22. Formatted Output

Java

Fall 2009

Instructor: Dr. Masoud Yaghini

Outline

- Formatting Output with printf
- Printing Integers
- Printing Floating-Point Numbers
- Printing Strings and Characters
- Printing with Field Widths and Precisions
- Using Flags in the printf Format String
- References

Formatting Output with printf

Introduction

- Method printf
  - Formats and outputs data to the standard output stream, System.out
  - Can perform
    - rounding
    - aligning columns
    - right/left justification
    - inserting literal characters
    - exponential format
    - fixed width and precision
    - date and time format
  - Java borrowed this feature from the C programming language

Introduction

- The **printf** method has the form

  printf(format-string, argument-list);

  - **Format String**
    - Describe the output format
    - Consist of fixed text and format specifier
    - Fixed text is output by printf just as it would be output by System.out methods print or println.
  - **Argument List**
    - contains the values that correspond to each format specifier in format-string.

Introduction

- **Format specifier**
  - Placeholder for a value
  - Specify the type of data to output
  - Begins with a percent sign (%) and is followed by a conversion character (such as: s or d)
    - e.g., %s, is a placeholder for a string value
    - e.g., %d, is a placeholder for an **int** value
  - Optional formatting information
    - Argument index, flags, field width, precision
    - Specified between % and conversion character

Printing Integers

Printing Integers

- Integer
  - Whole number (no decimal point): `25`, `0`, `-9`
  - Positive, negative, or zero
  - Only minus sign prints by default (later we shall change this)
- Example:
  - IntegerConversionTest.java
- Output:

```
26
26
-26
```

Printing Floating-Point Numbers

Printing Floating-Point Numbers

- Floating Point Numbers
  - Have a decimal point (33.5, 0.0 or -657.983)
- Conversion character:
  - **e or E**
    - Display a floating-point value in exponential notation.
    - `150.4582` is `1.504582 x 10²` in scientific
    - `150.4582` is `1.504582e+02` in exponential (**e** stands for exponent)
    - When conversion character **E** is used, the output is displayed in uppercase letters.
  - **f**
    - Display a floating-point value in decimal format.

Printing Floating-Point Numbers

- Conversion character: (cont.)
  - **g or G**
    - Display a floating-point value in either the floating-point format **f** or the exponential format **e** based on the magnitude of the value.
    - If the magnitude is less than 10^{-3}, or greater than or equal to 10^{7}, the floating-point value is printed with **e** (or **E**).
      - **Otherwise**, the value is printed in format **f**.
    - When conversion character **G** is used, the output is displayed in uppercase letters.

Printing Floating-Point Numbers

- Example:
  - FloatingNumberTest.java
- Output:

```
1.234568e+07
1.234568e+07
-1.234568e+07
1.234568E+07
12345678.900000
1.23457e+07
1.23457E+07
```

Printing Strings and Characters

Printing Strings and Characters

- Conversion character:
  - `c` and `C`
    - Require `char`
    - `C` displays the output in uppercase letters
  - `s` and `S`
    - `String`
    - `Object, i`mplicitly use object's `toString` method
    - `S` displays the output in uppercase letters

Printing Strings and Characters

- Example:
  - CharStringConversion.java
- Output:

```
a
A
This is a string
This is also a string
THIS IS ALSO A STRING
```

Printing with Field Widths and Precisions

Printing with Field Widths and Precisions

- Field width
  - Size of field in which data is printed
  - If width larger than data, default right justified
    - If field width too small, increases to fit data
    - Minus sign uses one character position in field
  - Integer width inserted between **%** and conversion specifier
    - e.g., **%4d** – field width of 4
  - Positive field width –> right justified
  - Can be used with all format specifiers except the line separator (**%n**)

Printing with Field Widths and Precisions

- Example:
  - FieldWidthTest.java
- Output:

```
    1
   12
  123
 1234
12345

   -1
  -12
 -123
-1234
-12345
```

Printing with Field Widths and Precisions

- Precision
  - Meaning varies depending on data type
  - Floating point
    - Number of digits to appear after decimal (**e** or **E** and **f**)
    - Maximum number of significant digits (**g** or **G**)
  - Strings
    - Maximum number of characters to be written from string
- Format
  - Use a dot (**.**) then precision number after **%**
    - e.g., **%.3f**

Printing with Field Widths and Precisions

- Field width and precision
  - Can both be specified
    - **%** width **.** Precision, e.g. **%5.3f**
  - Precision must be positive
    - Example: **printf(“%9.3f", 123.456789);**

Printing with Field Widths and Precisions

- Example:
  - PrecisionTest.java
- Output:

```
Using precision for floating-point numbers
   123.945
   1.239e+02
   124

Using precision for strings
   Happy Birth
```

Using Flags in the printf Format String

Using Flags in the printf Format String

- Flags
  - Supplement formatting capabilities
  - Place flag immediately to the right of percent sign
  - Several flags may be combined

Right justifying and left justifying values

- **-** (minus sign) Flag:
  - Left justify the output within the specified field.
- Example:
  - MinusFlagTest.java
- Output:

```
Columns:
0123456789012345678901234567890123456789
     hello         7         a  1.230000
hello     7         a         1.230000
```

Printing numbers with and without the + flag

- + (plus sign) Flag:
  - Display a plus sign preceding positive values and a minus sign preceding negative values.
- Example:
  - PlusFlagTest.java
- Output:

```
786    -786
+786   -786
```

Using the space flag

- **space Flag:**
  - Print a space before a positive value not printed with the + flag.
- Example:
  - SpaceFlagTest.java
- Output:

```
 547
-547
```

Printing with the 0 (zero) flag

- **0 (zero) Flag:**
  - Filling a field with leading zeros.
- Example:
  - ZeroFlagTest.java
- Output:

```
+00000452
000000452
      452
```

Using the comma (,) flag

- **, (comma) Flag:**
  - Use the locale-specific thousands separator (i.e., ',' for U.S. locale) to display decimal and floating-point numbers.
- Example:
  - CommaFlagTest.java
- Output:

```
58,625
58,625.21
12,345,678.90
```

Using the (flag

- **(Flags:**
  - Enclose negative numbers in parentheses.
- Example:
  - ParenthesesFlagTest.java
- Output:

```
50
(50)
(5.0e+01)
```

References

References

- H. M. Deitel and P. J. Deitel, **Java™ How to Program**, Sixth Edition, Prentice Hall, 2005. (Chapter 28)

The End