Java String Formatting

[Last Updated: Nov 30, 2017]

java.util.Formatter is an interpreter for getting C language printf-style formatted strings. In Java, we usually use following methods to format console output which internally use Formatter class:

```
System.out.printf(String format, Object... args)
```

```
System.out.printf(Locale l, String format, Object... args)
```

We can also use following methods of java.lang.String to get formatted strings:

```
String format(String format, Object... args)
```

```
String format(Locale l, String format, Object... args)
```

The 'format' parameter in above methods, usually consists of one or multiple formatting specifier. A formatting specifier starts with %, which is a way to specify various formatting attributes to get the desired results.

In following examples we will quickly go through different formatting options available.

n - line terminator

```
System.out.printf("abc%n456%n");
```

```
abc
456
```

s or S - a String

```
System.out.printf("%s%n", "this is my string");
System.out.printf("%S%n", "this is my string");
System.out.printf("%s%n", null);
System.out.printf("%s%n", 100);
System.out.printf("%s%n", new Object());
System.out.printf("'This is the example of %s.....'%n", "string");
```

```
this is my string
THIS IS MY STRING
null
100
java.lang.Object@247507be
'This is the example of string.....'
```

b or B - true/false

It will convert null/false to false , everything else to true

```
System.out.printf("%b%n", null);
System.out.printf("%b%n", false);
System.out.printf("%B%n", false);
```

```
false
false
FALSE
```

```
System.out.printf("%b%n", true);
System.out.printf("%b%n", "true");
System.out.printf("%b%n", "false");
System.out.printf("%b%n", "test");
System.out.printf("%b%n", 1);
System.out.printf("%b%n", 'c');
System.out.printf("%b%n", new Object());
```

```
true
true
true
true
true
true
true
```


c or C - a char

```
System.out.printf("%c%n", 'a');
System.out.printf("%C%n", 'a');
System.out.printf("%c%n", 100);
System.out.printf("%c%n", null);
```

```
a
A
d
null
```

For invalid char:

```
System.out.printf("%c%n", "aString");
```

```
java.util.IllegalFormatConversionException: c != java.lang.String
       at java.base/java.util.Formatter$FormatSpecifier.failConversion(F
       at java.base/java.util.Formatter$FormatSpecifier.printCharacter(F
       at java.base/java.util.Formatter$FormatSpecifier.print(Formatter.
......
```


Formatting with padding

For left padding, an integer is used between % and the conversion specifier:

```
System.out.printf("Result: %20s%n", "example");
```

```
Result:              example
```

For right padding additional - is used.

```
System.out.printf("Result: %-20s%n", "example");
System.out.printf("%-20s result%n", "example");
```

```
Result: example
example              result
```

Good for formatting multiple lines in columns:

```
for (int i = 7; i < 300; i += 50) {
    System.out.printf("[Item:%4s  %-4s]%n", i, i * 10);
}
```

```
[Item:   7  70  ]
[Item:  57  570 ]
[Item: 107  1070]
[Item: 157  1570]
[Item: 207  2070]
[Item: 257  2570]
```

Precision

This is used to limit chars.

Syntax: x.y where x= padding (width) and y= number of chars.

(For floating numbers y is used for decimal places - next sections.)

```
System.out.printf("%2.2s%n", "Hi there!");
System.out.printf("[%6.4s]%n", "What's up?");
System.out.printf("[%-6.4s]%n", "What's up?");
```

```
Hi
[  What]
[What  ]
```

d - byte/short/int/long/BigInteger formatting

```
System.out.printf("%d%n", 2);
System.out.printf("%d%n", (byte) 2);
System.out.printf("%d%n", 2L);
System.out.printf("%d%n", BigInteger.valueOf(2L));
```

```
2
2
2
2
```

Invalid input (strings and other objects are also invalid):

```
System.out.printf("%d%n", '2');
```

```
java.util.IllegalFormatConversionException: d != java.lang.Character
       at java.base/java.util.Formatter$FormatSpecifier.failConversion(Formatter.java:4331)
       at java.base/java.util.Formatter$FormatSpecifier.printInteger(Formatter.java:2846)
       at java.base/java.util.Formatter$FormatSpecifier.print(Formatter.java:2800)
......
```

Padding with zeros

0 is used just after % and then an int for padding (as we saw in 'Formatting with padding' above).

```
System.out.printf("%04d", 2);
```

```
0002
```

