Ivor Horton's Beginning Java®

Java 7 Edition

Ivor Horton

CONTENTS

Chapter 1

Introducing Java

WHAT YOU WILL LEARN IN THIS CHAPTER:

- The basic characteristics of the Java language
- How Java programs work on your computer
- Why Java programs are portable between different computers
- The basic ideas behind object-oriented programming
- How a simple Java program looks and how you can run it using the Java Development Kit
- What HTML is and how to use it to include a Java program in a web page

This chapter should give you an appreciation of what the Java language is all about. Understanding th details of what I discuss in this chapter is not important at this stage; you see all of the topics again greater depth in later chapters of the book. The intent of this chapter is to introduce you to the gener ideas that underpin what I cover through the rest of the book, as well as the major contexts in whic Java programs can be used and the kind of program that is applicable in each context.

WHAT IS JAVA ALL ABOUT?

Java is an innovative programming language that has become the language of choice for programs th need to run on a variety of different computer systems. First of all, Java enables you to write sma programs called *applets*. These are programs that you can embed in web pages to provide son intelligence. Being able to embed executable code in a web page introduces a vast range of excitin possibilities. Instead of being a passive presentation of text and graphics, a web page can b interactive in any way that you want. You can include animations, games, interactive transactio processing — the possibilities are almost unlimited.

Of course, embedding program code in a web page creates special security requirements. As a Internet user accessing a page with embedded Java code, you need to be confident that it won't d anything that might interfere with the operation of your computer or damage the data you have o your system. This implies that execution of the embedded code must be controlled in such a way th it prevents accidental damage to your computer environment, as well as ensure that any Java code th was created with malicious intent is effectively inhibited. Java implicitly incorporates measures minimize the possibility of such occurrences arising with a Java applet.

Java's support for the Internet and network-based applications generally doesn't end with applet For example, Java Server Pages (JSP) provides a powerful means of building a server application th can dynamically create and download HTML pages to a client that are precisely customized for th specific request that is received. Of course, the pages that are generated by JSP can themselves conta Java applets.

Java also enables you to write large-scale application programs that you can run unchanged on an ~~computer with an operating system environment in which Java is supported. This applies~~ to th majority of computers in use today. The slogan that was coined to illustrate the cross-platfor capability of Java, “write once, run anywhere,” has been amply demonstrated to be the case. You ca develop code on a PC and it will run on a Java-enabled cell phone. You can even write programs th work both as ordinary applications and as applets.

Java has matured immensely in recent years. The breadth of function provided by the standard co Java has grown incredibly. Java provides you with comprehensive facilities for building applicatio with an interactive graphical user interface (GUI), extensive image processing and graphi programming facilities, as well as support for XML, accessing relational databases an communicating with remote computers over a network. Just about any kind of application can now b programmed effectively in Java, with the implicit plus of complete portability.

Of course, Java is still developing and growing. The latest Java Development Kit, JDK 7, adds ma new facilities that include new language features as well as significant additions to the supporti libraries. You learn about all of these in this book.

FEATURES OF THE JAVA LANGUAGE

The most important characteristic of Java is that it was designed from the outset to be machi independent. You can run Java programs unchanged on any machine and operating syste combination that supports Java. Of course, there is still the slim possibility of the odd glitch, as yo are ultimately dependent on the implementation of Java on any particular machine, but Java progra are intrinsically more portable than programs written in other languages. An application written Java only requires a single set of source code statements, regardless of the number of differe computer platforms on which it is run. In any other programming language, the application frequent requires the source code to be tailored to accommodate different computer environments, particular if an extensive graphical user interface is involved. Java offers substantial savings in time an resources in developing, supporting, and maintaining major applications on several different hardwa platforms and operating systems.

Possibly the next most important characteristic of Java is that it is *object-oriented*. The objec oriented approach to programming is an implicit feature of all Java programs, so you find out wh this means later in this chapter. Object-oriented programs are easier to understand and less tim consuming to maintain and extend than programs that have been written without the benefit of usi objects.

Not only is Java object-oriented, but it also manages to avoid many of the difficulties an complications that are inherent in some object-oriented languages, making it easy to learn and ve straightforward to use. By and large, it lacks the traps and “gotchas” that arise in some oth programming languages. This makes the learning cycle shorter, and you need less real-world codi experience to gain competence and confidence. It also makes Java code easier to test.

Java has a built-in ability to support national character sets. You can write Java programs as easil for use in Greece or Japan as you can for English-speaking countries, assuming you are familiar wi the national languages involved, of course. You can even build programs from the outset to suppo several different national languages with automatic adaptation to the environment in which the co executes.

LEARNING JAVA

Java is not difficult to learn, but there is a great deal to it. Although the Java language is ver powerful, it is fairly compact, so acquiring an understanding of the Java language should take le time than you think. However, there's much more to Java than just the language. To be able t program effectively in Java, you need to understand the libraries that go with the language, and the are very extensive. It is also important to become familiar with open source projects, especially tho developed by the Apache folks.

