Commenting in Java

Katie Fraser
WIT program LWTA
Fall 2013

Java Code Conventions

- Java programs have two types of comments:
  - Documentation comments, delimited by **...*\
  - Implementation comments, delimited by * ...*\ or \\
- Doc comments describe the specification of the code, and can be read and understood independently of the source code
  - Often read by a ***user*** of the class
- Implementation comments explain particular details of the implementation
  - Often read by a ***developer*** modifying the class

Doc Comments and Javadoc

Terminology

- **API docs or API specs** - On-line or hardcopy descriptions of the API, intended primarily for programmers writing in Java.
- **Doc comments** - The special comments in the Java source code that are delimited by the /**...*/ delimiters. These comments are processed by the Javadoc tool to generate the API docs.
- **Javadoc** - The JDK tool that generates API documentation from documentation comments.

Javadoc

- Javadoc is a tool for generating API documentation from doc comments
- In order to work properly, Javadoc requires comments to be written in a particular format
- Doc comments are written in HTML and immediately precede a class or method declaration
- Doc comments have 2 parts:
  - Description
  - Block tags

Example

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. 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;
        }
 }
```

Example

Description

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. 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;
        }
}
```

Example

Description

Block tags

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. 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;
        }
}
```

Example after running Javadoc tool:

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`

Javadoc Description

- First sentence should be a summary sentence
- Javadoc tool copies this description to the class summary, so it should be concise and informative
- Particularly want to distinguish overloaded methods from each other
- Description should be as implementation-independent as possible

Javadoc Tags

- Javadoc tool parses special tags when they are embedded within a Java doc comment
- Tags start with “@” symbols
- Tags are case-sensitive
- Tags must start at the beginning of a line (after any asterisks and whitespace)
- Tags with the same name are grouped together

Common Javadoc Tags

- @author – author(s) of the class
- @version – software version number
- @param – the parameters of the method
- @throws – exceptions thrown by the method
- @return – values returned by the method

A full list of tags is given at the Javadoc Reference Page.

Order of Tags

- @author (classes and interfaces only, required)
- @version (classes and interfaces only, required)
- @param (methods and constructors only)
- @return (methods only)
- @exception (or @throws in Javadoc 1.2 onwards)
- @see
- @since
- @serial (or @serialField or @serialData)
- @deprecated

Style Guide

- Use <code> … </code> for Java keywords and names (class names, method names, etc.)
- Use in-line links using the `@link` tag
  - But: use links sparingly, as they make comments more difficult to read
- Omit parentheses for the general form of methods and constructors
  - e.g. `add(int,Object)` and `add(Object)`
  - Refer to the method in general as `add` rather than `add()`

Style Guide

- Use 3rd person rather than 2nd person
  - e.g. “Gets the label” rather than “Get the label”
- Begin method descriptions with a verb phrase
  - e.g. “Gets the label” rather than “This method gets the label”
- Begin class, interface, and field descriptions by simply stating what the object is.
  - e.g. “A button label” rather than “This field is a label”
- Use “this” rather than “the” when when referring to an object created from the current class.

Style Guide

- The description should provide additional information beyond what can be inferred from the API name.
- e.g. `public void setToolTipText(String text)`

```
/**
    * Sets the tool tip text.
    *
    * @param text  the text of the tool tip
    */
```

- Better:

```
/**
    * 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.
    */
```

Running Javadoc

- From the command line:

```
javadoc *.java
```

- In Eclipse:
  - Create a template by typing `/** <newline>`
  - To run Javadoc, choose the package or file for which you want to generate the documentation
  - Choose Project > Generate Javadoc
  - Choose source and destination files
  - Specify any extra options, then choose Finish

Running Javadoc

- In Netbeans:
  - To create Javadoc comments:
    - Right-click class file → Tools → Auto Comment
  - To generate Javadoc documentation:
    - Right-click class file → Tools → Generate Javadoc
  - For more information and a graphical tutorial, see: http://edu.netbeans.org/quicktour/javadoc.html

Implementation Comments

Implementation Comments

- Add comments to help others (and yourself) understand how the code works
- Describe what the code does, but also ***why*** it is doing what it does
- Four main styles:
  - Block
  - Single-line
  - Trailing
  - End-of-line

Implementation Comments

- Block comments can be used to provide a more detailed description of a piece of code.

```
/*
   * Here is a block comment. Block comments
   * can span several lines.
   */
```

- Single-line comments are used for short comments

```
/* A short, single-line comment. */
```

- In both cases, the comments should be indented to the same level as the code, and should be preceded by a blank line.

Implementation Comments

- Trailing comments can be used for very short comments

```
if (a == 2) {
      return TRUE;          /* special case */
  } else {
      return isPrime(a);    /* works only for odd a */
  }
```

- End-of-line comments can also be used for short comments, or to “comment out” lines

```
return false;  //not prime
```

- Note: For commenting out large sections of code, use your IDE's “comment-out” command instead

Sources

Information and examples were taken from the following sources:

- “How To Write Doc Comments for the Javadoc Tool”
  http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#styleguide
- Javadoc Reference Page
  http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html
- Java Language Specification, First Edition
  http://docs.oracle.com/javase/specs/#25995
- Code Conventions for the Java Programming Language
  http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html