APPENDIX

E Using the javadoc Utility

The Java JDK comes with a utility named `javadoc`, which you can use to automatically generate documentation for the classes, interfaces, and methods that you write. The `javadoc` utility produces HTML documentation with the same structure as the Java API documentation. To generate this documentation, there are two general steps necessary:

1. Write documentation comments in the Java source code file that you wish to document.
2. Run the `javadoc` utility, passing the name of the Java source code file as a command-line argument.

Let's take a closer look at each step.

Writing Documentation Comments

In the first step, you write into a Java source file special comments known as *documentation comments*. You typically do this as you are writing the Java code. A documentation comment may precede a class header, an interface header, or a method header. Each one contains information about the class, interface, or method that it precedes.

A documentation comment begins with `/**` and ends with `*/`. Between these two symbols you write information that the `javadoc` utility can use to generate documentation. This information can contain regular text descriptions, as well as special `javadoc` tags which begin with the `@` symbol. When you run the `javadoc` utility, it searches for these documentation comments and processes them.

To see examples of documentation comments, we will look at the `BankAccount` class that was first presented in Chapter 6. This class uses simple documentation comments for the class and its methods. First, here is the documentation for the class, followed by the class header:

```
/**
  The BankAccount class simulates a bank account.
*/

public class BankAccount
```

The text in a documentation comment for a class should contain a brief description of the class. This is the text that will be used to describe the class in the HTML file. Notice in this example, the description inside the documentation comment is indented with spaces. Any whitespace characters at the beginning of a line are ignored by the `javadoc` utility. In addition to whitespace characters, `javadoc` ignores all asterisk characters at the beginning of a line. For example, the following documentation comment will produce the same results as the previous one:

```
/**
 *
 * The BankAccount class simulates a bank account.
 *
 */
public class BankAccount
```

Although the extra asterisks are ignored by `javadoc`, they help the comment to visually stand out for any person reading the code.

A documentation comment for a method is structured in the following way:

- After the `/**` symbol, a brief description of the method appears. This text will be used to describe the method in the HTML documentation file. In addition, the first sentence of this description will be used as a summary of the method.
- If the method has parameters, documentation for each parameter will appear in the comment. A parameter's documentation begins with the `@param` tag, followed by the name of the parameter, followed by a description of the parameter.
- If the method returns a value, a description of the return value will appear in the comment. This description must begin with the `@return` tag, followed by a description of the return value.
- If the method throws exceptions, documentation for each exception will appear in the comment. Documentation for each exception begins with the `@exception` tag, followed by the name of the exception, followed by a description of the events that cause the exception.

For example, here is one of the `BankAccount` constructors, preceded by its documentation comment:

```
/**
 * The constructor initializes the balance
 * and interestRate fields.
 * @param startBalance The starting balance
 * @param intRate The interest rate
 */
public BankAccount(double startBalance,
                   double intRate)
{
   balance = startBalance;
   interestRate = intRate;
   interest = 0.0;
}
```

The documentation comment for this constructor is relatively simple, containing only a brief one-sentence description and two `@param` tags.

In the HTML file that is produced by `javadoc`, each method will have two major sections of documentation: a *summary section* and a *detail section*. The first sentence in the method's documentation comment is used as the summary of the method. Note `javadoc` considers the end of the sentence as a period followed by a whitespace character. For this reason, when a method description contains more than one sentence, you should always end the first sentence with a period followed by a whitespace character. The method's detail section will contain all of the text from the beginning of the comment to the first tag, or the end of the comment if it contains no tags.

The `@param` Tag

When the `javadoc` utility sees an `@param` tag inside of a method's documentation comments, it knows that the documentation for a parameter variable appears next. Each parameter should have its own `@param` tag, and follow this general format:

```
@param parameterName Description
```

In the general format, *`parameterName`* is the name of the parameter, and *`Description`* is a brief description of the parameter. When a method's documentation comments contain one or more `@param` tags, the `javadoc` utility will create a Parameters section in the method's documentation. This is where the descriptions of the method's parameters will be listed. Remember the following points about `@param` tag comments:

- All `@param` tags in a method's documentation comment must appear after the general description of the method.
- The description can span several lines. It ends at the end of the documentation comment (the `*/` symbol), or at the beginning of another tag.