In this book, the sequence in which you learn how the language works and how you apply it has bee carefully structured so that you gain expertise and confidence with programming in Java through relatively easy and painless process. As far as possible, each chapter avoids the use of things that yo haven't learned about already. A consequence, though, is that you won't be writing Java applicatior with application windows and a Graphical User Interface (GUI) right away. Although it may be a appealing idea, this would be a bit like learning to swim by jumping in the pool at the deep en Generally speaking, there is good evidence that by starting in the shallow end of the pool and learnir how to float before you try to swim, you minimize the chance of drowning, and there is a hig expectation that you can end up being a competent swimmer.

Java Programs

As I have already noted, there are two basic kinds of programs you can write in Java. Programs th are to be embedded in a web page are called Java applets, and normal standalone programs are calle Java applications. You can further subdivide Java applications into console applications, which onl support character output to your computer screen (console output typically goes to the command lir on a PC under Microsoft Windows, for example), and windowed applications, which can create ar manage multiple windows. The latter use the typical GUI mechanisms of window-based programs - menus, toolbars, dialogs, and so on.

While you are learning the Java language basics, you use console applications as examples understand how things work. These are applications that use simple command-line input and outpu With this approach you can concentrate on understanding the specifics of the language witho worrying about any of the complexity involved in creating and managing windows. After you a comfortable with using all the features of the Java language, you move on to window-base applications and applet examples.

Learning Java — the Road Ahead

Before starting out on any journey, it is always helpful to have an idea of where you're heading ar what route you should take, so let's take a look at a brief road map of where you're going with Jav There are seven broad stages you progress through in learning Java using this book:

1. The first stage is this chapter. It sets out some fundamental ideas about the structure of Jav programs and how they work. This includes such things as what object-oriented programming is a about and how an executable program is created from a Java source file. Getting these concep straight at the outset makes learning to write Java programs that much easier for you.

2. Next, in Chapters 2 to 4, I explain how statements are put together, what facilities you have fo storing basic data in a program, how you perform calculations, and how you make decisions base on the results of them. These are the nuts and bolts you need for the next stages.

3. In the third stage, in Chapters 5 and 6, you learn about *classes* — how you define them and ho you can use them. Classes are blueprints for objects, so this is where you learn the object-oriente characteristics of Java. By the time you are through this stage, you should have learned all th basics of how the Java language works so you are ready to progress further into how you can app it.

4. In the fourth stage, in Chapters 7 through 12, you learn how you deal with errors and how yo read and write files. Of course, file input/output is an essential capability in the majority applications.

5. The fifth stage is covered by Chapters 13 to 15. These chapters explain how you define gener class types, which are blueprints for creating sets of similar classes. You also learn about a rang of utility classes and capabilities from the support libraries that you can apply in many differe program contexts.

6. In the sixth stage, in Chapters 16 to 21, you learn in detail how you implement applications applets with a graphical user interface, and how you handle interactions with the user in th context. This amounts to applying the GUI capabilities provided by the Java class libraries. Yo also learn how you manage concurrent threads of execution within a Java program, which fundamental to effective GUI programming. When you finish this stage, you should be equipped write your own fully fledged applications and applets in Java.

7. In the last stage you learn about the Extensible Markup Language, XML, which is a powerf tool for representing data that is to be transferred from one computer to another. You apply th Java support classes for XML in a practical context, writing and reading XML files.

At the end of the book, you should be a knowledgeable Java programmer. The rest is down experience.

Throughout this book I use complete examples to explore how Java works. You should create an run all of the examples, even the simplest, preferably by typing them in yourself. Don't be afraid experiment with them. If there is anything you are not quite clear on, try changing an example arou to see what happens, or better still — write an example of your own. If you're uncertain how som aspect of Java that you have already covered works, don't look it up right away — try out a few thin and see if you can figure it out. Making mistakes is a very effective way to learn.

THE JAVA ENVIRONMENT

You can execute Java programs on a variety of computers using a range of operating systems. You Java programs run just as well on a PC running any supported version of Microsoft Windows as does on Linux or a Sun Solaris workstation. This is possible because a Java program does not execu directly on your computer. It runs on a standardized environment called the *Java 2 Platform* that h been implemented as software in the form of the *Java Runtime Environment (JRE)* on a wide varie of computers and operating systems. The Java Platform consists of two elements — a softwa implementation of a hypothetical computer called the *Java Virtual Machine (JVM)* and the *Ja Application Programming Interface (Java API)*, which is a set of software components that provid the facilities you need to write a fully fledged interactive application in Java.

A *Java compiler* converts the Java source code that you write into a binary program consisting *bytecodes*. Bytecodes are machine instructions for the JVM. When you execute a Java program, program called the *Java interpreter* inspects and deciphers the bytecodes for it, checks it out to ensu

that it has not been tampered with and is safe to execute, and then executes the actions that th
by~~tecodes specify within the JVM. A Java interpreter can run standalone, or it can be part of~~ a we
browser such as Google Chrome, Mozilla Firefox, or Microsoft Internet Explorer where it can b
invoked automatically to run applets in a web page.

Because your Java program consists of bytecodes rather than native machine instructions, it
completely insulated from the particular hardware on which it is run. Any computer that has the Jav
environment implemented handles your program as well as any other, and because the Java interpret
sits between your program and the physical machine, it can prevent unauthorized actions in th
program from being executed.

