APPENDIX E

Javadoc

In this appendix, you will:

- Learn about the Javadoc documentation generator
- Understand Javadoc comment types
- Generate Javadoc documentation

The Javadoc Documentation Generator

Javadoc is a documentation generator created by Sun Microsystems that allows you to generate Application Programming Interface (API) documentation in Hypertext Markup Language (HTML) format from Java source code. In Chapter 1, you learned that you can place both line and block comments anywhere in a program to provide documentation that can be useful both to yourself and others. A **Javadoc comment** is a special form of block comment that provides a standard way to document Java code. After you write Javadoc comments, they can be interpreted by special utility programs that generate an HTML document. The resulting HTML document provides an attractive format for the documentation when you open it in a browser. Most class libraries, both commercial and open source, provide Javadoc documents. If you have visited the Java Web site to research how to use a class, you most likely have viewed documentation created by the Javadoc utility.

In Chapter 1, you learned that block comments start with /* and end with */ and can span as many lines as necessary, and that Javadoc comments start with /** and end with */. For symmetry, many developers end their Javadoc comments with **/. By convention, asterisks start intermediate lines in a Javadoc comment. This is not required, but it helps you more easily distinguish comments from code.

Javadoc comments can contain tags. A **Javadoc tag** is a keyword within a comment that the Javadoc tool can process. Tags begin with an at-sign (@) and use a limited vocabulary of keywords. Some commonly used Javadoc tags include:

- `@author`: Describes the author of a document
- `@param`: Describes a parameter of a method or constructor
- `@return`: Describes the return type of a method
- `@throws`: Describes an exception a method may throw
- `@exception`: Describes an exception

Javadoc Comment Types

There are two types of Javadoc comments:

- Class-level comments that provide a description of a class
- Member-level comments that describe the purposes of class members

Class-level Javadoc comments provide a description of a class; you place class-level comments above the code that declares a class. Class-level comments frequently contain author tags and a description of the class. Figure E-1 shows a shaded class-level comment in a class.

```
/**
 * @author Joyce Farrell.
 * The Employee class contains data about one employee.
 * Fields include an ID number and an hourly pay rate.
 */
public class Employee
{
   private int idNum;
   private double hourlyPay;
   public Employee(int id, double pay)
   {
      idNum = id;
      hourlyPay = pay;
   }
   int getIdNum()
   {
      return idNum;
   }
   void setIdNum(int id)
   {
      idNum = id;
   }
}
```

Figure E-1 An `Employee` class with class-level comments

Member-level Javadoc comments describe the fields, methods, and constructors of a class. Method and constructor comments may contain tags that describe the parameters, and method comments may also contain return tags. Figure E-2 shows a class with some shaded member-level comments.

```
/**
 * @author Joyce Farrell.
 * The Employee2 class contains data about one employee.
 * Fields include an ID number and an hourly pay rate.
 */
public class Employee2
{
   /**
    * Employee ID number
    */
   private int idNum;
   /**
    * Employee hourly pay
    */
```

Figure E-2 An `Employee2` class with class-level and member-level comments *(continues)*

(*continued*)

```
   private double hourlyPay;
   /**
    * Sole constructor for Employee2
    */
   public Employee2(int id, double pay)
   {
      idNum = id;
      hourlyPay = pay;
   }
   /**
    * Returns the Employee2 ID number
    *
    * @return int
    */
   int getIdNum()
   {
      return idNum;
   }
   /**
    * Sets the Employee2 ID number
    *
    * @param id employee ID number
    */
   void setIdNum(int id)
   {
      idNum = id;
   }
}
```

Figure E-2 An `Employee2` class with class-level and member-level comments

Like all program comments, Javadoc comments *can* contain anything. However, you should follow the conventions for Javadoc comments. For example, developers expect all Javadoc comments to begin with an uppercase letter, and they recommend that method comments start with a verb such as "Returns" or "Sets." For more information, go to the Java Web site.

Generating Javadoc Documentation

To generate the Javadoc documentation from your class, you should do the following:

1. Create a folder in which to store your class. For example, you might store the Employee2.java file in a folder named Employee2.
2. Within the folder, you can create a Documents subfolder to hold the documentation that you generate. However, if you omit this step and use the syntax described in Step 3, the folder is created for you automatically.

