**Printf and format methods**

Notes below from: <https://docs.oracle.com/javase/tutorial/java/data/numberformat.html>

**The printf and format Methods**

The java.io package includes a PrintStream class that has two formatting methods that you can use to replace print and println. These methods, format and printf, are equivalent to one another. The familiar System.out that you have been using happens to be a PrintStream object, so you can invoke PrintStream methods on System.out. Thus, you can use format or printf anywhere in your code where you have previously been using print or println. For example,

System.out.format(.....);

The syntax for these two [java.io.PrintStream](https://docs.oracle.com/javase/8/docs/api/java/io/PrintStream.html%22%20%5Ct%20%22_blank)methods is the same:

public PrintStream format(String format, Object... args)

where format is a string that specifies the formatting to be used and args is a list of the variables to be printed using that formatting. A simple example would be

System.out.format("The value of " + "the float variable is " +

 "%f, while the value of the " + "integer variable is %d, " +

 "and the string is %s", floatVar, intVar, stringVar);

The first parameter, format, is a format string specifying how the objects in the second parameter, args, are to be formatted. The format string contains plain text as well as *format specifiers*, which are special characters that format the arguments of Object... args. (The notation Object... args is called *varargs*, which means that the number of arguments may vary.)

Format specifiers begin with a percent sign (%) and end with a *converter*. The converter is a character indicating the type of argument to be formatted. In between the percent sign (%) and the converter you can have optional flags and specifiers. There are many converters, flags, and specifiers, which are documented in [java.util.Formatter](https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html%22%20%5Ct%20%22_blank)

Here is a basic example:

int i = 461012;

System.out.format("The value of i is: %d%n", i);

The %d specifies that the single variable is a decimal integer. The %n is a platform-independent newline character. The output is:

The value of i is: 461012

The printf and format methods are overloaded. Each has a version with the following syntax:

public PrintStream format(Locale l, String format, Object... args)

To print numbers in the French system (where a comma is used in place of the decimal place in the English representation of floating point numbers), for example, you would use:

System.out.format(Locale.FRANCE,

 "The value of the float " + "variable is %f, while the " +

 "value of the integer variable " + "is %d, and the string is %s%n",

 floatVar, intVar, stringVar);

**An Example**

The following table lists some of the converters and flags that are used in the sample program, TestFormat.java, that follows the table.

|  |
| --- |
| **Converters and Flags Used in TestFormat.java** |
| **Converter** | **Flag** | **Explanation** |
| d |   | A decimal integer. |
| f |   | A float. |
| n |   | A new line character appropriate to the platform running the application. You should always use %n, rather than \n. |
| tB |   | A date & time conversion—locale-specific full name of month. |
| td, te |   | A date & time conversion—2-digit day of month. td has leading zeroes as needed, te does not. |
| ty, tY |   | A date & time conversion—ty = 2-digit year, tY = 4-digit year. |
| tl |   | A date & time conversion—hour in 12-hour clock. |
| tM |   | A date & time conversion—minutes in 2 digits, with leading zeroes as necessary. |
| tp |   | A date & time conversion—locale-specific am/pm (lower case). |
| tm |   | A date & time conversion—months in 2 digits, with leading zeroes as necessary. |
| tD |   | A date & time conversion—date as %tm%td%ty |
|   | 08 | Eight characters in width, with leading zeroes as necessary. |
|   | + | Includes sign, whether positive or negative. |
|   | , | Includes locale-specific grouping characters. |
|   | - | Left-justified.. |
|   | .3 | Three places after decimal point. |
|   | 10.3 | Ten characters in width, right justified, with three places after decimal point. |

**The following program shows some of the formatting that you can do with format. The output is shown within double quotes in the embedded comment:**

import java.util.Calendar;

import java.util.Locale;

public class TestFormat {

 public static void main(String[] args) {

 long n = 461012;

 System.out.format("%d%n", n); // --> "461012"

 System.out.format("%08d%n", n); // --> "00461012"

 System.out.format("%+8d%n", n); // --> " +461012"

 System.out.format("%,8d%n", n); // --> " 461,012"

 System.out.format("%+,8d%n%n", n); // --> "+461,012"

 double pi = Math.PI;

 System.out.format("%f%n", pi); // --> "3.141593"

 System.out.format("%.3f%n", pi); // --> "3.142"

 System.out.format("%10.3f%n", pi); // --> " 3.142"

 System.out.format("%-10.3f%n", pi); // --> "3.142"

 System.out.format(Locale.FRANCE,

 "%-10.4f%n%n", pi); // --> "3,1416"

 Calendar c = Calendar.getInstance();

 System.out.format("%tB %te, %tY%n", c, c, c); // --> "May 29, 2006"

 System.out.format("%tl:%tM %tp%n", c, c, c); // --> "2:34 am"

 System.out.format("%tD%n", c); // --> "05/29/06"

 }

}

**The DecimalFormat Class**

You can use the [java.text.DecimalFormat](https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html%22%20%5Ct%20%22_blank)class to control the display of leading and trailing zeros, prefixes and suffixes, grouping (thousands) separators, and the decimal separator. DecimalFormat offers a great deal of flexibility in the formatting of numbers, but it can make your code more complex.