In the past, there has been a penalty for all this flexibility and protection in the speed of execution
your Java programs. An interpreted Java program would typically run at only one-tenth of the speed
an equivalent program using native machine instructions. With present Java machin
implementations, much of the performance penalty has been eliminated, and in programs that are n
computation intensive — you really wouldn't notice this anyway. With the JVM that is supplied wi
the current Java 2 Development Kit (JDK) available from the Oracle website, there are very fe
circumstances where you notice any appreciable degradation in performance compared to a progra
compiled to native machine code.

Java Program Development

For this book you need the Java 2 Platform, Standard Edition (J2SE) version 7 or later. The JDK
available from www.oracle.com/technetwork/java/javase/downloads/index.html. You can choose fro
versions of the JDK for Solaris, Linux, and Microsoft Windows, and there are versions supportin
either 32-bit or 64-bit operating system environments.

Using a Program Code Editor

To create the Java program source files that you use with the JDK, you need some kind of code edito
There are several excellent professional Java program development tools available that provi
friendly environments for creating and editing your Java source code and compiling and debuggin
your programs. These are powerful tools for the experienced programmer that improve productivi
and provide extensive debugging capabilities. However, for learning Java using this book,
recommend that you resist the temptation to use any of these for the time being.

So why am I suggesting that you are better off *not* using a tool that makes programming easier an
faster? There are several reasons. Firstly, the professional development systems tend to hide a lot
things you need to get to a grip on if you are to get a full understanding of how Java works. Secondl
the professional development environments are geared to managing complex applications with a larg
amount of code, which introduces complexity that you really are better off without while you a
learning. Virtually all commercial Java development systems provide prebuilt facilities of their ow
to speed development. Although this is helpful for production program development, it really does g
in the way when you are trying to learn Java. A further consideration is that productivity feature
supported by a commercial Java development are sometimes tied to a specific version of the Java
Platform. This means that some features of the latest version of Java might not work. The profession
Java development tools that provide an Interactive Development Environment (IDE) are intende
primarily for knowledgeable and experienced programmers, so start with one when you get to the en
of the book.

So what *am* I recommending? Stick to using JDK 7 from Oracle with a simple program text editor ~~development environment for creating and managing your source code. Quite a number of sharewa~~ and freeware code editors around are suitable, some of which are specific to Java, and you should hav no trouble locating one. I can make two specific suggestions.

- I find that the JCreator editor from www.jcreator.com is particularly good and is easy to insta There's a free version, JCreator LE, and a paid version with more functionality. The free versic is perfectly adequate for learning but you may want to upgrade to the paid version when you hav reached the end of the book.
- Notepad++ is a free source code editor running in the Microsoft Windows environment that yo can download from http://notepad-plus-plus.org. It supports syntax highlighting for Java. It also handy as a simple editor for XML files.
- I recommend the NetBeans IDE that you can download from http://netbeans.org , which is als free. This is a sophisticated professional interactive development environment that supports n only Java, but also many other programming languages. Nonetheless, it is easy to use and there extensive online documentation. If you have plans to progress into programming using oth languages, there's a good chance that you will find the NetBeans IDE supports them.

Whichever code editor you choose to use, I recommend that you only use the simplest proje creation options. In particular you should avoid using any of the program project types that provi prebuilt skeleton program code while you are working your way through this book. By codi everything yourself you will maximizes your learning experience. Of course, after you hav assimilated everything in the book, you are ready to enhance your Java program developme capability with the full capabilities of a pro development tool.

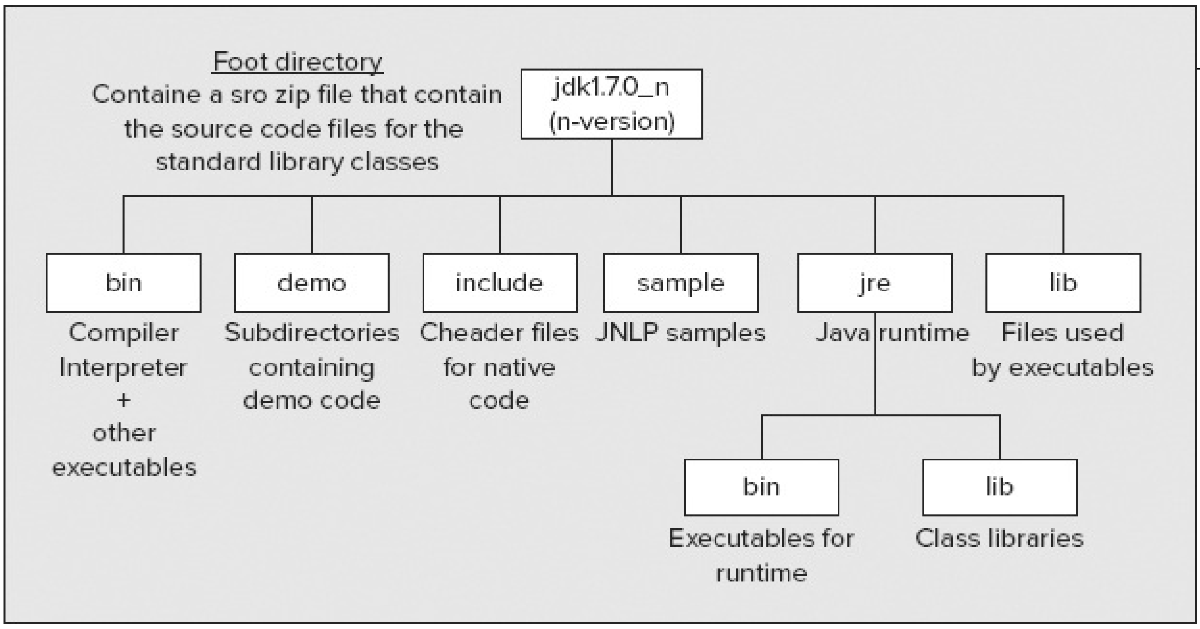
A good place to start looking if you want to investigate what other editors are available is th www.download.com website.