3. Go to the command prompt, and navigate to the directory that holds the Employee2.java file. (See Appendix A for information on finding the command prompt and changing directories.) From the command prompt, run the following command:

   ```
   javadoc -d Documents *.java
   ```

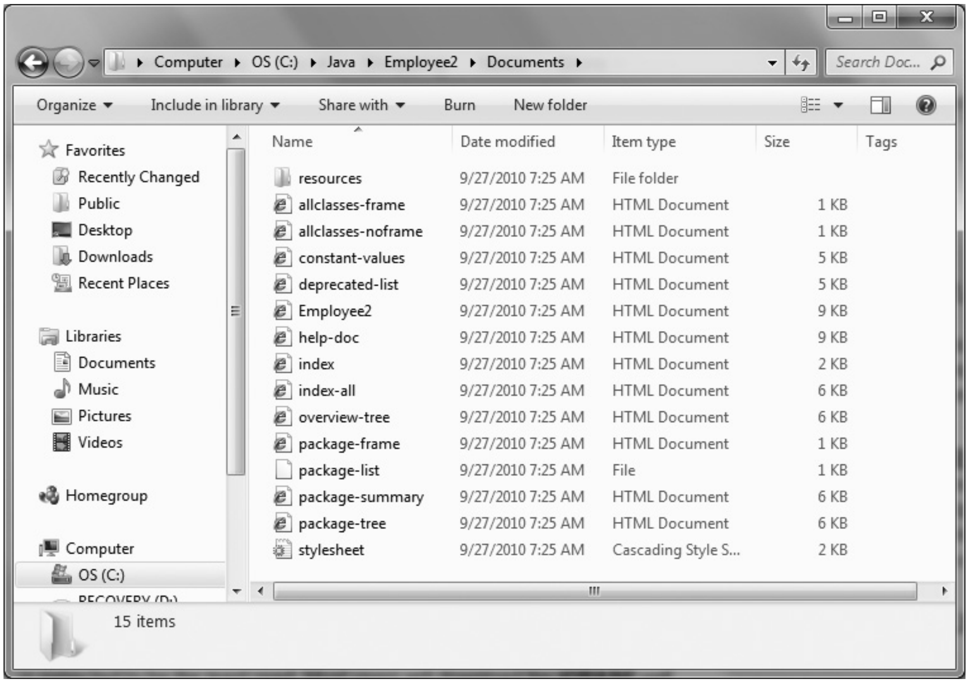
   The `-d` is the directory option. If you omit it, all the generated files are saved in the current directory. By including this option, you indicate that the files should be saved in the Documents directory.

To see the author's name in the resulting documentation, change the Javadoc command to the following:

```
javadoc -d Documents -author *.java
```

If you are using the jGRASP development environment to create your Java programs, you can execute the Javadoc command with a button click. You can download the jGRASP program from *http://jGRASP.org*.

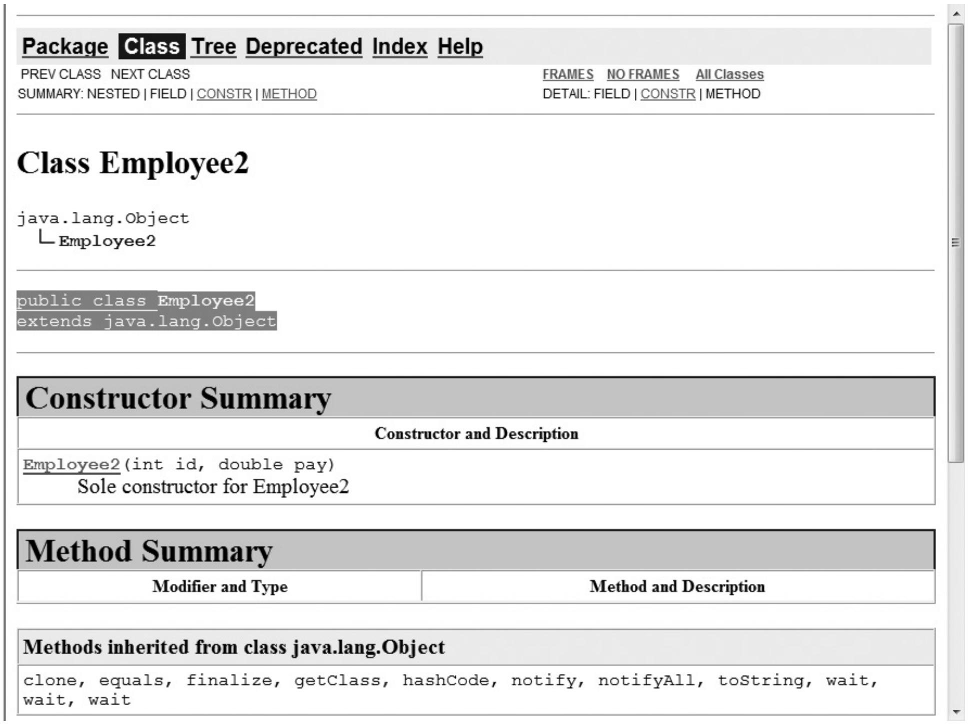
4. Navigate to the Documents folder. You will see a number of generated files, as shown in Figure E-3. The list includes HTML documents with information about all the constants in your class, all the deprecated methods in your class, and so on. (The `Employee2` class has no constants or deprecated methods, but you can open the files and view the format that the contents would take if they existed.)



