ICS 141 Programming with Objects

Jessica Maistrovich

Metropolitan State University

Comments

```
// Single-Line Comment

/*
* Multiple-Line comment
*/

/**
* JavaDoc comment
*/
```

JavaDoc Tags

TAG	PARAMETER	DESCRIPTION
@author	author_name	Describes an author
@param	desciption	provide information about method parameter or the input it takes
@see	reference	generate a link to other element of the document
@version	version-name	provide version of the class, interface or enum.
@return	desciption	provide the return value

https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

Format of a Doc Comment

A doc comment is written in HTML and must precede a class, field, constructor or method declaration. It is made up of two parts -- a description followed by block tags. In this example, the block tags are `@param`, `@return`, and `@see`.

```
/**
* Returns an Image object that can then be painted on the screen.
* The url argument must specify an absolute <a href="#{@link}">{@link URL}</a>. The name
* argument is a specifier that is relative to the url argument.
* <p>
* This method always returns immediately, whether or not the
* image exists. When this applet attempts to draw the image on
* the screen, the data will be loaded. The graphics primitives
* that draw the image will incrementally paint on the screen.
*
* @param  url  an absolute URL giving the base location of the image
* @param  name the location of the image, relative to the url argument
* @return      the image at the specified URL
* @see         Image
*/
public Image getImage(URL url, String name) {
try {
return getImage(new URL(url, name));
} catch (MalformedURLException e) {
return null;
}
}
```

https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

getImage

```
public Image getImage(URL url,
                      String name)
```

Returns an `Image` object that can then be painted on the screen. The `url` argument must specify an absolute URL. The `name` argument is a specifier that is relative to the `url` argument.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

`url` - an absolute URL giving the base location of the image.

`name` - the location of the image, relative to the `url` argument.

Returns:

the image at the specified URL.

See Also:

`Image`

https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

- **Add description beyond the API name.**

The best API names are "self-documenting", meaning they tell you basically what the API does. If the doc comment merely repeats the API name in sentence form, it is not providing more information. For example, if method description uses only the words that appear in the method name, then it is adding nothing at all to what you could infer. The ideal comment goes beyond those words and should always reward you with some bit of information that was not immediately obvious from the API name.

Avoid - The description below says nothing beyond what you know from reading the method name. The words "set", "tool", "tip", and "text" are simply repeated in a sentence.

```
/**
* Sets the tool tip text.
*
* @param text  the text of the tool tip
*/
public void setToolTipText(String text) {
```

Preferred - This description more completely defines what a tool tip is, in the larger context of registering and being displayed in response to the cursor.

```
/**
* Registers the text to display in a tool tip.   The text
* displays when the cursor lingers over the component.
*
* @param text  the string to display.  If the text is null,
*              the tool tip is turned off for this component.
*/
public void setToolTipText(String text) {
```

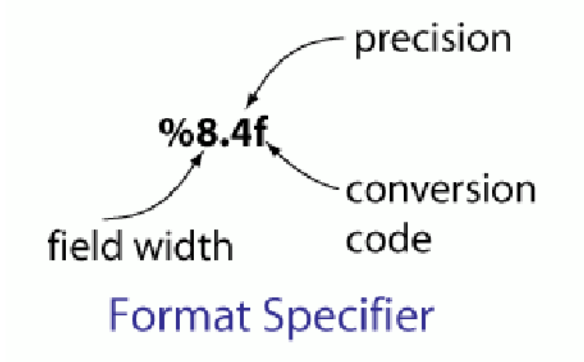
Try it!

- Add JavaDoc comments to your existing projects.

Formatting output

Formatting output

- Use DecimalFormat:
  - DecimalFormat numform = new DecimalFormat("0.00");
  - numform.format(<number want to format>);
- Use System.out.printf():
  - printf(formatString, value1, value2, value3, ...)