Installing the JDK

Detailed instructions on how to install the JDK for your particular operating system are available fro the JDK download website at www.oracle.com/technetwork/java/javase/downloads/index.html, so won't go into all the variations for different systems here. However, you should watch out for a fe things that may not leap out from the pages of the installation documentation.

First of all, the JDK and the documentation are separate, and you install them separately. If you a pushed for disk space, you don't have to install the documentation because you can access it onlin The current location for the online documentation for the JDK http://download.java.net/jdk7/docs/api but this could conceivably change. The documentatio download for the JDK consists of a ZIP archive containing a large number of HTML files structured i a hierarchy. You should install the JDK before you unzip the documentation archive. If you install th JDK to drive C: under Windows, the directory structure shown in Figure 1-1 is created.

FIGURE 1-1



Foot directory
Containe a sro zip file that contain the source code files for the standard library classes
jdk1.7.0_n (n-version)
bin
Compiler Interpreter + other executables
demo
Subdirectories containing demo code
include
Cheader files for native code
sample
JNLP samples
jre
Java runtime
lib
Files used by executables
bin
Executables for runtime
lib
Class libraries

This structure appears in your `C:\Program Files\Java` directory (or `C:\Program Files (×86)\Java` if you insta the 32-bit version of the JDK with a 64-bit version of Windows 7). The `jdk1.7.0` directory in Figure 1 is sometimes referred to as the root directory for Java. In some contexts it is also referred to as t Java home directory. The actual root directory name may have the release version number appende in which case the actual directory name is of the form `jdk1.7.0_01`, for example. Figure 1-1 shows t fundamental subdirectories to the root directory.

The `sample` directory contains sample applications that use JNLP, which is the Java Netwo Launching Protocol that is used for executing applications or applets from a network server witho the need for a browser or the need to download and install the code.

You don't need to worry about the contents of most of these directories, at least not when you g started. The installation process should add the path for the `jdk1.7.0_n\bin` directory to the paths define in your PATH environment variable, which enables you to run the compiler and the interpreter fro anywhere without having to specify the path to it. If you installed the JDK to your C: drive, then t path is `C:\Program Files\Java\jdk1.7.0_n\bin`.

A word of warning — if you have previously installed a commercial Java development produ check that it has not modified your PATH environment variable to include the path to its own Ja executables. If it has and the path precedes the path to JDK 7, when you try to run the Java compiler interpreter, you will get the versions supplied with the commercial product rather that those that cam with JDK 7. One way to fix this is to remove the path or paths that cause the problem. If you do want to remove the paths that were inserted for the commercial product, you have to use the full pa specification when you want to run the compiler or interpreter from the JDK.

The `jre` directory contains the Java Runtime Environment facilities that are used when you execute Java program. The classes in the Java libraries are stored in the `jre\lib directory`. They don't appe individually though. They are all packaged up in the archive, `rt.jar`. Leave this alone. The JRE tak care of retrieving what it needs from the archive when your program executes.

The CLASSPATH environment variable is a frequent source of problems and confusion to newcomers Java. The current JDK does not require CLASSPATH to be defined, and if it has been defined by some oth Java version or system, it may cause problems. Check to see whether CLASSPATH has been defined your system. If you have to keep the CLASSPATH environment variable — maybe because you want keep the system that defined it or you share the machine with someone who needs it — you have use a command-line option to define CLASSPATH temporarily whenever you compile or execute your Ja code. You see how to do this a little later in this chapter.

If ~~you want the JDK documentation installed in the hierarchy shown in Figure 1-1, then you shou~~ extract the documentation from the archive to the `jdk1.7.0_n` directory. This creates a new subdirector `docs` (or possibly `docs_www`), to the root directory, and installs the documentation files in that. To look the documentation, you just open the `index.html` file that is in the `docs` subdirectory.

Extracting the Source Code for the Class Libraries

The source code for the class libraries is included in the archive `src.zip` that you find in the `jdk1.7.0` ro directory. Many Java IDEs can access the contents of this zip directly without unpacking it. After yo have learned the basics of the Java language, browsing this source is very educational, and it can als be helpful when you are more experienced with Java in giving a better understanding of how thin work — or when they don't, why they don't.

You can extract the source files from the `src.zip` archive using the Winzip utility, the WinRA utility, the JAR utility that comes with the JDK, or any other utility that unpacks `.zip` archives — b be warned — there's a lot of it, and it takes a while!

Extracting the contents of `src.zip` to the root directory `\jdk1.7.0` creates a new subdirectory, `src`, a installs the source code in subdirectories to this. To look at the source code for a particular class, ju open the `.java` file that you are interested in using with your Java program editor or any plain te editor.

