

Part I: Fundamentals of Programming

By now you have heard a lot about Java and are anxious to start writing Java programs. The first part of the book is a stepping stone that will prepare you to embark on the journey of learning Java. You will begin to know Java and will develop fundamental programming skills. Specifically, you will learn how to write simple Java programs with primitive data types, control statements, methods, and arrays.

Chapter 1 Introduction to Java and JBuilder 4

Chapter 2 Primitive Data Types and Operations

Chapter 3 Control Statements

Chapter 4 Methods

Chapter 5 Arrays

Introduction to Java and JBuilder 4

Objectives

- Learn about Java and its history.
- Understand the relationship between Java and the World Wide Web.
- Become familiar with JBuilder 4.
- Create Java projects, compile and run Java programs with JBuilder 4.
- Understand Java runtime environment.
- Write a simple Java application.

Introduction

By now you have heard quite a lot about the exciting Java programming language. It must seem as if Java is everywhere! Your local bookstores are filled with Java books. There are articles about Java in every major newspaper and magazine. It is impossible to read a computer magazine without coming across the magic word *Java*. You must be wondering why Java is so hot. The answer is that it enables users to deploy applications on the Internet. In fact, this is its main distinguishing characteristic. The future of computing will be profoundly influenced by the Internet, and Java promises to remain a big part of that future. Java is *the* Internet programming language.

You are about to begin an exciting journey, learning a powerful programming language. Java is cross-platform, object-oriented, network-based, and multimedia-ready. After its inception in May 1995, Java quickly became an attractive language for developing Internet applications. This chapter introduces Java and its programming features, followed by a simple example of Java applications.

The History of Java

Java was developed by a team led by James Gosling at Sun Microsystems, a company best known for its Sun workstations. Originally called *Oak*, it was designed in 1991 for use in embedded consumer electronic applications. In 1995, renamed *Java*, it was redesigned for developing Internet applications. Java programs can be embedded in HTML pages and downloaded by Web browsers to bring live animation and interaction to Web clients.

The power of Java is not limited to Web applications, for it is a general-purpose programming language. It has full programming features and can be used to develop standalone applications. Java is inherently object-oriented. Although many object-oriented languages began strictly as procedural languages, Java was designed from the start to be object-oriented. Object-oriented programming (OOP) is a popular programming approach that is replacing traditional procedural programming techniques.

NOTE: One of the central issues in software development is how to reuse code. Object-oriented programming provides great flexibility, modularity, clarity, and reusability through method abstraction, class abstraction, and class inheritance—all of which you'll learn about in this book.

Characteristics of Java

Java has gained enormous popularity. Java's rapid rise and wide acceptance can be traced to its design and programming features, particularly its promise that you can write a program once and run the program anywhere. As stated in the Java-language white paper by Sun, Java is *simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, and dynamic*. Let's analyze these often-used buzzwords.

Java Is Simple

No language is simple, but Java is a bit easier than the popular object-oriented programming language C++, which was the dominant software-development language before Java. Java is partially modeled on C++, but greatly simplified and improved. For instance, pointers and multiple inheritance often make programming complicated. Java replaces the multiple inheritance in C++ with a simple language construct called an *interface*, and eliminates pointers.

Java uses automatic memory allocation and garbage collection, whereas C++ requires the programmer to allocate memory and collect garbage. Also, the number of language constructs is small for such a powerful language. The clean syntax makes Java programs easy to write and read. Some people refer to Java as "C++-ish" because it is like C++, but with more functionality and fewer negative aspects.

Java Is Object-Oriented

Computer programs are instructions to the computers. You tell a computer what to do through programs. Without programs, a computer is an empty machine. Computers do not understand human languages, so you need to use computer languages to communicate with computers. There are over one hundred programming languages. The most popular languages are:

- COBOL (COmmon Business Oriented Language)
- FORTTRAN (FORmula TRANslation)
- BASIC (Beginner All-purpose Symbolic Instructional Code)
- Pascal (Named for Blaise Pascal)
- Ada (Named for Ada Lovelace)
- C (Whose developer designed B first)
- Visual Basic (Basic-like visual language developed by Microsoft)
- Delphi (Pascal-like visual language developed by Borland)
- C++ (an object-oriented language, based on C)

Each language was designed with a specific purpose. COBOL was designed for business applications and now is used primarily for business data processing. FORTRAN was designed for mathematical computations and is used mainly for numeric computations. BASIC, as its name suggests, was designed to be learned and used easily. Pascal was designed to be a simple structural programming language. Ada was developed at the direction of the Department of Defense, and is mainly used in defense projects. C is popular among system software projects like writing compilers and operating systems. Visual Basic and Delphi are for rapid application development. C++ is the C language with object-oriented features.

All of these languages except C++ are known as *procedural programming languages*. Software systems developed using procedural programming languages are based on the paradigm of procedures. Object-oriented programming models the real world in terms of objects. Everything in the world can be modeled as an object. A circle is an object, a person is an object, and a window's icon is an object. Even a mortgage can be perceived as an object. A Java program is object-

oriented because programming in Java is centered on creating objects, manipulating objects, and making objects work together.

Part I, "Fundamentals of Programming," introduces primitive data types and operations, control statements, methods, and arrays. These are the fundamentals for all programming languages. You will learn object-oriented programming in Part II, "Object-Oriented Programming." Object-oriented programming provides great flexibility, modularity, and reusability. For years, object-oriented technology was perceived as elitist, requiring a substantial investment in training and infrastructure. Java has helped object-oriented technology enter the mainstream of computing. Its simple, clean syntax makes programs easy to write and read. Java programs are quite expressive in terms of designing and developing applications.

Java Is Distributed

Distributed computing involves several computers working together on a network. Java is designed to make distributed computing easy. Since networking capability is inherently integrated into Java, writing network programs is like sending and receiving data to and from a file. For example, Figure 1.1 shows three programs running on three different systems; the three programs communicate with each other to perform a joint task.

*****Insert Figure 1.1 (Same as Figure 1.1 in the 3rd Edition, p6)**

Figure 1.1

Java programs can run on different systems that work together.

Java Is Interpreted

You need an interpreter to run Java programs. The programs are compiled into Java Virtual Machine code called *bytecode*. The bytecode is machine-independent and can run on any machine that has a Java interpreter, as shown in Figure 1.2.

*****Insert Figure 1.2 (Same as Figure 1.2 in the 3rd Edition, p7)**

Figure 1.2

The Java interpreter executes Java bytecode.

Usually, a compiler, such as a C++ compiler, translates a

program in a high-level language to machine code. The code can only run on the native machine. If you run the program on other machines, it has to be recompiled on the native machine. For instance, if you compile a C++ program in Windows, the executable code generated by the compiler can only run on the Windows platform. With Java, you compile the source code once, and the bytecode generated by a Java compiler can run on any platform with a Java interpreter. The Java interpreter translates the bytecode into the machine language of the target machine.

Java Is Robust

Robust means *reliable*. No programming language can ensure complete reliability. Java puts a lot of emphasis on early checking for possible errors, because Java compilers can detect many problems that would first show up at execution time in other languages. Java has eliminated certain types of error-prone programming constructs found in other languages. It does not support pointers, for example, thereby eliminating the possibility of overwriting memory and corrupting data.

Java has a runtime exception-handling feature to provide programming support for robustness. Java forces the programmer to write the code to deal with exceptions. Java can catch and respond to an exceptional situation so that the program can continue its normal execution and terminate gracefully when a runtime error occurs.

Java Is Secure

As an Internet programming language, Java is used in a networked and distributed environment. If you download a Java applet (a special kind of program) and run it on your computer, it will not damage your system because Java implements several security mechanisms to protect your system against harm caused by stray programs. The security is based on the premise that *nothing should be trusted*.

Java Is Architecture-Neutral

The most remarkable feature of Java is that it is *architecture-neutral*, also known as platform-independent. With a Java Virtual Machine (JVM), you can write one program that will run on any platform, as shown in Figure 1.3.

*****Insert Figure 1.3 (Same as Figure 1.3 in the 3rd Edition, p8)**

Figure 1.3

Java bytecode can be executed on any platform that has a JVM.

Java's initial success stemmed from its Web programming capability. You can run Java applets from a Web browser, but Java is for more than just writing Web applets. You can also run standalone Java applications directly from operating systems, using a Java interpreter. Today, software vendors usually develop multiple versions of the same product to run on different platforms (Windows, OS/2, Macintosh, and various UNIX, IBM AS/400, and IBM mainframes). Using Java, developers need to write only one version to run on all platforms.

Java Is Portable

Java programs can be run on any platform without being recompiled, making them very portable. Moreover, there are no platform-specific features in the Java language. In some languages, such as Ada, the largest integer varies on different platforms. But in Java, the size of the integer is the same on every platform, as is the behavior of arithmetic. The fixed size of numbers makes the program portable.

The Java environment is portable to new hardware and operating systems. In fact, the Java compiler itself is written in Java.

Java's Performance

Java's performance is sometimes criticized. The execution of the bytecode is never as fast as it would be with a compiled language, such as C++. Because Java is interpreted, the bytecode is not directly executed by the system, but is run through the interpreter. However, the speed is more than adequate for most interactive applications, where the CPU is often idle, waiting for input or for data from other sources.

CPU speed has increased dramatically in the past few years, and this trend will continue. There are many ways to improve performance. Users of the earlier Sun Java Virtual Machine (JVM) have certainly noticed that Java is slow. However, the new JVM is significantly faster. The new JDK uses the technology known as just-in-time compilation, as shown in Figure 1.4. It compiles bytecode into native machine code, stores the native code, and reinvokes the native code when its bytecode is executed. Sun recently developed the Java HotSpot Performance Engine, which includes a compiler for optimizing the frequently used code. The HotSpot Performance Engine can be plugged into a JVM to dramatically boost its performance.

*****Insert Figure 1.4 (Same as Figure 1.4 in the 3rd Edition, p9)**

Figure 1.4

Just-in-time compiler compiles bytecode to the native machine code.

Java Is Multithreaded

Multithreading is a program's capability to perform several tasks simultaneously. For example, downloading a video file while playing the video would be considered multithreading. Multithread programming is smoothly integrated in Java, whereas in other languages, you have to call operating system-specific procedures to enable multithreading.

Multithreading is particularly useful in graphical user interface (GUI) and network programming. In GUI programming, there are many things going on at the same time. A user can listen to an audio recording while surfing a Web page. In network programming, a server can serve multiple clients at the same time. Multithreading is a necessity in multimedia and network programming.

Java Is Dynamic

Java was designed to adapt to an evolving environment. You can freely add new methods and properties to a class without affecting its clients. For example, in the Circle class, you can add a new data property to indicate the color of the circle, and a new method to obtain the circumference of the circle. The original client program that uses the Circle class remains the same.

Java and the World Wide Web

The World Wide Web is an electronic information repository that can be accessed on the Internet from anywhere in the world. You can use the Web to book a hotel room, buy an airline ticket, register for a college course, download the *New York Times*, chat with friends, and listen to live radio. There are countless activities you can do on the Internet. Today, many people spend a good part of their computer time surfing the Web for fun and profit.

The Internet is the infrastructure of the Web. The Internet has been around for more than thirty years, but has only recently become popular. The colorful World Wide Web and sophisticated Web browsers are the major reason for its popularity.

The primary authoring language for the Web is the Hypertext Markup Language (HTML). HTML is a markup language: a simple language for laying out documents, linking documents on the Internet, and bringing images, sound, and video alive on the Web. However, it cannot interact with the user except through simple forms. Web pages in HTML essentially are static and flat.

