Chapter 2

The Java Programming Environment

- Installing the Java Development Kit
- Choosing a Development Environment
- Using the Command-Line Tools
- Using an Integrated Development Environment
- Running a Graphical Application
- Building and Running Applets

In this chapter, you will learn how to install the Java Development Kit (JDK) and how to compile and run various types of programs: console programs, graphical applications, and applets. You run the JDK tools by typing commands in a shell window. However, many programmers prefer the comfort of an integrated development environment. We show you how to use a freely available development environment to compile and run Java programs. Although easier to learn, integrated development environments can be resource-hungry and tedious to use for small programs. As a middle ground, we show you how to use a text editor that can call the Java compiler and run Java programs. Once you have mastered the techniques in this chapter and picked your development tools, you are ready to move on to Chapter 3, where you will begin exploring the Java programming language.

Installing the Java Development Kit

The most complete and up-to-date versions of the Java Development Kit (JDK) are available from Sun Microsystems for Solaris, Linux, and Windows. Versions in various states of development exist for the Macintosh and many other platforms, but those versions are licensed and distributed by the vendors of those platforms.

NOTE: Some Linux distributions have prepackaged versions of the JDK. For example, on Ubuntu, you can install the JDK by simply installing the `sun-java6-jdk` package with `apt-get` or the Synaptic GUI.

Downloading the JDK

To download the Java Development Kit, you will need to navigate the Sun web site and decipher an amazing amount of jargon before you can get the software that you need. See Table 2–1 for a summary.

You already saw the abbreviation JDK for Java Development Kit. Somewhat confusingly, versions 1.2 through 1.4 of the kit were known as the Java SDK (Software Development Kit). You will still find occasional references to the old term. There is also a Java Runtime Environment (JRE) that contains the virtual machine but not the compiler. That is not what you want as a developer. It is intended for end users who have no need for the compiler.

Next, you'll see the term Java SE everywhere. That is the Java Standard Edition, in contrast to Java EE (Enterprise Edition) and Java ME (Micro Edition).

You will occasionally run into the term Java 2 that was coined in 1998 when the marketing folks at Sun felt that a fractional version number increment did not properly communicate the momentous advances of JDK 1.2. However, because they had that insight only after the release, they decided to keep the version number 1.2 for the *development kit*. Subsequent releases were numbered 1.3, 1.4, and 5.0. The *platform*, however, was renamed from Java to Java 2. Thus, we had Java 2 Standard Edition Software Development Kit Version 5.0, or J2SE SDK 5.0.

For engineers, all of this was a bit confusing, but that's why we never made it into marketing. Mercifully, in 2006, sanity prevailed. The useless Java 2 moniker was dropped and the current version of the Java Standard Edition was called Java SE 6. You will still see occasional references to versions 1.5 and 1.6—these are just synonyms for versions 5.0 and 6.

Finally, when Sun makes a minor version change to fix urgent issues, it refers to the change as an update. For example, the first update of the development kit for Java SE 6 is officially called JDK 6u1 and has the internal version number 1.6.0_01.

If you use Solaris, Linux, or Windows, point your browser to `http://java.sun.com/javase` to download the JDK. Look for version 6 or later and pick your platform. Don't worry if the software is called an "update." The update bundles contain the most current version of the whole JDK.

Sometimes, Sun makes available bundles that contain both the Java Development Kit and an integrated development environment. That integrated environment has, at different times of its life, been named Forte, Sun ONE Studio, Sun Java Studio, and Netbeans. We do not know what the eager beavers in marketing will call it when you approach the Sun web site. We suggest that you install only the Java Development Kit at this time. If you later decide to use Sun's integrated development environment, simply download it from `http://netbeans.org`.

Table 2–1 Java Jargon

Name	Acronym	Explanation
Java Development Kit	JDK	The software for programmers who want to write Java programs
Java Runtime Environment	JRE	The software for consumers who want to run Java programs
Standard Edition	SE	The Java platform for use on desktops and simple server applications
Enterprise Edition	EE	The Java platform for complex server applications
Micro Edition	ME	The Java platform for use on cell phones and other small devices
Java 2	J2	An outdated term that described Java versions from 1998 until 2006
Software Development Kit	SDK	An outdated term that described the JDK from 1998 until 2006
Update	u	Sun's term for a bug fix release
NetBeans	—	Sun's integrated development environment

After downloading the JDK, follow the platform-dependent installation directions. At the time of this writing, they are available at `http://java.sun.com/javase/6/webnotes/install/index.html`.

Only the installation and compilation instructions for Java are system dependent. Once you get Java up and running, everything else in this book should apply to you. System independence is a major benefit of Java.

NOTE: The setup procedure offers a default for the installation directory that contains the JDK version number, such as `jdk1.6.0`. This sounds like a bother, but we have come to appreciate the version number—it makes it easier to install a new JDK release for testing.

Under Windows, we strongly recommend that you do not accept a default location with spaces in the path name, such as `c:\Program Files\jdk1.6.0`. Just take out the `Program Files` part of the path name.