Compiling a Java Program

Java source code is always stored in files with the extension `.java`. After you have created the sour code for a program and saved it in a `.java` file, you need to process the source using a Java compile Using the compiler that comes with the JDK, you make the directory that contains your Java sour file the current directory, and then enter the following command:

```
javac MyProgram.java
```

Here, `javac` is the name of the Java compiler, and `MyProgram.java` is the name of the program source fi This command assumes that the current directory contains your source file. If it doesn't, the compil isn't able to find your source file. It also assumes that the source file corresponds to the Java langua as defined in the current version of the JDK. There is a command-line option, `-source`, that you can u to specify the Java language version, so for JDK 7, the preceding command to execute the compiler equivalent to the following:

```
javac -source 1.7 MyProgram.java
```

In practice you can ignore the `-source` command-line option unless you are compiling a Java progra that was written using an older version of the JDK. You get the current source version compiled b default. To compile code written for JDK 6 you would write:

```
javac -source 1.6 oldSourceCode.java
```

Here's a simple program that you use to can try out the compiler:

Available for download on Wrox.com

```
public class MyProgram {
  public static void main(String[] args) {
    ~~System.out.println("Rome wasn't burned in a day!");~~
  }
}
```

MyProgram.ja

This just outputs a line of text to the command line when it executes. As this is just to try out th compiler, I'm not explaining how the program works at this point. Of course, you must type the coc in exactly as shown and save it in a file with the name `MyProgram.java`. The file name without th extension is always the same as the class name in the code. If you have made any mistakes th compiler issues error messages.

If you need to override an existing definition of the `CLASSPATH` environment variable — perhap because it has been set by a Java development system you have installed — the command would b the following:

```
javac -classpath . MyProgram.java
```

The value of `CLASSPATH` follows the `-classpath` option specification and here it is just a period. A perio defines the path to the current directory, whatever that happens to be. This means that the compil looks for your source file or files in the current directory. If you forget to include the period, th compiler is not able to find your source files in the current directory. If you include the `-classpath` command-line option in any event it does no harm. If you need to add more paths to the `CLASSPA` specification, they must be separated by semicolons. If a path contains spaces, it must be delimited b double quotes.

Note that you should avoid storing your source files within the directory structure that was create for the JDK, as this can cause problems. Set up separate directories of your own to hold the sourc code for your programs and keep the code for each program in its own directory.

Assuming your program contains no errors, the compiler generates a bytecode program that is th equivalent of your source code. The compiler stores the bytecode program in a file with the san name as the source file, but with the extension `.class`. Java executable modules are always stored in file with the extension `.class`. By default, the `.class` file is stored in the same directory as the sourc file.

The command-line options I have introduced here are by no means all the options you hav available for the compiler. You are able to compile all of the examples in the book just knowing abou the options I have discussed. There is a comprehensive description of all the options within th documentation for the JDK. You can also specify the `-help` command-line option to get a summary the standard options you can use. To get a summary of the Java compiler options, enter the followin on the command line:

```
javac -help
```

To get usage information for the java application launcher, enter the following command:

```
java -help
```

If you are using some other product to develop your Java programs, you are probably using a muc more user-friendly, graphical interface for compiling your programs that doesn't involve enterin commands such as those shown in this section. However, the file name extensions for your source fi and the object file that results from it are just the same.

Executing a Java Application

To execute the bytecode program in the `.class` file with the Java interpreter in the JDK, you make th directory containing the `.class` file current and enter the command:

```
java MyProgram
```

Note that you use just the name `MyProgram` to identify the program, not the name of the file that th compiler generates, `MyProgram.class`. It is a common beginner's mistake to use the latter by analogy wi the compile operation. If you put a `.class` file extension on `MyProgram`, your program won't execute, an you get an error message:

```
Exception in thread "main" java.lang.NoClassDefFoundError: MyProgram/class
```

Although the `javac` compiler expects to find the name of a *file* that contains your source code, the `ja` interpreter expects the name of a *class* (and that class must contain a `main()` method, as I explain lat in this chapter). The class name is `MyProgram` in this case. The `MyProgram.class` file contains the compile `MyProgram` class. I explain what a class is shortly.

The `-enableassertions` option is necessary for programs that use *assertions*, and because you u assertions after you have learned about them it's a good idea to get into the habit of always using th option. You can abbreviate the `-enableassertions` option to `-ea` if you want. The previous command wi assertions enabled is:

```
java -enableassertions MyProgram
```

If you want to override an existing `CLASSPATH` definition, the option is the same as with the compile You can also abbreviate `--classpath` to `-cp` with the compiler or the Java interpreter. Here's how th command looks using that abbreviation:

```
java -ea -cp . MyProgram
```

To execute your program, the Java interpreter analyzes and then executes the bytecode instruction The JVM behaves identically in all computer environments that support Java, so you can be sure yo program is completely portable. As I already said, your program runs just as well on a Linux Jav implementation as it runs on an implementation for Microsoft Windows, Solaris, or any oth operating system that supports Java. (Beware of variations in the level of Java supported, thoug Some environments lag a little, so implementations supporting the current JDK version might b available later than under Windows or Solaris.)