Name	Date modified	Item type	Size	Tags
resources	9/27/2010 7:25 AM	File folder		
allclasses-frame	9/27/2010 7:25 AM	HTML Document	1 KB	
allclasses-noframe	9/27/2010 7:25 AM	HTML Document	1 KB	
constant-values	9/27/2010 7:25 AM	HTML Document	5 KB	
deprecated-list	9/27/2010 7:25 AM	HTML Document	5 KB	
Employee2	9/27/2010 7:25 AM	HTML Document	9 KB	
help-doc	9/27/2010 7:25 AM	HTML Document	9 KB	
index	9/27/2010 7:25 AM	HTML Document	2 KB	
index-all	9/27/2010 7:25 AM	HTML Document	6 KB	
overview-tree	9/27/2010 7:25 AM	HTML Document	6 KB	
package-frame	9/27/2010 7:25 AM	HTML Document	1 KB	
package-list	9/27/2010 7:25 AM	File	1 KB	
package-summary	9/27/2010 7:25 AM	HTML Document	6 KB	
package-tree	9/27/2010 7:25 AM	HTML Document	6 KB	
stylesheet	9/27/2010 7:25 AM	Cascading Style S...	2 KB	

Figure E-3 Contents of the `Employee2` Documents folder in Internet Explorer

The index.html file provides an index of all class interface, constructor, field, and method names; when you double-click it, the file opens in your default browser. Figure E-4 shows how the first part of the index.html file for `Employee2` appears in Internet Explorer. If you have searched the Java Web site for documentation, the format of the page in Figure E-4 is familiar to you. The class name and other information appear in a font and style consistent with other classes in the Java API. You can see information about the class constructor and the notes that you added in your comments. You see inheritance information—`Employee2` descends directly from `Object`. The format of this documentation is familiar to users, making it much easier for them to find what they need than if each developer created documentation formats independently.


Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
SUMMARY: NESTED | FIELD | CONSTR | METHOD
FRAMES NO FRAMES All Classes
DETAIL: FIELD | CONSTR | METHOD

Class Employee2

```
java.lang.Object
  └Employee2
```

```
public class Employee2
extends java.lang.Object
```

Constructor Summary

Constructor and Description
`Employee2(int id, double pay)` Sole constructor for Employee2

