CHAPTER 1

GETTING STARTED

The longest journey begins with a single step.

CHAPTER OBJECTIVES

In this chapter, you will learn about:

To attain proficiency with a computer programming language, you must get your hands dirty. In order to get your hands dirty, you must have appropriate tools at your disposal. Within this chapter, we will introduce the tools needed to create and run Java programs, and show you how to obtain and configure these tools on your system.

With these tools installed on your computer, this interactive workbook offers you the opportunity to both read about Java programming and become an active participant in your Java education as well. There is no substitute for experience, so let's hit the ground running!

LAB 1.1

LAB 1.1

COMPILING AND RUNNING A JAVA PROGRAM

LAB OBJECTIVES

After this lab, you will be able to:

- Install and Configure the Java Development Kit
- Compile and Run a Java Program
- Compile and Run a Java Applet

Before the introduction of the Java programming language, most software development was done in a platform-specific manner. Programs written in languages such as Fortran, C, and C++ were run through *compilers* to generate platform-specific executables. PCs, Macintoshes, and UNIX workstations all required their own version of an executable and often the programs would have to be specifically tailored for each platform.

Java programs must be run through a compiler as well. However, the Java compiler does not generate platform-specific code—it generates platform-independent *bytecodes*. These bytecodes can then be executed by a *Java Virtual Machine* (JVM). Strictly speaking, Java is not "Write Once, Run Anywhere" as claimed by Sun; realistically, it is "Write Once, Run on Any System With a JVM." Because JVMs are available for most interesting platforms (and even a few not-so-interesting ones), the language has fulfilled its promise of cross-platform compatibility.

To foster development with the Java programming language, Sun Microsystems provides a free software development kit (SDK), which includes a compiler, virtual machine, debugger, and a host of other useful development tools. This is called the Java Development Kit, or JDK, and the latest Windows and Solaris versions are always available from the Sun Web site at:

http://java.sun.com/products/jdk/1.2/index.html

If you are developing on a platform other than Windows or Solaris (good for you!), you can find information regarding JDKs for other platforms on the Web at:

http://java.sun.com/cgi-bin/java-ports.cgi

The JDK is free of charge and free of frills. There is no graphical user interface provided to these tools, so you can only use them from a command-line interface such as a DOS window or UNIX shell. The benefit of the JDK is that it is a reference implementation—if you can successfully compile and run programs using Sun's JDK, you can be sure your programs will run under any compatible JVM. If you have done previous software development with an integrated development environment (IDE), you may find this adjustment difficult. Many commercial Java IDEs are available, and there are even a few free IDEs (like Sun's Java Workshop product), but they will not be addressed within this text—use them at your own peril.

Do not confuse the JDK with the Java Runtime Environment (JRE). The JRE contains only the virtual machine and supporting code necessary to run (not develop) Java programs. The computer industry has an affinity for ambiguous acronyms, and if you install the JRE instead of the JDK you will find yourself DOA without an SDK.

LAB 1.1 EXERCISES

1.1.1 INSTALL AND CONFIGURE THE JAVA DEVELOPMENT KIT

The latest version of Sun's Java2 JDK can be found on the Web at:

http://java.sun.com/products/jdk/1.2/index.html

LAB 1.1

Download the JDK version appropriate for your platform and install it. The Windows version will attempt to install itself into **C:\JDK1.2.x**, where **x** indicates the dot-version of the current JDK. The Solaris version will attempt to install itself into your current working directory.

The JDK documentation, known as the Java API documentation or "Javadocs," can be downloaded from the same Web site. Download the HTML-formatted documentation and install it. The Javadocs are an excellent example of documentation done right, and we will be referring to this documentation throughout the text.

You didn't neglect to install the documentation, did you? Consider it a requirement for learning the language.

As you will shortly discover, Javadocs will become your second-best friend during your forray into the depths of Java. Your best friend, of course, is this interactive workbook!

Once the JDK is installed on your system, you must modify your **PATH** environment variable to include the location of your development tools. For those unfamiliar with the **PATH** variable, it tells your operating system where to look within your filesystem for programs.

ON WINDOWS 95, 98 OR NT . . .

If you are running Windows 95 or 98, you can modify your **PATH** within the **autoexec.bat** file in a manner similar to the following. Be sure to change the **JDK1.2.x** in the following line to match the location where you installed the JDK on your system.

```
PATH=%PATH%;C:\JDK1.2.x\Bin
```

This change will take effect next time you reboot Windows—like that is news to any Windows user.

Users of WindowsNT can make this modification within the Console application available within the Control Panel.

To double-check that your **PATH** modification has been successful, you can input the following from an MS-DOS Prompt window:

```
C:\WINDOWS>path
```

If your **PATH** was modified successfully, you should see the location of the **Bin** subdirectory of your JDK installation, similar to the following:

```
PATH=C:\WINDOWS;C:\WINDOWS\COMMAND;C:\JDK12~1.2\BIN
```

ON SOLARIS, UNIX OR LINUX OS . . .