Java programs can be run from a Web browser. Because Java is a full-blown programming language, you can make programs responsive and interactive with users. Java programs that run from a Web page are called *applets*. Applets use a modern graphical user interface, including buttons, text fields, text areas, option buttons, and so on. Applets can respond to user events, such as mouse movements and keystrokes.

Figure 1.5 shows an applet in an HTML page, and Figure 1.6 shows the HTML source. The source contains an applet tag, which specifies the Java applet. The HTML source can be seen by choosing View, Page Source from Netscape 6 Web Browser.

TIP: For a demonstration of Java applets, visit **www.javasoft.com/applets/**. This site provides a rich Java resource as well as links to other cool applet demo sites. **www.javasoft.com** is the official Sun Java Web site.

*****Insert Figure 1.5 (Same as Figure 1.5 in the 3rd Edition, p11)**

Figure 1.5

A Java applet for computing a mortgage is embedded in an HTML page. The user can compute the mortgage payment by using this applet.

*****Insert Figure 1.6 (Same as Figure 1.6 in the 3rd Edition, p11)**

Figure 1.6

The HTML source that contains the applet shows the tag specifying the Java applet.

The Java Language Specification

Computer languages have strict rules of usage. You must follow the rules when writing programs or else the computer will be unable to understand them. Sun Microsystems, the originator of Java, intends to retain control of this important new computer language—and for a very good reason:

to prevent it from losing its unified standards. The complete reference of Java standards can be found in *Java Language Specification, Second Edition*, by James Gosling, Bill Joy, and Guy Steele (Addison-Wesley, 2000).

The Java language specification is a technical definition of the language that includes syntax, constructs, and the *application programmer interface* (API), which contains predefined classes. The Java language syntax and constructs are stable, but the API is still expanding. At the JavaSoft Web site (www.javasoft.com), you can view the language specification and the latest version and updates to the Java API.

Sun releases each version of Java with a Java Development Toolkit, known as JDK. This is a primitive command-line tool set that includes software libraries, a compiler for compiling Java source code, an interpreter for running Java bytecode, and an applet viewer for testing Java applets, as well as other useful utilities.

NOTE: Sun announced the Java 2 name in December 1998, just as it released JDK 1.2. Java 2 is the overarching brand that applies to the latest Java technology. JDK 1.2 was the first version of the Java Development Toolkit that supported the Java 2 technology. This book is based on the latest version, JDK 1.3. The official name for JDK 1.3 is Java 2 SDK v 1.3. SDK stands for Software Development Toolkit. Since most Java programmers are familiar with the name JDK, this book uses the names Java SDK and JDK interchangeably.

Java Development Tools and JBuilder

JDK consists of a set of separate programs for developing and testing Java programs, each of which is invoked from a command line. Besides JDK, there are more than a dozen Java development tools on the market today. The major development tools are

JBuilder by Borland (www.borland.com)

Forte by Sun (www.javasoft.com)

Visual Café by WebGain (www.webgain.com)

Visual Age for Java by IBM (www.ibm.com)

These tools provide an *integrated development environment* (IDE) for developing Java programs. The basic functions of these tools are very similar. Editing, compiling, building,

debugging, and online help are integrated in one graphical user interface. Just enter source code in one window, or open an existing file in a window, then click a button, menu item, or function key to compile the source code. The use of development tools makes it easy and productive to develop Java programs. This book introduces Java programming with JBuilder 4.

JBuilder is easy to learn and easy to use. The JBuilder development team made a significant effort to simplify the user interface and make it easy to navigate through the programs, projects, classes, packages, and code elements. As a result, JBuilder has fewer windows than Microsoft Visual J++ and WebGain Visual Café. This makes JBuilder an ideal tool for beginners and for students who have little programming experience.

JBuilder 4, released in November 2001 by Borland, is a premier Java development tool for developing Java programs. Borland products are known to be "best of breed" in the Rapid Application Development tool market. Over the years, Borland has led the charge to create visual development tools like Delphi and C++ Builder. Borland is leading the way in Java development tools with JBuilder. JBuilder delivers 100% pure Java code and is one of the most popular Java tools used by professional Java developers. Many university students are using JBuilder Foundation to learn Java programming.

JBuilder 4 is written completely in Java. It uses the Swing component library, provided by JavaSoft (Sun Microsystems), as its foundation. This pure Java component library allows JBuilder to have a sophisticated user interface and remain 100% Java. The Swing library also gives the added benefit of a pluggable look-and-feel, which allows JBuilder to adjust its appearance to match the computer's native operating system UI without sacrificing the power and flexibility of the JBuilder UI. JBuilder 4 runs on the Windows 98, NT, and 2000, Linux, and Solaris platforms.

JBuilder 4 is available in three versions: JBuilder Foundation, JBuilder Professional, and JBuilder Enterprise.

[BL] The JBuilder Foundation is ideal for beginners to learn the basics of Java programming. The JBuilder Foundation is free for educational use and included in the companion CD-ROM. This book teaches Java programming with JBuilder 4 Foundation.

[BL] JBuilder Professional contains the essential components for developing Java applications and applets. It also contains the Borland JavaBeans components for creating database applications.

[BX] JBuilder Enterprise contains all the components in JBuilder Professional, plus support for creating distributed applications using CORBA, and for creating Java servlets, and Java Server Pages.

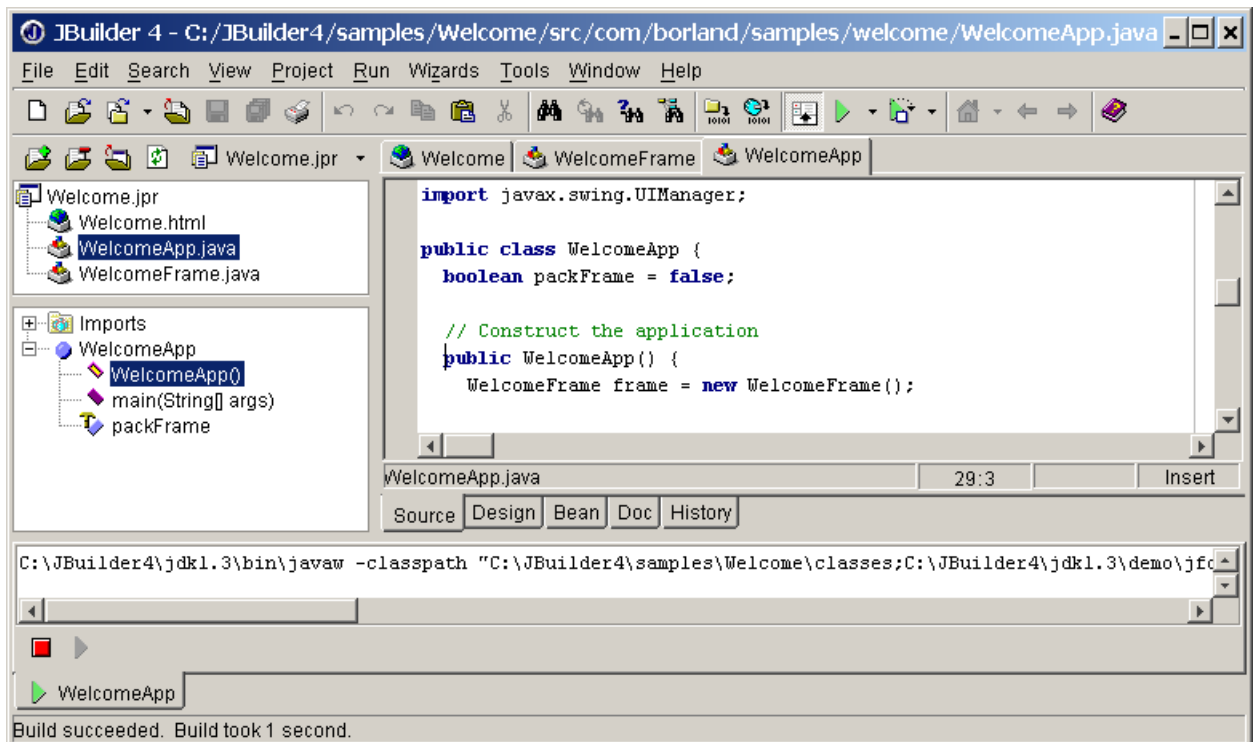
Getting Started with JBuilder 4

Assume you have successfully installed JBuilder Foundation on your machine. Start JBuilder from Windows, Linux, or Solaris. The main JBuilder user interface appears, as shown in Figure 1.7. If you don't see the Welcome project, choose Welcome Project (Sample) from the Help menu.

NOTE: For installing JBuilder 4 on Windows, Linux, and Solaris, please refer to the installation guide in the index.html file.

NOTE: The screen shots in this book are from JBuilder 4 Foundation. *If you use JBuilder 4 Professional, or JBuilder 4 Enterprise, your screen may look slightly different.*

***Insert Figure 1.7



***PD: Please leave some space on top and bottom for labels. Labels will be provided at Proof stage. Author.

Figure 1.7

The JBuilder user interface is a single window that performs functions for editing, compiling, debugging, and running programs.

The JBuilder user interface presents a single window for managing Java projects, browsing files, compiling, running, and debugging programs. This user interface is called *AppBrowser* in JBuilder 4. Note that the *AppBrowser* window and the main window are two separate windows in JBuilder 3, but are combined into one window in JBuilder 4.

Traditional IDE tools use many windows to accommodate various development tasks, such as editing, debugging, and browsing information. As a result, finding the window you need is often difficult. Because it is easy to get lost, beginners may be intimidated. For this reason, some new programmers prefer to use separate utilities, such as the JDK command line tools, for developing programs.

Borland is aware of the usability problem and has made significant efforts to simplify the JBuilder user interface. JBuilder introduces the *AppBrowser* window, which enables you to explore, edit, design, and debug projects all in one unified window.

The *AppBrowser* window primarily consists of the main menu, main toolbar, status bar, project pane, structure pane, and content pane.

The Main Menu

The main menu is similar to that of other Windows applications and provides most of the commands you need to use JBuilder, including those for creating, editing, compiling, running, and debugging programs. The menu items are enabled and disabled in response to the current context.

The Toolbar

The toolbar provides buttons for several frequently used commands on the menu bar. Clicking a toolbar is faster than using the menu bar. For some commands, you also can use function keys or keyboard shortcuts. For example, you can save a file in three ways:

- Select File, Save from the menu bar.
- Click the "save" toolbar button.
- Use the keyboard shortcut Ctrl+S.

TIP: You can display a label, known as *ToolTip*, for a button by pointing the mouse to the button without clicking.

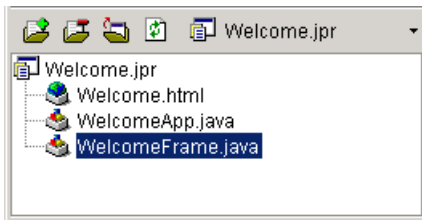
The Status Bar

The status bar displays a message to alert users to the operation status, such as file saved for the Save file command and compilation successful for the Compilation command.

The Project Pane

The *project pane* (known as the *navigation pane* in JBuilder 3) displays the contents of one or more projects opened in the AppBrowser. It consists of the following items, as shown in Figure 1.8.

*****Insert Figure 1.8**



*****PD: Need several labels for this figure. AU**

Figure 1.8

The project pane manages JBuilder projects.

- A small toolbar with four buttons (Add To Project, Remove From Project, Close Project, and Refresh).
- A drop-down list of all opened projects.
- A tree view of all the files that make up the active project.

