.Date;

@SuppressWarnings("deprecation")

**publicclass** FormatNotes12

{

**publicstaticvoid** main(String[] args)

{

Date date = **new**Date();

**long**millis = date.getTime();

**int** sec = date.~~getSeconds~~();

**int**hrs = date.~~getHours~~();

**int** min = date.~~getMinutes~~();

**int** mon = date.~~getMonth~~();

**long** year = date.~~getYear~~();

**int** day = date.~~getDay~~();

System.*out*.println("Millis from Date: "+millis);

System.*out*.println("Second from Date: "+sec);

System.*out*.println("Hours from Date: "+hrs);

System.*out*.println("Min from Date: "+min);

System.*out*.println("Month from Date: "+mon);

System.*out*.println("Year from Date: "+year);

System.*out*.println("Day from Date: "+day);

}

}

/\*

Here we are retrieving all things using this Date class object. Here we are fetching day, month, minutes, second, year, millisecond and hours using this Date class object reference i.e date.

Inside Date class these method written through which we can fetch all the things related date and time using this reference.

Here is the output of this program:

Millis from Date: 1441782658745

Second from Date: 58

Hours from Date: 12

Min from Date: 40

Month from Date: 8

Year from Date: 115

Day from Date: 3

\*/

Example:

**package**com.lara.FormatterNotes;

**import**java.util.Date;

**publicclass** FormatNotes13

{

**publicstaticvoid** main(String[] args)

{

Date date1 = **new**Date();

**long**millis = date1.getTime();

Date date2 = **new**Date(millis+24\*60\*60\*1000);

System.*out*.println("Tomorrow Date and Time: "+date2);

Date date3 = **new**Date(millis-24\*60\*60\*1000);

System.*out*.println("Yesterday Date and Time: "+date3);

}

}

/\*

Here we are fetching tomorrow and yesterday date and timing from this millis reference using also Date class Object reference. Here in this program we are first creating Date class object.

Using this object reference we are calling getTime() method from the Date class and store this values into millis long variable. Base on this millis value we are calling tomorrow date and time

as well as yesterday date and timing also.

Here is the output of this program:

Tomorrow Date and Time: ThuSep 10 12:51:02 IST 2015

Yesterday Date and Time: Tue Sep 08 12:51:02 IST 2015

\*/

**Calendar:**

Calendar class is an abstract class though we can’t create the object of this class. Inside Calendar class one method is there i.egetInstance() method through which we can get the instance or object of the class Calendar.

Example:

**package**com.lara.FormatterNotes;

**import**java.util.Calendar;

**import**java.util.Date;

**publicclass** FormatNotes14

{

**publicstaticvoid** main(String[] args)

{

//Here we are able to create the Calendar class using this getInstance() method which is available inside this Calendar class.

Calendar cal = Calendar.*getInstance*();

//Here we are only creating Date class object using this cal object reference of Calendar class.

Date date = cal.getTime();

System.*out*.println("Calling Date object before calling Calendar add method: "+date);

//Here we are adding one day from the existing day by using this add() method. Inside this method we are adding value of Date constant and one int value i.e 1.

cal.add(Calendar.*DATE*, 1);

date = cal.getTime();

System.*out*.println("Calling Date object After calling Calendar add method: "+date);

}

}

/\*

Here is the output of this program:

Calling Date object before calling Calendar add method: Wed Sep 09 13:37:24 IST 2015

Calling Date object After calling Calendar add method: ThuSep 10 13:37:24 IST 2015

\*/

Example:

packagecom.lara.FormatterNotes;

importjava.util.Calendar;

importjava.util.Date;

public class FormatNotes15

{

public static void main(String[] args)

{

//Here we are creating Calendar class object reference.

Calendar cal = Calendar.getInstance();

Date date = cal.getTime();

System.out.println("Calling Date Object before calling add() method of Calendar class: "+date);

cal.add(Calendar.DATE, -2);

date = cal.getTime();

System.out.println("Calling Date Object after calling add() method of Calendar class: "+date);

}

}

/\*

Here in this program we are using same thing as previous but little bit difference is in add method that instead of one day after we have passed -2 means we are checking the date and time

2 days before. So it will shows the date and time was 2 days back. Here we are printing todays date and time and also 2 days back date and time.

Here is the output of this program:

Calling Date Object before calling add() method of Calendar class: Wed Sep 09 14:27:36 IST 2015

Calling Date Object after calling add() method of Calendar class: Mon Sep 07 14:27:36 IST 2015

\*/

Example:

**package**com.lara.FormatterNotes;

**import**java.util.Calendar;

**import**java.util.Date;

**publicclass** FormatNotes16

{

**publicstaticvoid** main(String[] args)

{

//Here we are creating both Calendar class object using this getInstance() method and also for Date class object reference which is stored date and time using this cal reference.

Calendar cal = Calendar.*getInstance*();

Date date = cal.getTime();

System.*out*.println("Date is: "+date);

/\*Here using Calendar class object reference trying to get next month name using this add method by passing Calendar enum constant i.e MONTH and another one is passing 1 as next

\* month.

