TRAINING & REFERENCE

# murach's Java programming

## (Chapter 1)

Thanks for downloading this chapter from ***Murach's Java Programming***. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its "how-to" headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our **web site**. From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books on Java development.

Thanks for your interest in our books!

MIKE MURACH & ASSOCIATES, INC.
1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963
**murachbooks@murach.com** • **www.murach.com**
*Copyright © 2011 Mike Murach & Associates. All rights reserved.*

## What developers have said about previous editions of *Murach's Java*

"Finally there is a Java book for serious programmers doing real life business applications."

Donna Dean, IS Trainer, Chicago, Illinois

"I bought your Java book a week ago and I am already writing useful programs, not 'toys!'"

Richard Cooper, Programmer

"The absolute best teaching text for Java among the gazillions of self-hyped tutorials and skills books on the market today."

Posted at Amazon.com

"I love your Java book. It cuts right to the essential information, providing the perfect balance between too many details and too little information. Example apps are incredible. Keep up the good work."

Steve, Programmer, Denver, Colorado

"Terrific - Fantastic - Superlative! WELL worth several times the purchase price."

Posted at Amazon.com

"*Murach's Java* is now my go-to source for reference and learning and brushing up, well above and beyond the other books in my collection."

Jeff Salter, Sacramento Java Users Group (SacJUG)

"If I'd seen this book first, I would not have wasted money (and time) on 6 other books! This one is highly organized, clear, and very effective as a learning tool."

Posted at Amazon.com

"The style is clean, very user friendly, and simple. The narrative is totally accurate and focuses on important issues; it is not dumbed down. This book is a winner!"

Dr. Richard Wiener, Editor-in-Chief, *Journal of Object Technology*

# Contents

# Section 1

## Essential Java skills

This section gets you started quickly with Java programming. First, chapter 1 introduces you to Java applications and shows you how to use NetBeans to work with Java projects. Then, chapter 2 introduces you to the basic skills that you need for developing Java applications. When you complete these chapters, you'll be able to write, test, and debug simple applications of your own.

After that, chapter 3 presents the details for working with numeric data. Chapter 4 presents the details for coding control statements. Chapter 5 shows how to validate the data that's entered by the user. And chapter 6 shows how to thoroughly test and debug an application. These are the essential skills that you'll use in almost every Java application that you develop. When you finish these chapters, you'll be able to write solid programs of your own. And you'll have the background that you need for learning how to develop object-oriented programs.

# 1

# How to get started with Java and NetBeans

Before you can begin learning the Java language, you need to install Java. In addition, you need to choose an IDE or a text editor for working with Java. For this book, we recommend that you use the NetBeans IDE. Appendix A of this book shows you how to install both Java and NetBeans on a Windows system, and appendix B shows you how to install them on a Macintosh OS X system. Then, this chapter shows how to use the NetBeans IDE to create and work with a Java application. But first, this chapter presents some background information about Java.

# Introduction to Java

In 1996, Sun Microsystems released a new programming language called Java. Although Oracle bought Sun in 2010, Java remains one of the most widely used object-oriented programming languages.

## Toolkits and platforms

Figure 1-1 describes all major releases of Java starting with version 1.0 and ending with version 1.7. Throughout Java's history, the terms *Java Development Kit* (*JDK*) and *Software Development Kit* (*SDK*) have been used to describe the Java toolkit. In this book, we'll use the term *JDK* since it's the most current and commonly used term. In addition, different numbering schemes have been used to indicate the version of Java. For example, Java 5.0 and Java 6 refer to versions 1.5 and 1.6 of Java. In this book, we'll use the 1.x style of numbering since this numbering is used by the documentation for Java.

With versions 1.2 through 1.5 of the JDK, the *Standard Edition* (*SE*) of Java was known as *Java 2 Platform, Standard Edition* (*J2SE*), and the *Enterprise Edition* (*EE*) was known as the *Java 2 Platform, Enterprise Edition* (*J2EE*). Since version 1.6 of the JDK, the Standard Edition of Java has been known as *Java SE*, and the Enterprise Edition is known as *Java EE*. This book shows how to use Java SE 7, but it should also work for earlier and future versions of Java. That includes Java SE 8, which is scheduled for release in late 2012.

## How Java compares to C++ and C#

When Sun's developers created Java, they tried to keep the syntax for Java similar to the syntax for C++ so it would be easy for C++ programmers to learn Java. In addition, they designed Java so its applications can be run on any computer platform. In contrast, C++ needs to have a specific compiler for each platform. Java was also designed to automatically handle many operations involving the creation and destruction of memory. This is a key reason why it's easier to develop programs and write bug-free code with Java than with C++.

To provide these features, the developers of Java had to sacrifice some speed (or performance) when compared to C++. For many types of applications, however, Java's relative slowness is not an issue.

Microsoft's Visual C# language is similar to Java in many ways. Like Java, C# uses a syntax that's similar to C++ and that automatically handles memory operations. However, in practice, C# code only runs on Windows. Because of that, C# is a good choice for developing applications for a Windows-only environment. However, Java is a better choice if you need to develop cross-platform applications.

## Java timeline

| Year | Month | Event |
|---|---|---|
| 1996 | January | Sun releases Java Development Kit 1.0 (JDK 1.0). |
| 1997 | February | Sun releases Java Development Kit 1.1 (JDK 1.1). |
| 1998 | December | Sun releases the Java 2 Platform with version 1.2 of the Software Development Kit (SDK 1.2). |
| 1999 | August | Sun releases Java 2 Platform, Standard Edition (J2SE). |
| | December | Sun releases Java 2 Platform, Enterprise Edition (J2EE). |
| 2000 | May | Sun releases J2SE with version 1.3 of the SDK. |
| 2002 | February | Sun releases J2SE with version 1.4 of the SDK. |
| 2004 | September | Sun releases J2SE 5.0 with version 1.5 of the JDK. |
| 2006 | December | Sun releases Java SE 6 with version 1.6 of the JDK. |
| 2010 | April | Oracle buys Sun. |
| 2011 | July | Oracle releases Java SE 7 with version 1.7 of the JDK. |

## Operating systems supported by Java

Windows (XP, Vista, 7)

Linux

Solaris

Macintosh OS X

## Java compared to C++ and C#

| Feature | Description |
|---|---|
| Syntax | Java syntax is similar to C++ and C# syntax. |
| Platforms | Compiled Java code can be run on any platform that has a Java interpreter. Similarly, compiled C# code (MSIL) can be run on any system that has the appropriate interpreter. Currently, only Windows has an interpreter for MSIL. C++ code must be compiled once for each type of system that it is going to be run on. |
| Speed | C++ and C# run faster than Java, but Java is getting faster with each new version. |
| Memory | Both Java and C# handle most memory operations automatically, while C++ programmers must write code that manages memory. |

## Description

- Versions 1.2 through 1.4 of Java are called the *Software Development Kit* (*SDK*).
- Versions 1.5 through 1.7 of Java are called the *Java Development Kit* (*JDK*).

## Note

- Java SE 8 with version 1.8 of the JDK is expected to be released late in 2012.

Figure 1-1 Introduction to Java

## Applications, applets, and servlets

Figure 1-2 describes the three types of programs that you can create with Java. First, you can use Java to create *applications* that run directly on your computer. These are also known as *desktop applications*.

When you create these desktop applications, you can use a *graphical user interface* (*GUI*) to get user input and perform a calculation as shown at the top left of this figure. In chapter 15, you'll learn how to create these types of applications. Until then, you'll learn how to create another type of desktop application known as a *console application*. This type of application runs in the *console*, or *command prompt*, that's available from your operating system. An example of a console application is shown at the top right of this figure.

One of the unique characteristics of Java is that you can use it to create a special type of web-based application known as an *applet*. For instance, this figure shows an applet that works the same way as the applications above it. The main difference between an application and an applet is that an applet can be stored in an HTML page and can run inside a Java-enabled browser. As a result, you can distribute applets via the Internet or an intranet. In chapter 17, you'll learn how to create and deploy applets.

Although applets can be useful for creating a complex user interface within a browser, they have their limitations. First, you usually need to install a plug-in on each client machine, which isn't ideal for some types of applications. Second, since an applet runs within a browser on the client, it's not ideal for working with resources that run on the server, such as enterprise databases.

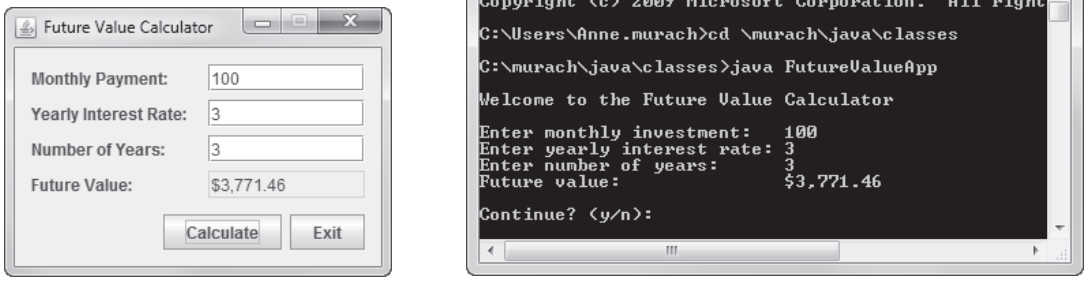
To provide access to enterprise databases, many developers use Java EE to create applications that are based on servlets. A *servlet* is a special type of Java application that runs on the server and can be called by a client, which is usually a web browser. This is also illustrated in this figure. Here, you can see that the servlet works much the same way as the applet. The main difference is that the code for the application runs on the server.