Right zero padding is not possible. Using '%0-4d' or '%-04d' will throw exception:

```
System.out.printf("%0-4d", 2);
```

```
java.util.IllegalFormatFlagsException: Flags = '-0'
       at java.base/java.util.Formatter$FormatSpecifier.checkNumeric(Formatter.java:3084)
       at java.base/java.util.Formatter$FormatSpecifier.checkInteger(Formatter.java:3039)
       at java.base/java.util.Formatter$FormatSpecifier.(Formatter.java:2782)
......
```

Comma formatted numbers

A comma is used between % and d

```
System.out.printf("%,d", 1000000);
```

```
1,000,000
```

For different locale

```
System.out.printf(Locale.GERMAN, "%,d", 1000000);
```

```
1.000.000
```

Always include + sign

```
for (int i = 1; i < 4; i++) {
    System.out.printf("%+d%n", i);
}
```

```
+1
+2
+3
```

Always include parentheses for negative numbers

```
for (int i = 1; i < 4; i++) {
    System.out.printf("%(d%n", -i);
}
```

```
(1)
(2)
(3)
```

Always include leading space for positive numbers

Only one space is allowed:

```
for (int i = 1; i < 4; i++) {
    System.out.printf("[% d]%n", i);
}
```

```
[ 1]
[ 2]
[ 3]
```

Precision cannot be applied to integers

```
System.out.printf("%6.4d", 123456);
```


Recent Tutorials

```
java.util.IllegalFormatPrecisionException: 4
       at java.base/java.util.Formatter$FormatSpecifier.checkInteger(Formatter.java:3041)
       at java.base/java.util.Formatter$FormatSpecifier.(Formatter.java:2782)
       at java.base/java.util.Formatter.parse(Formatter.java:2621)
......
```


f - float/double formatting

```
System.out.printf("%f%n", 1.33f);
System.out.printf("%f%n", 1.33d);
System.out.printf("%f%n", Double.valueOf(1.33d));
System.out.printf("%f%n", BigDecimal.valueOf(1.33d));
```

```
1.330000
1.330000
1.330000
1.330000
```

Applying precisions:

Syntax: x.y, where x is width (padding) and y is decimal places. Sometimes value of x is ignored, if it's smaller than the necessary chars (including the decimal) to display. Remember x is not to limit width but to add padding (spaces); y is to decrease/increase decimal places.

```
System.out.printf("[%4.2f]%n", 12.34567);
System.out.printf("[%5.2f]%n", 12.34567);
System.out.printf("[%6.2f]%n", 12.34567);
System.out.printf("[%7.2f]%n", 12.34567);
System.out.printf("[%-7.2f]%n", 12.34567);
System.out.printf("[%7.4f]%n", 12.3);
System.out.printf("[%8.4f]%n", 12.3);
```

```
[12.35]
[12.35]
[ 12.35]
[  12.35]
[12.35  ]
[12.3000]
[ 12.3000]
```

Always display decimal with # flag

The integer portion of the result always ends with a decimal point ('.'), even if the fractional portion is zero.

```
System.out.printf("[%#1.0f]%n", 1234d);
System.out.printf("[%1.0f]%n", 1234d);
```

```
[1234.]
[1234]
```

e or E - Scientific notation

Syntax: x.ye => y=precision and x=total width (padding)

```
System.out.printf("%1.2e%n", 123.45);
System.out.printf("[%10.2e]%n", 123.45);
```

```
System.out.printf("[%-10.1e]%n", 123.45);
System.out.printf("%5.2E%n", 123.45);
```

```
1.23e+02
[  1.23e+02]
[1.2e+02   ]
1.23E+02
```

g or G - Scientific notation

It depends on precision and rounding.

```
System.out.printf("%1.2g%n", 123.45);
System.out.printf("[%10.2g]%n", 123.45);
System.out.printf("[%-10.1g]%n", 123.45);
System.out.printf("[%-10.1G]%n", 123.45);
```

```
1.2e+02
[   1.2e+02]
[1e+02     ]
[1E+02     ]
```

Index based references

A variable reference can be used as X$ just after %, where X is the index.

