PROGRAMMING ENVIRONMENTS

ALTHOUGH THE JAVA LANGUAGE is highly standardized, the procedures for creating, compiling, and editing Java programs vary widely from one programming environment to another. There are two basic approaches: a command line environment, where the user types commands and the computer responds, and an integrated development environment (IDE), where the user uses the keyboard and mouse to interact with a graphical user interface. While there is just one common command line environment for Java programming, there is a wide variety of IDEs.

I cannot give complete or definitive information on Java programming environments in this section, but I will try to give enough information to let you compile and run the examples from this textbook, at least in a command line environment. There are many IDEs, and I can't cover them all here. I will concentrate on Eclipse, one of the most popular IDEs for Java programming, but some of the information that is presented will apply to other IDEs as well.

One thing to keep in mind is that you do not have to pay any money to do Java programming (aside from buying a computer, of course). Everything that you need can be downloaded for free on the Internet.

2.6.1 Java Development Kit

The basic development system for Java programming is usually referred to as the JDK (Java Development Kit). It is a part of Java SE, the Java "Standard Edition" (as opposed to Java for servers or for mobile devices). This book requires Java

Version 5.0 or higher. Confusingly, the JDKs that are part of Java Versions 5, 6, and 7 are sometimes referred to as JDK 1.5, 1.6, and 1.7. Note that Java SE comes in two versions, a Development Kit version (the JDK) and a Runtime Environment version (the JRE). The Runtime can be used to run Java programs and to view Java applets in Web pages, but it does not allow you to compile your own Java programs. The Development Kit includes the Runtime and adds to it the JDK which lets you compile programs. You need a JDK for use with this textbook.

Java was developed by Sun Microsystems, Inc., which is now a part of the Oracle corporation. Oracle makes the JDK for Windows and Linux available for free download at its Java Web site,http://www.oracle.com/technetwork/java. If you have a Windows computer, it might have come with a Java Runtime, but you might still need to download the JDK. Some versions of Linux come with the JDK either installed by default or on the installation media. If you need to download and install the JDK, be sure to get JDK 5.0 (or higher). As of August 2010, the current version of the JDK is **JDK 6**, and it can be downloaded from

```
http://www.oracle.com/technetwork/java/javase/downloads/index.h
tml
```

Mac OS comes with Java. Recent versions of Mac OS come with Java Version 5 or Version 6, so you will not need to download anything.

If a JDK is properly installed on your computer, you can use the command line environment to compile and run Java programs. Most IDEs also require Java to be installed, so even if you plan to use an IDE for programming, you probably still need a JDK, or at least a JRE.

2.6.2 Command Line Environment

Many modern computer users find the command line environment to be pretty alien and unintuitive. It is certainly very different from the graphical user interfaces that most people are used to. However, it takes only a little practice to learn the basics of the command line environment and to become productive using it.

To use a command line programming environment, you will have to open a window where you can type in commands. In Windows, you can open such a command window by running the program named cmd. In recent versions of Windows, it can be found in the "Accessories" submenu of the Start menu, under the name "Command Prompt". Alternatively, you can run `cmd` by using the "Run Program" feature in the Start menu, and entering "cmd" as the name of the program. In Mac OS, you want to run the Terminal program, which can be found in the Utilities folder inside the Applications folder. In Linux, there are several possibilities, including an old program called xterm. In Ubuntu Linux, you can use the "Terminal" command under "Accessories" in the "Applications" menu.

No matter what type of computer you are using, when you open a command window, it will display a prompt of some sort. Type in a command at the prompt and press return. The computer will carry out the command, displaying any output in the command window, and will then redisplay the prompt so that you can type another command. One of the central concepts in the command line environment is the current directory which contains the files to which commands that you type apply. (The words "directory" and "folder" mean the same thing.) Often, the name of the current directory is part of the command prompt. You can get a list of the files in the current directory by typing in the command dir (on Windows) or ls (on Linux and Mac OS). When the window first opens, the current directory is your home directory, where all your files are stored. You can change the current directory using the cd command with the name of the directory that you want to use. For example, to change into your Desktop directory, type in the command`cd Desktop` and press return.

You should create a directory (that is, a folder) to hold your Java work. For example, create a directory named `javawork` in your home directory. You can do this using your computer's GUI; another way to do it is to open a command window and enter the command `mkdir javawork`. When you want to work on programming, open a command window and enter the command `cd javawork` to change into your work directory. Of course, you can have more than one working directory for your Java work; you can organize your files any way you like.