The project pane shows a list of one or more files. In the case of the Project browser, you will see the project (.jpr) file first. Attached to that is a list of the files in the project. The list can include .java, .html, text, or image files. You select a file in the project pane by clicking it.

The content pane and the structure pane display information about the selected file. As you select different files in the project pane, each one will be represented in the content and structure panes.

The project pane shown in Figure 1.8 contains three files. The Add button is used to add new files to the project, and the Remove button to remove files from the project. For example, you can remove Welcome.html by selecting the file in the project pane and clicking the Remove button. You can then add the file back to the project as follows:

1. Click the Add button to display the Open dialog box shown in Figure 1.9.
2. Open Welcome.html. You will see Welcome.html displayed in the project pane.

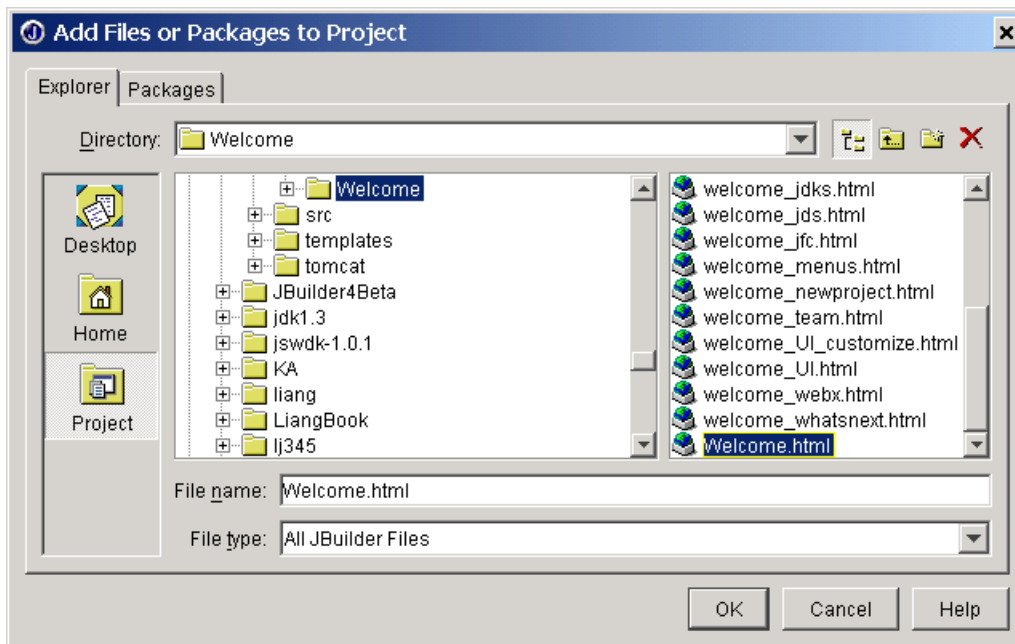


Figure 1.9

The Open dialog box enables you to open an existing file.

TIP: You can select multiple files by clicking the files with the CTRL key pressed, or select consecutive files with the SHIFT key pressed.

The Content Pane

The content pane displays all opened files as a set of tabs. To open a file in the content pane, double-click it in the

project pane. The content pane displays the detailed content of the selected file. The editor or viewer used is determined by the file's extension. If you click the WelcomeFrame.java file in the project pane, for example, you will see four tabs (Source, Design, Bean, and Doc) at the bottom of the content pane (see Figures 1.10 and 1.11). If you select the Source tab, you will see the JBuilder Java source code editor. This is a full-featured, syntax-highlighted programming editor.

If you select Welcome.html in the project pane, you will see the content pane become an HTML browser, as shown in Figure 1.10. If you choose the Source tab, you can view and edit the HTML code in the content pane, as shown in Figure 1.11.

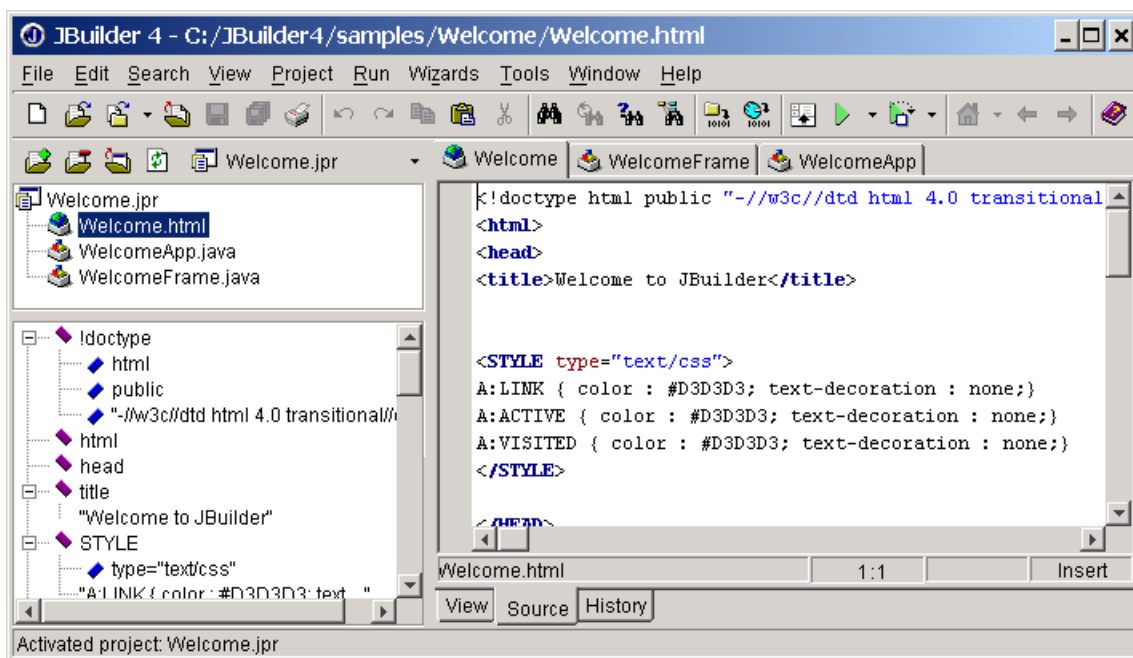


Figure 1.10

You can view HTML file in the content pane.

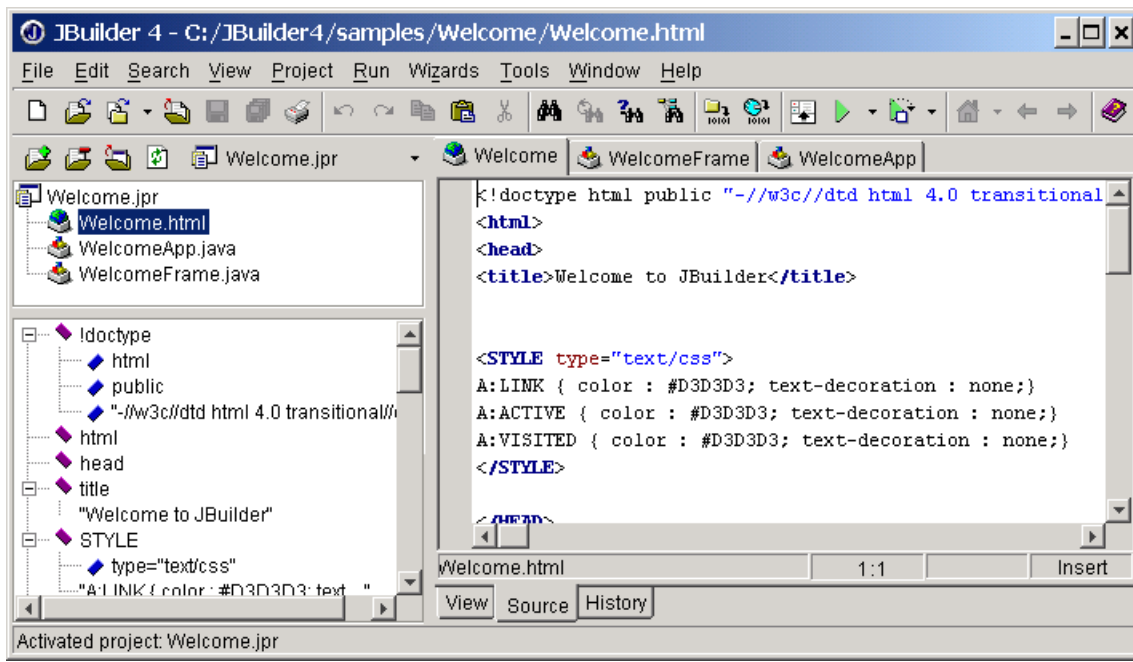


Figure 1.11

You can edit HTML files in the content pane.

The Structure Pane

The *structure pane* displays the structural information about the files you selected in the project pane. All the items displayed in the structure pane are in the form of a hierarchical indexed list. The expand symbol in front of an item indicates that it contains subitems. You can see the subitems by clicking on the expand symbol.

You also can use the structure pane as a quick navigational tool to the various structural elements in the file. If you select the WelcomeFrame.java file, for example, you see classes, variables, and methods in the structure pane. If you then click on any of those elements in the structure pane, the content pane will move to and highlight it in the source code.

If you click on the `jMenuFile` item in the structure pane, as shown in Figure 1.12, the content pane moves to and highlights the statement that defines the `jMenuFile` data field. This provides a much faster way to browse and find the elements of a .java file than scrolling through it.

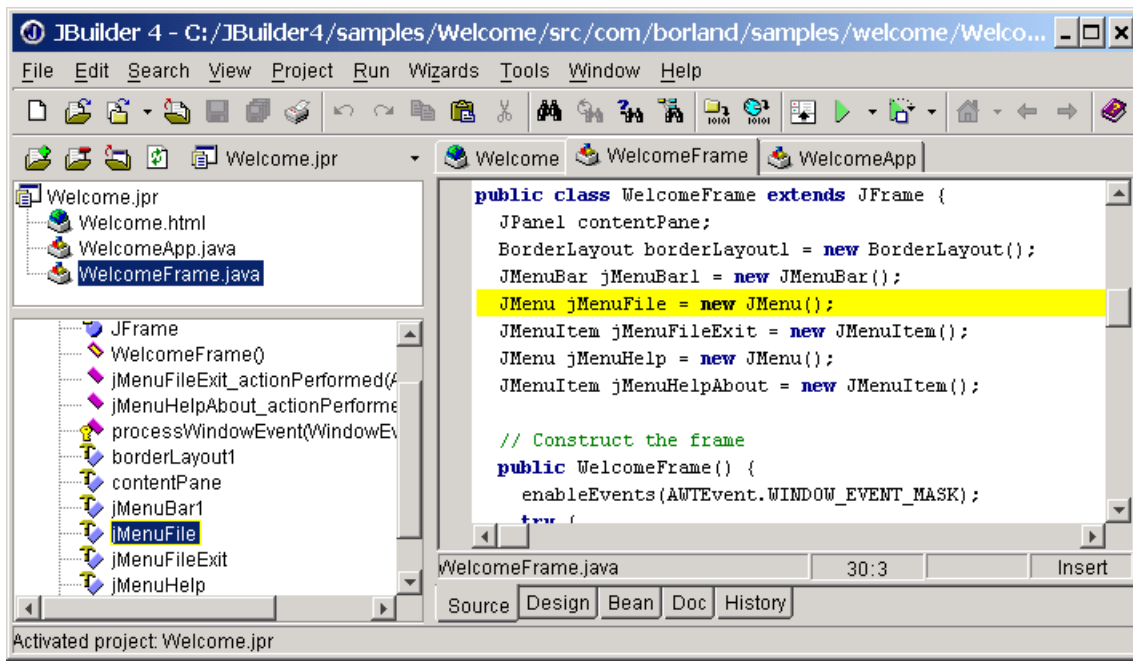


Figure 1.12

You can cruise through the source code from the structure pane.