Executing an Applet

The Java compiler in the JDK compiles both applications and applets. However, an applet is n executed in the same way as an application. You must embed an applet in a web page before it can b run. You can then execute it either within a Java-enabled web browser, or by using the `appletviewer`, bare-bones browser provided as part of the JDK. It is a good idea to use the `appletviewer` to run apple while you are learning. This ensures that if your applet doesn't work, it is almost certainly your co that is the problem, rather than some problem of Java integration with the browser.

If you have compiled an applet and included it in a web page stored as `MyApplet.html` in the curre directory that contains the `.class` file, you can execute it by entering the following command:

```
appletviewer MyApplet.html
```

So how do you put an applet in a web page?

The Hypertext Markup Language

The *Hypertext Markup Language*, or *HTML* as it is commonly known, is used to define a web pag When you define a web page as an HTML document, it is stored in a file with the extension `.html`. A HTML document consists of a number of elements, and each element is identified by *tags*. Th document begins with `<html>` and ends with `</html>`. These delimiters, `<html>` and `</html>`, are tags, ar each element in an HTML document should be enclosed between a similar pair of tags between ang brackets. All element tags are case-insensitive, so you can use uppercase or lowercase, or even mixture of the two. Here is an example of an HTML document consisting of a title and some oth text:

```
<html>
  <head>
    <title>This is the title of the document</title>
  </head>
  <body>
    You can put whatever text you like here. The body of a document can contain
    all kinds of other HTML elements, including <B>Java applets</B>.
    Note how each element always begins with a start tag identifying the
    element, and ends with an end tag that is the same as the start tag but
    with a slash added. The pair of tags around 'Java applets' in the previous
    sentence will display the text as bold.
  </body>
</html>
```

There are two elements that can appear directly within the `<html>` element, a `<head>` element and `<body>` element, as in the example. The `<head>` element provides information about the document, and not strictly part of it. The text enclosed by the `<title>` element tags that appears here within the `<hea` element are displayed as the window title when the page is viewed.

Other element tags can appear within the `<body>` element, and they include tags for headings, list tables, links to other pages, and Java applets. There are some elements that do not require an end ta because they are considered to be empty. An example of this kind of element tag is `<hr>`, whic specifies a horizontal rule, a line across the full width of the page. You can use the `<hr>` tag to divide page and separate one type of element from another. Another is `<center>`, which centers the output fro the applet.

Adding an Applet to an HTML Document

For many element tag pairs, you can specify an *element attribute* in the starting tag that defin additional or qualifying data about the element. This is how a Java applet is identified in an `<apple` tag. Here is an example of how you might include a Java applet in an HTML document:

Available for download on Wrox.com

```
<html>
  <head>
    <title> A Simple Program </title>
  </head>
  <body>
    <hr>
    <applet code = "MyFirstApplet.class"  width = 350  height = 200 >
    </applet>
    <hr/>
  </body>
</html>
```

Directory "MyFirstAppl

The two bolded lines between tags for horizontal lines specify that the bytecodes for the applet a contained in the file `MyFirstApplet.class`. The name of the file containing the bytecodes for the applet specified as the value for the `code` attribute in the `<applet>` tag. The other two attributes, `width` and `heigh` define the width and height of the region on the screen that is used by the applet when it execute These always have to be specified to run an applet. Here is the Java source code for a simple applet:

Available for download on Wrox.com

```
import javax.swing.JApplet;
import java.awt.Graphics;

public class MyFirstApplet extends JApplet {

  public void paint(Graphics g) {
    g.drawString("To climb a ladder, start at the bottom rung.", 20, 90);

  }
}
```

Directory "MyFirstAppl

You might get the following warning message when you compile this program:

```
warning: [serial] serializable class MyFirstApplet has no definition
 of serialVersionUID
```

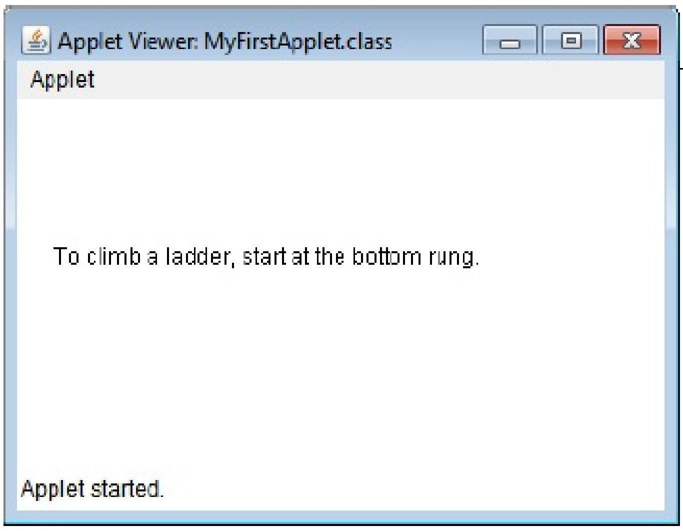
You can safely ignore this. You learn about serializable classes in Chapter 12.

