ST-1

Special Topics No. 1: JavaDoc Documentation Tool

In this document, we provide basic information on the standard Java API documentation tool called *javadoc*. Using the javadoc tool on the source code embedded with javadoc-style comments, we can generate the HTML-based documentation of the Java source code automatically. Online documentation for the standard Java API libraries is produced with this tool; so is the documentation for the javabook and galapagos packages.

Introduction

In this document, we will describe the use of a documentation tool called javadoc and how to comment the source code so the javadoc tool can produce the HTML-based documentation. The documentation for the standard Java API can be viewed by opening the HTML file named index.html in the docs/api subdirectory (e.g., jdk1.3/docs/api). The documentation for the javabook and galapagos packages is produced by using the javadoc tool, also. To use the javadoc tool, the source file must be commented in the javadoc-style comments with the appropriate tags. We will discuss the javadoc-style comments in this document. We will call the javadoc-style comments simply *javadoc comments* or *doc comments*.

1 Javadoc Comments

A javadoc comment begins with the `/**` marker and ends with the `*/` marker. The following is a javadoc comment:

```
/**
 * This is a <b>javadoc</b> comment.
 */
```

Because javadoc generates HTML file, any valid HTML can be embedded. In this example, the word 'javadoc' appears in bold. A javadoc comment may be composed of mutliple lines, for example:

```
/**
 * This is line one.
 * This is line two.
 *
 * This is intended as a new paragraph.
 */
```

Although the lines are separated in the source file, when the javadoc tool parses a javadoc comment, leading * characters on each line are discarded and blanks and tabs preceding the initial * characters are also discarded, except for the first line. The * characters are a marker we use to make the javadoc comments more readable in the source file; they will not appear in the generated HTML documents. Since the blanks and tabs are also removed, we need to use an HTML tag <p> to mark a new paragraph as in

```
/**
 * This is paragraph one.
 *
 * <p>
 * This is paragraph two.
 *
 * <p>
 * This is the last paragraph.
 */
```

Another useful HTML marker is <code>, which we can use to include a sample code in a javadoc comment. Any text between the <code> and </code> markers will appear in a Courier font. List markers such as <li> are also useful.

The first sentence of a javadoc comment should be written as a summary of the comment. The first sentence ends at the first period that is followed by a blank, tab, line delimiter, or a javadoc tag (we will explain the javadoc tag in the next section). When the javadoc tool generates the HTML file, the first sentence will appear in the member summary section at the top of the HTML file.

For the javadoc comments to be recognized as such by the javadoc tool, they must appear immediately before the class, interface, constructor, method, or data member declarations. This is why the import statements appear at the very beginning of the source files when javadoc comments are used. If you put the javadoc comment for the class before the import statements, it will be ignored.

2 Javadoc Tags

There are a number of special tags we can embed with the javadoc comments. These tags start with the “at” symbol @. We will mention only the more common ones. For a complete list, please refer to the resources given in the last section of this document.

Javadoc tags must start at the beginning of a line. If you have more than one line with the same tag (e.g., multiple @param tags), then place the tags together. This will ensure that the javadoc tool can tell where the list ends.

@author

Use this tag to create an author entry. You can have multiple @author tags. This tag is meaningful only for the class/interface javadoc comment.

@version

Use this tag to create a version entry. A javadoc comment may contain at most one @version tag. Version normally refers to the version of the software (such as the JDK) that contains this feature.

@see

Use this tag to add a hyperlinked "See Also" entry to the class. Here are three examples:

```
@see java.lang.String
@see String
@see javabook.JavaBookDialog
```

@param

Use this tag to add a parameter description for a method. This tag contains two parts: the first is the name of the parameter and the second is the description. The decription can be more than one line. Here's an example:

```
@param size the length of the passed array
```

@return

Use this tag to add a return type description for a method. This tag is meaningful only if the method's return is non-void. Here's an example:

```
@return true if the array is empty; otherwise return
        false
```

An example

Here's a bigger example:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

/**
 * This applet accepts the user's name via a <code>TextField</code> object.
 * When the user presses the ENTER key, the applet displays
 * a personalized greeting "Nice to meet you, <user name>."
 * with <user name> replaced by the actual name entered by the user.
 *
 * @author Dr. Caffeine
 * @version 1.0
 *
 */
public class GreetingApplet extends Applet implements ActionListener
{

   /**
    * To prompt user for input
    */
   private   Label     prompt;

   /**
    * To display the personalized greeting
    */
   private   Label     greeting;

   /**
    * To accept user input
    */
   private   TextField inputLine;

   /**
    * Default constructor that creates one TextField and
    * two Label objects. This GreetingApplet is set to be
    * an ActionListener of itself.
    */
   public GreetingApplet( )
   {
      //create GUI objects
      prompt    = new Label("Please enter your name:");
      greeting  = new Label( );
      inputLine = new TextField( );

      //add GUI objects to the applet
      add( prompt    );
      add( greeting  );
      add( inputLine );

      //add this applet as an action listener
      inputLine.addActionListener( this );
   }


   /**
    * Implements the abstract method defined in the interface
    * ActionListener. The method retrieves the text from the
    * TextField object ‘inputLine’ and displays the personalized
```

```
     * greeting using the Label object ‘greeting’.
     *
     * @param event the ActionEvent object to process
     */
    public void actionPerformed( ActionEvent event )
    {
       greeting.setText( "Nice to meet you,"
                        + inputLine.getText( ) + "." );
       add( greeting );
       doLayout();
    }
}
```

3 Generating the Javadoc Documentation

After we add javadoc comments to the source files, we use the javadoc command to generate the documentation. We run the javadoc as we run javac or java tools. After the command, we enter either the package name or the soure file names. For example to javadoc the complete javabook package, we write

```
javadoc javabook
```

By default, javadoc produces a set of HTML files, one .html file for every source file, that describe the public and protected classes, interfaces, constructors, methods, and data members. To change the default, we must specify the options. For example,

```
javadoc -private javabook
```

will produce the description of all classes and their method and data members. For other options, please refer to the homepage shown in the next section.

4 For More Information

The best source of information on the javadoc tool is the javadoc home page at

```
http://java.sun.com/products/jdk/javadoc/
```