In this book, we refer to the installation directory as *jdk*. For example, when we refer to the *jdk*`/bin` directory, we mean the directory with a name such as `/usr/local/jdk1.6.0/bin` or `c:\jdk1.6.0\bin`.

Setting the Execution Path

After you are done installing the JDK, you need to carry out one additional step: Add the *jdk*`/bin` directory to the execution path, the list of directories that the operating system traverses to locate executable files. Directions for this step also vary among operating systems.

- In UNIX (including Solaris and Linux), the procedure for editing the execution path depends on the shell that you are using. If you use the C shell (which is the Solaris default), then add a line such as the following to the end of your `~/.cshrc` file:

  ```
  set path=(/usr/local/jdk/bin $path)
  ```

  If you use the Bourne Again shell (which is the Linux default), then add a line such as the following to the end of your `~/.bashrc` or `~/.bash_profile` file:

  ```
  export PATH=/usr/local/jdk/bin:$PATH
  ```

- Under Windows, log in as administrator. Start the Control Panel, switch to Classic View, and select the System icon. In Windows NT/2000/XP, you immediately get the system properties dialog. In Vista, you need to select Advanced System Settings (see Figure 2–1). In the system properties dialog, click the Advanced tab, then click on the Environment button. Scroll through the System Variables window until you find a variable named `Path`. Click the Edit button (see Figure 2–2). Add the *jdk*`\bin` directory to the beginning of the path, using a semicolon to separate the new entry, like this:

  ```
  c:\jdk\bin;other stuff
  ```

  Save your settings. Any new console windows that you start have the correct path.

Here is how you test whether you did it right: Start a shell window. Type the line

```
java -version
```

and press the ENTER key. You should get a display such as this one:

```
java version "1.6.0_01"
Java(TM) SE Runtime Environment (build 1.6.0_01-b06)
Java HotSpot(TM) Client VM (build 1.6.0_01-b06, mixed mode, sharing)
```

If instead you get a message such as "java: command not found" or "The name specified is not recognized as an internal or external command, operable program or batch file", then you need to go back and double-check your installation.