When a web browser calls a servlet, the servlet performs its task and returns the result to the browser, typically in the form of an HTML page. For example, suppose a browser requests a servlet that displays all unprocessed invoices that are stored in a database. Then, when the servlet is executed, it reads data from the database, formats that data within an HTML page, and returns the HTML page to the browser.

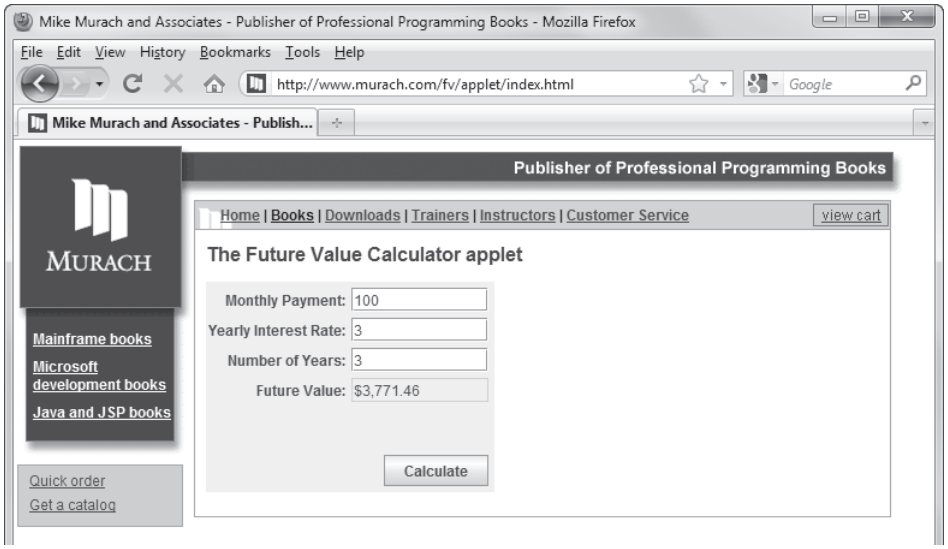
When you create a servlet-based application like the one shown here, all the processing takes place on the server and only HTML is returned to the browser. That means that anyone with an Internet or intranet connection, a web browser, and adequate security clearance can access and run a servlet-based application. Because of that, you don't need to install any special software on the client.

To make it easy to store the results of a servlet within an HTML page, the Java EE specification provides for *JavaServer Pages* (JSPs). Most developers use JSPs together with servlets when developing server-side Java applications. Although servlets and JSPs aren't presented in this book, we cover this topic in a companion book, *Murach's Java Servlets and JSP*. For more information about this book, please visit our web site at www.murach.com.

## A GUI application and a console application




## An applet




## A servlet

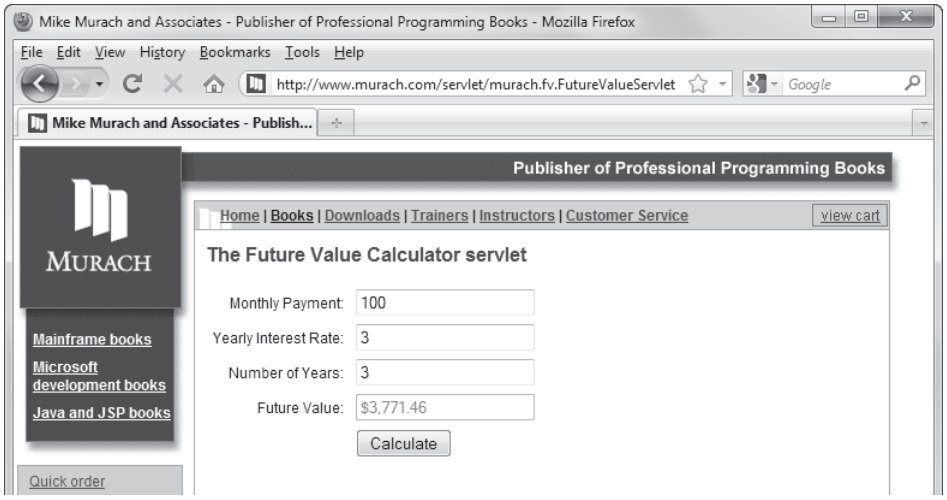



Figure 1-2 Applications, applets, and servlets

## The code for the console version of the Future Value application

To give you an idea of how the code for a Java application works, figure 1-3 presents the code for the console version of the Future Value application that you saw in figure 1-2.

If you have experience with other programming languages, you may be able to understand much of this code already. If not, don't worry! You'll learn how all of this code works in the next few chapters. For now, here's a brief explanation of this code.

Most of the code for this application is stored in a *class* named FutureValueApp. This class begins with an opening brace ({) and ends with a closing brace (}). Within this class, two *methods* are defined. These methods also begin with an opening brace and end with a closing brace, and they are indented to clearly show that they are contained within the class.

The first method, named main, is the *main method* for the application. The code within this method is executed automatically when you run the application. In this case, the code displays the data the user sees on the console, accepts the data the user enters at the console, and calculates the future value.

The second method is named calculateFutureValue. This method is called from the main method and calculates the future value based on the data the user enters.

## The code for the Future Value application

```
import java.util.Scanner;
import java.text.NumberFormat;

public class FutureValueApp
{
    public static void main(String[] args)
    {
        System.out.println("\nWelcome to the Future Value Calculator\n");

        Scanner sc = new Scanner(System.in);
        String choice = "y";

        while (choice.equalsIgnoreCase("y"))
        {
            // get the input from the user
            System.out.print("Enter monthly investment:   ");
            double monthlyInvestment = sc.nextDouble();
            System.out.print("Enter yearly interest rate: ");
            double interestRate = sc.nextDouble();
            System.out.print("Enter number of years:      ");
            int years = sc.nextInt();

            // calculate the future value
            double monthlyInterestRate = interestRate/12/100;
            int months = years * 12;
            double futureValue = calculateFutureValue(
                monthlyInvestment, monthlyInterestRate, months);

            // format and display the result
            NumberFormat currency = NumberFormat.getCurrencyInstance();
            System.out.println("Future value:                " +
                               currency.format(futureValue) + "\n");

            // see if the user wants to continue
            System.out.print("Continue? (y/n): ");
            choice = sc.next();
            System.out.println();
        }
    }

    private static double calculateFutureValue(double monthlyInvestment,
    double monthlyInterestRate, int months)
    {
        double futureValue = 0;
        for (int i = 1; i <= months; i++)
            futureValue = (futureValue + monthlyInvestment) *
                          (1 + monthlyInterestRate);
        return futureValue;
    }
}
```

Figure 1-3 The code for the console version of the Future Value application

## How Java compiles and interprets code

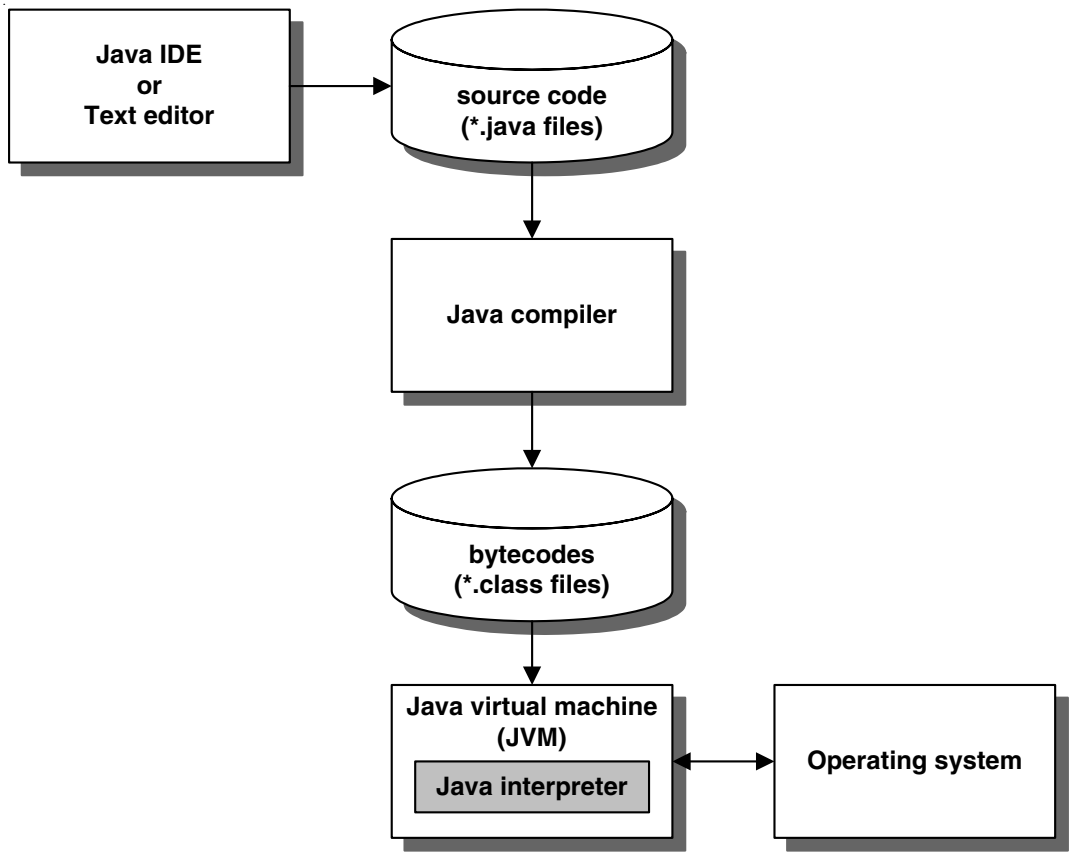
When you develop a Java application, you create one or more *classes*. For each class, you write the Java statements that direct the operation of the class. Then, you use a Java tool to translate the Java statements into instructions that can be run by the computer. This process is illustrated in figure 1-4.

