String Formatting

Jim Wilson
MOBILE SOLUTIONS DEVELOPER & ARCHITECT
@hedgehogjim jwhh.com

Overview

The advantage of format specifiers

How to use a format specifier

Width and precision

Controlling value appearance with flags

Argument index

String Concatenation & StringBuilder

Sometimes require heavy focus on details

- Need to explicitly assemble each piece
- Customizing appearance of numeric values is generally handled manually

String Formatting

Format specifiers

- Focus is on describing desired result
- Can control many aspects of appearance

Supported by a variety of methods

- String.format
- System.out.printf
- Formatter.format

Concatenation vs. Formatting

```
My nephews are 17, 15, 8, and 6 years old
```

```
int david = 17, dawson = 15, dillon = 8, gordon = 6;


String s1 =
  "My nephews are " + david + ", " + dawson + ", “
   dillon + ", and " + gordon + " years old";


String s2 = String.format(
  "My nephews are %d, %d, %d, and %d years old",
  david, dawson, dillon, gordon);
```

Concatenation vs. Formatting

```
The average age between each is 3.6666666666666665 years
```

```
int david = 17, dawson = 15, dillon = 8, gordon = 6;
double avgDiff = ((david - dawson) +  (dawson - dillon) +
      (dillon - gordon)) / 3.0d;

String s3 =
  "The average age between each is " + avgDiff + " years";
```

Concatenation vs. Formatting

```
The average age between each is 3.7 years
```

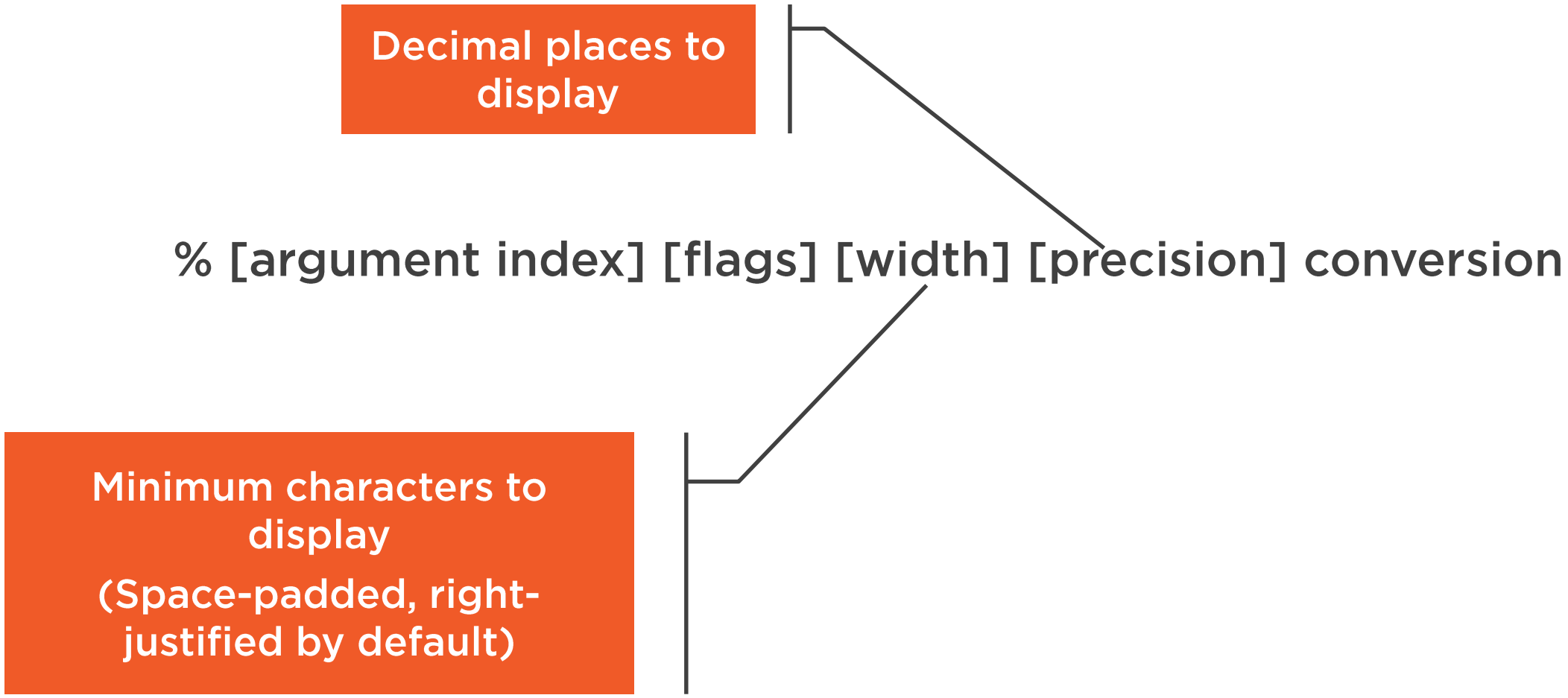
```
int david = 17, dawson = 15, dillon = 8, gordon = 6;
double avgDiff = ((david - dawson) +  (dawson - dillon) +
      (dillon - gordon)) / 3.0d;

String s4 = String.format(
  "The average age between each is %.1f years"
```

