**Java String format Example**

In this example we are going to see how to format Strings in Java. Developers familiar with C, will find the methods used here resembling printf function. And the formatting specifications are very similar. In fact, [PrintStream](http://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html) class in Java, has a member function called printf designed to work similarly with C’s printf.

Let’s see a simple example on how you can use it.

**1. Using printf**

The complete signature of printf is printf(String format, Object... args). The first argument is a String that describes the desired formatting of the output. From there on, printf can have multiple number of arguments of any type. At runtime, these arguments will be converted to String and will be printed according to the formatting instructions.

The most basic rule that string formatting follows is this:

|  |  |
| --- | --- |
| 1 | "% [argument index] [flag] [width] [.precision] type" |

So let’s see what all of the above mean :

* % is a special character denoting that a formatting instruction follows.
* [argument index] explicitly denoted the index of the arguments to be formatted. If it not present, arguments will be formatted in the same order as they appear in the arguments list.
* [flag] is a special formatting instruction. For example, the + flag specifies that a numeric value should always be formatted with a sign, and the 0 flag specifies that 0 is the padding character. Other flags include – that is pad on the right , + pad on the left (if the formatted object is a string). Note that some flags cannot be combined with certain other flags or with certain formatted objects.
* [width] denotes the minimum number of output characters for that Object.
* [.precession] denotes the precision of floating point numbers in the output. That is basically the number of decimal digits you wish to print on the output. But it can be used for other types to truncate the output width
* type along with %, are the only mandatory formatting arguments. type simply denotes the type of the object that will be formatted in the output. For integers that is d, for strings that is s, for floating point numbers that is f, for integers with hex format that is x.

Let’s see a basic example

*StringFromatExample.java:*

|  |  |  |
| --- | --- | --- |
| 01 | package com.javacodegeeks.core.string; | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | public class StringFromatExample { | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | public static void main(String[] args) { | |
| 06 |  |

|  |  |
| --- | --- |
| 07 | System.out.printf("Integer : %d\n",15); |
| 08 | System.out.printf("Floating point number with 3 decimal digits: %.3f\n",1.21312939123); | |

|  |  |  |
| --- | --- | --- |
| 09 | System.out.printf("Floating point number with 8 decimal digits: %.8f\n",1.21312939123); | |
| 10 |  |

|  |  |  |
| --- | --- | --- |
| 11 | System.out.printf("String: %s, integer: %d, float: %.6f", "Hello World",89,9.231435); | |
| 12 | } |

|  |  |
| --- | --- |
| 13 | } |

The above program will print out :

Integer : 15

Floating point number with 3 decimal digits: 1.213

Floating point number with 8 decimal digits: 1.21312939

String: Hello World, integer: 89, float: 9.231435

So let’s see some basic rules around string formatting.

**2. Formatting a String**

Here is a basic list of the most important rules when you want to format a String.

**Integer formatting**

* %d       : will print the integer as it is.
* %6d    : will pint the integer as it is. If the number of digits is less than 6, the output will be padded on the left.
* %-6d  : will pint the integer as it is. If the number of digits is less than 6, the output will be padded on the right.
* %06d : will pint the integer as it is. If the number of digits is less than 6, the output will be padded on the left with zeroes.

Here is an example:

*StringFromatExample.java:*

|  |  |  |
| --- | --- | --- |
| 01 | package com.javacodegeeks.core.string; | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | public class StringFromatExample { | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | public static void main(String[] args) { | |
| 06 |  |

|  |  |  |
| --- | --- | --- |
| 07 | System.out.printf("%-12s%-12s%s\n","Column 1","Column 2","Column3"); | |
| 08 | System.out.printf("%-12d%-12d%07d\n",15,12,5); |

|  |  |  |
| --- | --- | --- |
| 09 | } | |
| 10 | } |

The above program will print out :

Column 1 Column 2 Column3

15 12 0000005

**String formatting**

* %s       : will print the string as it is.
* %15s    : will pint the string as it is. If the string has less than 15 characters, the output will be padded on the left.
* %-6s  : will pint the string as it is. If the string has less than 6 characters, the output will be padded on the left.
* %.8d : will print maximum 8 characters of the string.

*StringFromatExample.java:*

|  |  |  |
| --- | --- | --- |
| 01 | package com.javacodegeeks.core.string; | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | public class StringFromatExample { | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | public static void main(String[] args) { | |
| 06 |  |

|  |  |  |
| --- | --- | --- |
| 07 | System.out.printf("%-12s%-12s%s\n","Column 1","Column 2","Column3"); | |
| 08 |  |

|  |  |  |
| --- | --- | --- |
| 09 | System.out.printf("%-12.5s%s", "Hello World","World"); | |
| 10 | } |

|  |  |
| --- | --- |
| 11 |  |
| 12 | } | |

The above program will print out :

Column 1 Column 2 Column3

Hello World

**Floating point formatting**

* %f       : will print the number as it is.
* %15f    : will pint the number as it is. If the number has less than 15 digits, the output will be padded on the left.
* %.8f : will print maximum 8 decimal digits of the number.
* %9.4f : will print maximum 4 decimal digits of the number. The output will occupy 9 characters at least. If the number of digits is not enough, it will be padded

*StringFromatExample.java:*

|  |  |  |
| --- | --- | --- |
| 01 | package com.javacodegeeks.core.string; | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | public class StringFromatExample { | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | public static void main(String[] args) { | |
| 06 |  |

|  |  |  |
| --- | --- | --- |
| 07 | System.out.printf("%-12s%-12s\n","Column 1","Column 2"); | |
| 08 |  |

|  |  |  |
| --- | --- | --- |
| 09 | System.out.printf("%-12.5f%.20f", 12.23429837482,9.10212023134); | |
| 10 | } |

|  |  |
| --- | --- |
| 11 |  |
| 12 | } | |

The above program will print out :

Column 1 Column 2

12.23430 9.10212023134000000000

As you can see if you truncate the number of decimal digits, some of the precision is lost. On the other hand if you specify more decimal numbers in the formatting options, the number will be padded is necessary.

**3. Using String.format**

If you don’t want to print out the String and just want to format it for later use, you can use the static format method of the Stringclass (sort of like sprintf in C). It works in exactly the same way as printf as far as formatting is concerned, but it doesn’t print the String, it returns a new formatted String.

Let’s see an example:

*StringFromatExample.java:*

|  |  |  |
| --- | --- | --- |
| 01 | package com.javacodegeeks.core.string; | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | public class StringFromatExample { | |
| 04 |  |

|  |  |  |
| --- | --- | --- |
| 05 | public static void main(String[] args) { | |
| 06 |  |

|  |  |  |
| --- | --- | --- |
| 07 | String s = String.format("%-12.5f%.20f", 12.23429837482,9.10212023134); | |
| 08 |  |

|  |  |  |
| --- | --- | --- |
| 09 | System.out.println(s); | |
| 10 | } |

|  |  |
| --- | --- |
| 11 | } |

The above program will print out :

12.23430 9.10212023134000000000