The most basic commands for using Java on the command line are javac and java; `javac` is used to compile Java source code, and `java` is used to run Java stand-alone applications. If a JDK is correctly installed on your computer, it should recognize these commands when you type them in on the command line. Try typing the commands `java -version` and `javac -version` which should tell you which version of Java is installed. If you get a message such as "Command not found," then Java is not correctly installed. If the "java" command works, but "javac" does not, it means that a Java Runtime is installed rather than a Development Kit. (On Windows, after installing the JDK, you need to modify the Windows PATH variable to make this work. See the JDK installation instructions for information about how to do this.)

To test the `javac` command, place a copy of *TextIO.java* into your working directory. (If you downloaded the Web site of this book, you can find it in the directory named `source`; you can use your computer's GUI to copy-and-paste this file into your working directory. Alternatively, you can navigate to *TextIO.java* on the book's Web site and use the "Save As" command in your Web browser to save a copy of the file into your working directory.) Type the command:

```
javac  TextIO.java
```

This will compile `TextIO.java` and will create a bytecode file named `TextIO.class` in the same directory. Note that if the command succeeds, you will not get any response from the computer; it will just redisplay the command prompt to tell you it's ready for another command.

To test the `java` command, copy sample program `Interest2.java` from this book's source directory into your working directory. First, compile the program with the command

```
javac  Interest2.java
```

Remember that for this to succeed, *TextIO* must already be in the same directory. Then you can execute the program using the command

```
java  Interest2
```

Be careful to use **just the name** of the program, *Interest2*, with the `java` command, not the name of the Java source code file or the name of the compiled class file. When you give this command, the program will run. You will be asked to enter some information, and you will respond by typing your answers into the command window, pressing return at the end of the line. When the program ends, you will see the command prompt, and you can enter another command.

You can follow the same procedure to run all of the examples in the early sections of this book. When you start work with applets, you will need a different way to run the applets. That will be discussed later in the book.

To create your own programs, you will need a text editor. A text editor is a computer program that allows you to create and save documents that contain plain text. It is important that the documents be saved as plain text, that is without any special encoding or formatting information. Word processor documents are not appropriate, unless you can get your word processor to save as plain text. A good text editor can make programming a lot more pleasant. Linux comes with several text editors. On Windows, you can use notepad in a pinch, but you will probably want something better. For Mac OS, you might download the free TextWrangler application. One possibility that will work on any platform is to use jedit, a good programmer's text editor that is itself written in Java and that can be downloaded for free from www.jedit.org.

To create your own programs, you should open a command line window and `cd` into the working directory where you will store your source code files. Start up your text editor program, such as by double-clicking its icon or selecting it from a Start menu. Type your code into the editor window, or open an existing source code file that you want to modify. Save the file. Remember that the name of a Java source code file must end in ".java", and the rest of the file name must match the name of the class that is defined in the file. Once the file is saved in your working directory, go to the command window and use the`javac` command to compile it, as discussed above. If there are syntax errors in the code, they will be listed in the command window. Each error message contains the line number in the file where the computer found the error. Go back to the editor and try to fix the errors, **save your changes**, and then try the `javac` command again. (It's usually a good idea to just work on the first few errors; sometimes fixing those will make other errors go away.) Remember that when the `javac` command finally succeeds, you will get no message at all. Then you can use the `java` command to run your program, as described above. Once you've compiled the program, you can run it as many times as you like without recompiling it.

That's really all there is to it: Keep both editor and command-line window open. Edit, save, and compile until you have eliminated all the syntax errors. (Always remember to save the file before compiling it -- the compiler only sees the saved file, not the version in the editor window.) When you run the program, you might find that it has semantic errors that cause it to run incorrectly. In that case, you have to go back to the edit/save/compile loop to try to find and fix the problem.

2.6.3 IDEs and Eclipse

In an Integrated Development Environment, everything you need to create, compile, and run programs is integrated into a single package, with a graphical user interface that will be familiar to most computer users. There are many different IDEs for Java program development, ranging from fairly simple wrappers around the JDK to highly complex applications with a multitude of features. For a beginning programmer, there is a danger in using an IDE, since the difficulty of learning to use the IDE, on top of the difficulty of learning to program, can be overwhelming. However, for my own programming, I generally use the EclipseIDE, and I introduce my students to it after they have had some experience with the command line. Eclipse has a variety of features that are very useful for a beginning programmer. And even though it has many advanced features, its design makes it possible to use Eclipse without understanding its full complexity. Eclipse is used by many professional programmers and is probably the most commonly used Java IDE.

Eclipse is itself written in Java. It requires Java 1.4 or higher to run, and Java 5.0 or higher is recommended. For use with this book, you should be running Eclipse with Java 5.0 or higher. Eclipse requires a Java Runtime Environment, not necessarily a JDK. You should make sure that the JRE or JDK, Version 5.0 or higher is installed on your computer, as described above, **before** you install Eclipse. Eclipse can be downloaded for free from eclipse.org. You can download the "Eclipse IDE for Java Developers."