To start, you enter and edit the Java *source code* for a class. These are the Java statements like the ones you saw in figure 1-3 that tell the application what to do. Then, you use the *Java compiler* to compile the source code into a format known as Java *bytecodes*. At this point, the bytecodes can be run on any platform that has a *Java interpreter* to *interpret* (or translate) the Java bytecodes into code that can be understood by the underlying operating system.

Since Java interpreters are available for all major operating systems, you can run Java on most platforms. This is what gives Java applications their *platform independence*. In contrast, C++ requires a specific compiler for each type of platform that its programs are going to run on. When a platform has a Java interpreter installed on it, it can be considered an implementation of a *Java virtual machine* (*JVM*).

In addition, most modern web browsers can be Java enabled. This allows applets, which are bytecodes that are downloaded from the Internet or an intranet, to run within a web browser. To make this work, Sun developed (and Oracle now maintains) the *Java Plug-in*. This piece of software is similar to other browser plug-ins such as Apple QuickTime. It allows the browser to run the current version of the Java interpreter. You'll learn more about this in chapter 17.

## How Java compiles and interprets code




## Description

- When you develop a Java application, you develop one or more *classes*.
- You can use a Java IDE or any text editor to create, edit, and save the *source code* for a Java class. Source code files have the *java* extension.
- The *Java compiler* translates Java source code into a *platform-independent* format known as Java *bytecodes*. Files that contain Java bytecodes have the *class* extension.
- The *Java interpreter* executes Java bytecodes. Since Java interpreters exist for all major operating systems, Java bytecodes can be run on most platforms. A Java interpreter is an implementation of a *Java virtual machine* (*JVM*).
- Most modern web browsers can be Java enabled. This lets applets run within these browsers. Oracle provides a tool known as the *Java Plug-in* that allows you to specify the version of the Java interpreter that you want to use.

Figure 1-4 How Java compiles and interprets code

## Introduction to Java IDEs

To develop Java applications, you typically use an *Integrated Development Environment (IDE)*. Although you can use a simple text editor, an IDE provides features that can make developing Java applications considerably easier. Figure 1-5 describes some of the features of the most popular IDEs.

Note that all of the IDEs listed in this figure are either free or have a free edition. That makes them particularly attractive to students as well as programmers who are learning on their own. Most of these IDEs also run on all modern operating systems.

The first two IDEs listed in this figure, NetBeans and Eclipse, are arguably the two most popular Java IDEs. Both of these IDEs provide all of the features listed in this figure. For example, both of these IDEs help you complete your code and notify you of potential compile-time errors. They both automatically compile your code before you run it. And they both include a debugger that lets you perform standard debugging functions like setting breakpoints, stepping through code, and viewing the values of variables.

The default installation of NetBeans also provides a feature for building graphical user interfaces (GUIs). To use this GUI builder, you can drag controls onto a form on the design surface. Then, you can move, size, and align the controls and set properties of the controls that determine how they look. As you do this, the code that displays the GUI is automatically generated. Finally, you can generate event handlers for the events that you want to handle and then write the code that handles these events.

On the other hand, the default installation of Eclipse does not provide a GUI builder. However, several free GUI builder plug-ins are available for Eclipse. These GUI builders provide features similar to the GUI builder that's provided by NetBeans.

The last three IDEs listed in this figure aren't as popular as NetBeans and Eclipse. However, we have included them here to give you an idea of the wide range of IDE choices that are available for Java. In addition, other Java IDEs are available that aren't included here.

For this book, we recommend using NetBeans because we think it's more intuitive and easier to use than Eclipse, especially for beginners. Once you're done with this book, you can switch to whatever IDE you prefer. Fortunately, once you learn how to use one IDE, it's fairly easy to learn to use another one.

## Popular Java IDEs

| IDE | Description |
|---|---|
| NetBeans | A free, open-source IDE that runs on most modern operating systems. |
| Eclipse | A free, open-source IDE that runs on most modern operating systems. |
| IntelliJ IDEA | The Community Edition of this IDE is a free, open-source IDE that runs on most modern operating systems. |
| JCreator LE | The Lite Edition (LE) of this IDE is free, but the code is not open-source, and it only runs on Windows. |
| BlueJ | A free IDE that's designed for teaching Java to first-year students and runs on most modern operating systems. |

## Features provided by most IDEs

- A code editor with code completion and error detection.
- Automatic compilation of classes when you run the application.
- A debugger that lets you set breakpoints, step through code, and view the values of active variables.
- A GUI builder that lets you create graphical user interfaces by dragging controls onto a form, setting properties, and writing code that handles the events that are triggered when a user interacts with the form.

## Description

- To develop Java applications, you typically use an *Integrated Development Environment* (*IDE*) like those listed above. All of these IDEs are either free or have free editions.

Figure 1-5 Introduction to Java IDEs

# How to use NetBeans to work with existing projects

Now that you have some background information about Java, you're ready to start working with existing NetBeans projects. In particular, you're ready to learn how to open and run any of the applications for this book. You can download these applications from our web site as described in appendix A (Windows) or appendix B (Mac OS X).

## Introduction to Java projects and the NetBeans IDE

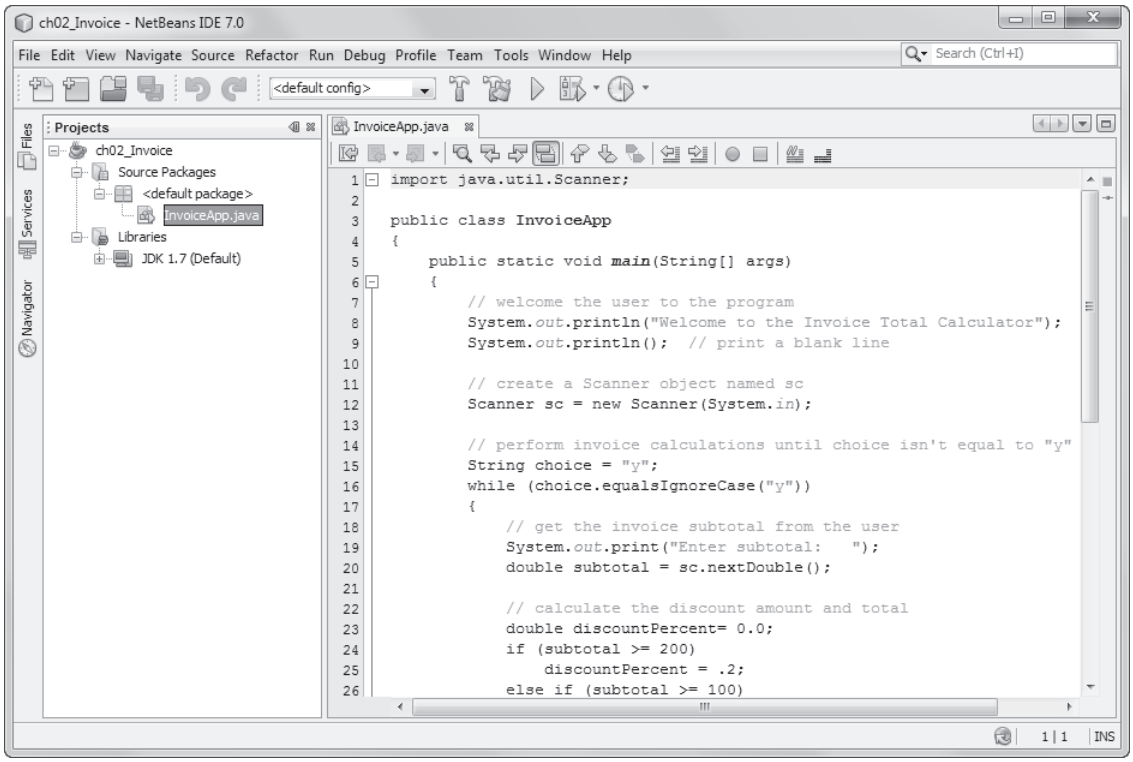
Figure 1-6 shows the NetBeans IDE with an open Java project. In NetBeans, a *project* is a folder that contains all the files for an application. In this example, the project is named ch02_Invoice.

In the Projects window, you can see that the folder for the ch02_Invoice project contains two subfolders. The first one, named Source Packages, contains the source files for the application. The second one, named Libraries, contains the Java libraries that are used by the application. In this case, the application uses just the JDK 1.7 libraries, but you can add others.