Method Summary

Modifier and Type	Method and Description

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait,
wait, wait
```

Figure E-4 The `Employee2` class documentation in Internet Explorer

The Javadoc tool will run on .java source files that are stub files with no method bodies. This means you can write documentation comments and run the Javadoc tool when you are first designing classes, before you have written implementations for the class's methods.

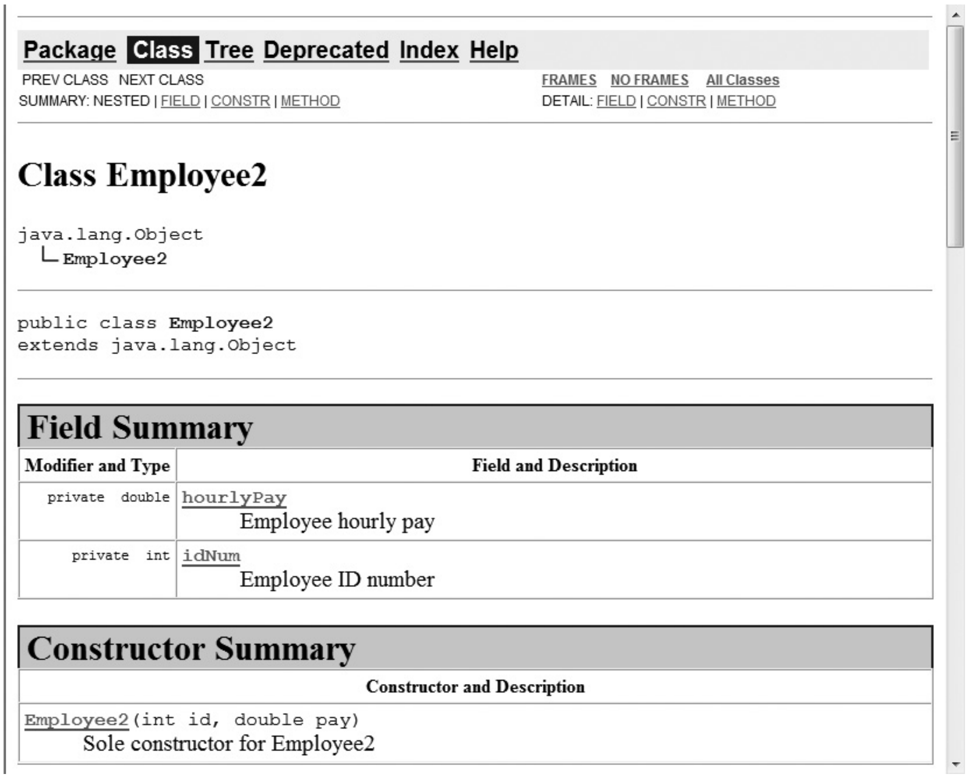
Writing acceptable Javadoc comments requires adherence to some style standards. For example, professionals recommend that multiple @author tags should be listed in chronological order, with the creator of the class listed at the top, and that multiple @param tags should be listed in argument-declaration order. Additionally, Javadoc comments can provide hyperlinks that allow navigation from one document to another. For example, when a class contains a field that is an object of another class, you might want to link to the other class's documentation. For more information, see the recommendations from Java developers at the Java Web site.

Specifying Visibility of Javadoc Documentation

By default, Javadoc documents only `public` and `protected` members of an API. In other words, even if you write Javadoc comments for `private` members, the comments do not appear in the generated documentation unless you take special action to make them visible. Although the index.html file contains details about the `Employee2` class's constructor and methods, there is no information about the `private` fields `idNum` and `hourlyPay`. To generate that documentation, you must specify `private` visibility by using the following `javadoc` command:

```
javadoc -d Documents -private *.java
```

Figure E-5 shows the documentation generated by this command. You can see that the newly generated documentation includes a Field Summary section. It lists the fields in alphabetical order preceded by their access specifiers and data types. Each field identifier is followed by the appropriate description that was provided in the Javadoc comment in the source code.


Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

Class Employee2

```
java.lang.Object
  └Employee2
```

```
public class Employee2
extends java.lang.Object
```

Field Summary

Modifier and Type	Field and Description
private double	hourlyPay Employee hourly pay
private int	idNum Employee ID number

Constructor Summary

Constructor and Description
Employee2(int id, double pay) Sole constructor for Employee2

Figure E-5 The `Employee2` class documentation when `private` members are included

You can specify four types of visibility:

- `public`—Displays `public` members only
- `protected`—Displays `public` and `protected` members only; this is the default option
- `package`—Displays package classes and members in addition to `public` and `protected` members
- `private`—Displays all members

Key Terms

Javadoc is a documentation generator created by Sun Microsystems that allows you to generate Application Programming Interface (API) documentation in Hypertext Markup Language (HTML) format from Java source code.

A **Javadoc comment** is a special form of block comment that provides a standard way to document Java code.

A **Javadoc tag** is a keyword within a comment that the Javadoc tool can process.

Class-level Javadoc comments provide a description of a class; you place class-level comments above the code that declares a class.

Member-level Javadoc comments describe the fields, methods, and constructors of a class.