Another popular choice of IDE is Netbeans, which provides many of the same capabilities as Eclipse. Netbeans can be downloaded from netbeans.org, and Oracle offers downloads of Netbeans on its Java web site. I like Netbeans a little less than Eclipse, and I won't say much about it here. It is, however, quite similar to Eclipse.

The first time you start Eclipse, you will be asked to specify a workspace, which is the directory where all your work will be stored. You can accept the default name, or provide one of your own. When startup is complete, the Eclipse window will be filled by a large "Welcome" screen that includes links to extensive documentation and tutorials. You can close this screen, by clicking the "X" next to the word "Welcome"; you can get back to it later by choosing "Welcome" from the "Help" menu.

The Eclipse GUI consists of one large window that is divided into several sections. Each section contains one or more views. If there are several views in one section, then there will be tabs at the top of the section to select the view that is displayed in that section. Each view displays a different type of information. The whole set of views is called a perspective. Eclipse uses different perspectives, that is different sets of views of different types of information, for different tasks. For compiling and running programs, the only perspective that you will need is the "Java Perspective," which is the default. As you become more experiences, you might want to the use the

"Debug Perspective," which has features designed to help you find semantic errors in programs.

The Java Perspective includes a large area in the center of the window where you will create and edit your Java programs. To the left of this is the Package Explorer view, which will contain a list of your Java projects and source code files. To the right are some other views that I don't find very useful, and I suggest that you close them by clicking the small "X" next to the name of each view. Several other views that**will** be useful while you are compiling and running programs appear in a section of the window below the editing area. If you accidently close one of the important views, such as the Package Explorer, you can get it back by selecting it from the "Show View" submenu of the "Window" menu.

To do any work in Eclipse, you need a project. To start a Java project, go to the "New" submenu in the "File" menu, and select the "Java Project" command. In the window that pops up, it is only necessary to fill in a "Project Name" for the project and click the "Finish" button. The project name can be anything you like. The project should appear in the "Package Explorer" view. Click on the small triangle next to the project name to see the contents of the project. Assuming that you use the default settings, there should be a directory named "src," which is where your Java source code files will go. It also contains the "JRE System Library"; this is the collection of standard built-in classes that come with Java.

To run the *TextIO* based examples from this textbook, you must add the source code file *TextIO.java* to your project. If you have downloaded the Web site of this book, you can find a copy of *TextIO.java* in the source directory. Alternatively, you can navigate to the file on-line and use the "Save As" command of your Web browser to save a copy of the file onto your computer. The easiest way to get *TextIO* into your project is to locate the source code file on your computer and drag the file icon onto the project name in the Eclipse window. If that doesn't work, you can try using copy-and-paste: Right-click the file icon (or control-click on Mac OS), select "Copy" from the pop-up menu, right-click the project name in the Eclipse window, and select "Paste". If you also have trouble with that, you can try using the "Import" command in Eclipse's "File" menu; select "File System" (under "General") in the window that pops up, click "Next", and provide the necessary information in the next window. (Unfortunately, using the file import window is rather complicated. If you find that you have to use it, you should consult the Eclipse documentation about it.) In any case, *TextIO* should appear in the src directory of your project, inside a packagenamed "default package". Once a file is in this list, you can open it by double-clicking it; it will appear in the editing area of the Eclipse window.

To run any of the Java programs from this textbook, copy the source code file into your Eclipse Java project in the same way that you did for TextIO.java. To run the program, right-click the file name in the Package Explorer view (or control-click in Mac OS). In the menu that pops up, go to the "Run As" submenu, and select "Java Application". The program will be executed. If the program writes to standard output, the output will appear in the "Console" view, in the area of the Eclipse window below the editing area. If the program uses *TextIO* for input, you will have to type the required input into the "Console" view **-- click the "Console" view before you start typing**, so that the characters that you type will be sent to the correct part of the window. (Note that if you don't like doing I/O in the "Console" view, you can use an alternative version of *TextIO.java* that opens a separate window for I/O. You can find this "GUI" version of TextIO in a directory named `TextIO-GUI` inside this textbook's source directory.)

You can have more than one program in the same Eclipse project, or you can create additional projects to organize your work better. Remember to place a copy of *TextIO.java* in any project that requires it.

To create your own Java program, you must create a new Java class. To do this, right-click the Java project name in the "Project Explorer" view. Go to the "New" submenu of the popup menu, and select "Class". (Alternatively, there is a small icon at the top of the Eclipse window that you can click to create a new Java class.) In the window that opens, type in the name of the class, and click the "Finish" button. The class name must be a legal Java identifier. Note that you want the name of the class, not the name of the source code file, so don't add ".java" at the end of the name. The class should appear inside the "default package," and it should automatically open in the editing area so that you can start typing in your program.