Within the Source Packages folder, the source files can be organized into *packages*. In this case, no package was specified for the project, so the default package is used. When you develop simple applications like the one shown here, that's usually acceptable. For more complex applications, though, you'll want to use two or more packages as shown in chapter 10.

The application shown here consists of a single source file named InvoiceApp.java. You can see part of this file in the NetBeans *code editor*. You'll learn more about working with this code editor later in this chapter. For now, I just want to point out that this file defines a single class. Because this class contains the main method for the application, it's called the *main class*. When you run an application, the main method in the main class is executed by default.

## NetBeans with a Java project open




## Description

- A NetBeans *project* consists of a top-level folder that contains the subfolders and files for an application.
- The Source Packages subfolder contains the .java files that make up the project. These files define *classes* that are later compiled into .class files.
- By default, a project consists of a single class that contains the *main method*. The main method is the starting point for the application, and the class that contains it is called the *main class*.
- The .java files that make up a project can be organized into one or more *packages*. If you don't specify a package for the main class when you create a project, it's stored in the default package.
- The Libraries subfolder contains the *libraries* that are available to your project. These libraries contain the Java classes that you can use in your projects. By default, you can use the classes in the JDK libraries.
- The folders, files, and libraries that make up a Java project are listed in the Projects window. If this window isn't visible, you can display it by using the Window→Projects command. Then, you can expand and collapse the nodes in this window by clicking on the plus and minus signs.
- You can display and work with the source code in a .java file in the *code editor window*. For details, see figure 1-11.

Figure 1-6 Introduction to Java projects and the NetBeans IDE

## How to open, close, and delete a project

To open a project in NetBeans, you use the Open Project dialog box shown in figure 1-7. This dialog box lets you navigate to the folder that contains the project you want to open. In this figure, for example, the Open Project dialog box shows all of the existing NetBeans projects in this folder:

```
C:\murach\java\netbeans\book_apps
```

To clearly indicate when a folder contains a Java project, the Open Project dialog box displays a small coffee cup icon to the left of the folder name. Then, you select the project you want to open and click the Open Project button.

When you're done working with a project, you can close it to remove it from the Projects window. To do that, you can use one of the techniques described in this figure.

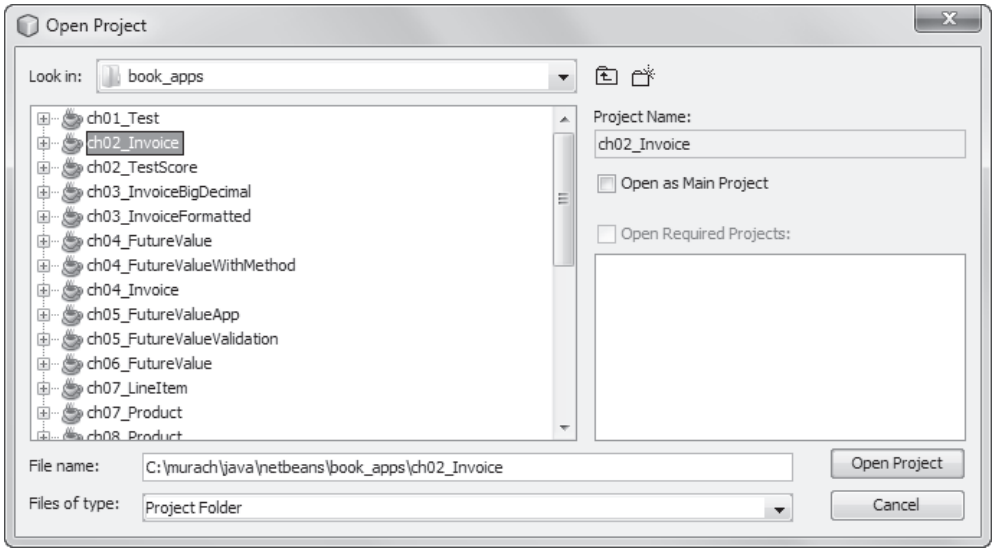
You can also delete a project if you decide that you no longer want to work with it in NetBeans. Before the project is deleted, NetBeans will prompt you to confirm the deletion. Then, by default, NetBeans deletes all of the files for the project except for the source files. That way, you can work with those files outside of NetBeans if you want to. If you want to delete the source files as well, you can select the "Also Delete Sources" option in the dialog box that's displayed.

## How to compile and run a project

Figure 1-7 also describes how to compile and run a project. An easy way to run a project is to press F6. Then, if the project has been modified since the last time it was compiled, NetBeans automatically compiles the project and runs the main method in the main class.

If you want to compile a project without running it, you can use the Build command as described in this figure. You can also use the Clean and Build command to compile the project and remove any files that are no longer needed. This sometimes helps to get a project to work correctly after you have copied, moved, or renamed some of its files.

### The dialog box for opening a project




### How to open, close, and delete a project

- To open a project, click the Open Project button in the toolbar or select the File→Open Project command. Then, use the Open Project dialog box that's displayed to locate and select the project and click the Open Project button.
- You can also open a project by using the File→Open Recent Project command and then selecting the project from the list that's displayed.
- To close a project, right-click on the project in the Projects window and select the Close command, or select the project and then use the File→Close Project command.
- To delete a project, right-click on the project in the Projects window and select the Delete command. When you do, you'll have the option of deleting just the files that NetBeans uses to manage the project or deleting all the folders and files for the project.

### How to compile and run a project

- To run a project, press F6, use the Run→Run Project command, or click the Run Project button in the toolbar.
- When you run a project, NetBeans automatically compiles it. As a result, you usually don't need to compile a project separately.
- To compile a project without running it, you can right-click on the project in the Projects window and select the Build command.
- To delete all compiled files for a project and compile them again, you can right-click on the project and select the Clean and Build command. This removes files that are no longer needed and compiles the entire project.

### Mac OS X note

- To enable right-clicking with Mac OS X, you can edit the system preferences for the mouse.

Figure 1-7 Basic skills for working with existing projects

## How to use the Output window with a console application

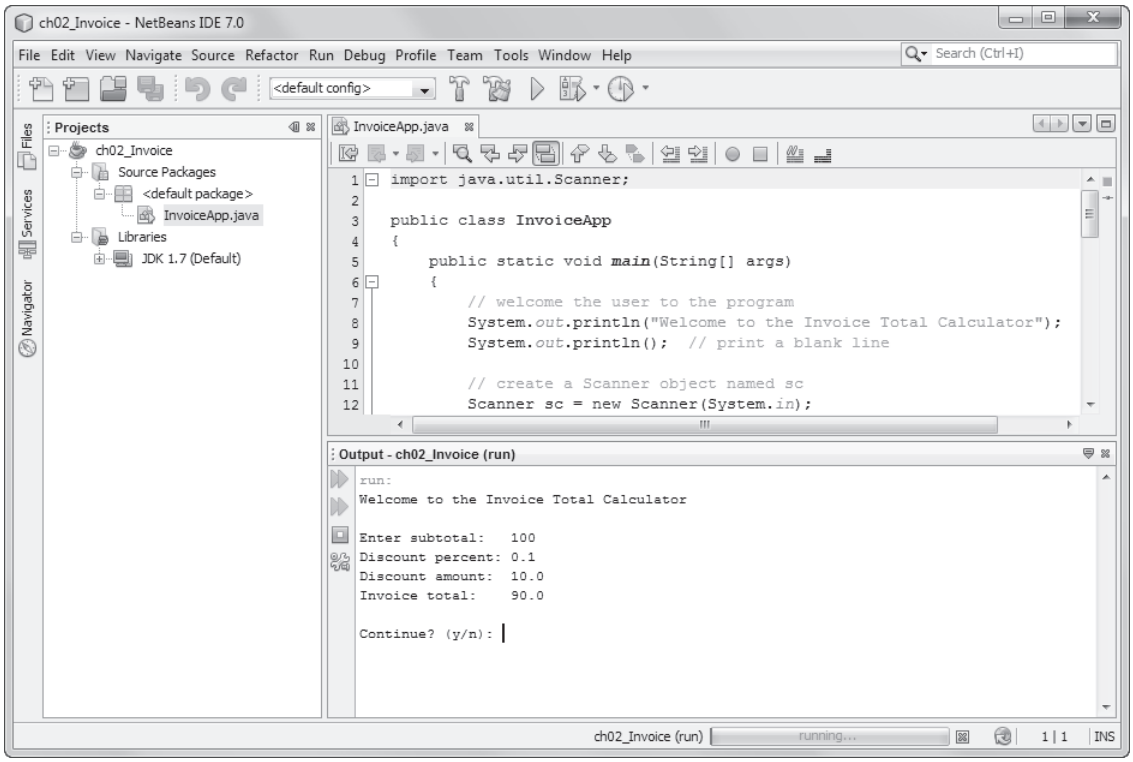
When you run a console application in NetBeans, any data that's written to the console is displayed in the Output window. In addition, The Output window can accept input. This is illustrated in figure 1-8.

The project shown in this figure is for a simple application that accepts a subtotal. Then, this application calculates and displays the discount percent, discount amount, and invoice total based on that subtotal. You'll see the code for this application in the next chapter. For now, just focus on the data in the Output window.

