Javadoc specifications

A programmer who wants to use your program should be able to look at the specification of the program, its documentation, without having to look at the code itself. Java has a facility for extracting documentation from your program —provided that you have written your documentation appropriately. This requires placing *javadoc comments* before each class, field declaration, and method declaration. These comments have the form:

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

To show you how this works, look at class `Elephant` in DrJava. And here is the folder in which file Elephant.java appears. Notice that we have put an empty directory doc in it. We suggest you do this also.

Now, return to the DrJava application, and click item **Javadoc** on the right of the tool bar at the top of the DrJava window. After waiting a while, a browser window will open, which contains the javadac spec in it, in the same format as the Java API specifications. Look at that! Take a look at directory `doc`; you will see lots of files, all of which were produced by hitting the javadoc button.

Now let's look at the specification in detail.

1. Note that it has a page for class `Elephant` and that it extends `Object`. Look at the comment for the class; it is precisely the javadoc comment that appeared just before the class definition.

2. *The Field Summary part* contains a list of public fields, with the javadoc comments that were extracted from them. Note that only the first sentence is given —that is, up to the first period. If you click on the field name, you get the "field detail", and the whole extracted comment will appear there.

Let's go back to the summaries.

3. *The Constructor summaries* give information about all the constructors, again with the javadoc comments that you put on them.

4. *Here are method summaries*. Note that there is no comment for procedure `setName`. Why not? Let's go back to the program and look at the comment that is there. Aah! It is not a javadoc comment because there is only one asterisk. Javadoc comments require *two* asterisks, as in the spec of function `getName`.

As in the field summary, in the method summary, we can click on the name of a method to get to the method detail.

5. Finally, you can see methods that are inherited from superclasses, in this case, class `Object`. We can click on `equals` to get to its method summary and then on its name to get to the method detail. Hey, notice how it says that `equals` should be reflexive, symmetric, and transitive. Let's go back.

You may see a navigation window

When you click the javadoc button, a navigation window may appear, which asks you to select a directory into which the Javadoc files will be placed. We always select directory `doc` within the folder for the project. After you select the folder, DrJava creates the Javadoc files —be patient; it can take a few moments. Finally, a new browser window will appear with the javadoc spec in it.

Your duty as a Java programmer

It's your duty as a Java programmer to write javadoc specs for *all* public components. It is also your duty to click the javadoc button and to read *all* the extracted specs carefully to make sure they make sense, are precise, are thorough. If the purpose of a method call cannot be understood from the method spec, then there is something wrong with it. Here are points to consider.

Since javadoc documentation is meant mainly for users of classes, javadoc generally extracts specifications only for public classes, methods, and fields —those that the user can use. However, in the preferences, you can set a switch that forces javadoc to extract javadoc comments from private entities as well.

1. *The javadoc spec for a class* should explain briefly what an instance of the class is. For example:

```
/** An instance of the class is a shape, like a polygon. */
```

2. *The javadoc spec for a variable* should state the meaning of the variable and any constraints on it.

3. *The javadoc spec for a method* should be a precise and thorough spec, as we have explained elsewhere. If one cannot tell how to write a call on the method from that spec, then the spec is no good.