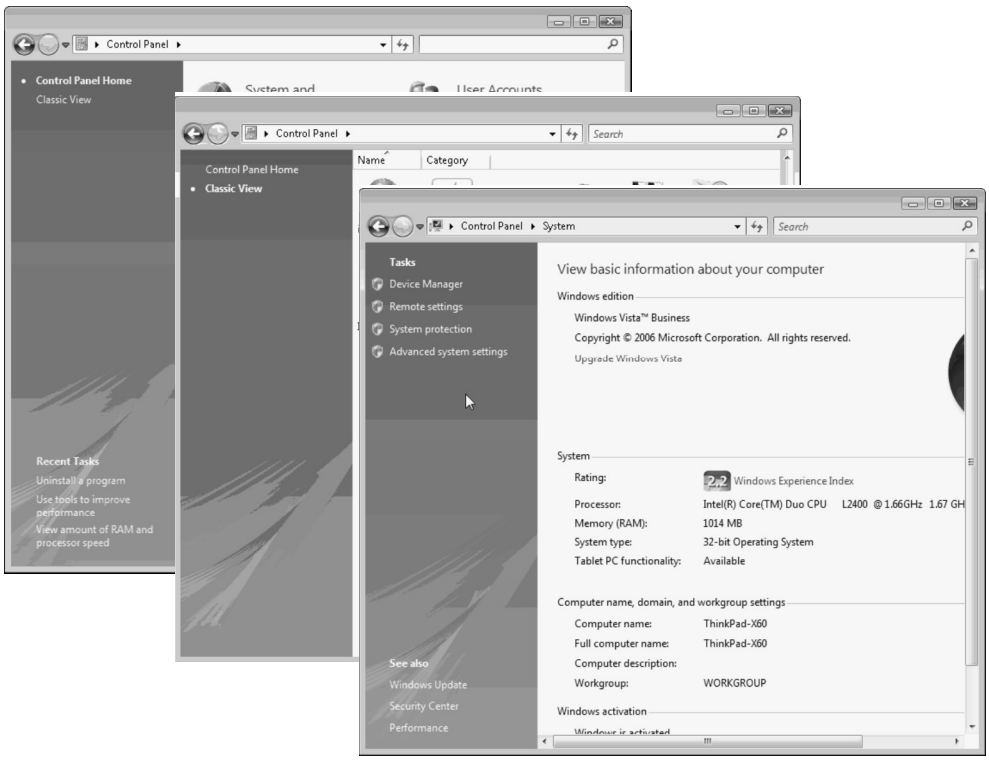

Figure 2–1 Launching the system properties dialog in Windows Vista

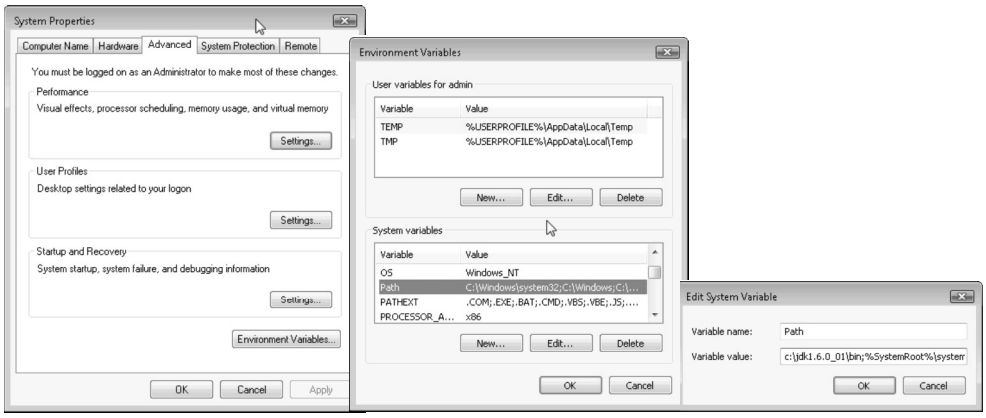

Figure 2–2 Setting the `Path` **environment variable in Windows Vista**

NOTE: In Windows, follow these instructions to open a shell window. If you use Windows NT/2000/XP, select the "Run" option from the Start menu and type `cmd`. In Vista, simply type `cmd` into the "Start Search" field in the Start menu. Press ENTER, and a shell window appears.

If you've never seen one of these, we suggest that you work through a tutorial that teaches the basics about the command line. Many computer science departments have tutorials on the Web, such as `http://www.cs.sjsu.edu/faculty/horstman/CS46A/windows/tutorial.html`.

Installing the Library Source and Documentation

The library source files are delivered in the JDK as a compressed file `src.zip`, and you must unpack that file to get access to the source code. We highly recommend that you do that. Simply do the following:

1. Make sure the JDK is installed and that the *jdk*`/bin` directory is on the execution path.
2. Open a shell window.
3. Change to the *jdk* directory (e.g., `cd /usr/local/jdk1.6.0` or `cd c:\jdk1.6.0`).
4. Make a subdirectory `src`

   ```
   mkdir src
   cd src
   ```

5. Execute the command

   ```
   jar xvf ../src.zip
   ```

   (or `jar xvf ..\src.zip` on Windows)

TIP: The `src.zip` file contains the source code for all public libraries. To obtain even more source (for the compiler, the virtual machine, the native methods, and the private helper classes), go to `http://download.java.net/jdk6`.

The documentation is contained in a compressed file that is separate from the JDK. You can download the documentation from `http://java.sun.com/javase/downloads`. Simply follow these steps:

1. Make sure the JDK is installed and that the *jdk*`/bin` directory is on the execution path.
2. Download the documentation zip file and move it into the *jdk* directory. The file is called `jdk-`*version*`-doc.zip`, where *version* is something like `6`.
3. Open a shell window.
4. Change to the *jdk* directory.
5. Execute the command

   ```
   jar xvf jdk-version-doc.zip
   ```

   where *version* is the appropriate version number.

Installing the Core Java *Program Examples*

You should also install the *Core Java* program examples. You can download them from `http://horstmann.com/corejava`. The programs are packaged into a zip file `corejava.zip`. You should unzip them into a separate directory—we recommend you call it `CoreJavaBook`. Here are the steps:

1. Make sure the JDK is installed and the *jdk*/bin directory is on the execution path.
2. Make a directory CoreJavaBook.
3. Download the corejava.zip file to that directory.
4. Open a shell window.
5. Change to the CoreJavaBook directory.
6. Execute the command

```
jar xvf corejava.zip
```

Navigating the Java Directories

In your explorations of Java, you will occasionally want to peek inside the Java source files. And, of course, you will need to work extensively with the library documentation. Table 2–2 shows the JDK directory tree.

Table 2–2 Java Directory Tree

Directory Structure	Description
jdk	(The name may be different, for example, jdk5.0)
bin	The compiler and tools
demo	Look here for demos
docs	Library documentation in HTML format (after expansion of j2sdk*version*-doc.zip)
include	Files for compiling native methods (see Volume II)
jre	Java runtime environment files
lib	Library files
src	The library source (after expanding src.zip)

The two most useful subdirectories for learning Java are docs and src. The docs directory contains the Java library documentation in HTML format. You can view it with any web browser, such as Netscape.

TIP: Set a bookmark in your browser to the file docs/api/index.html. You will be referring to this page a lot as you explore the Java platform.

The src directory contains the source code for the public part of the Java libraries. As you become more comfortable with Java, you may find yourself in situations for which this book and the on-line information do not provide what you need to know. At this point, the source code for Java is a good place to begin digging. It is reassuring to know that you can always dig into the source to find out what a library function really does. For example, if you are curious about the inner workings of the System class, you can look inside src/java/lang/System.java.