The `@return` Tag

Here is another method from the `BankAccount` class, which has a return value:

```
/**
 * The getBalance method returns the balance.
 * @return The value in the balance field.
 */
public double getBalance()
{
   return balance;
}
```

Because this method returns a value, the documentation comment contains an `@return` tag. The documentation for the return value follows this general format:

```
@return  Description
```

In the general format, `Description` is a brief description of the return value. When a method's documentation comments contain an `@return` tag, the `javadoc` utility will create a Returns section in the method's documentation. This is where the description of the method's return value will be listed. Remember the following points about `@return` tag comments:

- The `@return` tag in a method's documentation comment must appear after the general description of the method.
- The description can span several lines. It ends at the end of the documentation comment (the `*/` symbol), or at the beginning of another tag.

The `@exception` and `@throws` Tags

The `@exception` and `@throws` *tags* both do the same thing: specify an exception that is thrown by a method. Here are the general formats of both tags:

```
@exception  ExceptionName Description
@throws  ExceptionName Description
```

`ExceptionName` is the name of an exception and `Description` is a description of the circumstances that cause the exception. When a method's documentation comments contains an `@exception` tag or a `@throws` tag, the `javadoc` utility will create a Throws section in the method's documentation. This is where the description of the method's return value will be listed. Remember the following points about `@exception` and `@throws` tag comments:

- An `@exception` tag or `@throws` tag in a method's documentation comment must appear after the general description of the method.
- The description can span several lines. It ends at the end of the documentation comment (the `*/` symbol), or at the beginning of another tag.

For example, the following code shows a method named `FileRead`, which throws an `IOException` when it fails to read a file:

```
/**
 * This method opens a file and reads its contents.
 * @param FileName Specifies the file to read.
 * @exception IOException Failed to read the file.
 */
public void FileRead(String filename) throws IOException
{
   // Body of the method appears here...
}
```

Running `javadoc`

After creating one or more source code files containing documentation comments, your next step is to run the `javadoc` utility. You run `javadoc` from the operating system command prompt. Here is the general format of the `javadoc` command:

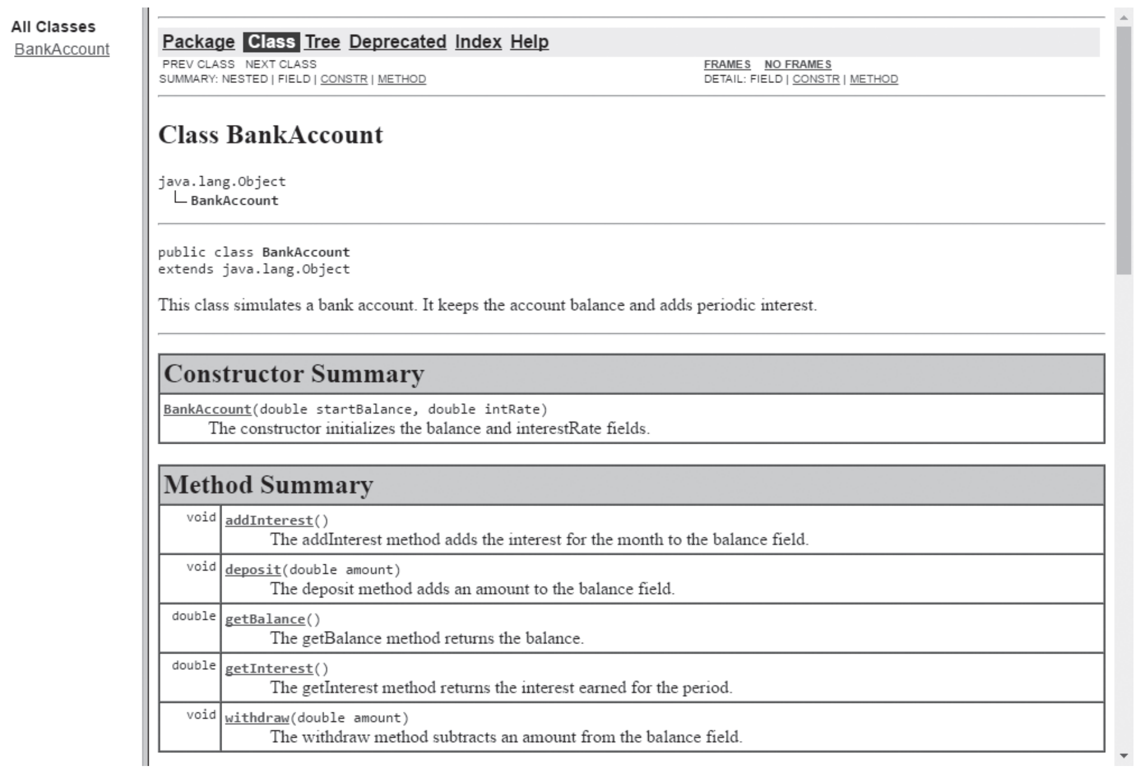
```
javadoc  SourceFileList
```

`SourceFileList` is one or more names of source code files. The files specified as arguments will be read by `javadoc` and documentation will be produced for each of them. For example, the following command will produce documentation for the `BankAccount` class:

```
javadoc BankAccount.java
```

After this command executes, several HTML files will be created in the same directory as the source code file. One of these files will have the same name as the class file. In this case, it will be named *BankAccount.html*. This is the file that you open in your browser to view the documentation. Figure E-1 shows the documentation for the `BankAccount` class.

Figure E-1 `BankAccount` class documentation (Oracle Corporate Counsel)



All Classes
BankAccount

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

Class BankAccount

```
java.lang.Object
  └ BankAccount
```

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

This class simulates a bank account. It keeps the account balance and adds periodic interest.

Constructor Summary
`BankAccount(double startBalance, double intRate)` The constructor initializes the balance and interestRate fields.

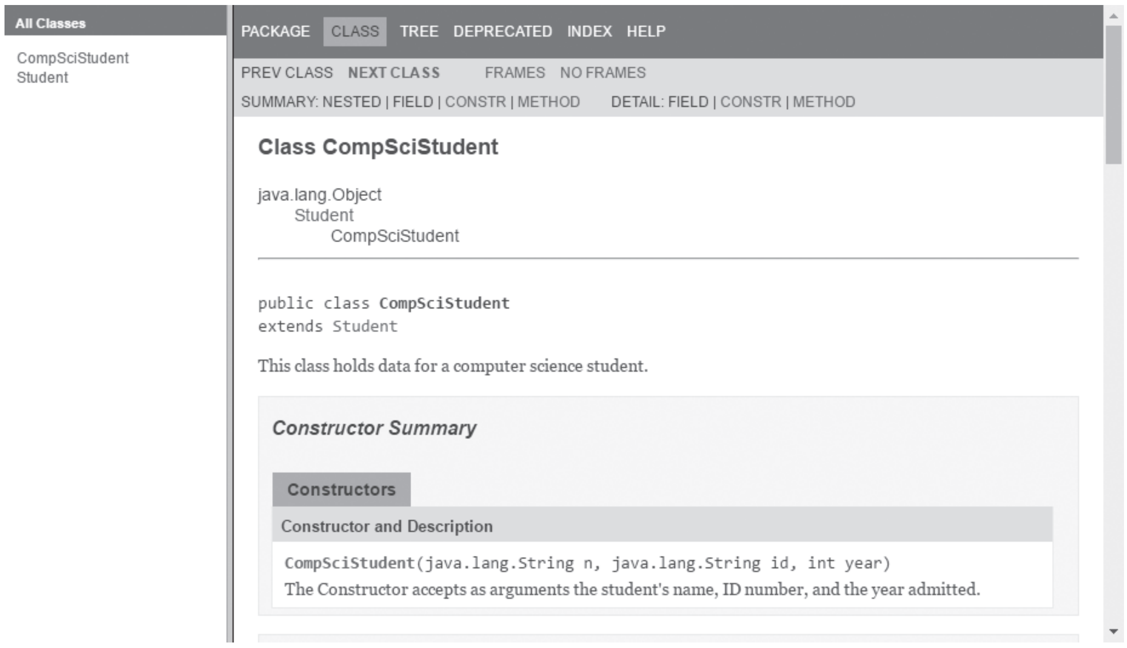
Method Summary	
`void`	`addInterest()` The addInterest method adds the interest for the month to the balance field.
`void`	`deposit(double amount)` The deposit method adds an amount to the balance field.
`double`	`getBalance()` The getBalance method returns the balance.
`double`	`getInterest()` The getInterest method returns the interest earned for the period.
`void`	`withdraw(double amount)` The withdraw method subtracts an amount from the balance field.