Parts of a Format Specifier

% conversion

Parts of a Format Specifier



Common Format Conversions

	Meaning	Type	Example Value	Result
d	Decimal	Integral	32	32
x X	Hex	Integral	32	20
f	Decimal	Floating point	123.0	123.000000
e E	Scientific notation	Floating point	123.0	1.230000e+02
s	String	General	“Hello”	Hello

Format Flags

Flag	Meaning
#	Include radix

Format Flag:

Main.java

```
int iVal = 32;


String s1 = String.format("%d", iVal);          32
String s2 = String.format("%x", iVal);          20


String s3 = String.format("%#x", iVal);         0x20
String s4 = String.format("%#X", iVal);         0X20
```

Format Flags

Flag	Meaning
#	Include radix
0	Zero-padding
-	Left justify

Width and Format Flags: 0 and -

Main.java

```
int w = 5, x = 235, y = 481, z = 12;


s1 = String.format("W:%d X:%d", w, x);
s2 = String.format("Y:%d Z:%d", y, z);


s3 = String.format("W:%4d X:%4d", w, x);
s4 = String.format("Y:%4d Z:%4d", y, z);
```

```
W:5 X:235
Y:481 Z:12


W:   5 X: 235
Y: 481 Z:  12
```

Width and Format Flags: 0 and -

Main.java

```
int w = 5, x = 235, y = 481, z = 12;


s5 = String.format("W:%04d X:%04d", w, x);
s6 = String.format("Y:%04d Z:%04d", y, z);


s7 = String.format("W:%-4d X:%-4d", w, x);
s8 = String.format("Y:%-4d Z:%-4d", y, z);
```

```
W:0005 X:0235
Y:0481 Z:0012


W:5    X:235
Y:481  Z:12
```

Format Flags

Flag	Meaning
#	Include radix
0	Zero-padding
-	Left justify
,	Include grouping separator

Format Flags: ,

Main.java

```
int iVal = 1234567;
double dVal = 1234567.0d;

s1 = String.format("%d", iVal);
s2 = String.format("%,d", iVal);
s3 = String.format("%,.2f", dVal);
```

```
1234567
1,234,567
1,234,567.00
```

Format Flags

Flag	Meaning
#	Include radix
0	Zero-padding
-	Left justify
,	Include grouping separator
space	Leading space when positive number
+	Always show sign
(	Enclose negative values in parenthesis

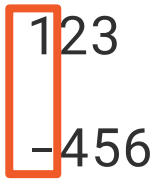
Format Flags: Space, +, and (

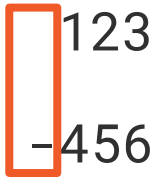
Main.java

```
int iPosVal = 123, iNegVal = -456;

s1 = String.format("%d", iPosVal);
s2 = String.format("%d", iNegVal);

s3 = String.format("% d", iPosVal);
s4 = String.format("% d", iNegVal);
```





Format Flags: Space, +, and (

Main.java

```
int iPosVal = 123, iNegVal = -456;

s5 = String.format("%+d", iPosVal);
s6 = String.format("%+d", iNegVal);

s7 = String.format("%(d", iPosVal);
s8 = String.format("%(d", iNegVal);
s9 = String.format("% (d", iPosVal);
```

```
+123
-456

123
(456)
 123
```

Argument Index

Index	Meaning
Not specified	Corresponds sequentially to argument
index$	Index of argument to use (1-based)
<	Corresponds to same argument as previous format specifier

Argument Index

Main.java

```
int valA = 100, valB = 200, valC = 300;

s1 = String.format("%d %d %d",
     valA, valB, valC);

s2 = String.format("%3$d %1$d %2$d",
     valA, valB, valC);

s3 = String.format("%2$d %<d %1$d",
     valA, valB, valC);
```

```
100 200 300

300 100 200

200 200 100
```

Summary

Format specifiers

- Focus is on describing desired result

Providing formatting info

- Always starts with %
- Always ends with conversion

Summary

Precision

- Decimal places to display

Width

- Minimum characters to display

Summary

Flags

- Alignment and padding
- Leading zeros
- Number grouping
- Sign handling

Argument index

- By default values are used in order
- Can tie a specifier to specific value
- Remember that indices are 1-based