Choosing a Development Environment

If your programming experience comes from using Microsoft Visual Studio, you are accustomed to a development environment with a built-in text editor and menus to

compile and launch a program along with an integrated debugger. The basic JDK contains nothing even remotely similar. You do *everything* by typing in commands in a shell window. This sounds cumbersome, but it is nevertheless an essential skill. When you first install Java, you will want to troubleshoot your installation before you install a development environment. Moreover, by executing the basic steps yourself, you gain a better understanding of what the development environment does behind your back.

However, after you have mastered the basic steps of compiling and running Java programs, you will want to use a professional development environment. In the last decade, these environments have become so powerful and convenient that it simply doesn't make much sense to labor on without them. Two excellent choices are the freely available Eclipse and NetBeans programs. In this chapter, we show you how to get started with Eclipse since it is still a bit slicker than NetBeans, although NetBeans is catching up fast. Of course, if you prefer a different development environment, you can certainly use it with this book.

In the past, we recommended the use of a text editor such as Emacs, JEdit, or TextPad for simple programs. We no longer make this recommendation because the integrated devlopment environments are now so fast and convenient.

In sum, we think that you should know how to use the basic JDK tools, and then you should become comfortable with an integrated development environment.

Using the Command-Line Tools

Let us get started the hard way: compiling and launching a Java program from the command line.

1. Open a shell window.
2. Go to the `CoreJavaBook/v1ch02/Welcome` directory. (The `CoreJavaBook` directory is the directory into which you installed the source code for the book examples, as explained in the section "Installing the *Core Java* Program Examples" on page 20.)
3. Enter the following commands:

```
javac Welcome.java
java Welcome
```

You should see the output shown in Figure 2–3 in the shell window.

Congratulations! You have just compiled and run your first Java program.

What happened? The `javac` program is the Java compiler. It compiles the file `Welcome.java` into the file `Welcome.class`. The `java` program launches the Java virtual machine. It executes the bytecodes that the compiler placed in the class file.

NOTE: If you got an error message complaining about the line

```
for (String g : greeting)
```

then you probably use an older version of the Java compiler. Java SE 5.0 introduced a number of very desirable features to the Java programming language, and we take advantage of them in this book.

If you are using an older version of Java, you need to rewrite the loop as follows:

```
for (int i = 0; i < greeting.length; i++)
   System.out.println(greeting[i]);
```

Figure 2–3 Compiling and running `Welcome.java`

The `Welcome` program is extremely simple. It merely prints a message to the console. You may enjoy looking inside the program shown in Listing 2–1 (we explain how it works in the next chapter).

Listing 2–1 `Welcome.java`

```
1. /**
2.  * This program displays a greeting from the authors.
3.  * @version 1.20 2004-02-28
4.  * @author Cay Horstmann
5.  */
6. public class Welcome
7. {
8.    public static void main(String[] args)
9.    {
10.      String[] greeting = new String[3];
11.      greeting[0] = "Welcome to Core Java";
12.      greeting[1] = "by Cay Horstmann";
13.      greeting[2] = "and Gary Cornell";
14.
15.      for (String g : greeting)
16.         System.out.println(g);
17.   }
18. }
```

Troubleshooting Hints

In the age of visual development environments, many programmers are unfamiliar with running programs in a shell window. Any number of things can go wrong, leading to frustrating results.

Pay attention to the following points:

- If you type in the program by hand, make sure you pay attention to uppercase and lowercase letters. In particular, the class name is `Welcome` and not `welcome` or `WELCOME`.
- The compiler requires a *file name* (`Welcome.java`). When you run the program, you specify a *class name* (`Welcome`) without a `.java` or `.class` extension.
- If you get a message such as "Bad command or file name" or "javac: command not found", then go back and double-check your installation, in particular the execution path setting.
- If `javac` reports an error "cannot read: Welcome.java", then you should check whether that file is present in the directory.

  Under UNIX, check that you used the correct capitalization for `Welcome.java`. Under Windows, use the `dir` shell command, *not* the graphical Explorer tool. Some text editors (in particular Notepad) insist on adding an extension `.txt` after every file. If you use Notepad to edit `Welcome.java`, then it actually saves it as `Welcome.java.txt`. Under the default Windows settings, Explorer conspires with Notepad and hides the `.txt` extension because it belongs to a "known file type." In that case, you need to rename the file, using the `ren` shell command, or save it again, placing quotes around the file name: `"Welcome.java"`.
- If you launch your program and get an error message complaining about a `java.lang.NoClassDefFoundError`, then carefully check the name of the offending class.

  If you get a complaint about `welcome` (with a lowercase `w`), then you should reissue the `java Welcome` command with an uppercase `W`. As always, case matters in Java.