If you want to produce documentation for multiple source code files, simply separate the names of the files with spaces. Here is an example:

```
javadoc Student.java CompSciStudent.java
```

After this command executes, the resulting HTML files will contain documentation for the `Student` and `CompSciStudent` classes. One of the HTML files will be named *index.html*. You can open this file in your browser to view the documentation for each class. Figure E-2 shows an example. This document displays a frame on the left side of the screen containing links to each class's documentation.

Figure E-2 A documentation file for multiple classes (Oracle Corporate Counsel)


All Classes
CompSciStudent
Student
PACKAGE CLASS TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class CompSciStudent

java.lang.Object
    Student
        CompSciStudent

```
public class CompSciStudent
extends Student
```

This class holds data for a computer science student.

Constructor Summary

Constructors

Constructor and Description
`CompSciStudent(java.lang.String n, java.lang.String id, int year)` The Constructor accepts as arguments the student's name, ID number, and the year admitted.

The `@author` and `@version` Tags

The `@author` tag may be used in a class's documentation comment to identify the author or authors of the class. The general format of the tag is:

```
@author  AuthorName
```

If there is more than one author, you use a separate tag for each author.

The `@version` tag may also be used in a class's documentation comment to identify the version of the class. The general format of the tag is:

```
@version  VersionNumber
```

Here is an example of the `BankAccount` class's documentation comment, modified to use these tags:

```
/**
 * The BankAccount class simulates a bank account.
 * @author Herbert Dorfmann
 * @version 1.0
 */

public class BankAccount
```

When you use these tags, you must also provide the `–author` and `–version` options on the command line when invoking the `javadoc` utility. Here is an example:

```
javadoc –author –version BankAccount.java
```

This will cause the resulting HTML documentation to contain the author name(s) and the version number. If you do not provide these command line options, javadoc will ignore the @author and @version tags.

Embedding HTML in Documentation Comments

You can embed HTML tags in documentation comments and javadoc will include those tags when it generates the documentation files. For example, the following code shows a method with the and tags embedded. The tag causes its enclosed text to be displayed in code font, and the tag causes its enclosed text to be displayed in italics.

```
/**
 * The  addInterest  method adds the interest
 * for the  month  to the  balance  field.
 */

public void addInterest()
{
   interest = balance * interestRate;
   balance = balance + interest;
}
```

The -public, -private, and -protected Command Line Options

By default, the javadoc utility generates documentation only for public and protected classes, interfaces, and members. You can alter this, however, by using a command line option.

The -public option causes javadoc to generate documentation only for public classes, interfaces, and members. Here is an example:

```
javadoc -public Class1.java Class2.java Class3.java
```

If any of the classes or any of the class members are not public, javadoc will not generate documentation for them.

The -private option causes javadoc to generate documentation for all classes, interfaces, and members. Here is an example:

```
javadoc -private Class1.java Class2.java Class3.java
```

This command will generate documentation for all the classes and their members, regardless of their access specification.

The -protected option causes javadoc to generate documentation only for public and protected classes, interfaces, and members. This is the same as javadoc's default behavior. Here is an example:

```
javadoc -protected Class1.java Class2.java Class3.java
```