Eclipse has several features that aid you as you type your code. It will underline any syntax error with a jagged red line, and in some cases will place an error marker in the left border of the edit window. If you hover the mouse cursor over the error marker or over the error itself, a description of the error will appear. Note that you do not have to get rid of every error immediately as you type; some errors will go away as you type in more of the program. If an error marker displays a small "light bulb," Eclipse is offering to try to fix the error for you. Click the light bulb to get a list of possible fixes, then double click the fix that you want to apply. For example, if you use an undeclared variable in your program, Eclipse will offer to declare it for you. You can actually use this error-correcting feature to get Eclipse to write certain types of code for you! Unfortunately, you'll find that you won't understand a lot of the proposed fixes until you learn more about the Java language, and it is **not** a good idea to apply a fix that you don't understand -- often that will just make things worse in the end.

Eclipse will also look for spelling errors in comments and will underline them with jagged red lines. Hover your mouse over the error to get a list of possible correct spellings.

Another essential Eclipse feature is content assist. Content assist can be invoked by typing Control-Space. It will offer possible completions of whatever you are typing at the moment. For example, if you type part of an identifier and hit Control-Space, you will get a list of identifiers that start with the characters that you have typed; use the up and down arrow keys to select one of the items in the list, and press Return or Enter. (Or hit Escape to dismiss the list.) If there is only one possible completion when you hit Control-Space, it will be inserted automatically. By default, Content Assist will also pop up automatically, after a short delay, when you type a period or certain other characters. For example, if you type "`TextIO.`" and pause for just a fraction of a second, you will get a list of all the subroutines in the *TextIO* class. Personally, I find this auto-activation annoying. You can disable it in the Eclipse Preferences. (Look under Java / Editor / Content Assist, and turn off the "Enable auto activation" option.) You can still call up Code Assist manually with Control-Space.

Once you have an error-free program, you can run it as described above, by right-clicking its name in the Package Explorer and using "Run As / Java Application". You can also right-click on the program itself in an editor window. If you find a problem when you run it, it's very easy to go back to the editor, make changes, and run it again. Note that using Eclipse, there is no explicit "compile" command. The source code files in your project are automatically compiled, and are re-compiled whenever you modify them.

If you use Netbeans instead of Eclipse, the procedures are similar. You still have to create new project (of type "Java Application"). You can add an existing source code file to a project by dragging the file onto the "Source Packages" folder in the project, and you can create your own classes by right-clicking the project name and selecting New/Java Class. To run a program, right-click the file that contains the main routine, and select the "Run File" command. Netbeans has a "Code Completion" feature that is similar to Eclipse's "Content Assist." One thing that you have to watch with Netbeans is that it might want to create classes in (non-default) packages; when you create a New Java Class, make sure that the "Package" input box is left blank.

2.6.4 The Problem of Packages

Every class in Java is contained in something called a package. Classes that are not explicitly put into a different package are in the "default" package. Almost all the examples in this textbook are in the default package, and I will not even discuss packages in any depth until Section 4.5. However, some IDEs might force you to pay attention to packages.

When you create a class in Eclipse, you might notice a message that says that "The use of the default package is discouraged." Although this is true, I have chosen to use it anyway, since it seems easier for beginning programmers to avoid the whole issue of packages, at least at first. Some IDEs, like Netbeans, are even less willing than Eclipse to use the default package: Netbeans inserts a package name automatically in the class creation dialog, and you have to delete that name if you want to create the class in the default package. If you do create a class in a package, the source code starts with a line that specifies which package the class is in. For example, if the class is in a package named `test.pkg`, then the first line of the source code will be

```
package test.pkg;
```

In an IDE, this will not cause any problem unless the program you are writing depends on *TextIO*. You will not be able to use *TextIO* in a program unless *TextIO* is in the same package as the program. You can put TextIO in a named, non-default package, but you have to modify the source code file *TextIO.java* to specify the package: Just add a `package` statement like the one shown above to the very beginning of the file, with the appropriate package name. (The IDE might do this for you, if you copy TextIO.java into a non-default package.) Once you've done this, the example should run in the same way as if it were in the default package.

By the way, if you use packages in a command-line environment, other complications arise. For example, if a class is in a package named `test.pkg`, then the source code file must be in a subdirectory named "pkg" inside a directory named "test" that is in turn inside your main Java working directory. Nevertheless, when you compile or execute the program, you should be in the main directory, not in a subdirectory. When you compile the source code file, you have to include the name of the directory in the command: Use "`javac test/pkg/ClassName.java`" on Linux or Mac OS, or "`javac test\pkg\ClassName.java`" on Windows. The command for executing the program is then "`java test.pkg.ClassName`", with a period separating the package name from the class name.

`Source : http://math.hws.edu/javanotes/c2/s6.html#basics.6.2`