  If you get a complaint about `Welcome/java`, then you accidentally typed `java Welcome.java`. Reissue the command as `java Welcome`.
- If you typed `java Welcome` and the virtual machine can't find the `Welcome` class, then check if someone has set the `CLASSPATH` environment variable on your system. (It is usually not a good idea to set this variable globally, but some poorly written software installers in Windows do just that.) You can temporarily unset the `CLASSPATH` environment variable in the current shell window by typing

  ```
  set CLASSPATH=
  ```

  This command works on Windows and UNIX/Linux with the C shell. On UNIX/Linux with the Bourne/bash shell, use

  ```
  export CLASSPATH=
  ```
- If you get an error message about a new language construct, make sure that your compiler supports Java SE 5.0.
- If you have too many errors in your program, then all the error messages fly by very quickly. The compiler sends the error messages to the standard error stream, so it's a bit tricky to capture them if they fill more than the window can display.

  Use the `2>` shell operator to redirect the errors to a file:

  ```
  javac MyProg.java 2> errors.txt
  ```

TIP: The excellent tutorial at http://java.sun.com/docs/books/tutorial/getStarted/cupojava/ goes into much greater detail about the "gotchas" that beginners can run into.

Using an Integrated Development Environment

In this section, we show you how to compile a program with Eclipse, an integrated development environment that is freely available from http://eclipse.org. Eclipse is written in Java, but because it uses a nonstandard windowing library, it is not quite as portable as Java itself. Nevertheless, versions exist for Linux, Mac OS X, Solaris, and Windows.

There are other popular IDEs, but currently, Eclipse is the most commonly used. Here are the steps to get started:

1. After starting Eclipse, select File -> New Project from the menu.
2. Select "Java Project" from the wizard dialog (see Figure 2–4). These screen shots were taken with Eclipse 3.2. Don't worry if your version of Eclipse looks slightly different.

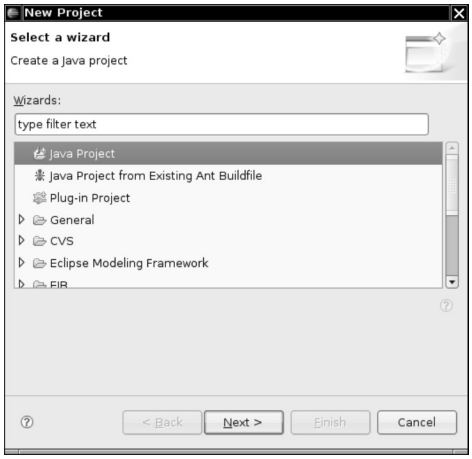

Figure 2–4 New Project dialog in Eclipse

3. Click the "Next" button. Supply the project name "Welcome" and type in the full path name of the directory that contains Welcome.java (see Figure 2–5).
4. Be sure to *uncheck* the option labeled "Create project in workspace".
5. Click the "Finish" button. The project is now created.

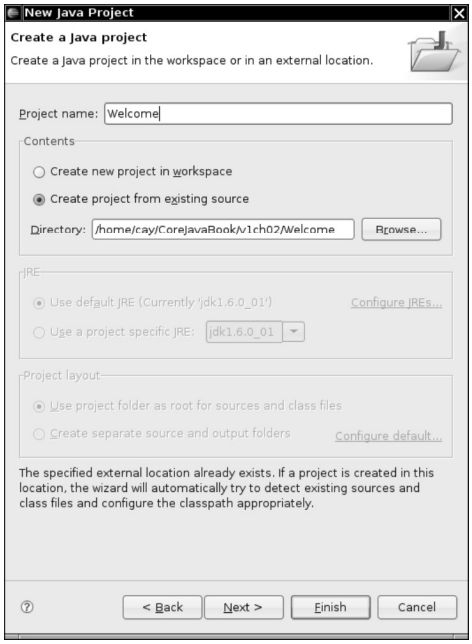

Figure 2–5 Configuring an Eclipse project

6. Click on the triangle in the left pane next to the project window to open it, and then click on the triangle next to "Default package". Double-click on `Welcome.java`. You should now see a window with the program code (see Figure 2–6).

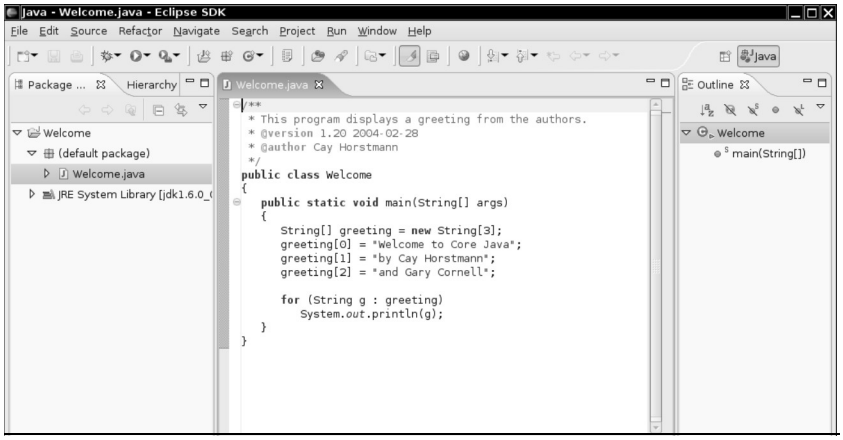

Figure 2–6 Editing a source file with Eclipse

7. With the right mouse button, click on the project name (Welcome) in the leftmost pane. Select Run -> Run As -> Java Application. An output window appears at the bottom of the window. The program output is displayed in the output window (see Figure 2–7).