If your default shell is a Bourne-shell derivative, you can modify your **PATH** similar to the following in your **.profile** file. Modify the **/usr/local/jdk1.2.x/bin** below to reflect the **bin** subdirectory where you installed the JDK.

```
PATH=$PATH:/usr/local/jdk1.2.x/bin
export PATH
```

To double-check your modification, open a new shell and input the following:

```
echo $PATH
```

If the modification was successful, you should see the location of the **bin** subdirectory of your JDK installation, similar to the following:

```
/bin:/usr/bin:/usr/local/bin:/usr/local/jdk1.2.x/bin
```

IF YOU HAVE ALREADY INSTALLED A PREVIOUS VERSION OF THE JDK ON EITHER PLATFORM . . .

Similar to the **PATH** environment variable used by your operating system, Java uses another environment named **CLASSPATH**. The **CLASSPATH** environment variable informs the JVM where it should look for Java program files (classes). If you do not have this defined on your system, don't worry about it. If you do have this defined on your system, be sure to remove an entry for **classes.zip** if it is present—this entry is needed for older Java1.1 releases, but will only confuse matters with Java2. Use the same steps as outlined previously for your **PATH** environment variable.

CONGRATULATIONS!

You have just installed and configured the JDK. Answer the following questions:

From an MS-DOS prompt or UNIX shell, type the following:

```
java –version
```

a) What was displayed?

1.1.2 COMPILE AND RUN A JAVA PROGRAM

Java source code can be written with whatever editor you are most comfortable. On Windows systems this might be notepad, edit, or brief. On UNIX systems this is probably vi, emacs, or pico.

Java source code files adhere to a strict naming convention. This will be discussed in following chapters, but it is important to know that filenames are case-sensitive (*MyApp* is not the same as *myapp*) and all end with a **.java** extension.

Create your first Java program by firing-up your favorite editor and entering the following code exactly as it is written. Save this file as **Welcome.java**:

```
import javax.swing.*;
public class Welcome
{
    public static void main(String[] argv)
    {
          JOptionPane.showMessageDialog(
                null,
                "Welcome to the wonderful world of
Java!",
                "Welcome, new Programmer",
                JOptionPane.PLAIN_MESSAGE);
          System.exit(0);
    }
}
```

Once the file is saved, start up an MS-DOS Prompt or UNIX shell. Navigate to the directory that contains Welcome.java and run the following command:

```
javac Welcome.java
```

a) What were your results from running this command?

Without changing directories, run the following command:

```
java Welcome
```

b) What were your results from running this command?

1.1.3 COMPILE AND RUN A JAVA APPLET

Fire up your favorite editor and create the following file, named **WelcomeApplet.java**:

```
import java.awt.*;
import java.applet.*;
public class WelcomeApplet extends Applet
{
    Label textLabel;
    public void init()
    {
          textLabel = new Label(
              "Welcome to the wonderful world of Java!");
          textLabel.setAlignment(Label.CENTER);
          this.add(textLabel);
    }
}
```

Start an MS-DOS Prompt or UNIX shell and navigate to the directory that contains WelcomeApplet.java. Run the following command:

```
javac WelcomeApplet.java
```

a) What were your results from running this command?

Once again, start an editor and create the following file named **WelcomeApplet.html**:

```
<HTML>
<TITLE>Welcome</TITLE>
<APPLET CODE="WelcomeApplet.class" WIDTH=400
HEIGHT=200>
If you can read this, your browser does not support
Java!
</APPLET>
</HTML>
```

From within the same directory, run the following command:

```
appletviewer WelcomeApplet.html
```

b) What were your results from running this command?

LAB 1.1 EXERCISE ANSWERS

1.1.1 ANSWERS

From an MS-DOS prompt or UNIX shell, type the following:

```
java –version
```

a) What was displayed?

Answer: If everything worked properly, you should have seen output similar to the following:

```
java version "1.2.2"
Classic VM (build JDK-1.2.2-W, native threads,
symcjit)
```

The **java** program is responsible for starting a JVM. The argument **–version** indicates that you are only interested in the version of the JVM, and do not care to run a Java program.

Depending upon the current version of the JDK, you may be running a slightly newer version than shown in the output above. So long as your version string is of the format **1.2.x**, where **x** represents the dot-release of the JDK, you are doing well. If by some chance an older version was displayed, such as **1.0.x** or **1.1.x**, you may have an old version of Java on your system. If you do not wish to remove the old version of Java, you should modify your **PATH** environment variable so that the 1.2 JDK entry is placed before any other JDK entries.