Following example is without referencing an index:

```
String test = "myString";
System.out.printf("%1.2s - %1.4s", test, test);
```

```
my - mySt
```

Using the reference:

```
String test2 = "myString";
System.out.printf("%1$1.2s - %1$1.4s", test2);
```

```
my - mySt
```

Using multiple references:

```
System.out.printf("%2$s | %3$1.4f | %1$,d", 1333, "hello", 5.4444);
```

```
hello | 5.4444 | 1,333
```

t or T - Date time formatting

```
System.out.printf("Hours: %tH%n", new Date());
System.out.printf("Mins: %tM%n", new Date());
System.out.printf("Secs: %tS%n", new Date());
```

```
Hours: 22
Mins: 09
Secs: 08
```

```
Date date = new Date();
System.out.printf("%tH:%tM:%tS%n", date, date, date);
//using index references
System.out.printf("%1$tH:%1$tM:%1$tS%n", date);
```

```
22:09:08
22:09:08
```

T can be used for %tH:%tM:%tS% format (last example):

```
System.out.printf("%tT", new Date());
```

```
22:09:08
```

Time in am/pm format

I - for 12 hr clock
p - for am or pm

```
System.out.printf("%1$tI:%1$tM %1$tp", new Date());
```

```
10:09 pm
```

Time in milli/nanoseconds

L - milliseconds
N - nanoseconds

```
System.out.printf("%1$tT %1$tL %1$tN", new Date());
```

```
22:09:08 357 357000000
```

TimeZone info

z - timezone offset
Z - timezone id

```
System.out.printf("%1$tT %1$tz%n", new Date());
System.out.printf("%1$tT %1$tZ%n", new Date());
```

```
22:09:08 -0600
22:09:08 CST
```

Time since epoch

s - epoch seconds
Q - epoch millis

```
System.out.printf("epoch sec: %1$ts%n", new Date());
System.out.printf("epoch millis: %1$tQ%n", new Date());
```

```
epoch sec: 1512014948
epoch millis: 1512014948440
```

Month

B - full month name
b - abbreviated month name
m - year of month number 01 - 12

```
Date dt = Date.from(ZonedDateTime.of(LocalDate.of(2017, 2, 1).atStartOfDay(),
        ZoneId.systemDefault()).toInstant());
System.out.printf("%tB%n", dt);
System.out.printf("%tb%n", dt);
System.out.printf("%tm%n", dt);
```

```
February
Feb
02
```

Spring Boot - Unit Testing Web Application V Embedded Server
Java - Different ways to Set Nested Field Va By Reflection
Java - Different ways to Set Field Value by Reflection
Installing Python 2.7 on windows
Installing Cassandra And Intro To CQLSH
Cassandra - Getting Started with Java
Java 14 - Switch Expressions And Statement Examples
Kafka - Manually Assign Partition To A Consu
Kafka - Understanding Topic Partitions
Spring Boot - Unit Testing Web Application V Mock MVC
Kafka - Introduction to Kafka Admin API
Java 14 - Helpful NullPointerException
Java 14 - Pattern Matching for instanceof
Spring Boot - Application Unit Testing with @SpringBootTest
Installing and Running Kafka
Kafka - Getting Started
Spring Boot - Different Ways To Pass Applica Properties
Reactor - Transforming into Publishers and Delaying any Error with flatMapDelayError() flatMapSequentialDelayError()
Reactor - Transforming into Publisher and Maintaining the source order with flatMapSequential()
Reactor - Transforming into Publishers and t flattening by using flatMap operation
Reactor - Using transform Operation
Reactor - Mapping items
Reactor - Getting Started
Java 13 - Text Blocks (JEP 355)
Spring Cloud - Hystrix CircuitBreaker, Threa Local Context Propagation
Spring Cloud - Circuit Breaker Hystrix Event Listener
Spring Cloud - Circuit Breaker Hystrix, Chan Default Thread Pool Properties
Spring Cloud - Circuit Breaker Hystrix, concu requests and default thread pool size
Spring Cloud - Circuit Breaker, Specifying Hy configuration in application.properties file
Spring Cloud - Hystrix Circuit Breaker, Settir Configuration Properties Using @HystrixProp
Spring Cloud - Hystrix Circuit Breaker, gettir failure exception in fallback method
Spring Cloud - Circuit Breaker Hystrix Basics
TypeScript - Standard JavaScript's built-in o Support
JavaScript - Async Processing with JavaScrip Promise
TypeScript - Applying Mixins

Day

A - full name
a - abbreviated
d - day of month, 01 - 31

```
System.out.printf("%tA%n", new Date());
System.out.printf("%ta%n", new Date());
System.out.printf("%td%n", new Date());
```

```
Wednesday
Wed
29
```

Year

Y - four digit year
y - two digit year

```
System.out.printf("%tY%n", new Date());
System.out.printf("%ty%n", new Date());
```

```
2017
17
```