Here, the application started by displaying a welcome message. Then, it displayed a prompt indicating that I should enter a subtotal. In response, I typed "100" and pressed Enter. When I did, the application displayed the calculations and then asked me if I wanted to continue. At this point, the application is still running, and I can enter "y" to perform another calculation or "n" to end the application.

When you're learning Java, it's common to create applications that use the console to display output and get input. Because of that, the first three sections of this book teach you Java using console applications. Then, section 4 of this book will teach you how to create modern applications with graphical user interfaces.

## An application that uses the Output window for input and output




## Description

- When you run an application that prints data to the console, that data is displayed in the Output window.
- When you run an application that requests input from the console, the Output window pauses to accept the input. Then, you can click in the Output window, type the input, and press the Enter key.
- In addition to displaying output and accepting input, the Output window can display other information. For example, it can display messages when the application is compiled, and it can display errors that are encountered when an application is run.

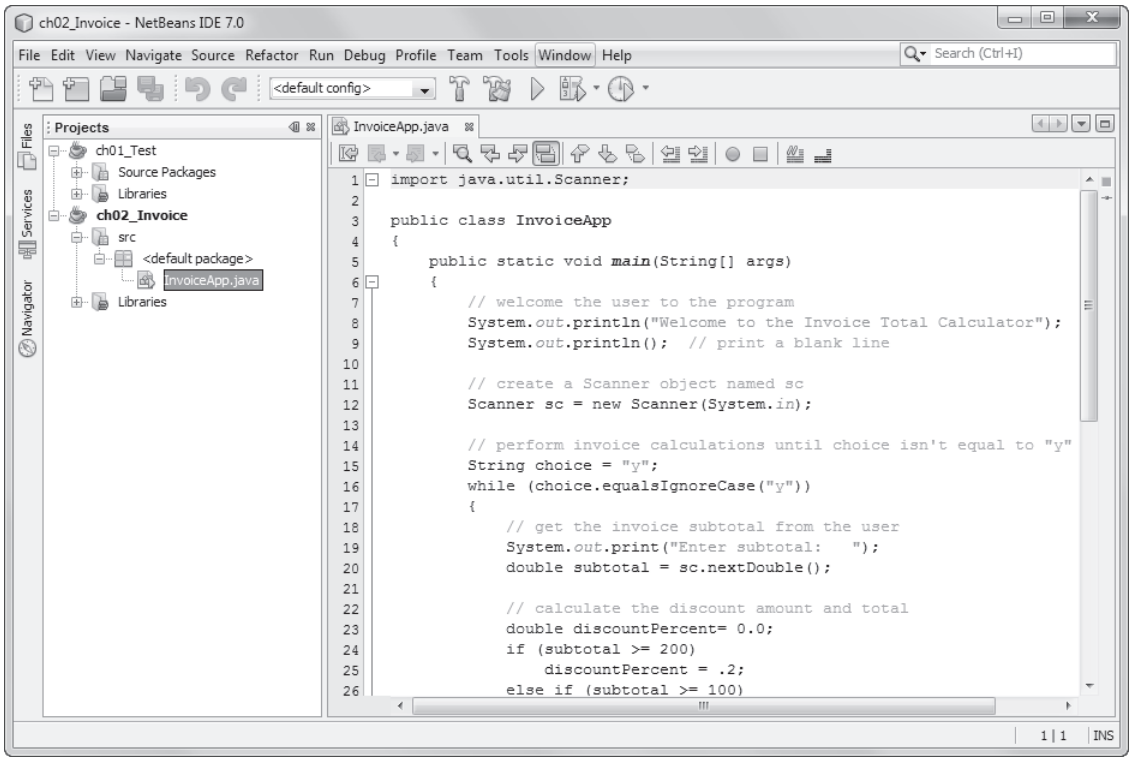
Figure 1-8 How to use the Output window with a console application

## How to work with two or more projects

Up to this point, I've shown you how to work with a single project in NetBeans. However, NetBeans lets you open and work with two or more projects at the same time. If, for example, you want to run some of the projects from the download for this book before you start creating your own projects, you can open those projects in NetBeans at the same time. You'll get a chance to do that in the first exercise for this chapter.

Figure 1-9 presents the skills for working with two or more projects. To start, you can run a project by selecting it in the Projects window and then using the techniques you learned in figure 1-7. Alternatively, you can set one of the projects as the *main project* using one of the techniques in this figure. When you do that, NetBeans identifies the project by boldfacing it in the Projects window. Then, that project is run automatically when you use one of the standard techniques for running a project. To run a project other than the main project, you have to right-click on the project or the file that contains the main method for the project and select the Run or Run File command.

## NetBeans with two open projects




## Description

- NetBeans lets you open and work with two or more projects at the same time.
- When you work with two or more projects, you can set one project as the *main project*. To do that, right-click on the project and select Set as Main Project. Or, when you open the project, select the Open as Main Project option.
- After you set a main project, you can run that project by pressing F6, by using the Run→Run Main Project command, or by clicking the Run Main Project button in the toolbar. The Run Main Project command and toolbar button replace the Run Project command and toolbar button when a main project is set.
- To run a project other than the main project, right-click on the project and select the Run command, or right-click on the file that contains the main method you want to run and select the Run File command.
- If you don't set a main project, you can run any project by selecting that project in the Projects window and then using standard techniques.

Figure 1-9 How to work with two or more projects

# How to use NetBeans to develop new projects

Now that you know how to work with existing Java projects in NetBeans, you're ready to learn how to develop new Java projects. That's what you'll learn in the remainder of this chapter.

## How to create a new project

Figure 1-10 presents the dialog boxes for creating a Java application. You use the New Project dialog box to choose the type of project you want to create. In most cases, you'll create a Java Application project as shown here. Then, when you click the Next button, NetBeans displays a New Java Application dialog box like the second one in this figure.

The New Java Application dialog box lets you enter a name and location for the project. In this figure, for example, the project name is "ch01_Test" and it will be stored in this folder:

```
C:\murach\java\netbeans\book_apps
```

If you install the source code for this book as described in the appendix, all of the applications presented in this book will be stored within this folder.

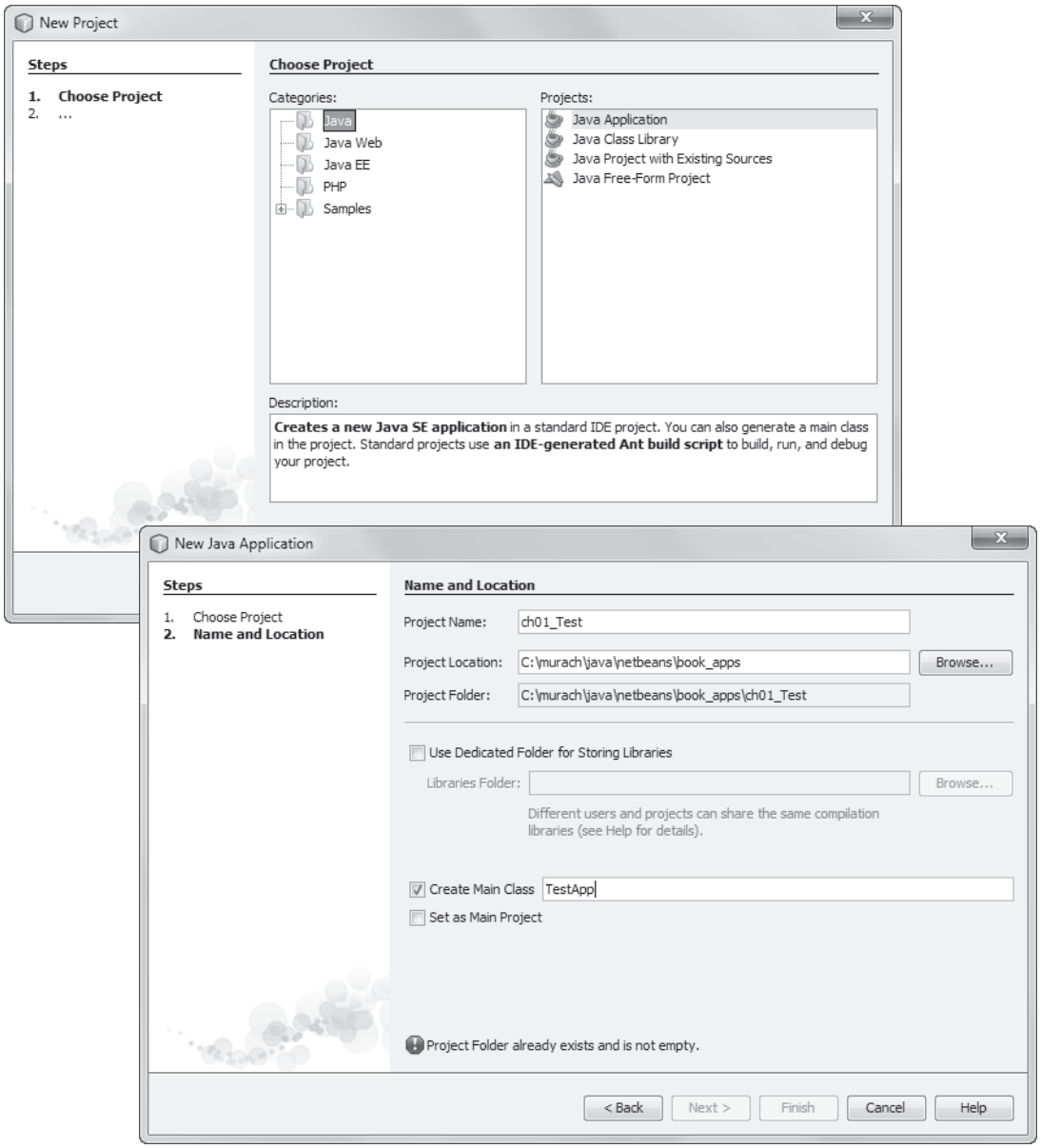
By default, when you create a Java application in NetBeans, NetBeans generates a main class with a main method. If that's not what you want, you can remove the check mark from the "Create Main Class" option. In most cases, though, you'll leave this option checked. Then, you can enter a name for the main class and, optionally, the package that contains it. For the project in this figure, for example, NetBeans suggested ch01_test.ch01_Test, where "ch01_test" is the name of the package and "ch01_Test" is the name of the class.