Note that Java is case-sensitive. You can't enter `public` with a capital `P` — if you do, the progra won't compile. This applet just displays a message when you run it. The mechanics of how t message gets displayed are irrelevant here — the example is just to illustrate how an applet goes in an HTML page. If you compile this code and save the previous HTML page specification in the fi `MyFirstApplet.html` in the same directory as the Java applet `.class` file, you can run the applet usi `appletviewer` from the JDK with the command:

```
appletviewer MyFirstApplet.html
```

This displays the window shown in Figure 1-2:

FIGURE 1-2



```
To climb a ladder, start at the bottom rung.
```

```
Applet started.
```

Because the height and width of the window for the applet are specified in pixels, the physic dimensions of the `appletviewer` window depend on the resolution and size of your monitor.

OBJECT-ORIENTED PROGRAMMING IN JAVA

As I said at the beginning of this chapter, Java is an object-oriented language. When you use programming language that is not object-oriented, you must express the solution to every proble essentially in terms of numbers and characters — the basic kinds of data that you can manipulate the language. In an object-oriented language like Java, things are different. Of course, you still hav numbers and characters to work with — these are referred to as the *primitive data types* — but you c define other kinds of entities that are relevant to your particular problem. You solve your problem i terms of the entities or objects that occur in the context of the problem. This not only affects how program is structured, but also the terms in which the solution to your problem is expressed.

If your problem concerns baseball players, your Java program is likely to have `BaseballPlayer` objec in it; if you are producing a program dealing with fruit production in California, it may well hav objects that are `Oranges` in it. Apart from seeming to be an inherently sensible approach to constructi programs, object-oriented programs are usually easier to understand.

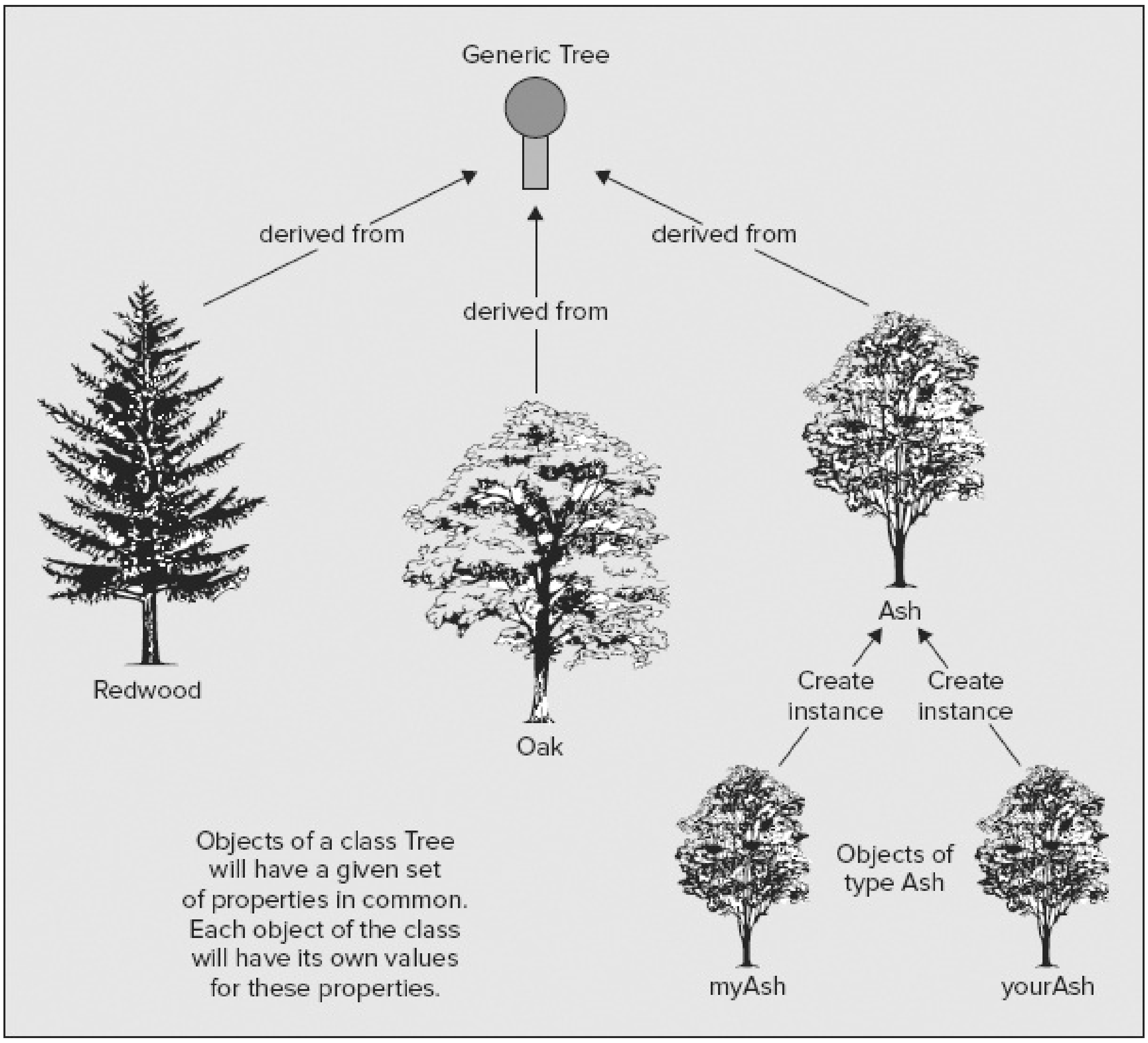
In Java almost everything is an object. If you haven't delved into object-oriented programmi before, or maybe because you have, you may feel this is a bit daunting. But fear not. Objects in Ja are particularly easy. So easy, in fact, that you are going to start out by understanding some of t ideas behind Java objects right now. In that way you can be on the right track from the outset.