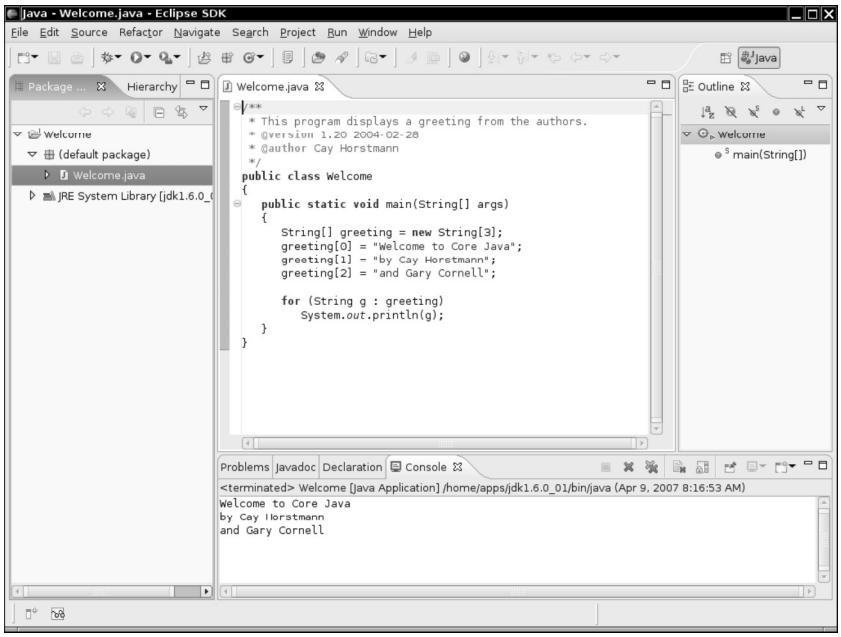

Figure 2–7 Running a program in Eclipse

Locating Compilation Errors

Presumably, this program did not have typos or bugs. (It was only a few lines of code, after all.) Let us suppose, for the sake of argument, that your code occasionally contains a typo (perhaps even a syntax error). Try it out—ruin our file, for example, by changing the capitalization of `String` as follows:

```
public static void main(string[] args)
```

Now, run the compiler again. You will get an error message that complains about an unknown `string` type (see Figure 2–8). Simply click on the error message. The cursor moves to the matching line in the edit window, where you can correct your error. This behavior allows you to fix your errors quickly.

TIP: Often, an Eclipse error report is accompanied by a lightbulb icon. Click on the lightbulb to get a list of suggested fixes.

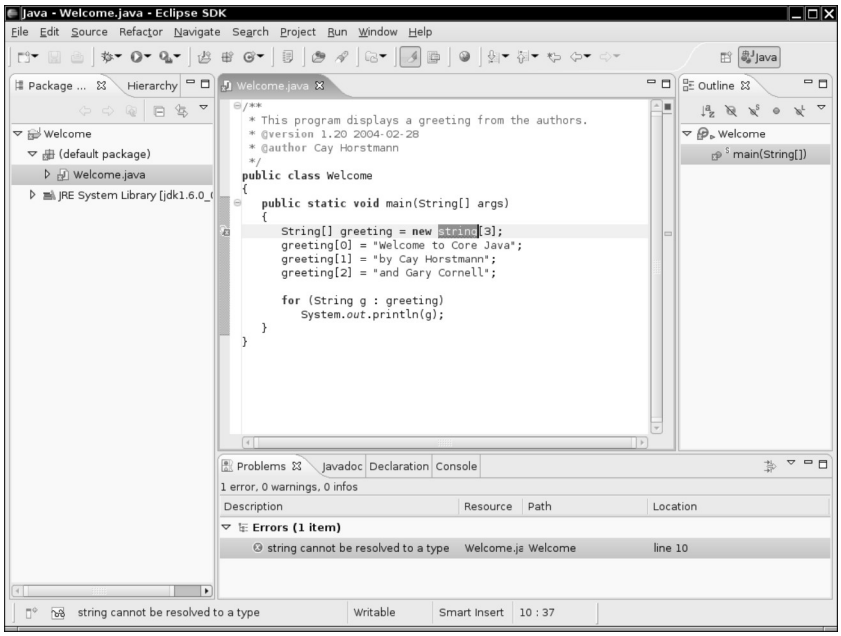

Figure 2–8 Error messages in Eclipse

These instructions should give you a taste of working in an integrated environment. We discuss the Eclipse debugger in Chapter 11.

Running a Graphical Application

The `Welcome` program was not terribly exciting. Next, we will demonstrate a graphical application. This program is a simple image file viewer that just loads and displays an image. Again, let us first compile and run it from the command line.