In this figure, I deleted the package name, and I changed the name of the class to TestApp. As a result, NetBeans created a project named ch01_Test that contains a main class named TestApp. Because I didn't specify a package name, this class is stored in the default package.

Like the Open Project dialog box you saw in figure 1-7, the New Java Application dialog box includes an option that determines if the new project is set as the main project. If you know that you want a project set as the main project when you create it, you should select this option. Otherwise, you can set the main project later using the technique you learned in figure 1-9.

When this dialog box is complete, you can click the Finish button to create the project and the class that contains the main method. Then, NetBeans creates a folder that corresponds with the project name, and it creates some additional files that it uses to configure the project.

## The dialog boxes for creating a new project




## Description

- To create a new project, use the File→New Project command or click the New Project button in the toolbar to display the New Project dialog box. Then, select a project type, click the Next button and complete the dialog box that's displayed.
- To create a Java Application project, enter the project name and location and the name you want to use for the main class. You can also enter the name of the package that will contain the main class, but that's not necessary.

Figure 1-10 How to create a new project

## How to set the Java version for a project

In some cases, you'll want to change the version of Java that a project uses. For example, you might want the project to run on computers with earlier versions of Java. Then, you can use the Project Properties dialog box in figure 1-11 to change the version for the project.

To use an earlier version of Java, you start by displaying the Sources category. Then, you select the version you want to use from the Source/Binary Format drop-down list. In this figure, I selected JDK 6. As a result, any language features that were added after Java 6 won't be available to the project. In addition, the bytecodes that are generated by the compiler for the project will run under Java 6 and later versions.

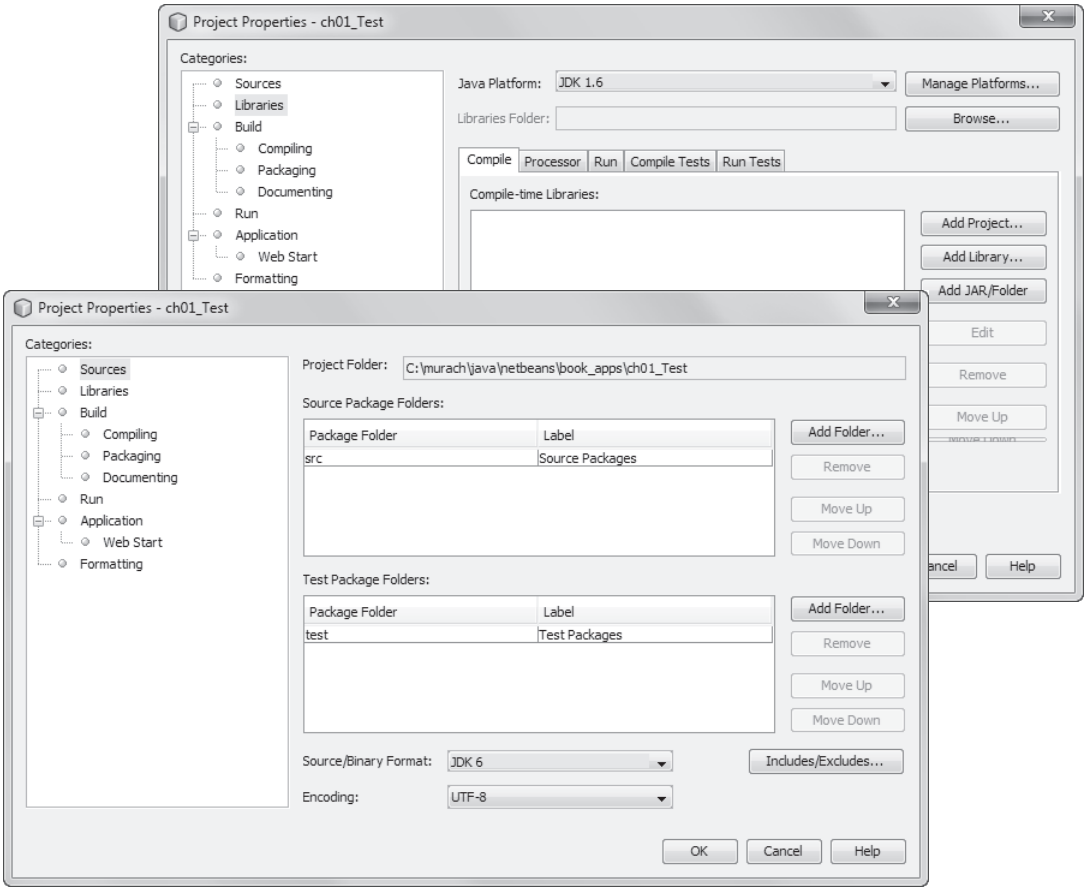
Although setting the Source/Binary Format option will keep you from using language features that were added with a later version of Java, it won't keep you from using features of the JDK libraries for later versions. In many cases, that's not a problem. If you want to be sure that you don't use any of the new features, though, you should change the Java platform that's used by the project. To do that, just select the JDK version from the Java Platform drop-down list in the Libraries category of the Project Properties dialog box.

Note that the Java Platform drop-down list only includes the versions you have installed on your computer. Before you can select a different version, then, you may have to install it as described in the appendix A (Windows) or appendix B (Mac OS X). After you do that, you can click the Manage Platforms button to the right of the Java Platform drop-down list to display a dialog box that lets you add the version you installed.

You might also want to change the Java version for a project if a newer version becomes available and you want to use some of the features of that version. After you install the new version, you can use the JDK libraries for that version in a project by selecting the JDK from the Java Platform drop-down list. In addition, if you want to use the new language features of that version and you don't need the project to run under earlier versions of Java, you can select the new JDK from the Source/Binary Format drop-down list.

Note that to use a newer version, you must set the Java Platform option first. Otherwise, that JDK won't be available in the Source/Binary Format list.

## The Project Properties dialog box




## Description

- The Project Properties dialog box lets you set various properties that affect the project. To display this dialog box, right-click on the project in the Projects window and select the Properties command.
- To set the version of the Java language and compiled bytecodes the project uses, select the version from the Source/Binary Format drop-down list in the Sources category. Then, the compiled bytecodes for the project will run under this version of Java and later. In addition, the project can only use language features for the specified version of Java.
- To set the version of the JDK libraries that are available to the project, select the version from the Java Platform drop-down list in the Libraries category. Then, your project can only use the features that are available from that version of the JDK libraries. For this to work, the version of the JDK you want to use must be installed on your system.
- To be sure that the JDK libraries are compatible with the language features and bytecodes, select the same version of the JDK from the Java Platform and the Source/Binary Format drop-down lists.