Creating a Project

A project is like a holder that ties all the files together. The information about each JBuilder project is stored in a project file with a .jpr or .jpx file extension. The project file contains a list of all the files and project settings and properties. JBuilder uses this information to load and save all the files in the project and compile and run the programs. For simplicity, this book creates one project to hold all the examples in a chapter. This project is created as follows:

1. Choose File, New Project to bring up the Project wizard dialog box, as shown in Figure 1.13.
2. Type **chapter1** in the Project name field. You may choose either jpr or jpx for the project type. In this book, I choose jpr. Type **c:/example** in the Root path field, and **empty** the Project directory name field, Source directory name field, and Output directory name field. Type **bak** in the Backup directory name. Click Next to display Step 2 of 3 of the Project wizard, as shown in Figure 1.14.
3. Click Next to display Step 3 of 3 of the Project wizard, as shown in Figure 1.15.

4. Fill in the title, author, company, and description fields. These optional fields provide a description for the project.
5. Click Finish. The new project is displayed in the project pane, as shown in Figure 1.16. The Project wizard created the project file (chapter1.jpr) and an HTML file (chapter1.html), and placed them in **c:\example**. The project file stores the information about the project, and the HTML file is used to describe the project.

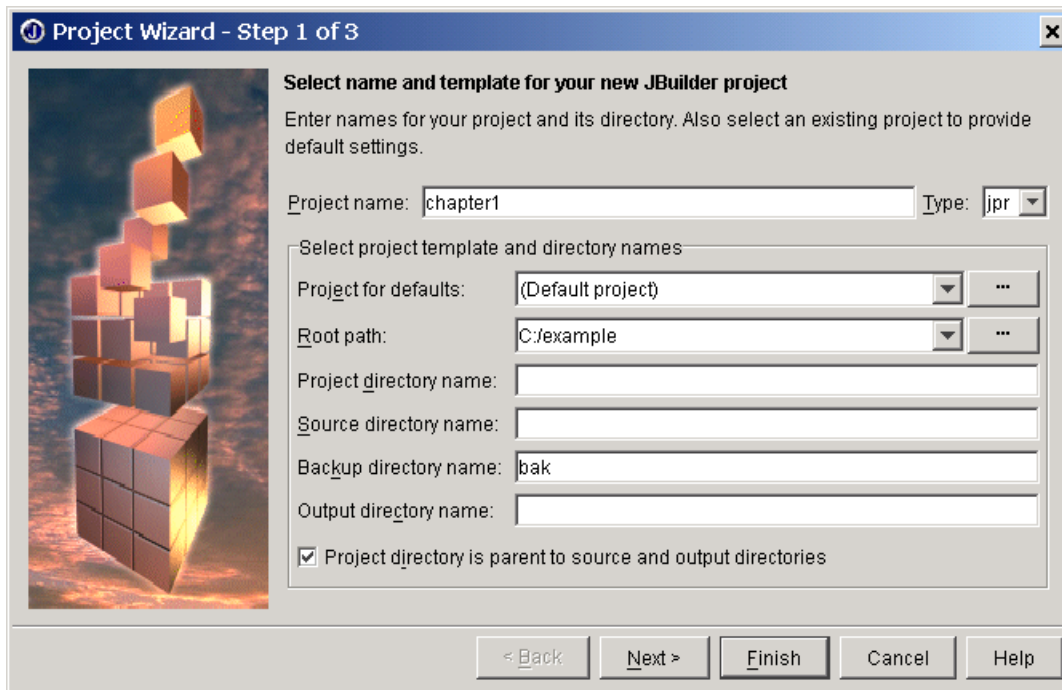


Figure 1.13

The Project wizard dialog box enables you to specify the project file with other optional information.

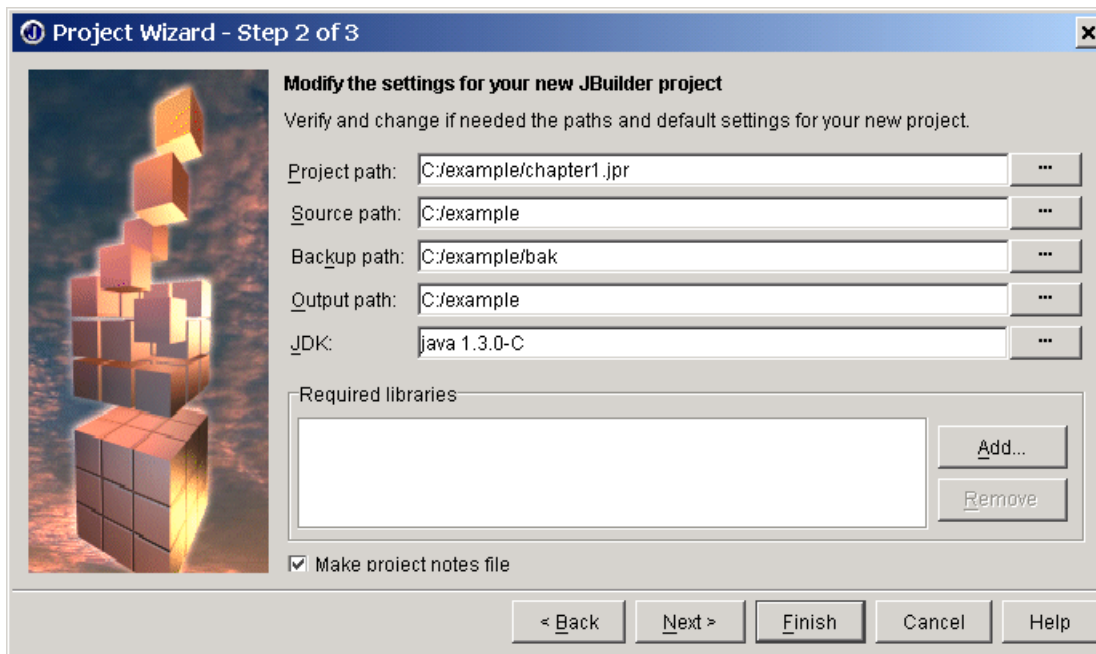


Figure 1.14

The Project wizard Step 2 of 3 enables you to modify project settings.

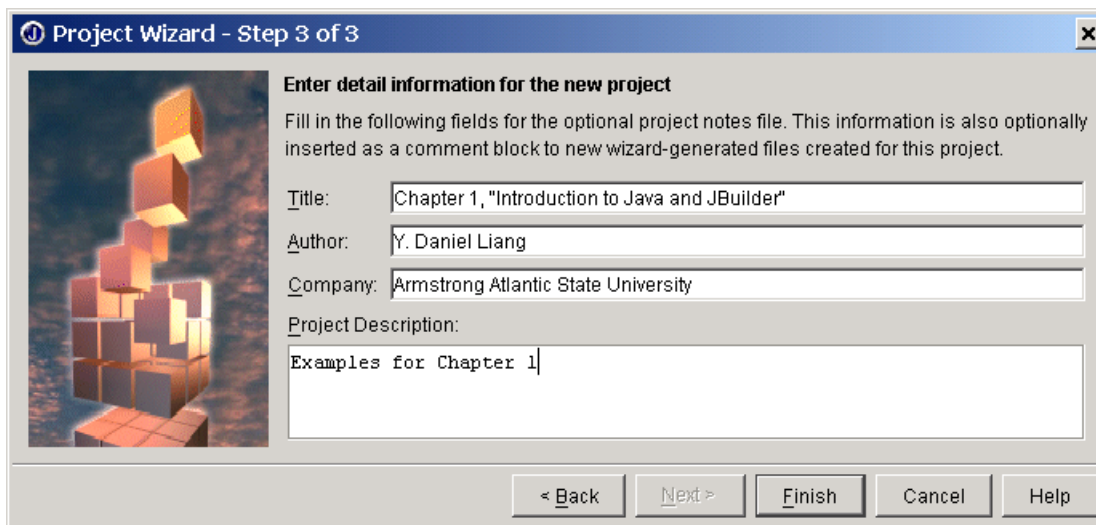


Figure 1.15

Project wizard Step 3 of 3 collects optional information for the project.

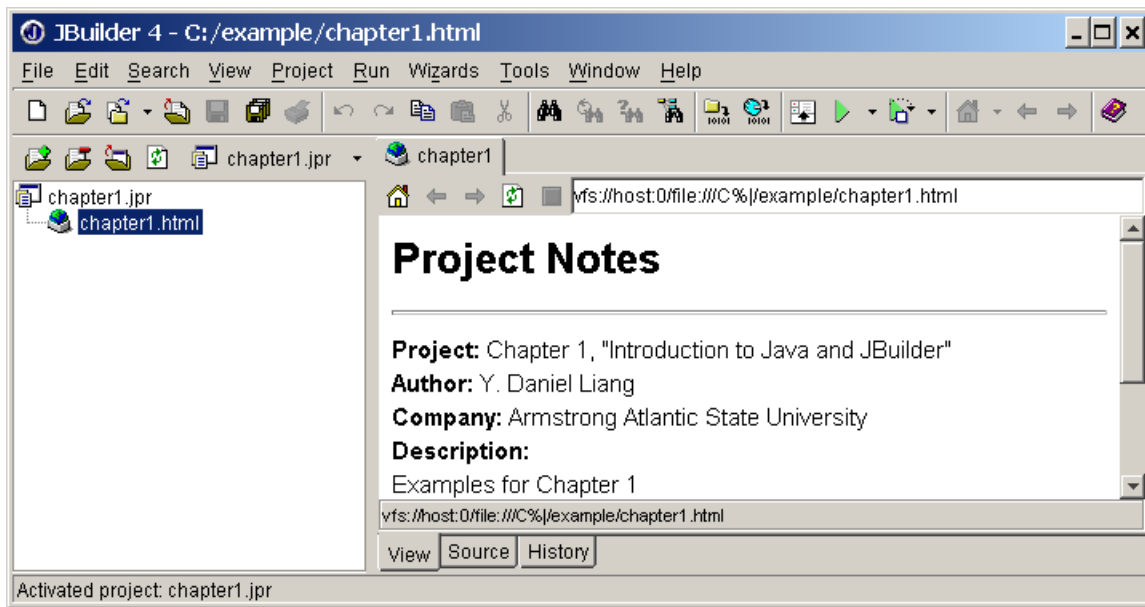


Figure 1.16

A new project is created with the .jpr file and .html file.

NOTE: JBuilder automatically generates many backup files. I use **bak** as the root directory for all these backup files so they can be easily located and removed.

CAUTION: Creating a project is a preliminary step before developing Java programs. Creating projects incorrectly is a common problem for new JBuilder users, which could lead to frustrating mistakes. You may create your project exactly as shown in this section, or change the word *example* to your name like *liang*, so that your project source path, and output path are `c:\liang` in Figure 1.14.

A Simple Java Program

Let's begin with a simple Java program that displays the message "Welcome to Java!" on the console.

Example 1.1 A Simple Java Program

This program shows how to write a simple Java application and demonstrates the compilation and execution of an application.

```
// Welcome.java: This application program prints Welcome to Java!
package chapter1;

public class Welcome
{
```



```

public static void main(String[] args)
{
    System.out.println("Welcome to Java!");
}

```

Example Review

In this program, `println("Welcome to Java!")` is actually the statement that prints the message. So why use the other statements in the program? Because computer languages have rigid styles and strict syntax, and you need to write code that the Java compiler understands.

