29. Formatted Output

Java

Summer 2008

Instructor: Dr. Masoud Yaghini

Outline

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

Introduction

Introduction

- Method `printf`
  - Formats and outputs data to the standard output stream, `System.out`
- Class `Formatter`
  - Formats and outputs data to a specified destination
    - E.g., a string or a file output stream

Introduction

- `printf`
  - Precise output formatting
    - Conversion specifications: flags, field widths, precisions, etc.
  - Can perform
    - rounding
    - aligning columns
    - right/left justification
    - inserting literal characters
    - exponential format
    - octal and hexadecimal 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
    - E.g., `%s`, is a placeholder for a string value
    - %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)

Formatted Output

```java
package chapter28;

// IntegerConversionTest.java
// Using the integral conversion characters.

public class IntegerConversionTest
{
   public static void main( String args[] )
   {
      System.out.printf( "%d\n", 26 );
      System.out.printf( "%d\n", +26 );
      System.out.printf( "%d\n", -26 );
   } // end main
} // end class IntegerConversionTest
```

- Program 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.

Formatted Output

```java
package chapter28;

// FloatingNumberTest.java
// Using floating-point conversion characters.

public class FloatingNumberTest
{
   public static void main( String args[] )
   {
      System.out.printf( "%e\n", 12345678.9 );
      System.out.printf( "%e\n", +12345678.9 );
      System.out.printf( "%e\n", -12345678.9 );
      System.out.printf( "%E\n", 12345678.9 );
      System.out.printf( "%f\n", 12345678.9 );
      System.out.printf( "%g\n", 12345678.9 );
      System.out.printf( "%G\n", 12345678.9 );
   } // end main
} // end class FloatingNumberTest
```

- Program 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`
      - Implicitly use object's `toString` method
    - `S` displays the output in uppercase letters

Common Programming Error

- Using %c to print a string causes an IllegalFormatConversionException—a string cannot be converted to a character.

Formatted Output

```java
package chapter28;

// CharStringConversion.java
// Using character and string conversion characters.

public class CharStringConversion
{
   public static void main( String args[] )
   {
      char character = 'a'; // initialize char
      String string = "This is also a string"; // String object

      System.out.printf( "%c\n", character );
      System.out.printf( "%C\n", character );
      System.out.printf( "%s\n", "This is a string" );
      System.out.printf( "%s\n", string );
      System.out.printf( "%S\n", string );
   } // end main
} // end class CharStringConversion
```

- Program 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
  - Can be used with all format specifiers except the line separator (%n)

Formatted Output

```java
package chapter28;

// FieldWidthTest.java
// Right justifying integers in fields.

public class FieldWidthTest
{
   public static void main( String args[] )
   {
      System.out.printf( "%4d\n", 1 );
      System.out.printf( "%4d\n", 12 );
      System.out.printf( "%4d\n", 123 );
      System.out.printf( "%4d\n", 1234 );
      System.out.printf( "%4d\n\n", 12345 ); // data too large

      System.out.printf( "%4d\n", -1 );
      System.out.printf( "%4d\n", -12 );
      System.out.printf( "%4d\n", -123 );
      System.out.printf( "%4d\n", -1234 ); // data too large
      System.out.printf( "%4d\n", -12345 ); // data too large
   } // end main
} // end class RightJustifyTest
```

- Program 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
      - %5.3f
  - Negative field width – left justified
  - Positive field width – right justified
  - Precision must be positive
    - Example:
      - printf("%9.3f", 123.456789);

Formatted Output

```java
package chapter28;

// PrecisionTest.java
// Using precision for floating-point numbers and strings.
public class PrecisionTest
{
   public static void main( String args[] )
   {
      double f = 123.94536;
      String s = "Happy Birthday";

      System.out.printf( "Using precision for floating-point numbers\n" );
      System.out.printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );

      System.out.printf( "Using precision for strings\n" );
      System.out.printf( "\t%.11s\n", s );
   } // end main
} // end class PrecisionTest
```

- Program 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.

Formatted Output

```java
package chapter28;

// MinusFlagTest.java
// Right justifying and left justifying values

public class MinusFlagTest
{
   public static void main( String args[] )
   {
      System.out.println( "Columns:" );
      System.out.println( "0123456789012345678901234567890123456789" );
      System.out.printf( "%10s%10d%10c%10f\n", "hello", 7, 'a', 1.23 );
      System.out.printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );
   } // end main
} // end class MinusFlagTest
```

- Program 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.

Formatted Output

```java
package chapter28;

// PlusFlagTest.java
// Printing numbers with and without the + flag.

public class PlusFlagTest
{
   public static void main( String args[] )
   {
      System.out.printf( "%d\t%d\n", 786, -786 );
      System.out.printf( "%+d\t%+d\n", 786, -786 );
   } // end main
} // end class PlusFlagTest
```

- Program output:

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

Using the space flag

- space Flag:
  - Print a space before a positive value not printed with the + flag.

Formatted Output

```java
package chapter28;

// SpaceFlagTest.java
// Printing a space before non-negative values.

public class SpaceFlagTest
{
   public static void main( String args[] )
   {
      System.out.printf( "% d\n% d\n", 547, -547 );
   } // end main
} // end class SpaceFlagTest
```

- Program output:

```
547
-547
```

Printing with the 0 (zero) flag

- 0 (zero) Flag:
  - Filling a field with leading zeros.

Formatted Output

```java
package chapter28;

// ZeroFlagTest.java
// Printing with the 0 (zero) flag fills in leading zeros.

public class ZeroFlagTest
{
   public static void main( String args[] )
   {
      System.out.printf( "%+09d\n", 452 );
      System.out.printf( "%09d\n", 452 );
      System.out.printf( "% 9d\n", 452 );
   } // end main
} // end class ZeroFlagTest

```

- Program 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.

Formatted Output

```java
package chapter28;

// CommaFlagTest.java
// Using the comma (,) flag to display numbers with thousands separator.

public class CommaFlagTest
{
   public static void main( String args[] )
   {
      System.out.printf( "%,d\n", 58625 );
      System.out.printf( "%,.2f\n", 58625.21 );
      System.out.printf( "%,.2f", 12345678.9 );
   } // end main
} // end class CommaFlagTest
```

- Program output:

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

Using the (flag

- (Flags:
  - Enclose negative numbers in parentheses.

Formatted Output

```java
package chapter28;

// ParenthesesFlagTest.java
// Using the ( flag to place parentheses around negative numbers.

public class ParenthesesFlagTest
{
   public static void main( String args[] )
   {
      System.out.printf( "%(d\n", 50 );
      System.out.printf( "%(d\n", -50 );
      System.out.printf( "%(.1e\n", -50.0 );
   } // end main
} // end class parenthesesFlagTest
```

- Program 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