Figure 1-11 How to set the Java version for a project

## How to work with Java source code and files

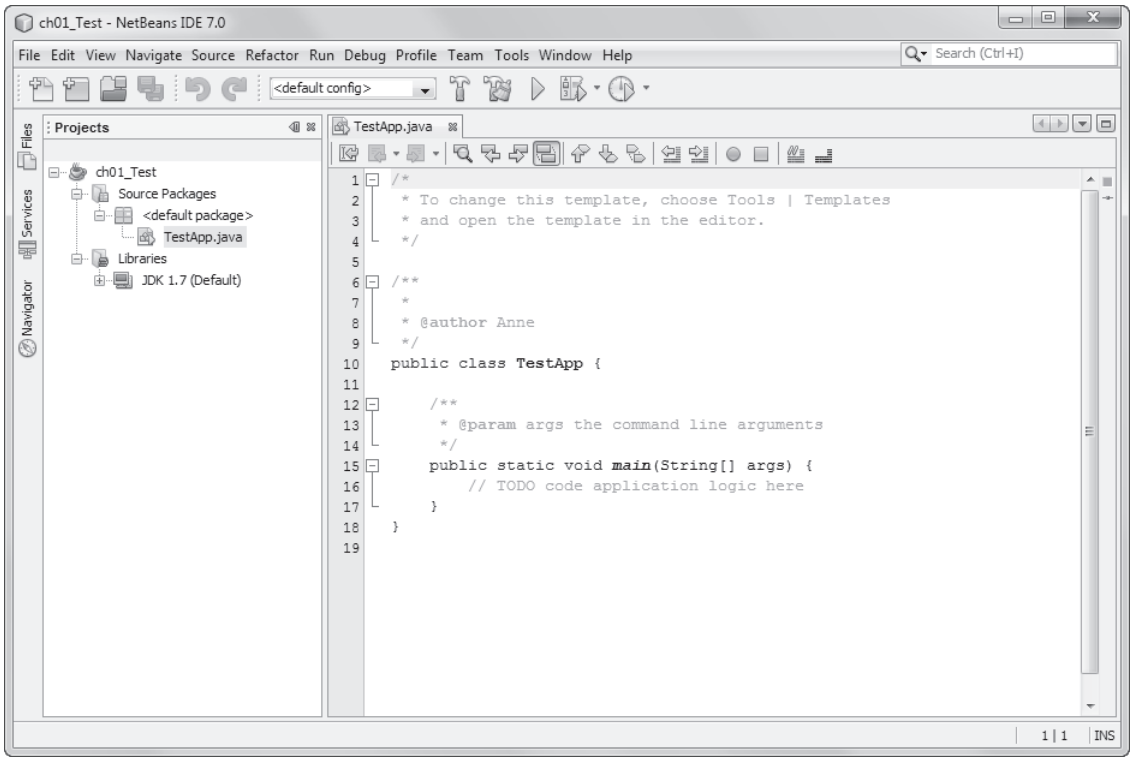
When you create a new project that contains a class with a main method, the class is typically opened in a new code editor window as shown in figure 1-12. To make it easier to for you to recognize the Java syntax, the code editor uses different colors for different language elements. In addition, NetBeans provides standard File and Edit menus and keystroke shortcuts that let you save and edit the source code. For example, you can press Ctrl+S to save your source code, and you can use standard commands to cut, copy, and paste code.

When you create a project with a main class, NetBeans generates some code for you. In this figure, for example, NetBeans generated the code that declares the class, the code that declares the main method, and comments that describe the class and method. Although you can delete or modify the class and method declarations, you won't usually do that. However, you may want to delete or modify some or all of the comments.

If the source code you want to work with isn't displayed in a code editor window, you can use the Projects window to navigate to the .java file and then double-click on it to open it in a code editor window. In this figure, for example, you can see the TestApp.java file in the Projects window. Notice that this file is stored in the default package of the Source Packages folder, since no package was specified when the project was created.

You can also rename or delete a .java file from the Projects window. To do that, just right-click on the file and select the appropriate command. If you rename a file, NetBeans automatically changes both the name of the .java file and the name of the class. Since the name of the .java file must match the name of the class, this is usually what you want.

## Net Bean's code editor with the starting source code for a project




## Description

- To open a .java file in the code editor, double-click on it in the Projects window. Then, you can use normal editing techniques to work with the source code.
- To collapse the code for a method or comment, click the minus sign (-) to its left. Then, a plus sign (+) appears to the left of the method or comment, and you can click the plus sign to display the code again.
- To save the source code for a file, use the File→Save command (Ctrl+S) or click the Save All Files button in the toolbar. This automatically compiles the file so it doesn't have to be compiled when the project is run.
- To rename a file, right-click on it, select the Refactor→Rename command, and enter the new name in the resulting dialog box.
- To delete a file, you can right-click on it, select the Delete command, and confirm the deletion in the resulting dialog box.

Figure 1-12 How to work with Java source code and files

## How to use the code completion feature

Figure 1-13 shows how to use the *code completion feature*. This feature prevents you from making typing mistakes, and it allows you to discover what fields and methods are available from various classes and objects. In this figure, for example, I started to enter a statement that prints text to the console.

First, I entered "sys" and pressed Ctrl+Spacebar (both keys at the same time). This displayed a list with the System class as the only option. Then, I pressed the Enter key to automatically enter the rest of the class name.

Next, I typed a period. This displayed a list of fields and methods available from the System class. Then, I used the arrow keys to select the field named out and pressed the Enter key to automatically enter that field name.

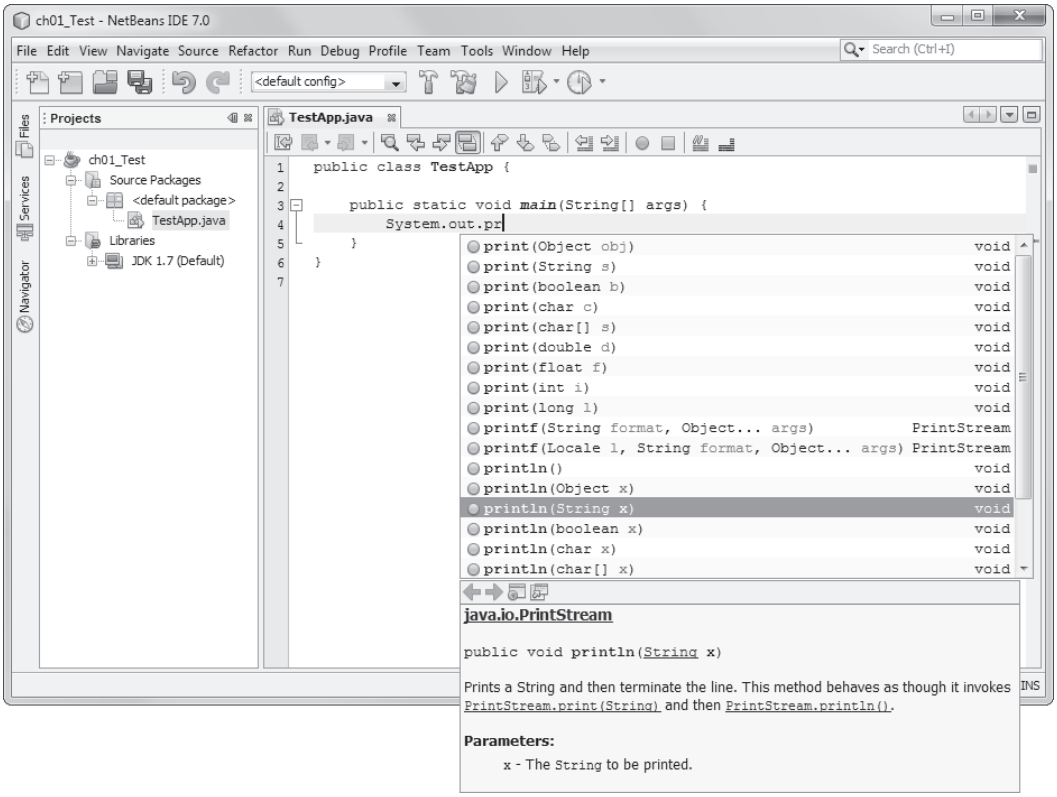
Finally, I typed another period. This displayed a long list of method names. Then, I typed "pr" to scroll down the list to the methods that start with "pr", and I used the arrow keys to select one of the println methods as shown in the figure. At this point, I could press Enter to have NetBeans enter the method into the editor for me.

When you use code completion, it automatically enters opening and closing parentheses and arguments whenever they're needed. In this figure, for example, you can see that the println method that I've selected is followed by a set of parentheses that contains a string argument. When I inserted this method into the code editor, the parentheses and arguments were inserted and the argument was highlighted so I could enter a value for it.

The code completion feature can also make it easy for you to enter values for string variables. If you type a quotation mark to identify a string value, the code completion feature automatically enters both opening and closing quotation marks and places the cursor between the two. At this point, you can enter the text for the string.

If you experiment with the code completion feature, you'll quickly see when it helps you enter code more quickly and when it makes sense to enter the code yourself. In addition, you'll see that it helps you understand the kinds of fields and methods that are available to the various classes and objects that you're working with. This will make more sense as you learn about object-oriented programming in Java beginning in the next chapter.

## The code editor with a code completion list




## Description

- You can use the *code completion feature* to help you enter the names of classes and objects and select from the methods and fields that are available for a class or object.
- To activate the code completion feature for entering a class or object name, press Ctrl+Spacebar after entering one or more letters of the class or object name. Then, a list of all the classes and objects that start with those letters is displayed.
- To activate the code completion feature for a method or field of a class or object, enter a period after a class or object name. Then, a list of all the methods and fields for that class or object is displayed.
- To insert an item from a code completion list, select the item and then press the Enter key. If the item requires parentheses, they're added automatically. If the item requires one or more arguments, default values are added for those arguments and the first argument is highlighted so you can enter its value. Then, you can press the Tab key and enter the values for any remaining arguments.
- If you enter the opening quote for a string value, the code completion feature automatically adds the closing quote and places the cursor between the two quotes so you can enter a value.