The first line in the code is the `package` statement. Its purpose is for grouping and organizing classes. In this book, I will use the `package` statement to group classes chapter-by-chapter. So all the classes in *Chapter I* will have the package statement:

```
package chapter1;
```


Every Java program must have at least one class. Each class begins with a class declaration that defines data and methods for the class. In this example, the class name is `Welcome`.

The class contains a method named `main`. The `main` method in this program contains the `println` statement. The `main` method is invoked by the interpreter.

Creating a Java Program

There are many ways to create a Java program in JBuilder. This book will show you how to use various wizards to create certain types of Java programs. In this section, you will first learn how to create Java programs using the Class wizard.

The following are the steps to create a Java program for Example 1.1:

1. Open the `chapter1.jpr` project if it is not in the project pane. To open it, choose File, Reopen to display a submenu consisting of the most recently opened projects and files, as shown in Figure 1.17. Select the project if it is in the menu. Otherwise, choose File, Open to locate and open `chapter1.jpr`. The project file is the one with the  icon.

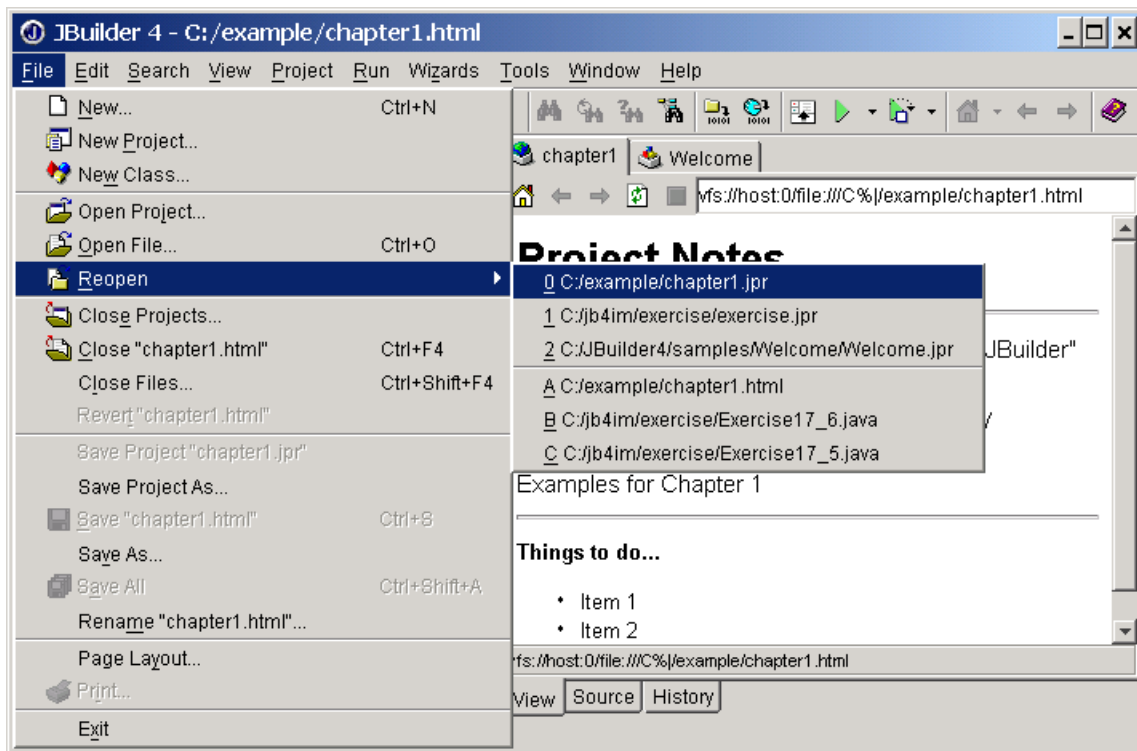


Figure 1.17

Recently used projects can be reopened by choosing File, Reopen.

2. Choose File, New Class to display the Class wizard, as shown in Figure 1.18.

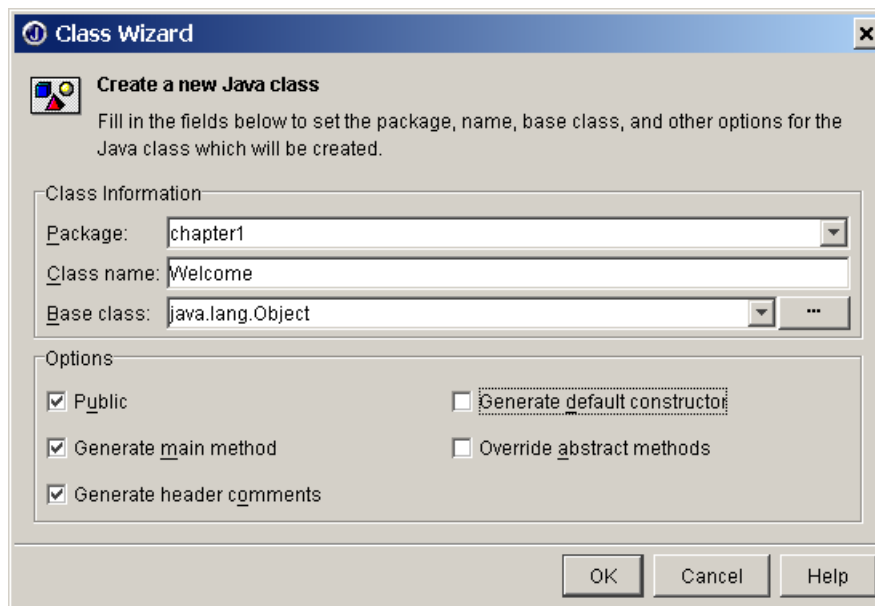


Figure 1.18

You can use the Class wizard to create a template for a new class.

3. In the Class wizard, type Welcome in the Class name field, and check the options Public, Generate main method, and Generate header comments in the Options section, as shown in Figure 1.18. Click OK to generate Welcome.java, as shown in Figure 1.19.

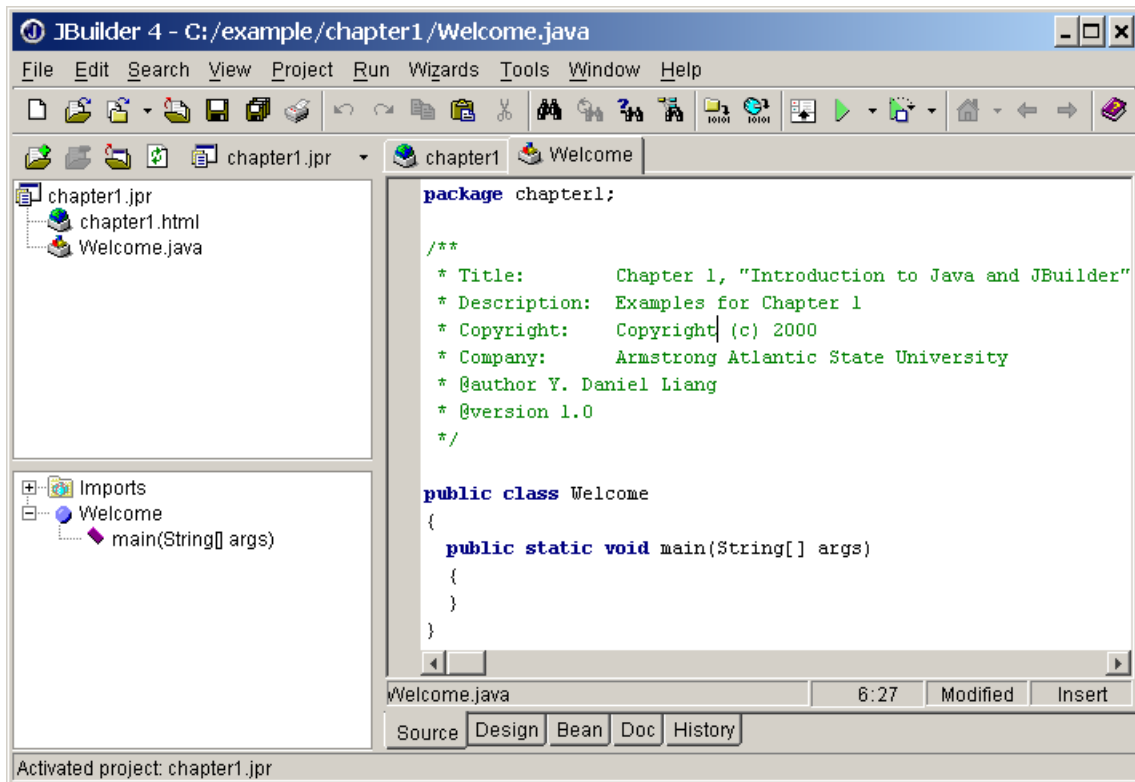


Figure 1.19

The program Welcome.java is generated by the Class wizard.

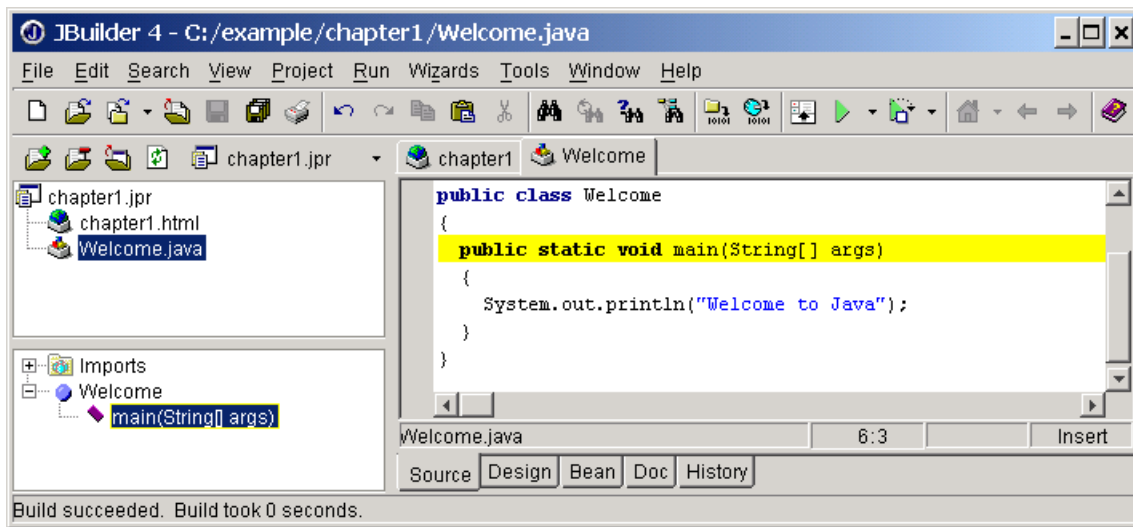


Figure 1.20

The program Welcome is typed in the content pane.

4. Type Example 1.1 in the content pane, as shown in Figure 1.20. Select File, Save All to save all your work. You should see a confirmation message in the status bar that indicates the files are saved.

NOTE: As you type, the code completion assistance may automatically come up to give you suggestions for completing the code. For instance, when you type a dot (.) after System and pause for a second, JBuilder displays a popup menu with suggestions to complete the code, as shown in Figure 1.21. You can then select from the menu to complete the code.

CAUTION: Java source programs are case-sensitive. It would be wrong, for example, to replace main in the program with Main. Program file names are case-sensitive on UNIX and generally not case-sensitive on PCs, but file names are case-sensitive in JBuilder.