\*/

cal.add(Calendar.*MONTH*, 1);

date = cal.getTime();

System.*out*.println("Month: "+date);

/\*

\* Here we are trying to fetch the two months back name based on current month. in an argument section we passed -2 means it will call two months back name.

\*/

cal.add(Calendar.*MONTH*, -2);

date = cal.getTime();

System.*out*.println("Fetching Month after calling add() method: "+date);

}

}

/\*

Here is the output of this program:

Date is: Wed Sep 09 14:43:38 IST 2015

Month: Fri Oct 09 14:43:38 IST 2015

Fetching Month after calling add() method: Sun Aug 09 14:43:38 IST 2015

Here its calling two months back including current month also. So output is showing as Aug month.

\*/

Example:

**package**com.lara.FormatterNotes;

**import**java.util.Calendar;

**import**java.util.Date;

**publicclass** FormatNotes17

{

**publicstaticvoid** main(String[] args)

{

//Here we have created two object one is Calendar class object using getInstance() method and another is class Date object and trying to print Current Date and Time using this date refer.

Calendar cal = Calendar.*getInstance*();

Date date = cal.getTime();

System.*out*.println("Current Date and Time: "+date);

//Here we are changing the current date and time we are trying to print next year by passing argument as 1 in argument section of add() method of Calendar class.

cal.add(Calendar.*YEAR*, 1);

date = cal.getTime();

System.*out*.println("Calling date object after calling add() method of Calendar class: "+date);

//Here we are calling two year back including current year using add method argument as -2.

cal.add(Calendar.*YEAR*, -2);

date = cal.getTime();

System.*out*.println("Calling date object after calling add() method of Calendar class: "+date);

}

}

/\*

Here is the output of this program:

Current Date and Time: Wed Sep 09 14:58:49 IST 2015

Calling date object after calling add() method of Calendar class: Fri Sep 09 14:58:49 IST 2016

Calling date object after calling add() method of Calendar class: Tue Sep 09 14:58:49 IST 2014

\*/

Example:

**package**com.lara.FormatterNotes;

**import**java.util.Calendar;

**import**java.util.Date;

**publicclass** FormatNotes18

{

**publicstaticvoid** main(String[] args)

{

Calendar cal = Calendar.*getInstance*();

cal.roll(Calendar.*DATE*, 30);

Date date = cal.getTime();

System.*out*.println("Curent Date & Time: "+date);

cal.add(Calendar.*DATE*, 30);

date = cal.getTime();

System.*out*.println("Current Date and Time: "+date);

}

}

/\*

Here in this program we are using roll() method of Calendar class through which are rolling date and reach to the final current date and time. So while we are creating object of class Calendar

usinggetInstance() method then after that we are calling roll() method and passing two parameter as an argument i.e one is Calendar class enum constant i.e Date and another one is

argument value. This argument value is deciding how much time it will roll and upto when. So we have to mentioned over there any int value in argument part of roll() method.

This roll() method will roll the date till counter not finish. Here we have mentioned 30 days means it will roll 30 days and print that date and time. Counter will start from current date and

time.

Here is the output of this program:

Curent Date & Time: Wed Sep 09 15:28:04 IST 2015

Current Date and Time: Fri Oct 09 15:28:04 IST 2015

\*/

**Date Format:**

Date class object can be formatted into different style then we are using DateFormat class or formatting date object into different style. It is an abstract class. We can’t create object of this class also like Calendar class using new operation.

Example:

package com.lara.FormatterNotes;

import java.text.DateFormat;

import java.util.Calendar;

import java.util.Date;

public class FormatNotes19

{

public static void main(String[] args)

{

Calendar cal = Calendar.getInstance();

Date date = cal.getTime();

System.out.println("Currrent Date and Time: "+date);

DateFormat df = DateFormat.getDateInstance();

String dateFormat = df.format(date);

System.out.println("Formatted Date is: "+dateFormat);

}

}

/\*

Here in this program we are trying to change the format of Date class object. So using DateFormat class method i.e format(). This format() method is taking one argument as date object

which is changing the original format to another format mean user defined format.

Here is the output of this program:

Currrent Date and Time: Wed Sep 09 16:09:01 IST 2015

Formatted Date is: Sep 9, 2015

Here this format() method will formatted only date not time. Its showing only date, month and year only.

\*/

Example:

package com.lara.FormatterNotes;

import java.text.DateFormat;

import java.util.Date;

public class FormatNotes20

{

public static void main(String[] args)

{

Date date = new Date();

System.out.println("Current Date without format: "+date);

//Here we are passing argument as an SHORT value which is static final in DateFormat class and passing this value in getDateInstance() method.

DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT);

String formatDate = df.format(date);

System.out.println("Formatted Date: "+formatDate);

}

}

/\*

Here is the output of this program:

Current Date without format: Wed Sep 09 16:15:45 IST 2015

Formatted Date: 9/9/15

\*/

Example:

package com.lara.FormatterNotes;

import java.text.DateFormat;

import java.util.Date;

public class FormatNotes21

{

public static void main(String[] args)

{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Date date = new Date();

System.out.println("Current Date before Medium format: "+date);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);

String formatDate = df.format(date);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Medium Formatted Date: "+formatDate);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

DateFormat df2 = DateFormat.getDateInstance(DateFormat.DATE\_FIELD);

String date\_Filled = df2.format(date);

System.out.println("Date\_Filled Formatted Date: "+date\_Filled);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

DateFormat df6 = DateFormat.getDateInstance(DateFormat.DEFAULT);

String DEFAULT = df6.format(date);

System.out.println("DEFAULT formatted Date: "+DEFAULT);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

DateFormat df7 = DateFormat.getDateInstance(DateFormat.ERA\_FIELD);

String ERA\_FIELD = df7.format(date);

System.out.println("ERA\_FIELD formatted Date: "+ERA\_FIELD);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

DateFormat df8 = DateFormat.getDateInstance(DateFormat.FULL);

String FULL = df8.format(date);

System.out.println("FULL formatted Date: "+FULL);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

DateFormat df13 = DateFormat.getDateInstance(DateFormat.LONG);

String LONG = df13.format(date);

System.out.println("LONG formatted Date: "+LONG);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

DateFormat df16 = DateFormat.getDateInstance(DateFormat.MONTH\_FIELD);

String MONTH\_FIELD = df16.format(date);

System.out.println("MONTH\_FIELD formatted Date: "+MONTH\_FIELD);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

DateFormat df21 = DateFormat.getDateInstance(DateFormat.YEAR\_FIELD);

String YEAR\_FIELD = df21.format(date);

System.out.println("YEAR\_FIELD formatted Date: "+YEAR\_FIELD);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

/\*

Here is the output of this program:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Current Date before Medium format: Wed Sep 09 17:15:18 IST 2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Medium Formatted Date: Sep 9, 2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Date\_Filled Formatted Date: 9/9/15

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DEFAULT formatted Date: Sep 9, 2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ERA\_FIELD formatted Date: Wednesday, September 9, 2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FULL formatted Date: Wednesday, September 9, 2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LONG formatted Date: September 9, 2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MONTH\_FIELD formatted Date: Sep 9, 2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

YEAR\_FIELD formatted Date: September 9, 2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

Example:

package com.lara.FormatterNotes;

import java.text.DateFormat;

import java.text.ParseException;

import java.util.Date;

public class FormatNotes22

{

public static void main(String[] args)

{

Date date = new Date();

System.out.println("Current Date without format: "+date);

DateFormat df = DateFormat.getDateInstance(DateFormat.FULL);

String formatDate = df.format(date);

System.out.println("Formatted Date: "+formatDate);

try

{

Date date1 = df.parse(formatDate);

System.out.println(date1);

}

catch(ParseException ex)

{

ex.printStackTrace();

}

}

}

/\*

Here is the output of this program:

Current Date without format: Wed Sep 09 17:27:33 IST 2015

Formatted Date: Wednesday, September 9, 2015

Wed Sep 09 00:00:00 IST 2015

\*/

Example:

package com.lara.FormatterNotes;

import java.text.DateFormat;

import java.util.Date;

import java.util.Locale;

public class FormatNotes23

{

public static void main(String[] args)

{

Locale loc = new Locale("fr");

Date date = new Date();

DateFormat df = DateFormat.getDateInstance(DateFormat.FULL, loc);

String formatDate = df.format(date);

System.out.println("Formatted Date: "+formatDate);

}

}

/\*

Here is the output of this program:

Formatted Date: mercredi 9 septembre 2015

Here we have create one Locale class object where we have passed one value into it. That value is in String format and that value is indicates to country name. Pass this Locale value into

DateFormat class getInstance() method.

\*/

Example:

package com.lara.FormatterNotes;

import java.text.NumberFormat;

import java.util.Locale;

public class FormatNotes24

{

public static void main(String[] args)

{

double num = 12345.0909;

NumberFormat nf1 = NumberFormat.getInstance(Locale.ITALY);

String formatNum = nf1.format(num);

System.out.println("Formated Number is : "+formatNum);

}

}

/\*

Here in this program we are trying to change the one number format into other country number format. Here we have created a double value which is passing into format() method of

NumberFormat class using NumberFormat class object reference. While we are creating NumberFormat class object using getInstance() method then inside this method using Locale class

we have call enum value i.e ITALY which is a country name with will indicate the country language. So when we are passing double value then it will check which country format user wants

based on that it will convert into that country language.

Here is the output of this program:

Formated Number is : 12.345,091

\*/

Example:

package com.lara.FormatterNotes;

import java.text.NumberFormat;

import java.util.Locale;

public class FormatNotes25

{

public static void main(String[] args)

{

double num = 2345678.9898;

NumberFormat nf = NumberFormat.getInstance(Locale.UK);

String formatNum = nf.format(num);

System.out.println("Formatted Number is: "+formatNum);

}

}

/\*

Here same thing is happen with as previous only little bit is difference is that here we have mentioned locality as UK.

Here is the output of this program:

Formatted Number is: 2,345,678.99

\*/