The example that follows creates a DecimalFormat object, myFormatter, by passing a pattern string to the DecimalFormat constructor. The format() method, which DecimalFormatinherits from NumberFormat, is then invoked by myFormatter—it accepts a double value as an argument and returns the formatted number in a string:

Here is a sample program that illustrates the use of DecimalFormat:

import java.text.\*;

public class DecimalFormatDemo {

 static public void customFormat(String pattern, double value ) {

 DecimalFormat myFormatter = new DecimalFormat(pattern);

 String output = myFormatter.format(value);

 System.out.println(value + " " + pattern + " " + output);

 }

 static public void main(String[] args) {

 customFormat("###,###.###", 123456.789);

 customFormat("###.##", 123456.789);

 customFormat("000000.000", 123.78);

 customFormat("$###,###.###", 12345.67);

 }

}

The output is:

123456.789 ###,###.### 123,456.789

123456.789 ###.## 123456.79

123.78 000000.000 000123.780

12345.67 $###,###.### $12,345.67

The following table explains each line of output.

|  |
| --- |
| **DecimalFormat.java Output** |
| **Value** | **Pattern** | **Output** | **Explanation** |
| 123456.789 | ###,###.### | 123,456.789 | The pound sign (#) denotes a digit, the comma is a placeholder for the grouping separator, and the period is a placeholder for the decimal separator. |
| 123456.789 | ###.## | 123456.79 | The value has three digits to the right of the decimal point, but the pattern has only two. The format method handles this by rounding up. |
| 123.78 | 000000.000 | 000123.780 | The pattern specifies leading and trailing zeros, because the 0 character is used instead of the pound sign (#). |
| 12345.67 | $###,###.### | $12,345.67 | The first character in the pattern is the dollar sign ($). Note that it immediately precedes the leftmost digit in the formatted output. |

**What is printf method in Java?**

The Java printf method is used to write the formatted strings. The ‘f’ in printf keyword means **formatted**. The printf method belongs to the [PrintStream](https://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html%22%20%5Ct%20%22_blank) and [PrintWriter](https://docs.oracle.com/javase/7/docs/api/java/io/PrintWriter.html%22%20%5Ct%20%22_blank) classes.

**The syntax of using the printf method**

For both classes (PrintStream and PrintWriter), the syntax of using the printf method is the same with two variations:

**The first way of using printf:**

printf(String format, Object… args)

Where:

* The **format** parameter specifies the format string that may contain fixed text and format specifiers. The **format specifiers** are explained in coming section.
* The args are the arguments in the format string referred by the format specifiers.
* If args are more than format specifiers then no error is thrown. The extra arguments are just ignored.

**Second way:**

printf(Locale loc, String format, Object… args)

* There you may specify a locale. If ***loc*** is null then no [locale](https://docs.oracle.com/javase/7/docs/api/java/util/Locale.html) is applied. A locale specifies the cultural, geographical or political region.

In the following section, I will show you examples of using the printf Java method to create formatted strings.

**An example of printf with different specifiers**

Before you see the list of specifiers in the following section, have a look at an example of using the printf with a few specifiers. In the example, the decimal integer, float number, a character, and a string is used with printf for writing a formatted string as follows:



|  |  |
| --- | --- |
| 1234567891011121314151617181920 | public class string\_b {           public static void main(String args[]) {                   int x = 20;                   float y = 5.5f;                   char c = 'J';                   String str = "Hello Java";                                     //Displaying formatted string                   System.out.printf("The formatted string: %d %f %c %s", x ,y, c, str);        } } |

**The output:**

The formatted string: 20 5.500000 J Hello Java

**List of format specifiers that can be used in printf**

The format specifiers include the following:

* conversion characters
* precision
* flags
* width

% conversion-character [flags] [width] [.precision]

Following is the list of conversion characters that you may use in the printf:

* %d – for signed decimal integer
* %f – for the floating point
* %o – octal number
* %c – for a character
* %s – a string
* %i – use for integer base 10
* %u – for unsigned decimal number
* %x – hexadecimal number
* %% – for writing % (percentage)
* %n – for new line = \n

**The flags list include:**

* **–** justify left
* **+** if you require to output the + or minus in the formatted string.
* **^** uppercase
* **0** for zero-padded numeric values

The **width option** specifies the maximum number of characters to be written in the output.

You may use the **precision** for the number of digits after the decimal point for floating numbers.

**An example of specifying precision in printf**

In the first example, you might notice the display of float value. For the variable ***y=5.5***, the printf displayed: 5.500000.