1. Open a shell window.
2. Change to the directory `CoreJavaBook/v1ch02/ImageViewer`.
3. Enter the following:

```
javac ImageViewer.java
java ImageViewer
```

A new program window pops up with our ImageViewer application (see Figure 2–9).

Now, select File -> Open and look for an image file to open. (We supplied a couple of sample files in the same directory.)

To close the program, click on the Close box in the title bar or pull down the system menu and close the program. (To compile and run this program inside a text editor or an integrated development environment, do the same as before. For example, for Emacs, choose JDE -> Compile, then choose JDE -> Run App.)

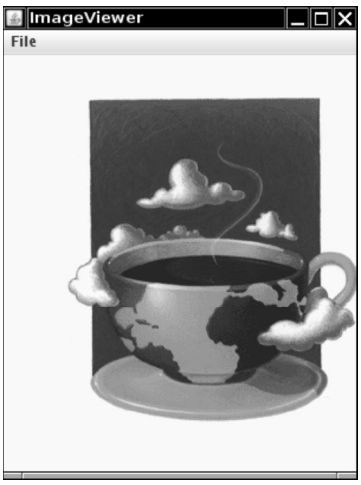

Figure 2–9 Running the ImageViewer application

We hope that you find this program interesting and useful. Have a quick look at the source code. The program is substantially longer than the first program, but it is not terribly complex if you consider how much code it would take in C or C++ to write a similar application. In Visual Basic, of course, it is easy to write or, rather, drag and drop, such a program. The JDK does not have a visual interface builder, so you need to write code for everything, as shown in Listing 2–2. You learn how to write graphical programs like this in Chapters 7 through 9.

Listing 2–2 ImageViewer.java

```
1. import java.awt.EventQueue;
2. import java.awt.event.*;
3. import java.io.*;
4. import javax.swing.*;
5.
6. /**
7.  * A program for viewing images.
8.  * @version 1.22 2007-05-21
9.  * @author Cay Horstmann
10.  */
11. public class ImageViewer
12. {
13.    public static void main(String[] args)
14.    {
15.       EventQueue.invokeLater(new Runnable()
16.          {
17.             public void run()
18.             {
```

Listing 2–2 ImageViewer.java (continued)

```
19.                JFrame frame = new ImageViewerFrame();
20.                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21.                frame.setVisible(true);
22.             }
23.          });
24.    }
25. }
26.
27. /**
28.  * A frame with a label to show an image.
29.  */
30. class ImageViewerFrame extends JFrame
31. {
32.    public ImageViewerFrame()
33.    {
34.       setTitle("ImageViewer");
35.       setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
36.
37.       // use a label to display the images
38.       label = new JLabel();
39.       add(label);
40.
41.       // set up the file chooser
42.       chooser = new JFileChooser();
43.       chooser.setCurrentDirectory(new File("."));
44.
45.       // set up the menu bar
46.       JMenuBar menuBar = new JMenuBar();
47.       setJMenuBar(menuBar);
48.
49.       JMenu menu = new JMenu("File");
50.       menuBar.add(menu);
51.
52.       JMenuItem openItem = new JMenuItem("Open");
53.       menu.add(openItem);
54.       openItem.addActionListener(new ActionListener()
55.          {
56.             public void actionPerformed(ActionEvent event)
57.             {
58.                // show file chooser dialog
59.                int result = chooser.showOpenDialog(null);
60.
61.                // if file selected, set it as icon of the label
62.                if (result == JFileChooser.APPROVE_OPTION)
63.                {
64.                   String name = chooser.getSelectedFile().getPath();
65.                   label.setIcon(new ImageIcon(name));
66.                }
67.             }
68.          });
```

Listing 2–2 ImageViewer.java (continued)

```
69.       JMenuItem exitItem = new JMenuItem("Exit");
70.       menu.add(exitItem);
71.       exitItem.addActionListener(new ActionListener()
72.          {
73.             public void actionPerformed(ActionEvent event)
74.             {
75.                System.exit(0);
76.             }
77.          });
78.    }
79.
80.    private JLabel label;
81.    private JFileChooser chooser;
82.    private static final int DEFAULT_WIDTH = 300;
83.    private static final int DEFAULT_HEIGHT = 400;
84. }
```

Building and Running Applets

The first two programs presented in this book are Java *applications,* stand-alone programs like any native programs. On the other hand, as we mentioned in the last chapter, most of the hype about Java comes from its ability to run *applets* inside a web browser. We want to show you how to build and run an applet from the command line. Then we will load the applet into the applet viewer that comes with the JDK. Finally, we will display it in a web browser.

First, open a shell window and go to the directory `CoreJavaBook/v1ch02/WelcomeApplet`, then enter the following commands:

```
javac WelcomeApplet.java
appletviewer WelcomeApplet.html
```

Figure 2–10 shows what you see in the applet viewer window.

Figure 2–10 WelcomeApplet applet as viewed by the applet viewer

The first command is the now-familiar command to invoke the Java compiler. This compiles the `WelcomeApplet.java` source into the bytecode file `WelcomeApplet.class`.