Common formats shortcuts

R - %tH:%tM
T - %tH:%tM:%tS
r - %tI:%tM:%tS %Tp
D - %tm/%td/%ty
F - %tY-%tm-%td
c - %ta %tb %td %tT %tZ %tY, e.g. Sun Jul 20 16:17:00 EDT 1969.

```
System.out.printf("%tR%n", new Date());
System.out.printf("%tT%n", new Date());
System.out.printf("%tr%n", new Date());
System.out.printf("%tD%n", new Date());
System.out.printf("%tF%n", new Date());
System.out.printf("%tc%n", new Date());
```

```
22:09
22:09:08
10:09:08 PM
11/29/17
2017-11-29
Wed Nov 29 22:09:08 CST 2017
```

JPA Criteria API - Modifying and Reusing Qu Objects
JPA Criteria API - Delete Operations
JPA Criteria API - Update Operations
JPA Criteria API - Case Expressions with CriteriaBuilder.selectCase()
JPA Criteria API - Calling Database Function CriteriaBuilder.function()
Java 12 - Getting JFileChooser shortcuts par files with FileSystemView# getChooserShortcutPanelFiles()
Java 12 - CompletionStage's new methods f exception handling
Java 12 - Comparing files with Files.mismat
Java 12 - Composing Collectors with Collectors.teeing()
Java 12 - Compact Number Formatting supp
Java 12 - String Changes
Java 12 - Switch Expression (JEP 325)
Git - Understanding HEAD
Git - Creating and switching branches using 'branch' and 'checkout' commands
TypeScript - Parameter Decorator
TypeScript - Property Decorators
TypeScript - Accessor Decorators
Jackson - JSON to Java tree model
Installing MongoDB On Windows 10 and Get started with MongoDB Compass
MongoDb - Getting Started with Java
JPA Criteria API - CriteriaBuilder Date Time Operations
JPA Criteria API - Collection Operations size(index()
JPA Criteria API - CriteriaBuilder Arithmetic Operations
JPA Criteria API - CriteriaBuilder String Manipulation Methods
TypeScript - Method Decorator
Obtaining Property Descriptor by using Object.getOwnPropertyDescriptor()
Git - Hosting and Accessing Remote Reposit over SSH
Groovy - Class File Vs Script File
Groovy - Automatic JavaBean properties
Groovy - Using @PackageScope for package private visibility
Groovy - Default Visibility of Classes and Me
Groovy - Imports
Groovy - Operator Overloading
Groovy Operators - Call Operator
Getting Started with GPU programming usin Aparapi Framework
Extract files from Windows 10 Backup image Mounting/Attaching VHD/VHDX
Jackson JSON - Deep merging with @JsonM Annotation

Jackson JSON - Updating Existing objects wi JSON input during Deserialization

Linux - What is the superuser home dir?

Java CompletableFuture - Understanding CompletionStage.whenComplete() method

Java - Converting FileTime To Formatted Str and vice versa

JPA Criteria API - Using all(), any(), some() methods of CriteriaBuilder

Git - Remote repository basics

See Also

java.util.Formatter

Create Extensible Applications using Java ServiceLoader

JavaBeans components quick tutorial

Arrays Equality

Handling exceptions globally with UncaughtExceptionHandler

Watching a directory for changes

JMX quick start

Basic Authentication with URLConnection

Understanding @Inherited meta annotation

Java - HttpServer Example

Core Java Tutorials

Java 15 Features

Java 14 Features

Java 13 Features

Java 12 Features

Java 11 Features

Java 10 Features

Java 9 Module System

Java 9 Misc Features

Java 9 JShell

Recent Tutorials

Java 16 - Records Features, Quick Walk-through

Java 16 - Introduction to Records

Spring - Injecting beans into Arrays/Collections, Using @Qualifiers And Specifying the Ordering

Injecting Collections - Injecting Beans Into Arrays And Collections, ordering with Ordered Interface

Spring - Injecting beans Into Arrays and Lists, ordering with @Ordered annotation

Spring - Injecting beans into Arrays and Collections, selecting elements with @Qualifier annotation

Spring - Injecting multiple Beans Into Arrays and Collections

Spring - Arrays and Collections As Beans

Spring - Using @ComponentScan#excludeFilters to exclude classes from sca based on annotations

Spring - Using @ComponentScan#includeFilters to scan non component clas based on annotations

Spring - Implementing ApplicationContextAware Interface

Spring - Using excludeFilters attribute of @ComponentScan to exclude comp classes

Spring - Using @ComponentScan to scan non component classes via includel attribute

Share

Next P