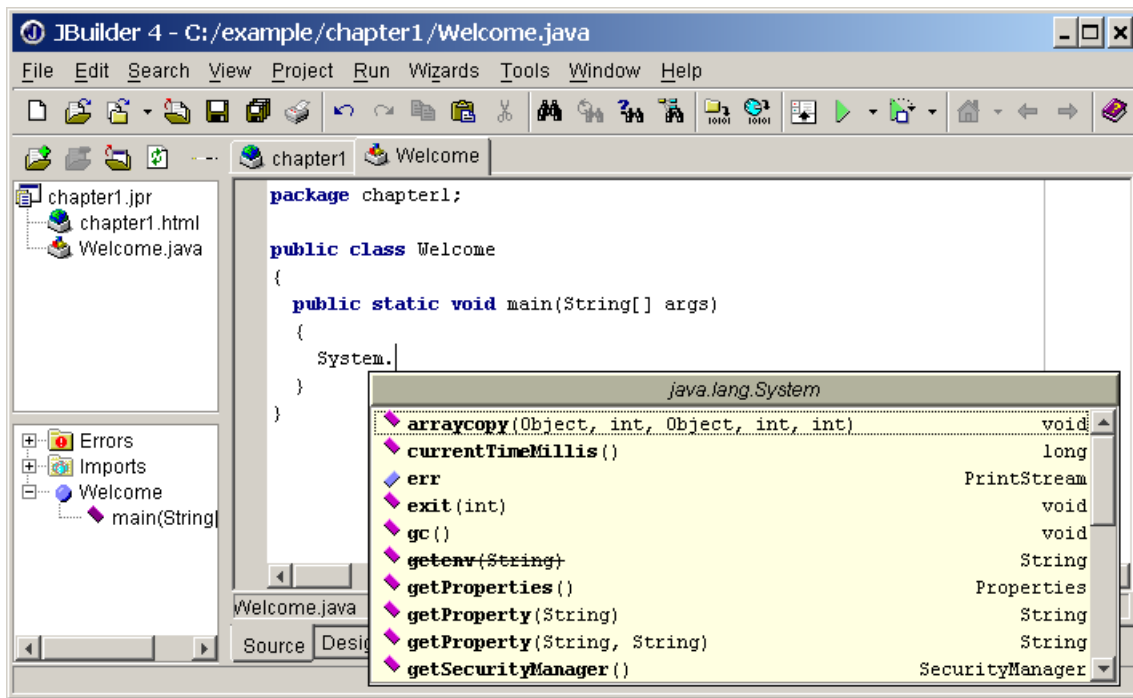


Figure 1.21

The Code Insight popup menu is automatically displayed to help you complete the code.

Compiling a Java Program

Before your program can be executed, you have to first create the program and then compile it. This process is iterative, as shown in Figure 1.22. If your program has compilation errors, you have to fix them by modifying the program, then recompile it. If your program has runtime errors or does not produce the correct result, you have to modify the program, recompile it, and execute it again.

*****Insert Figure 1.22 (Same as Figure 1.7 in the 3rd Edition, p14)**


Figure 1.22

The Java program development process consists of creating/modifying source code, compiling, and executing programs, in an iterative fashion.

The Java source file must end with the extension `.java` and should have the exact same name as the public class name. For example, the file for the source code in Example 1.1 should be named **Welcome.java**, since the public class name is Welcome.

To compile **Welcome.java**, use one of the following methods.
(Be sure that **Welcome.java** is selected in the project pane.)

[BL] Select Project, Make "Welcome.java" from the menu bar.

[BL] Click the Make toolbar button ().

[BX] Point to Welcome.java in the project pane, right-click the mouse button to display a popup menu (see Figure 1.23), and choose Make from the menu. (I found this method most useful.)

***Insert Figure 1.23

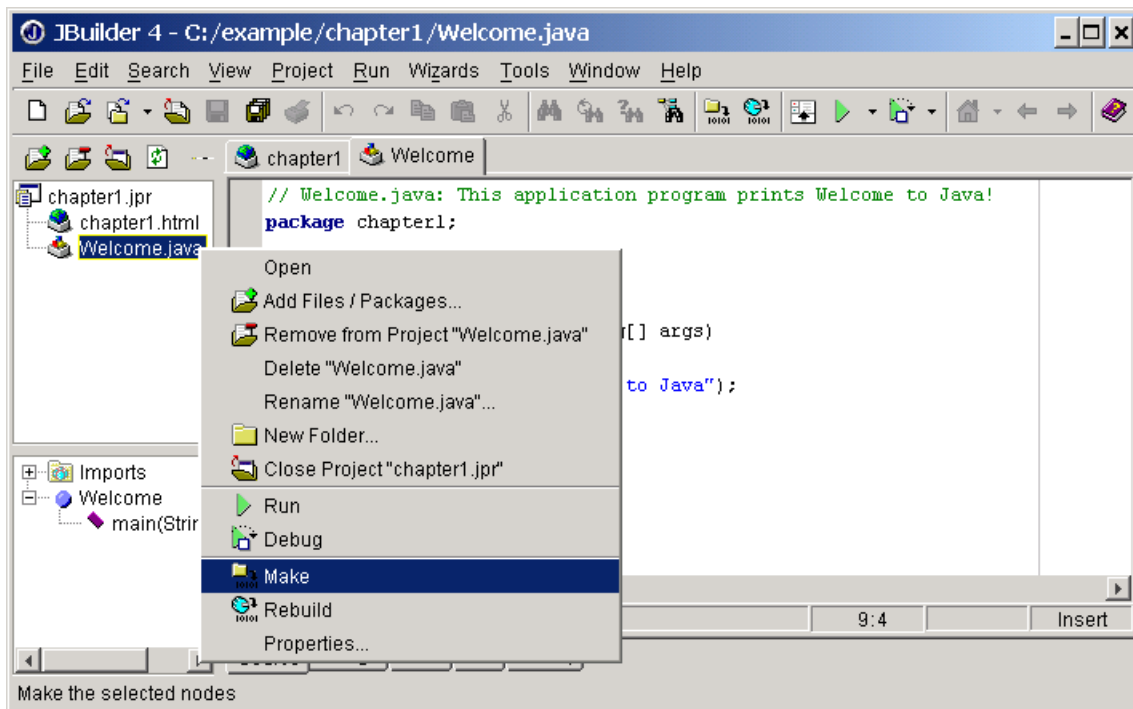


Figure 1.23

Point the mouse to the file in the project pane and right-click it to display a popup menu that contains the commands for processing the file.

The compilation status is displayed on the status bar. If there are no syntax errors, the compiler generates a file named **Welcome.class**. This file is not an object file as generated by other high-level language compilers. This file is called the *bytecode*. The bytecode is similar to machine instructions, but is architecture-neutral and can run on any

platform that has the Java interpreter and runtime environment. This is one of Java's primary advantages: Java bytecode can run on a variety of hardware platforms and operating systems.

NOTE: The bytecode is stored in Output path\Package Name. Therefore, Welcome.class is stored in **c:\example\chapter1**, since the Output path is set to **c:\example** (see Figure 1.14) and the package name is **chapter1**. The file structures for the examples in this book are shown in Figure 1.24.

*****Insert Figure 1.24**

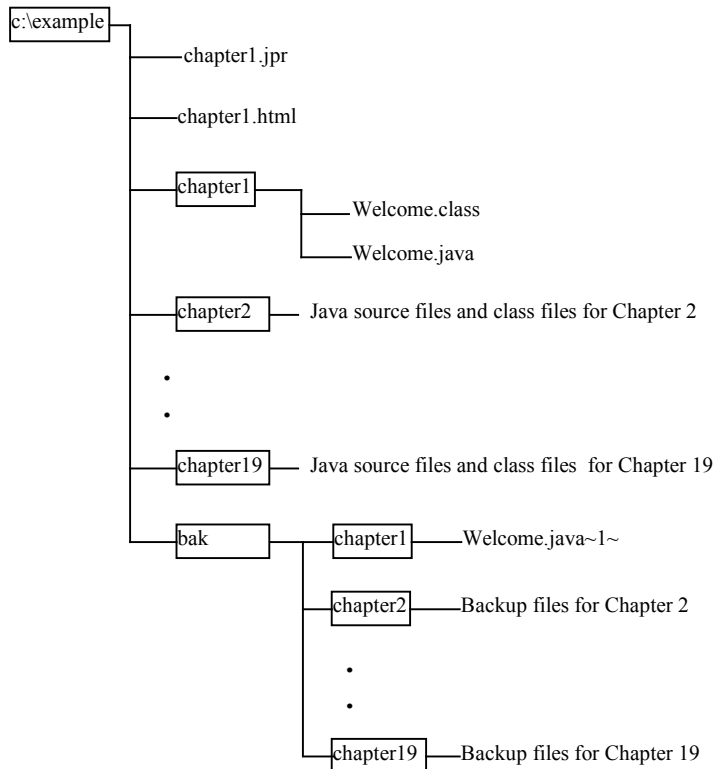



Figure 1.24

Welcome.java and Welcome.class are placed in c:\example\chapter1.

Executing a Java Application

To run Welcome.class, point to Welcome.java in the project pane and right-click the mouse button to display a popup menu. Choose Run from the popup menu.

NOTE: The Run command invokes the Compile command if the program is not compiled or was modified after the last compilation.

NOTE: You could run a program by selecting Run, Run "Welcome.java" from the main menu, or by clicking the Run toolbar button (), but then you have to specify a main class in the Runtime Properties dialog box. So it is more convenient to run a program from the project pane.

When this program executes, JBuilder displays the output in the message pane, as shown in Figure 1.25. When the program finishes, press Ctrl+C to close the DOS window.

*****Insert Figure 1.25**

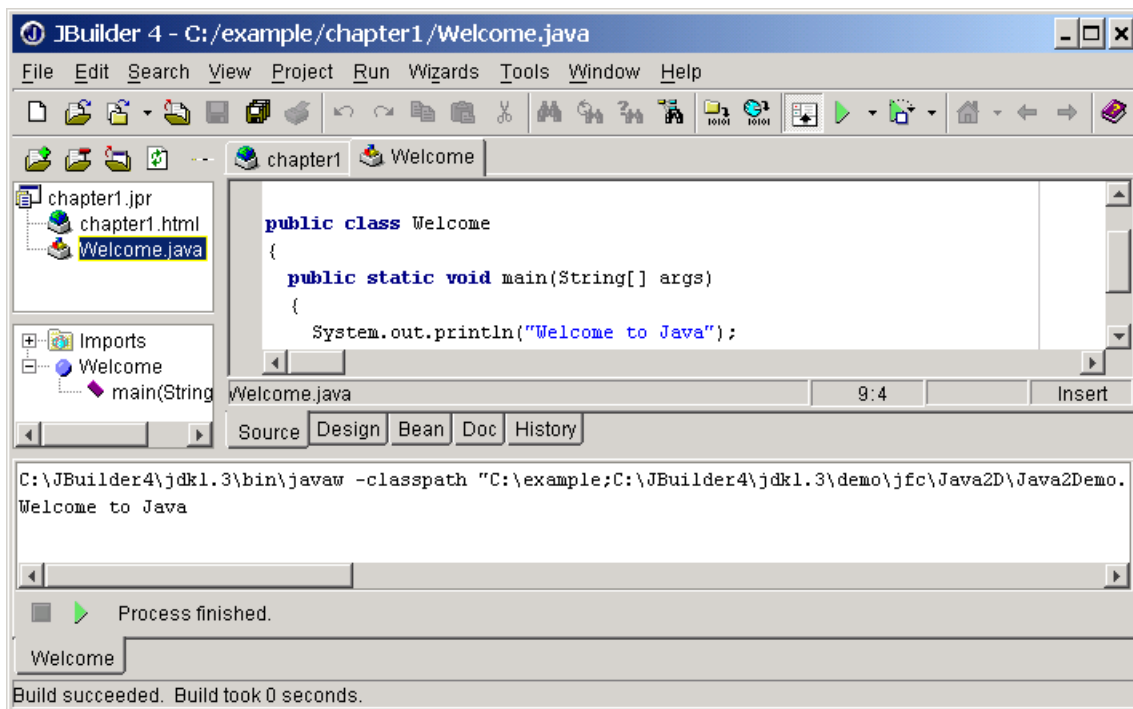


Figure 1.25

The execution result is shown in the message pane.