Conversion Code	Type	Example Output
b	boolean	true
c	character	B
d	integer (output as decimal)	221
o	integer (output as octal)	335
x	integer (output as hex)	dd
f	floating point	45.356
e	floating point (scientific notation)	-8.756e+05
s	String	Hello World
n	newline	

DecimalFormat

https://docs.oracle.com/javase/7/docs/api/java/text/DecimalFormat.html

DecimalFormat numberFormater = new DecimalFormat(Format string – "0.00“);

numberFormater.format(number want to format);

value of double	format pattern	output string
123.456	"000.000"	"123.456"
123.456	"000.0"	"123.5"
123.456	"000"	"123"
89.008	"000.000"	"089.008"
89.008	"0000.0000"	"0089.0080"
89.008	"0.00"	"89.01"
89.008	"0."	"89."

Patterns

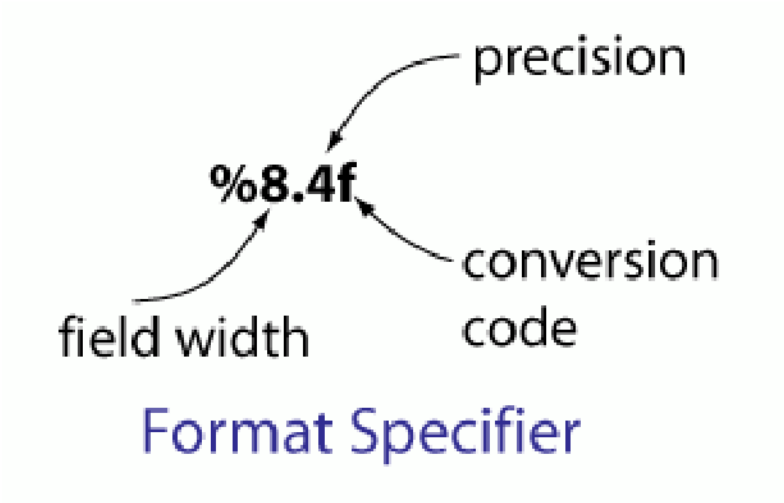
Value	Pattern	Output	Explanation
123456.789	###,###.###	123,456.789	The pound sign (#) denotes a digit, the comma is a placeholder for the grouping separator, and the period is a placeholder for the decimal separator.
123456.789	###.##	123456.79	The value has three digits to the right of the decimal point, but the pattern has only two. The format method handles this by rounding up.

Patterns

Value	Pattern	Output	Explanation
123.78	000000.000	000123.780	The pattern specifies leading and trailing zeros, because the 0 character is used instead of the pound sign (#).
12345.67	$###,###.###	$12,345.67	The first character in the pattern is the dollar sign ($). Note that it immediately precedes the leftmost digit in the formatted output.
12345.67	\ u00A5###,###.###	¥12,345.67	The pattern specifies the currency sign for Japanese yen (¥) with the Unicode value 00A5.

Print format aka printf()

System.out.printf(String with one or more format specifier, value1, value2, ...)



Conversion Code	Type	Example Output
b	boolean	true
c	character	B
d	integer (output as decimal)	221
o	integer (output as octal)	335
x	integer (output as hex)	dd
f	floating point	45.356
e	floating point (scientific notation)	-8.756e+05
s	String	Hello World
n	newline	

```
public class ConvCodes
{
  public static void main ( String[] args )
  {
    boolean r = true;
    char b = 'B';
    int adrs = 221;
    long date = 1666;
    double x = -875612.0014;
    String fire = "Great Fire:";

    System.out.printf("Roses are Red:%10b%n", r);
    System.out.printf("Answer to Question 1:%5c%n", b);
    System.out.printf("Sherlock Lived at:%3d%c%n", adrs, b);
    System.out.printf("%s %d%n", fire, date);
    System.out.printf("Usual format: %8.3f Scientific format: %8.3e%n", x, x);
  }
}
```

Conversion Code	Type
b	boolean
c	character
d	integer (output as decimal)
o	integer (output as octal)
x	integer (output as hex)
f	floating point
e	floating point (scientific notation)
s	String
n	newline

Here is the output:

```
Roses are Red:      true
Answer to Question 1:    B
Sherlock Lived at:221B
Great Fire: 1666
Usual format: -875612.001 Scientific format: -8.756e+05
```