JAVADOC

To use an API effectively, you need good documentation for it. The documentation for most Java APIs is prepared using a system called Javadoc. For example, this system is used to prepare the documentation for Java's standard packages. And almost everyone who creates a toolbox in Java publishes Javadoc documentation for it.

Javadoc documentation is prepared from special comments that are placed in the Java source code file. Recall that one type of Java comment begins with `/*` and ends with `*/`. A Javadoc comment takes the same form, but it begins with `/**` rather than simply `/*`. You have already seen comments of this form in some of the examples in this book, such as this subroutine from Section 4.3:

```
/**
 * This subroutine prints a 3N+1 sequence to standard output, using
 * startingValue as the initial value of N.  It also prints the number
 * of terms in the sequence. The value of the parameter, startingValue,
 * must  be a positive integer.
 */

static void print3NSequence(int startingValue) { ...
```

Note that the Javadoc comment must be placed just **before** the subroutine that it is commenting on. This rule is always followed. You can have Javadoc comments for subroutines, for member variables, and for classes. The Javadoc comment always immediately **precedes** the thing it is commenting on.

Like any comment, a Javadoc comment is ignored by the computer when the file is compiled. But there is a tool called `javadoc` that reads Java source code files,

extracts any Javadoc comments that it finds, and creates a set of Web pages containing the comments in a nicely formatted, interlinked form. By default, `javadoc` will only collect information about `public` classes, subroutines, and member variables, but it allows the option of creating documentation for non-public things as well. If `javadoc` doesn't find any Javadoc comment for something, it will construct one, but the comment will contain only basic information such as the name and type of a member variable or the name, return type, and parameter list of a subroutine. This is **syntactic** information. To add information about semantics and pragmatics, you have to write a Javadoc comment.

As an example, you can look at the documentation Web page for *TextIO* by following this link: TextIO Javadoc documentation. The documentation page was created by applying the `javadoc` tool to the source code file, *TextIO.java*. If you have downloaded the on-line version of this book, the documentation can be found in the `TextIO_Javadoc` directory.

In a Javadoc comment, the `*`'s at the start of each line are optional. The `javadoc` tool will remove them. In addition to normal text, the comment can contain certain special codes. For one thing, the comment can contain HTML mark-up commands. HTML is the language that is used to create web pages, and Javadoc comments are meant to be shown on web pages. The `javadoc` tool will copy any HTML commands in the comments to the web pages that it creates. You'll learn some basic HTML in Section 6.2, but as an example, you can add `<p>` to indicate the start of a new paragraph. (Generally, in the absence of HTML commands, blank lines and extra spaces in the comment are ignored. Furthermore, the characters & and < have special meaning in HTML and should not be used in Javadoc comments except with those meanings; they can be written as `&amp;` and `&lt;`.)

In addition to HTML commands, Javadoc comments can include doc tags, which are processed as commands by the `javadoc` tool. A doc tag has a name that begins with the character `@`. I will only discuss three tags: `@param`, `@return`, and `@throws`. These tags are used in Javadoc comments for subroutines to provide information about its parameters, its return value, and the exceptions that it might throw. These tags **must** be placed at the end of the comment, after any description of the subroutine itself. The syntax for using them is:

```
@param  parameter-name   description-of-parameter

@return  description-of-return-value

@throws  exception-class-name   description-of-exception
```

The **descriptions** can extend over several lines. The description ends at the next doc tag or at the end of the comment. You can include a `@param` tag for every parameter of the subroutine and a `@throws` for as many types of exception as you want to document. You should have a `@return` tag only for a non-void subroutine. These tags do not have to be given in any particular order.

Here is an example that doesn't do anything exciting but that does use all three types of doc tag:

```
/**
 * This subroutine computes the area of a rectangle, given its
width
 * and its height.  The length and the width should be positive
numbers.
 * @param width the length of one side of the rectangle
 * @param height the length the second side of the rectangle
 * @return the area of the rectangle
 * @throws IllegalArgumentException if either the width or the
height
```

```
     *    is a negative number.
     */
    public static double areaOfRectangle( double length, double
    width ) {
        if ( width < 0  ||  height < 0 )
           throw new IllegalArgumentException("Sides must have
    positive length.");
        double area;
        area = width * height;
        return area;
    }
```

I will use Javadoc comments for many of my examples. I encourage you to use them in your own code, even if you don't plan to generate Web page documentation of your work, since it's a standard format that other Java programmers will be familiar with.

If you do want to create Web-page documentation, you need to run the `javadoc` tool. This tool is available as a command in the Java Development Kit that was discussed in Section 2.6. You can use `javadoc`in a command line interface similarly to the way that the `javac` and `java` commands are used. Javadoc can also be applied in the Eclipse integrated development environment that was also discussed inSection 2.6: Just right-click the class, package, or entire project that you want to document in the Package Explorer, select "Export," and select "Javadoc" in the window that pops up. I won't go into any of the details here; see the documentation.

Source : http://math.hws.edu/javanotes/c4/s5.html