TIP: If the message pane is not displayed, choose View, Messages to display it.

Anatomy of a Java Program

The program in Example 1.1 has the following components:

Comments

Package

Reserved words

Modifiers

Statements

Blocks

Classes

Methods

The main method

To build a program, you need to understand these basic elements. The following sections explain each of them.

Comments

The first line in the program is a *comment*, which documents what the program is and how the program is constructed. Comments help programmers and users to communicate and understand the program. Comments are not programming statements and are ignored by the compiler. In Java, comments are preceded by two slashes (`//`) in a line, or enclosed between `/*` and `*/` in multiple lines. When the compiler sees `//`, it ignores all text after `//` in the same line. When it sees `/*`, it scans for the next `*/` and ignores any text between `/*` and `*/`.

Here are examples of the two types of comments:

```
// This application program prints Welcome to Java!
```

```
/* This application program prints Welcome to Java! */
```

A recommended comment style is given in the Section, "Programming Style and Documentation," in Chapter 2, "Primitive Data Types and Operations."

NOTE: In addition to the two comment styles, `//` and `/*`, Java supports comments of a special type, referred to as *javadoc comments*. javadoc comments begin with `/**` and end with `*/`. They are used for documenting classes and data and methods. They can be extracted into an HTML file using JDK's `javadoc` command. For more information, visit

*****End of NOTE (Editor: Please leave this line here to alert the layout person. I will use "End of CAUTION", "End of NOTE, and End of TIP, for CAUTION, NOTE, and TIP with multiple paragraphs. AU)**

Packages

The second line (package chapter1;) in the program specifies a package name chapter1 for the class Welcome. JBuilder compiles the source code in Welcome.java, generates Welcome.class, and stores Welcome.class in output path\chapter1. Output path is specified in the Project wizard, as shown in Figure 1.14. The package statement is optional. If it is used, it must be the first statement in the program except comments.

Reserved Words

Reserved words or keywords are words that have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word class, it understands that the word after class is the name for the class. Other reserved words in Example 1.1 are public, static, and void. Their use will be introduced later in the book.

TIP: Because Java is case-sensitive, public is a reserved word, but Public is not. Nonetheless, for clarity and readability, it would be best to avoid using reserved words in other forms. (See Appendix A, "Java Keywords.")

Modifiers

Java uses certain reserved words called *modifiers* that specify the properties of the data, methods, and classes and how they can be used. Examples of modifiers are public and static. Other modifiers are private, final, abstract, and protected. A public datum, method, or class can be accessed by other programs. A private datum or method cannot be accessed by other programs. Modifiers are discussed in Chapter 5, "Programming with Objects and Classes."

Statements

A *statement* represents an action or a sequence of actions. The statement System.out.println("Welcome to Java!") in the program in Example 1.1 is a statement to display the greeting "Welcome to Java!" Every statement in Java ends with a semicolon (;).

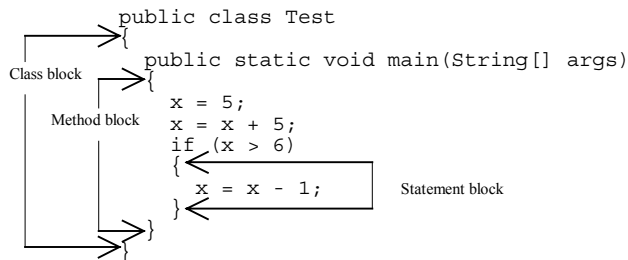
The following lines of code are statements:

```
x = 5;  
  
x = x + 5;
```

The first statement assigns 5 to the variable x, and the second adds 5 to the variable x.

Blocks

Braces in a program form a *block* that groups components of a program. In Java, each block begins with an open brace ({) and ends with a closing brace (}). Each class has a *class block* that groups the data and methods of the class. Each method has a *method block* that groups the statements in the method. Blocks can be *nested*, meaning that one block can be placed within another, as shown in the following code.



Classes

The *class* is the essential Java construct. A class is a template or blueprint for objects. To program in Java, you must understand classes and be able to write and use them. The mystery of the class will continue to be unveiled throughout this book. For now, though, understand that a program is defined by using one or more classes. Every Java program has at least one class, and programs are contained inside a class definition enclosed in blocks. The class can contain data declarations and method declarations.

Methods

What is `System.out.println`? It is a *method*: a collection of statements that performs a sequence of operations to display a message on the console. `println` is predefined as part of the standard Java language. It can be used even without fully understanding the details of how it works. It is used by invoking a calling statement with string arguments. The string arguments are enclosed within parentheses. In this case, the argument is `"Welcome to Java!"`. You can call the same `println` method with a different argument to print a

different message.

The main Method

You can create your own method. Every Java application must have a user-declared main method that defines where the program begins. The main method provides the control of program flow. The Java interpreter executes the application by invoking the main method.

The main method looks like this:

```
public static void main(String[] args)
{
    // Statements;
}
```

NOTE: You are probably wondering about such points as why the main method is declared this way. Your questions cannot be fully answered yet. For the time being, you will just have to accept that this is how things are done. You will find the answers in the coming chapters.

Managing Projects in JBuilder (Optional)

JBuilder uses a project file to store project information. You cannot edit the project file manually; it is modified automatically, however, whenever you add or remove files from the project or set project options. You can see the project file as a node at the top of the project tree in the project pane (see Figure 1.8). JBuilder uses a Project Properties dialog box for setting project properties and provides a Project wizard to facilitate creating projects.

NOTE: The project properties can be modified in the Project Properties dialog box after a project file is created. However, there is no need to change any properties if you have set the path properties correctly in the Project wizard in JBuilder 4. For this reason, this section is marked optional.

Setting Project Properties

JBuilder uses Default Project Properties dialog box to set default environment properties for all the projects. An individual project has its own Project Properties dialog box, which can be used to set project-specific properties.

To display the default Project Properties dialog box, select Project, Default Properties, as shown in Figure 1.26. To display the Project Properties dialog box, select Project, Properties, as shown in Figure 1.27. You can also right-

click the project file in the project pane and choose the Properties command to display the Project Properties dialog box.

The Default Project Properties dialog box (Figure 1.26) and the Project Properties dialog box (Figure 1.27) look the same but have different titles. Both dialog boxes contain Paths, General, Run, Debug, and Code Style. The Team tab is only available in JBuilder Professional and Enterprise. You can set options in these pages for the current project or the default project, depending on whether the dialog box is for the current project or for the default project. JBuilder Professional and JBuilder Enterprise Edition have more options in the Project Properties dialog box.

***Insert Figure 1.26

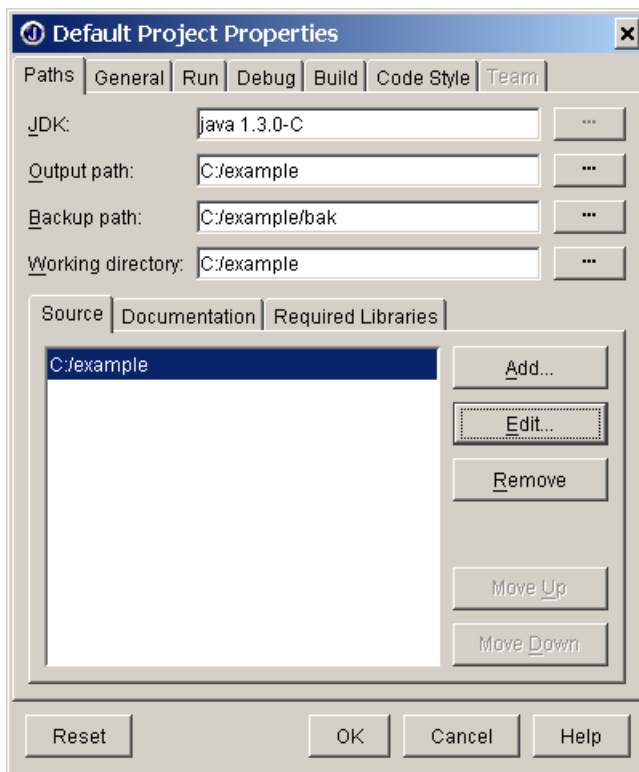


Figure 1.26

The Default Project Properties dialog box enables you to set default properties to cover all the projects.

*** Figure 1.27

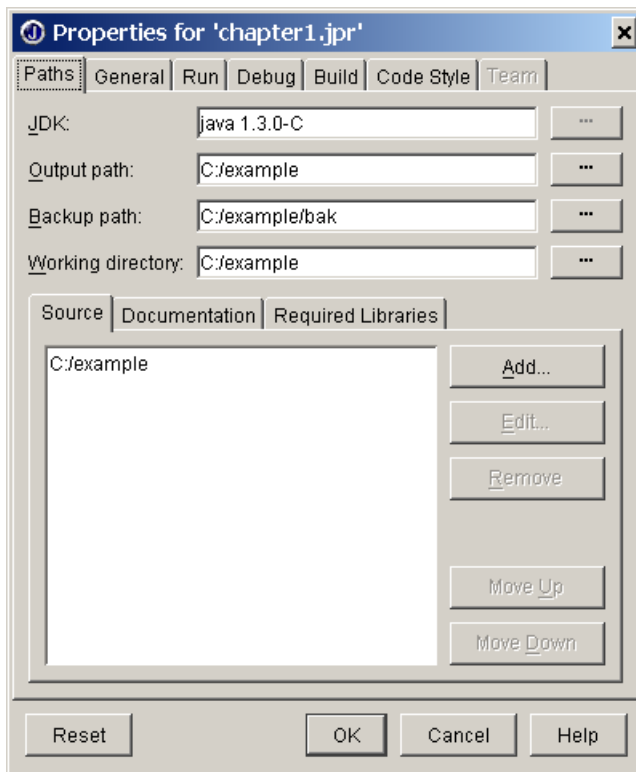


Figure 1.27

Each project keeps its own project properties.

The Paths Page

The Paths page of the Project Properties dialog box sets the following options:

- [BL] JDK version.
- [BL] Output path where the compilation output is stored.
- [BL] Backup path where the backup files are stored.
- [BL] Working directory where the temporary files are stored.
- [BL] Source page where the source files are stored.
- [BL] Documentation page where the source files are stored.
- [BL] Required Libraries page where the libraries are searched for compiling and running.

JBuilder can compile and run against JDK 1.1 or JDK 1.2. To set up the list of available JDKs, click the ellipsis button to display the Available JDK Versions dialog box for adding new JDK compilers. This feature is available only in JBuilder Professional and Enterprise.

Setting proper paths is necessary for JBuilder to locate the associated files in the right directory for compiling and running the programs in the project. The Source page specifies one or more paths for the source file. The Output path specifies the path where the compiler places the .class files. Files are placed in a directory tree that is based on the Output path and the package name. For example, if the Output path is c:\example and the package name is chapter1 in the source code, the .class file is placed in c:\example\chapter1. The Backup path specifies a directory where the backup source files are stored. JBuilder automatically backups the files before committing any change to the current file. Note the .java files are backed in the backup directory, but the other files like .html and .jpr are backed in their original directory.