This doesn't mean you are going to jump in with all the precise nitty-gritty of Java that you need f describing and using objects. You are just going to get the concepts straight at this point. You do thi by taking a stroll through the basics using the odd bit of Java code where it helps the ideas along. A the code that you use here is fully explained in later chapters. Concentrate on understanding the notio of objects first. Then you can ease into the specific practical details as you go along.

So What Are Objects?

Anything can be thought of as an object. Objects are all around you. You can consider `Tree` to be particular class of objects: trees in general. The notion of a `Tree` in general is a rather abstract conce — although any tree fits the description, it is more useful to think of more specific types of tre Hence, the Oak tree in my yard which I call `myOak`, the Ash tree in your yard which you ca `thatDarnedTree`, and a `generalSherman`, the well-known redwood, are actual instances of specific types of tre ~~subclasses of `Tree` that in this case happen to be `Oak`, `Ash`, and `Redwood`. Note how we drop into the~~ jargo here — *class* is a term that describes a specification for a collection of objects with commo properties. Figure 1-3 shows some classes of trees and how you might relate them.

FIGURE 1-3



Generic Tree
derived from
derived from
derived from
Redwood
Oak
Ash
Create instance
Create instance
Objects of a class Tree will have a given set of properties in common. Each object of the class will have its own values for these properties.
Objects of type Ash
myAsh
yourAsh

A class is a specification, or blueprint — expressed as a piece of program code — that defines wh goes to make up a particular sort of object. A subclass is a class that inherits all the properties of th parent class, but that also includes extra specialization. Particular classes of `Tree`, such as `Oak` or `As` have all the characteristics of the most general type, `Tree`; otherwise, they could not be considered to b such. However, each subclass of `Tree`, such as `Oak`, has its own characteristics that differentiate o objects from other types of `Tree`.

Of course, you define a class specification to fit what you want to do in your application contex There are no absolutes here. For my trivial problem, the specification of a `Tree` class might just consi of its species name and its height. If you are an arboriculturalist, then your problem with trees ma require a much more complex class, or more likely a set of classes, that involves a mass of arbore characteristics.

Every object that your program uses must have a corresponding class definition somewhere f objects of that type. This is true in Java as well as in other object-oriented languages. The basic idea a class in programming parallels that of classifying things in the real world. It is a convenient an well-defined way to group things together.

An *instance* of a class is a technical term for an existing object of that class. `Ash` is a specification f

a type of object, and `yourAsh` representing a particular tree is an object constructed to that specificatio
~~`yourAsh` is an instance of the class `Ash`. After you have a class defined you can create objects~~, o
instances, of that class. This raises the question of what differentiates an object of a given class fro
an object of another class, an `Ash` class object, say, from a `Redwood` object. In other words, what sort
information defines a class?

What Defines a Class of Objects?

You may have already guessed the answer. A class definition identifies all the parameters that defin
an object of that particular class type, at least, so far as your needs go. Someone else might define th
class differently, with a larger or smaller set of parameters to define the same sort of object — it a
depends on what you want to do with the class. You decide what aspects of the objects you include t
define that particular class of object, and you choose them depending on the kinds of problems th
you want to address using the objects of the class. Let's think about a specific class of objects.

If you were defining a class `Hat`, for example, you might use just two parameters in the definitio
You could include the type of hat as a string of characters such as `"Fedora"` or `"Baseball cap"` and its si
as a numeric value. The parameters that define an object of a class are referred to as *instan
variables* or *attributes* of a class, or class *fields*. The instance variables can be basic types of data su
as numbers, but they can also be other class objects. For example, the name of a `Hat` object could be
type `String` — the class `String` defines objects that are strings of characters.

Of course there are lots of other things you could include to define a `Hat` if you wanted to, `color`, f
example, which might be another string of characters such as `"Blue."` To specify a class you just deci
what set of attributes meet your requirements, and those are what you use. This is called *da
abstraction* in the parlance of the object-oriented aficionado because you just abstract the attribut
you want to use from the myriad possibilities for a typical object.

In Java the definition of the class `Hat` would look something like this:

```
class Hat {
  // Stuff defining the class in detail goes here.
  // This could specify the name of the hat, the size,
  // maybe the color, and whatever else you felt was necessary.
}
```

WARNING *Because the word* `class` *has this special role in Java it is called a* keyword, *and it is reserved for use only in this context. There are lots of other keywords in Java that you pick up as we go along. You need to remember that you must not use any of keywords for any other purposes. If you want to know what they all are, the complete set is in Appendix A.*

The name of the class follows the word `class`, and the details of the definition appear between th
curly braces.

I'm not going into the detail of how the class `Hat` is defined because you don't need it at this poi
The lines appearing between the braces are not code; they are actually *program comments* becau
they begin with two successive forward slashes. The compiler ignores anything on a line that follo
two successive forward slashes, so you can use this to add explanations to your programs. Generall
the more useful comments you can add to your programs, the better. You see in Chapter 2 that the
are other ways you can write comments in Java.

Each object of your class has a particular set of values defined that characterize that particul