*Set* methods are certainly important from a software engineering standpoint, because they can perform validity checking. *Set* and *get* methods have another important software engineering advantage, discussed in the following *Software Engineering Observation*.

**Software Engineering Observation 8.16**

*Accessing* **`private`** *data through* set *and* get *methods not only protects the instance variables from receiving invalid values, but also insulates clients of the class from the representation of the instance variables. Thus, if the representation of the data changes (typically, to reduce the amount of storage required, improve performance or enhance the class in other ways), only the method implementations need to change—the clients need not change as long as the interface provided by the methods remains the same.*

### 8.8.1 Executing an Applet that Uses Programmer-Defined Packages

After compiling the classes in Fig. 8.8 and Fig. 8.9, you can execute the applet from a command window with the command

```
appletviewer TimeTest5.html
```

As we discussed when we introduced packages earlier in this chapter, the interpreter can locate packaged classes in the current directory. The **`appletviewer`** is a Java application that executes a Java applet. Like the interpreter, the **`appletviewer`** can load standard Java classes and extension classes installed on the local computer. However, the **`appletviewer`** does not use the class path to locate classes in programmer-defined packages. For an applet, such classes should be bundled with the applet class in an archive file called a *Java Archive (JAR)* file. Remember that applets normally are downloaded from the Internet into a Web browser (see Chapter 3 for more information). Bundling the classes and packages that compose an applet enables the applet and its supporting classes to be downloaded as a unit, then executed in the browser (or via the Java Plug-in for browsers that do not support Java 2).

To bundle the classes in Fig. 8.8 and Fig. 8.9, open a command window and change directories to the location in which **`TimeTest5.class`** is stored. In that same directory should be the **`com`** directory that begins the package directory structure for class **`Time3`**. In that directory, issue the following command

```
jar cf TimeTest5.jar TimeTest5.class com\*.*
```

to create the JAR file. [*Note:* This command uses **`\`** as the directory separator from the MS-DOS prompt. UNIX would use **`/`** as the directory separator.] In the preceding command, **`jar`** is the *Java archive utility* used to create JAR files. Next are the options for the **`jar`** utility—**`cf`**. The letter **`c`** indicates that we are creating a JAR file. The letter **`f`** indicates that the next argument in the command line (**`TimeTest5.jar`**) is the name of the JAR file to create. Following the options and JAR file name are the actual files that will be included in the JAR file. We specified **`TimeTest5.class`** and **`com\*.*`**, indicating that **`TimeTest5.class`** and all the files in the **`com`** directory should be included in the JAR file. The **`com`** directory begins the package that contains the **`.class`** file for the **`Time3`**. [*Note:* You can include selected files by specifying the path and file name for each individ-

ual file.] It is important that the directory structure in the JAR file match the directory structure for the packaged classes. Therefore, we executed the **jar** command from the directory in which **com** is located.

To confirm that the files were archived directly, you can issue the command

```
jar tvf TimeTest5.jar
```

which produces the listing in Fig. 8.10. In the preceding command, the options for the **jar** utility are ***tvf***. The letter ***t*** indicates that the table of contents for the JAR should be listed. The letter ***v*** indicates that the output should be verbose (the verbose output includes the file size in bytes and the date and time each file was created, in addition to the directory structure and file name). The letter ***f*** specifies that the next argument on the command line is the JAR file to use.

The only remaining issue is to specify the archive as part of the applet's HTML file. In prior examples, **<applet>** tags had the form

```
<applet code = "ClassName.class" width = "width" height = "height">
</applet>
```

To specify that the applet classes are located in a JAR file, use an **<applet>** tag of the form:

```
<applet code = "ClassName.class" archive = "archiveList"
   width = "width" height = "height">
</applet>
```

The ***archive*** *attribute* can specify a comma-separated list of archive files for use in an applet. Each file in the **archive** list will be downloaded by the browser when it encounters the **<applet>** tags in the HTML document. For the TimeTest5 applet, the applet tag would be

```
<applet code = "TimeTest5.class" archive = "TimeTest5.jar"
   width = "400" height = "115">
</applet>
```

Try loading this applet into your Web browser. Remember that you *must* either have a browser that supports Java 2 (such as Netscape Navigator 6) or convert the HTML file for use with the Java Plug-in (as discussed in Chapter 3).

```
   0 Fri May 25 14:13:14 EDT 2001 META-INF/
  71 Fri May 25 14:13:14 EDT 2001 META-INF/MANIFEST.MF
2959 Fri May 25 13:42:32 EDT 2001 TimeTest5.class
   0 Fri May 18 17:35:18 EDT 2001 com/deitel/
   0 Fri May 18 17:35:18 EDT 2001 com/deitel/jhtp4/
   0 Fri May 18 17:35:18 EDT 2001 com/deitel/jhtp4/ch08/
1765 Fri May 18 17:35:18 EDT 2001 com/deitel/jhtp4/ch08/Time3.class
```

Fig. 8.10 Contents of **TimeTest5.jar**.