# Producing Formatted Output

Hui Chen

Department of Computer & Information Science

Brooklyn College

# Objectives

- To format output using the **System.out.printf** and **String.format** method (§4.6).

# Outline

- Discussed
  - The Math class and its methods and constants
  - The char data type and The Character class
  - The String data type and operations
  - Three “big” example programs
- We still have a problem. How do we format the output nicely on the console?
  - Define output format using format string
  - Using the System.out.printf method with the format string
  - Using the String.format method with the format string

# Producing Formatted Output

- There is often a need in our program to produce outputs with well defined format.
- Every method which produces formatted output requires a *format string* and an *argument list*.
  - System.out.printf(…)
  - String.format(…)

# Examples

- Consider the following examples:
- Example 1.

```
java.util.Calendar rightNow =
java.util.Calendar.getInstance();

String person = "Duke";

String s = String.format("%2$s's Birthday: %1$tm
%1$te,%1$tY", rightNow, duke);
```

- Example 2.

```
double r = 5.0; double area = Math.PI * r*r;

System.out.printf("The area of the circle with
radius  %4.2f is %6.2f\n", r, area);
```

# From Examples: Format Strings and Argument List

- Example 1.

```
String s = String.format("%2$s's Birthday: %1$tm
%1$te,%1$tY", rightNow, duke);
```

  - Format String: `"%2$s's Birthday: %1$tm %1$te,%1$tY"`
  - Argument List: `rightNow, duke`

- Example 2.

```
System.out.printf("The area of the circle with radius
%4.2f is %6.2f\n", r, area);
```

  - Format String: `"The area of the circle with radius %4.2f is %6.2f\n"`
  - Argument List: `r, area`

# Define Format String

- The format string is a String which may contain fixed text and one or more embedded format specifiers.
- Examples:

```
"%2$s's Birthday: %1$tm %1$te,%1$tY"

"The area of the circle with radius  %4.2f
is %6.2f\n"
```

- The underlined are format specifiers, the rest the fixed text.

# Format Specifiers

- The format specifiers for general, character, and numeric types have the following syntax:

```
%[argument_index$][flags][width][.precision]conversion
```

- Any thing in [] are optional

# More about Format Specifiers

- argument_index: a decimal integer,  indicating the position (starting from 1) of the argument in the argument list, e.g., 1$, 2$
- flags: a set of characters that modify the output format. The set of valid flags depends on the conversion.
- width: a positive decimal integer, indicating the minimum number of characters to be written to the output.
- precision: a non-negative decimal integer usually used to restrict the number of characters. The specific behavior depends on the conversion.
- conversion: required is a character indicating how the argument should be formatted. The set of valid conversions for a given argument depends on the argument's data type.

# Conversion

- General - may be applied to any argument type
- Character - may be applied to basic types which represent Unicode characters: char, Character, byte, Byte, short, and Short. This conversion may also be applied to the types int and Integer when Character.isValidCodePoint(int) returns true
- Numeric
  - Integral - may be applied to Java integral types: byte, Byte, short, Short, int and Integer, long, Long, and BigInteger (but not char or Character)
  - Floating Point - may be applied to Java floating-point types: float, Float, double, Double, and BigDecimal
- Date/Time - may be applied to Java types which are capable of encoding a date or time: long, Long, Calendar, Date and TemporalAccessor
- Percent - produces a literal '%' ('\u0025')
- Line Separator - produces the platform-specific line separator

# Several Frequently Used Conversions

| Conversion | Argument Category | Description |
|---|---|---|
| 'b', 'B' | general | If the argument *arg* is null, then the result is "false". If *arg* is a boolean or Boolean, then the result is the string returned by String.valueOf(arg). Otherwise, the result is "true". |
| 's', 'S' | general | If the argument *arg* is null, then the result is "null". If *arg* implements Formattable, then arg.formatTo is invoked. Otherwise, the result is obtained by invoking arg.toString(). |
| 'c', 'C' | character | The result is a Unicode character |
| 'd' | integral | The result is formatted as a decimal integer |
| 'f' | floating point | The result is formatted as a decimal number |
| 't', 'T' | date/time | Prefix for date and time conversion characters. |

# Flags

*y* means the flag is supported for the indicated argument types.

| Flag | General | Character | Integral | Floating Point | Date / Time | Description |
|---|---|---|---|---|---|---|
| '-' | y | y | y | y | y | The result will be left-justified. |
| '#' | y | - | y | y | - | The result should use a conversion-dependent alternate form |
| '+' | - | - | y | y | - | The result will always include a sign |
| ' ' | - | - | y | y | - | The result will include a leading space for positive values |
| '0' | - | - | y | y | - | The result will be zero-padded |
| '(' | - | - | y | y | - | The result will enclose negative numbers in parentheses |

# Example Format Specifiers

Recall: `%[argument_index$][flags][width][.precision]conversion`

Examples:

%1$-10d

%2$(10d

%10.3f

%-10.3f

%010.3f

%(010d

%20s

%1c

# Using Format Strings

- Consider two methods
  - System.out.printf(fmt_string, argument_list) method
  - String.format(fmt_string, argument_list)

# Using Format Strings: Example

- Print out a well formatted table with table header like

| Name | Level | GPA |
|---|---|---|
| Jane Doe | Freshman | 3.456 |

```
String headFmt = "%20s %10s %5s";

String dataFmt = "%20s %10s %5.3f";

System.out.printf(headFmt+"\n", "Name", "Level", "GPA");

String dataRow = String.format(dataFmt, "Jane Doe", "Freshman", gpa);

System.out.println(dataRow);
```

# Reference for Format String

- Reference: https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html

# Questions