All the libraries you need for this book have already been taken care of. If you need to add the custom libraries to the project, click the Add button to display a list of available libraries. To remove a library, click the Remove button. To switch the order of libraries, use the Move Up and Move Down buttons.

The General Page

The General page (Figure 1.28) allows you to specify an encoding scheme used in JBuilder, enable or disable automatic source package discovery, and modify javadoc fields.

*****Insert Figure 1.28**

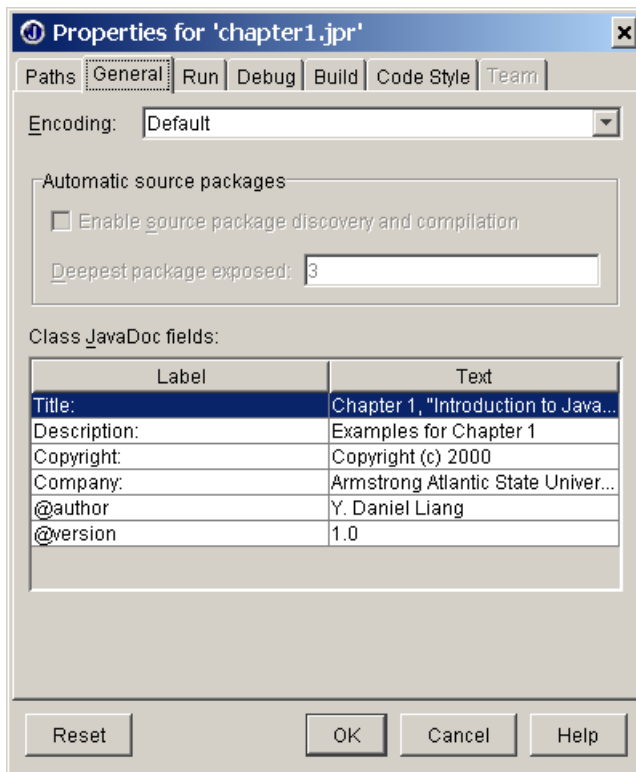


Figure 1.28

The General page has options for selecting encoding type, enabling/disabling automatic source package discovery, and modifying javadoc fields.

The "Encoding" choice menu specifies the encoding that controls how the compiler interprets characters beyond the ASCII character set. If no setting is specified, the default native-encoding converter for the platform is used.

You can enable or disable automatic source package. This is a useful feature available only in JBuilder professional and Enterprise. With automatic source package enabled, all packages in the project's source path automatically appear in the project pane, so you can navigate through the files without switching projects.

The Class Javadoc fields section specifies the javadoc tags generated in the class files when Generate Header Comments is enabled in a wizard.

The Run Page, Debug Page, and Build Page

The Run page sets runtime configuration. The Debug page sets debug options. The Build page (Figure 1.29) sets compiler options. The options are applied to all the files in the project, as well as to files referenced by these files, stopping at packages that are marked "stable."

***Insert Figure 1.29

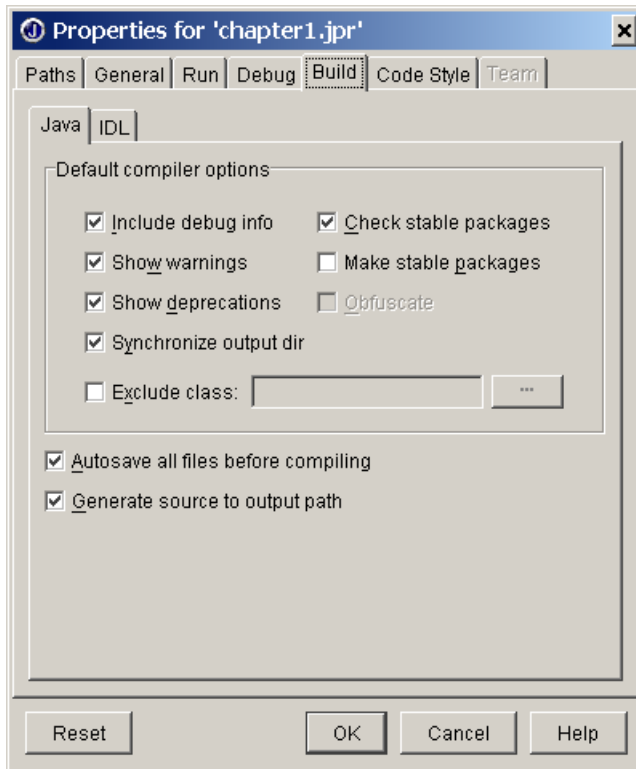


Figure 1.29

The Build page sets compiler options.

The option "Include debug information" includes symbolic debug information in the .class file when you compile, make, or rebuild a node. The option "Show warnings" displays compiler warning messages. The option "Show deprecations" displays all deprecated classes, methods, properties, events, and variables used in the API. The option "Synchronize output dir" deletes class files on the output path for which you don't have source files before compiling.

The Option "Check stable packages" checks files in the packages marked "stable" to see whether they and their imported classes need to be recompiled. This option shortens the edit/recompile cycle by not rechecking stable packages.

The option "Make Package Stable" checks all the classes of a package on the first build and marks the package "stable." If this option is unchecked, only the referenced classes of this package will be compiled, and the package will not be marked "stable." This option should be unchecked when working with partial projects. It is especially useful for working with a library of classes with no source code.

The option "Obfuscate" makes your programs less vulnerable to decompiling. Decompiling means to translate the Java bytecode to Java source code. After decompiling your obfuscated code, the generated source code contains altered symbol names for private symbols. This feature is available only in JBuilder Professional and Enterprise.

The "Exclude Class" choice lets you specify that a .class file is excluded from being compiled.

The option "Autosave all files before compiling" automatically saves all files in the project before each compile.

The option "Generate source to output path" is applicable only to RMI (Remote Method Invocation) and IDL (Interface Definition Language) files. These files are used in multi-tier Java applications, which are covered in my *Rapid Java Application Development Using JBuilder 4* text.

The Code Style Page

The Code Style page (Figure 1.30) enables you to specify the code style of the program generated by JBuilder.

***Insert Figure 1.30

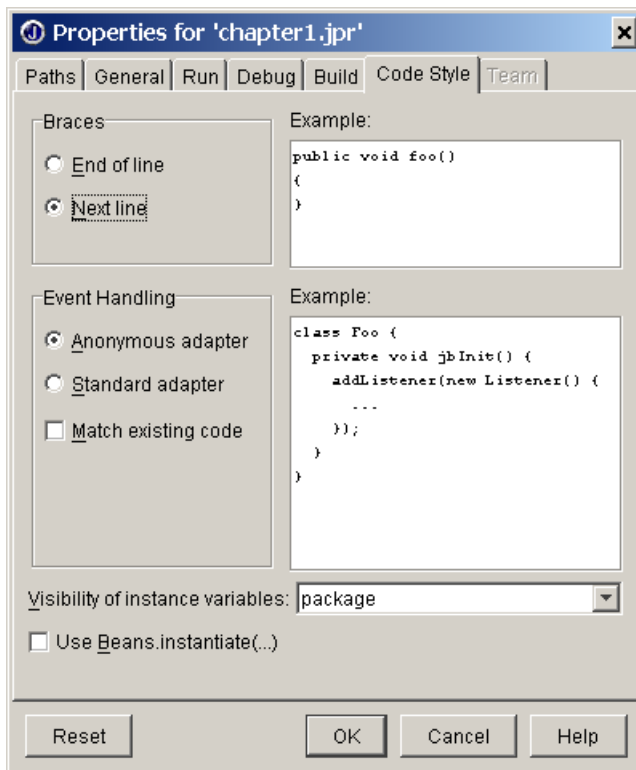


Figure 1.30

The Code Style page of the Project Properties dialog box sets the options for the code style of the program generated by JBuilder.

With the option "End of line" selected, JBuilder will generate the code with opening braces inserted at the end of the line; otherwise, the opening braces are inserted at the beginning of the new line.

In the Event Handling section, you can choose one of the options to tell JBuilder to generate event-handling code using anonymous adapter, standard adapter, or matching the existing style of event-handling code. The adapters are used in the UI designer in JBuilder. For more information on adapters, please refer to Appendix H, "Rapid Java Application Development Using JBuilder."

The option "Use Beans.instantiate" tells JBuilder to instantiate objects using Beans.instantiate() instead of the new operator. This feature is discussed in my book *Rapid Java Application Development Using JBuilder 4*.

Chapter Summary

In this chapter, you learned about Java and the relationship between Java and the World Wide Web. Java is an Internet programming language, and since its inception in 1995, it has quickly become a premier language for building software.

Java is platform-independent, meaning that you can write a program once and run it anywhere. Java is a simple, object-oriented programming language with built-in graphics programming, input and output, exception handling, networking, and multithreading support.

Java source files end with the .java extension. Every class is compiled into a separate file called a bytecode that has the same name as the class and ends with the .class extension.

Every Java program is a set of class definitions. The keyword class introduces a class definition. The contents of the class are included in a block. A block begins with an open brace ({) and ends with a closing brace (}). Methods are contained in a class. A Java application must have a main method. The main method is the entry point where the application program starts when it is executed.

You have begun to use JBuilder to create Java applications. You learned how to create projects, create and add files to the project, and compile and run applications.

Review Questions

- 1.1 Briefly summarize the history of Java.
- 1.2 Java is object-oriented. What are the advantages of object-oriented programming?
- 1.3 Can Java run on any machine? What is needed to run Java on a computer?
- 1.4 What are the input and output of a Java compiler?
- 1.5 List some Java development tools. Are these tools like JBuilder and Forte different languages from Java, or are they dialects or extensions of Java?
- 1.6 What is the relationship between Java and HTML?
- 1.7 Explain the Java keywords. List some Java keywords you learned in this chapter.
- 1.8 Is Java case-sensitive? What is the case for Java keywords?
- 1.9 What is the Java source filename extension, and what is the Java bytecode filename extension?
- 1.10 What is a comment? What is the syntax for a comment in Java? Is the comment ignored by the compiler?
- 1.11 What is the statement to display a string on the console?
- 1.12 Identify and fix errors in the following code:

```
public class Welcome
{
    public void Main(String[] args)
    {
        System.out.println('Welcome to Java!');
    }
}
```
- 1.13 How do you create a Java project in JBuilder?
- 1.14 How do you compile a Java program in JBuilder?
- 1.15 How do you run a Java program in JBuilder?
- 1.16 Suppose the output path is c:\smith, and the package statement in the source code in package homework.csci1301, where is the .class file stored after successful compilation in JBuilder?
 - a. in c:\jbuilder4\projects\classes.
 - b. in c:\smith.
 - c. in c:\smith\homework.
 - d. in c:\smith\homework\csci1301.
- 1.17 Suppose the source path is c:\smith, and you specified exercise1 in the Package field in the Class

wizard, and Test in the Class name field. Where is the generated Test.java stored?

- a. in c:\jbuilder4\projects\src.
- b. in c:\smith.
- c. in c:\smith\exercise1.
- d. in c:\smith\exercise1\test.

Programming Exercises

1.1 The text created a project named *chapter1*. Now create your project. Use your name as the project name. Create a Java application named WelcomeHTML.java in the project. WelcomeHTML.java displays a message "Welcome to HTML" using System.out.println to display a message on the message pane in JBuilder 4.