This time, however, you do not run the `java` program. You invoke the `appletviewer` program instead. This program is a special tool included with the JDK that lets you quickly test an applet. You need to give this program an HTML file name, rather than the name of a Java class file. The contents of the `WelcomeApplet.html` file are shown below in Listing 2–3.

Listing 2–3 `WelcomeApplet.html`

```
<html>
   <head>
      <title>WelcomeApplet</title>
   </head>
   <body>
      <hr/>
      <p>
         This applet is from the book
         <a href="http://www.horstmann.com/corejava.html">Core Java</a>
         by <em>Cay Horstmann</em> and <em>Gary Cornell</em>,
         published by Sun Microsystems Press.
      </p>
      <applet code="WelcomeApplet.class" width="400" height="200">
         <param name="greeting" value ="Welcome to Core Java!"/>
      </applet>
      <hr/>
      <p><a href="WelcomeApplet.java">The source.</a></p>
   </body>
</html>
```

If you are familiar with HTML, you will notice some standard HTML instructions and the `applet` tag, telling the applet viewer to load the applet whose code is stored in `WelcomeApplet.class`. The applet viewer ignores all HTML tags except for the `applet` tag.

Unfortunately, the browser situation is a bit messy.

- Firefox supports Java on Windows, Linux, and Mac OS X. To experiment with applets, just download the latest version, visit `http://java.com`, and use the version checker to see whether you need to install the Java Plug-in.
- Some versions of Internet Explorer have no support for Java at all. Others only support the very outdated Microsoft Java Virtual Machine. If you run Internet Explorer, go to `http://java.com` and install the Java Plug-in.
- If you have a Macintosh running OS X, then Safari is integrated with the Macintosh Java implementation, which supports Java SE 5.0 at the time of this writing.

Provided you have a browser that supports a modern version of Java, you can try loading the applet inside the browser.

1. Start your browser.
2. Select File -> Open File (or the equivalent).
3. Go to the `CoreJavaBook/v1ch02/WelcomeApplet` directory. You should see the `WelcomeApplet.html` file in the file dialog. Load the file.
4. Your browser now loads the applet, including the surrounding text. It will look something like Figure 2–11.

You can see that this application is actually alive and willing to interact with the Internet. Click on the Cay Horstmann button. The applet directs the browser to display Cay's web page. Click on the Gary Cornell button. The applet directs the browser to pop up a mail window, with Gary's e-mail address already filled in.

Figure 2–11 Running the WelcomeApplet applet in a browser

Notice that neither of these two buttons works in the applet viewer. The applet viewer has no capabilities to send mail or display a web page, so it ignores your requests. The applet viewer is good for testing applets in isolation, but you need to put applets inside a browser to see how they interact with the browser and the Internet.

TIP: You can also run applets from inside your editor or integrated development environment. In Emacs, select JDE -> Run Applet from the menu. In Eclipse, use the Run -> Run as -> Java Applet menu option.

Finally, the code for the applet is shown in Listing 2–4. At this point, do not give it more than a glance. We come back to writing applets in Chapter 10.

Listing 2–4 WelcomeApplet.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.net.*;
4. import javax.swing.*;
5.
6. /**
7.  * This applet displays a greeting from the authors.
8.  * @version 1.22 2007-04-08
9.  * @author Cay Horstmann
10. */
```

Listing 2–4 WelcomeApplet.java (continued)

```
11. public class WelcomeApplet extends JApplet
12. {
13.    public void init()
14.    {
15.       EventQueue.invokeLater(new Runnable()
16.          {
17.             public void run()
18.             {
19.                setLayout(new BorderLayout());
20.
21.                JLabel label = new JLabel(getParameter("greeting"), SwingConstants.CENTER);
22.                label.setFont(new Font("Serif", Font.BOLD, 18));
23.                add(label, BorderLayout.CENTER);
24.
25.                JPanel panel = new JPanel();
26.
27.                JButton cayButton = new JButton("Cay Horstmann");
28.                cayButton.addActionListener(makeAction("http://www.horstmann.com"));
29.                panel.add(cayButton);
30.
31.                JButton garyButton = new JButton("Gary Cornell");
32.                garyButton.addActionListener(makeAction("mailto:gary_cornell@apress.com"));
33.                panel.add(garyButton);
34.
35.                add(panel, BorderLayout.SOUTH);
36.             }
37.          });
38.    }
39.
40.    private ActionListener makeAction(final String urlString)
41.    {
42.       return new ActionListener()
43.          {
44.             public void actionPerformed(ActionEvent event)
45.             {
46.                try
47.                {
48.                   getAppletContext().showDocument(new URL(urlString));
49.                }
50.                catch (MalformedURLException e)
51.                {
52.                   e.printStackTrace();
53.                }
54.             }
55.          };
56.    }
57. }
```

In this chapter, you learned about the mechanics of compiling and running Java programs. You are now ready to move on to Chapter 3, where you will start learning the Java language.