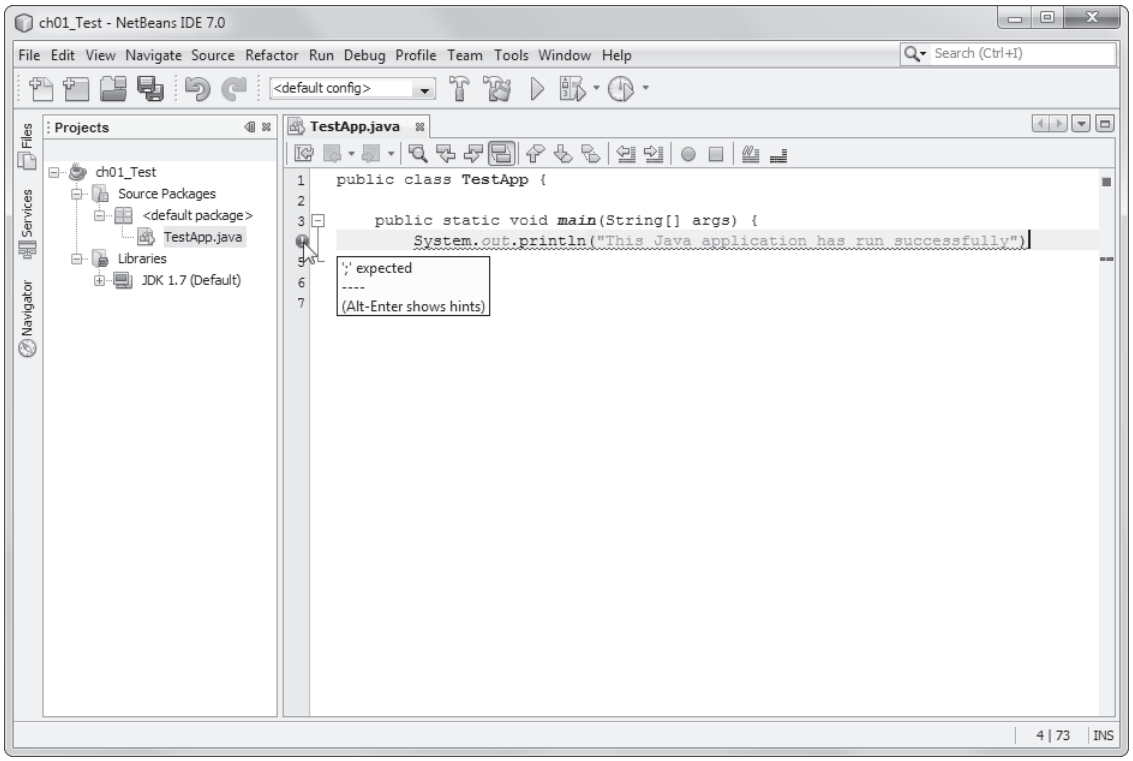
Figure 1-13 How to use the code completion feature

## How to detect and correct syntax errors

In NetBeans, a *syntax error* is caused by a statement that won't compile. As you enter text into the code editor, NetBeans displays syntax errors whenever it detects them. In figure 1-14, for example, NetBeans displays an error that indicates that a semicolon needs to be entered to complete the statement. This error is marked with a red icon to the left of the statement. In addition, the statement that contains the error is marked with a wavy red underline.

If you position the mouse cursor over the red error icon or over the statement itself, NetBeans displays a description of the error. In this figure, for example, the description indicates that NetBeans expects a semicolon at the end of the statement. As a result, you can fix the error by typing the semicolon.

## The code editor with an error displayed




## Description

- NetBeans often detects *syntax errors* as you enter code into the code editor.
- When NetBeans detects a syntax error, it displays a red error icon to the left of the statement in error and it places a red wavy line under the statement.
- To get more information about a syntax error, you can position the mouse pointer over the error icon. Or, you can move the cursor to the line that contains the error and press Alt+Enter.

Figure 1-14 How to detect and correct syntax errors

# Perspective

In this chapter, you were introduced to Java, and you learned how to use NetBeans to create and run a Java application. With that as background, you're ready to learn how to write your own Java applications. But first, I recommend that you familiarize yourself with NetBeans by doing the exercises at the end of this chapter.

## Summary

- You use the *Java Development Kit* (*JDK*) to develop Java applications. This used to be called the *Software Development Kit* (*SDK*) for Java.
- As of version 6, the *Standard Edition* (*SE*) of Java is called *Java SE*. In older versions, it was called the *Java 2 Platform, Standard Edition* (*J2SE*).
- You can use Java SE to create *applications* (also known as *desktop applications*) that run on your computer and a special type of Internet-based application known as an *applet*.
- A desktop application can use a *graphical user interface* (*GUI*) or a *console* to display output and get user input. Applications that use a console to interact with the user are known as *console applications*.
- You can use the *Enterprise Edition* (*EE*) of Java, which is known as *Java EE*, to create server-side applications using *servlets* and *JavaServer Pages* (*JSPs*).
- The *Java compiler* translates *source code* into a *platform-independent* format known as Java *bytecodes*.
- Any machine that has a Java interpreter installed on it can be considered an implementation of a *Java virtual machine* (*JVM*).
- An *Integrated Development Environment* (*IDE*) such as NetBeans can make working with Java easier.
- In NetBeans, a *project* is a folder that contains all of the files that make up an application.
- Java code is stored in *classes*. To organize multiple classes, you can store them in *packages*.
- The *main class* of an application is the class that contains the *main method*, which is the starting point of the application.
- If an application prints text to the *console*, NetBeans displays the text in the Output window. NetBeans also allows you to enter input into the Output window.
- When multiple projects are open, NetBeans identifies the *main project* by boldfacing its name in the Projects window.
- You can use the NetBeans *code editor* to enter and edit code. As you enter code, you can use the *code completion feature* to help you enter the names of classes and objects and select from fields and methods.

## Before you do the exercises for this chapter

Before you do any of the exercises in this book, you need to install the JDK and NetBeans. In addition, you need to install the source code for this book from our web site (www.murach.com). See appendix A (Windows) or appendix B (Mac OS X) for details.

## Exercise 1-1 Use NetBeans to open and run two projects

This exercise guides you through the process of using NetBeans to open and run two console applications.

### Open and run the Invoice application

1. Start NetBeans. When the Start Page is displayed, review the information on its tabs. Then, close this page.
2. Open the project named ch01_ex1_Invoice. The project should be stored in this directory:
   `C:\murach\java\netbeans\ex_starts`
3. Open the InvoiceApp.java file in the code editor and review its code to get an idea of how this application works.
4. Press F6 to run the application. Enter a subtotal when you're prompted, and then enter "n" when you're asked if you want to continue.

### Open and run the Test Score application

5. Open the project named ch01_ex2_TestScore. When you do, make sure to select the "Open as Main Project" option. Then, open the TestScoreApp.java file in the code editor and review its code.
6. Click the Run Project button in the toolbar to run the application. Enter one or more grades when you're prompted, and enter 999 to end the application.

### Set the main project and run the applications again

7. Set the Invoice application as the main project. Then, press F6 to run this application. When you're done, end the application.
8. Right-click on the Test Score application and select the Run command to run this application. When you're done, end the application.
9. Close both projects.

## Exercise 1-2 Use NetBeans to develop an application

This exercise guides you through the process of using NetBeans to enter, save, compile, and run a simple application.

### Enter the source code and run the application

1. Start NetBeans if it isn't already open.
2. Select the File→New Project command from the NetBeans menu system. Then, use the resulting dialog boxes to create a Java Application project named ch01_Test that contains a main class named TestApp. Store the project in this directory:

```
C:\murach\java\netbeans\ex_starts
```

3. Modify the generated code for the TestApp class so it looks like this (type carefully and use the same capitalization):

```
public class TestApp
{
    public static void main(String[] args)
    {
        System.out.println(
            "This Java application has run successfully.");
    }
}
```

4. Press F6 to compile and run the application. This should display "This Java application has run successfully." in the Output window.

### Use the code completion feature

5. Enter the statement that starts with System.out again, right after the first statement. This time, type "sys" and then press Ctrl+Spacebar. Then, use the code completion feature to select the System class, and complete the statement.
6. Enter this statement a third time, right after the second statement. This time, type System, enter a period, and select out from the list that's displayed. Then, enter another period, select println(String x), and complete the statement. You should now have the same statement three times in a row.
7. Run the application again to see that the message is displayed three times in a row in the Output window.

### Introduce and correct a syntax error

8. In the code editor window, delete the semicolon at the end of the first println statement, and NetBeans will display an error icon to the left of the statement.
9. Correct the error, and NetBeans will remove the error icon.
10. Use the File→Save command (Ctrl+S) to save the changes.

# How to build your Java programming skills

*Mike Murach, Publisher*

The easiest way is to let **Murach's Java Programming** be your guide! So if you've enjoyed this chapter, I hope you'll get your own copy of the book today. And don't miss its companion text for web programming, **Murach's Java Servlets and JSP**. You can use both books to:

- Teach yourself how to code desktop and web applications in Java
- Take advantage of all the time- and work-saving features of the NetBeans IDE as you develop Java applications
- Understand how object-oriented programming really works to create your own 3-tier database applications, the way the pros do
- Pick up a new skill whenever you want or need to by focusing on material that's new to you
- Look up coding details or refresh your memory on forgotten details when you're in the middle of developing a Java application
- Loan to your colleagues who will be asking you more and more questions about Java programming

To get your copy of either or both books, you can order online at **www.murach.com** or call us at 1-800-221-5528 (toll-free in the U.S. and Canada). And remember, when you order directly from us, this book comes with my personal guarantee:

### 100% Guarantee

*You must be satisfied. Each book you buy directly from us must outperform any competing book or course you've ever tried, or send it back within 90 days for a full refund…no questions asked.*

Thanks for your interest in Murach books!