If you received an error message similar to

```
Bad command or file name
```

or

```
java: command not found
```

your **PATH** environment variable may not be set correctly. Review the Lab and double-check that the JDK bin directory is in your **PATH**.

1.1.2 ANSWERS

Once the file is saved, start up an MS-DOS Prompt or UNIX shell. Navigate to the directory that contains Welcome.java and run the following command:

```
javac Welcome.java
```

a) What were your results from running this command?

Answer: If all went well, you should have received no output to your console.

The **javac** program is the Java compiler. It translates source code into Java bytecodes. If the compiler was successful, it translated your **Welcome.java** source code into Java bytecodes and stored them in a new class file named **Welcome.class**.

If you received any error messages from the **javac** command, check the source code to be sure it was input exactly as shown in the Lab. Java is case-sensitive, so be sure you did not mix character cases.

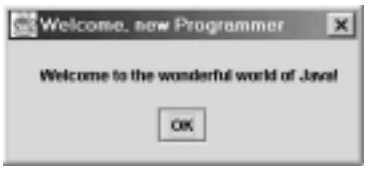
Without changing directories, run the following command:

```
java Welcome
```

b) What were your results from running this command?

Answer: You should have been greeted by a popup window that contained the text "Welcome to the wonderful world of Java!".

The **java** command started a JVM that ran the **Welcome** program. This simple program merely created a popup window that contained a friendly welcome message similar to the following:



If you received an error from the java command similar to the following:

```
Exception in thread "main" java.lang.NoClass
DefFoundError: Welcome
```

the JVM was unable to locate the **Welcome.class** file. Try running the command again, from the same directory which contains the **Welcome .class** file.

1.1.3 ANSWERS

Start an MS-DOS Prompt or UNIX shell and navigate to the directory which contains WelcomeApplet.java. Run the following command:

```
javac WelcomeApplet.java
```

a) What were your results from running this command?

Answer: Similar to the previous lab section, you should have received no output if all went well.

If the WelcomeApplet.java source code compiled properly, you received no messages from the Java compiler. If you did receive error messages from the compiler, double-check the source code to be sure it appears exactly as shown within the lab. You don't think *we* made a typo, do you?!

From within the same directory, run the following command:

```
appletviewer WelcomeApplet.html
```

b) What were your results from running this command?

Answer: You should have been greeted by an applet that displayed the text message "Welcome to the wonderful world of Java!"

The **appletviewer** program is included with the JDK as an alternative to running applets within a web browser, and provides an interface that makes testing of applets much less cumbersome than within a web browser.

More importantly, at the time of this writing, neither Netscape Navigator nor Microsoft Internet Explorer include standard support for Java2. For this reason, we will assume throughout the workbook that **appletviewer** will be used to run all the example applets.

If you did not see an applet displayed, check your **WelcomeApplet.html** file to be sure it does not deviate from that which is shown within the Lab. Also, be sure that both the **WelcomeApplet.html** and **WelcomeApplet.class** files exist within the same directory from which you ran **appletviewer**.

What do you mean Java2 is not supported by my browser?!

Relax—in an effort to provide an upgrade to the JVM of your browser, Sun has created a free browser plug-in which allows you to run the JVM of your choice within the web browser.

Check your browser documentation to see if it supports Java2. If it does not, check Sun's Java Plug-In web site for details:

http://java.sun.com/products/plugin

LAB 1.1 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions.

1) The Java compiler transforms source code into what platform-independent format?

a) _____ opcodes
b) _____ bytecodes
c) _____ virtual machines
d) _____ applets

2) What command runs the Java compiler?
- **a)** _____ java
- **b)** _____ appletviewer
- **c)** _____ jdb
- **d)** _____ javac

3) Which of the following commands is used to run a Java application?
- **a)** _____ java
- **b)** _____ appletviewer
- **c)** _____ jdb
- **d)** _____ javac

4) Which of the following commands is used to run a Java applet?
- **a)** _____ java
- **b)** _____ appletviewer
- **c)** _____ jdb
- **d)** _____ javac

5) The freely-available collection of tools which can be used to create Java programs is known as what?
- **a)** _____ JRE
- **b)** _____ JDK
- **c)** _____ JVM
- **d)** _____ JNI

Quiz answers appear in Appendix A, Section 1.1.

CHAPTER 1

TEST YOUR THINKING

The purpose of this chapter was just to make sure you could run a Java program. There's not much point in learning a computer language if you can't use it.

1) Since you are going to be programming frequently in this book, we suggest that you setup a "programming environment" for yourself. A programming environment is a configuration of your computer, or computer account, which allows you to write, compile, and run computer programs. We recommend that you setup your login environment to allow you to write, compile, and run Java programs. Once you have successfully completed the labs of this chapter, you should "lock in" that configuration so the next time you login to use your computer, you don't have to worry about the configuration again. This will save you a great deal of time in the future.