You may use the precision that specifies the number of digits after the decimal point for floating numbers.  See an example below where two double type variables are declared and assigned the values as follows:

a = 35.55845

b = 40.1245414

By using printf, I will display two digits after the decimal point for*variable a* and four digits for *b*:



|  |  |
| --- | --- |
| 123456789101112 | public class string\_b {           public static void main(String args[]) {                   double a = 35.55845, b = 40.1245414;                   //printf precision                   System.out.printf("x = %.2f %n b = %.4f", a, b);        } } |

The output:

x = 35.56

b = 40.1245

**The example of formatting strings**

The following example shows formatting the string with ‘%s’ specifier.



|  |  |
| --- | --- |
| 12345678910111213141516171819 | public class string\_b {           public static void main(String args[]) {                  String str = "Hello Printf";                  char x = 'z';                  System.out.printf("%s", str);                                   //Displaying with upper case                  System.out.printf("%n%S", str,x);           } } |

**The output:**

Hello Printf

HELLO PRINTF

The ‘%S’ displays the string in upper case.

**Example of displaying date with locale**

The following example displays the current date set in the local system. The second statement displays the current day:



|  |  |
| --- | --- |
| 123456789101112131415 | import java.util.Date; public class string\_b {           public static void main(String args[]) {            Date dpf = new Date();            System.out.printf("Current date/time: %tc", dpf);                  System.out.printf("%n Name of the Day, Today: %tA/%TA\n",dpf, dpf);           } } |

**The output:**

Current date/time: Mon Nov 20 16:45:19 2017

Name of the Day, Today: Monday/MONDAY

**Displaying date in day, Month name and year example**

See the following example for displaying the local date in the following format:

20 November, 2014



|  |  |
| --- | --- |
| 12345678910111213 | import java.util.Date; public class string\_b {           public static void main(String args[]) {            Date dpf = new Date();            System.out.printf("Local date: %1$td %1$tB, %1$tY", dpf);           } } |

The Output:

Local date: 11 December, 2014

**Formatting time example**

The following example shows formatting the local time by using date object with Java printf():



|  |  |
| --- | --- |
| 123456789101112131415 | import java.util.Date; public class string\_b {           public static void main(String args[]) {                  Date timepf = new Date();                                   System.out.printf( "The current local time: %1$tI:%1$tM:%1$tS %1$tZ", timepf );           } } |

The output:

The current local time: 06:06:33 PKT

**Using argument index example**

The **‘$’** can be used to refer the number of argument in the printf Java method. This is useful for situation where you have multiple variables of the same type.

For example, in one of the above example, we used single float. If we have two or more float variable, then how to identify both? This is where argument index ‘$’ plays its role.

You may use the ‘$’ as follows:

* ‘%5$ – means fifth argument
* ‘%2$ means the second argument
* ‘%10$ means tenth argument and so on.

See a demo below where I declared char, int and float variables – two for each type. All are used in the string formatting and see how all are differentiated:

[**See online demo and code**](https://www.jquery-az.com/java/demo.php?ex=177.0_1)



|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425 | public class string\_b {           public static void main(String args[]) {                  char a = 'x' , b = 'y';                  int  int\_a = 20, int\_b = 30;                  float flt\_c = 5.5f, flt\_d = 7.7f;                    System.out.printf( "Second arg:  %2$c", a,b, int\_a, flt\_c,flt\_d);                  System.out.printf( "%nFourth arg:  %4$d", a,b, int\_a, int\_b,flt\_c, flt\_d);                  System.out.printf( "%nFifth arg:  %5$.2f", a,b, int\_a, int\_b,flt\_c, flt\_d);                              } } |

The output:



**Displaying ‘%’ in formatted string**

Since the percentage sign (%) is used for formatting strings in Java then how we may display % in the formatted strings? The following example shows that:



|  |  |
| --- | --- |
| 1234567891011 | public class string\_b {           public static void main(String args[]) {                  float marks = 98.50f;                  System.out.printf( "I got %.2f%% marks. Hurray!", marks );           } } |

**The output:**

I got 98.50% marks. Hurray!

So, using the percentage twice is the way out for including a % sign in the Java formatted string using printf.

**Displaying the hashcode example**

For displaying the hashcode, use the **%h specifier**. See the example code below:



|  |  |
| --- | --- |
| 12345678910111213 | public class string\_b {           public static void main(String args[]) {                                      System.out.printf( "'Java' hashcode is %h\n", "Java" );                     System.out.printf( "'java' hashcode is %h\n", "java" );           } } |

**The output:**

‘Java’ hashcode is 231e42

‘java’ hashcode is 31aa22

**The code for displaying commas with big numbers**

In different scenarios, you may require displaying commas with numbers e.g. showing the amount as:

47,145,25.55

The following example shows how you may do this with int, float and double numbers formatting:



|  |  |
| --- | --- |
| 12345678910111213141516171819 | public class string\_b {           public static void main(String args[]) {                  int amt\_a = 478547;                  long amt\_b = 14578478;                  double amt\_c = 14575457457.121f;                     System.out.printf( "Int: %,d\n", amt\_a );                     System.out.printf( "long: %,d\n", amt\_b );                     System.out.printf( "double: %,.2f", amt\_c );           } } |

The output:

Int: 478,547

long: 14,578,478

double: 14,575,457,280.00