How to use NetBeans with *Murach's Java SE 6*

NetBeans is a software framework for developing *Integrated Development Environments* (*IDEs*). In particular, it is used to develop the NetBeans IDE, which is an IDE for Java. NetBeans is open-source, available for free, and runs on all modern operating systems.

This tutorial has been designed to work with our beginning Java book, *Murach's Java SE 6*. To make it easy for you to use this tutorial with our Java book, each topic in this tutorial includes references to the related chapters in our book whenever that's necessary or helpful.

Mike Murach & Associates, Inc.
1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963
murachbooks@murach.com • www.murach.com

How to get started with NetBeans

Before you install the NetBeans IDE, you should install the need to install the *Java Development Kit* (*JDK*) for Java SE 6 as described in figure 1-4 in chapter 1 of *Murach's Java SE 6*. In addition, if you want to use NetBeans with our book, you should download the NetBeans version of the source code for our book from our web site (www.murach.com). To do that, download the zip file for this source code and unzip it into the C:\murach\java6 folder. If this folder doesn't already exist, you can create it.

How to download and install the NetBeans IDE

Once you have installed Java SE 6, you're ready to install the NetBeans IDE. Since most Java development is still done under Windows, figure 1 shows how to install NetBeans on a Windows system. In summary, once you download the exe file for the NetBeans installation program, you run this installation program and respond to the resulting dialog boxes. When you do, the installation program will install NetBeans and create a shortcut to the NetBeans.exe file in your Start menu. Since this works like most Windows installation programs, you shouldn't have any trouble doing this.

If you encounter any problems, you can view the documentation that's available from the NetBeans web site and consult the troubleshooting tips. If you want to install NetBeans on another operating system such as Linux, Solaris, or Mac OS X, you can follow the instructions that are available from the NetBeans website.

The NetBeans web site

```
www.netbeans.org
```

How to download and install NetBeans

1. Go to the NetBeans web site.
2. Download the NetBeans IDE installer. For NetBeans 5.5 on a Windows system, the exe file should be named something like netbeans-5_5-windows.exe.
3. Run the install file and respond to the resulting dialog boxes.

The default install folder

```
C:\Program Files\netbeans-5.5
```

Description

- Although this procedure is for downloading and installing NetBeans for Windows, you can use a similar procedure for non-Windows systems.
- For information about installing NetBeans on other operating systems or about trouble-shooting installation problems, you can refer to the documentation that's available from the NetBeans web site.

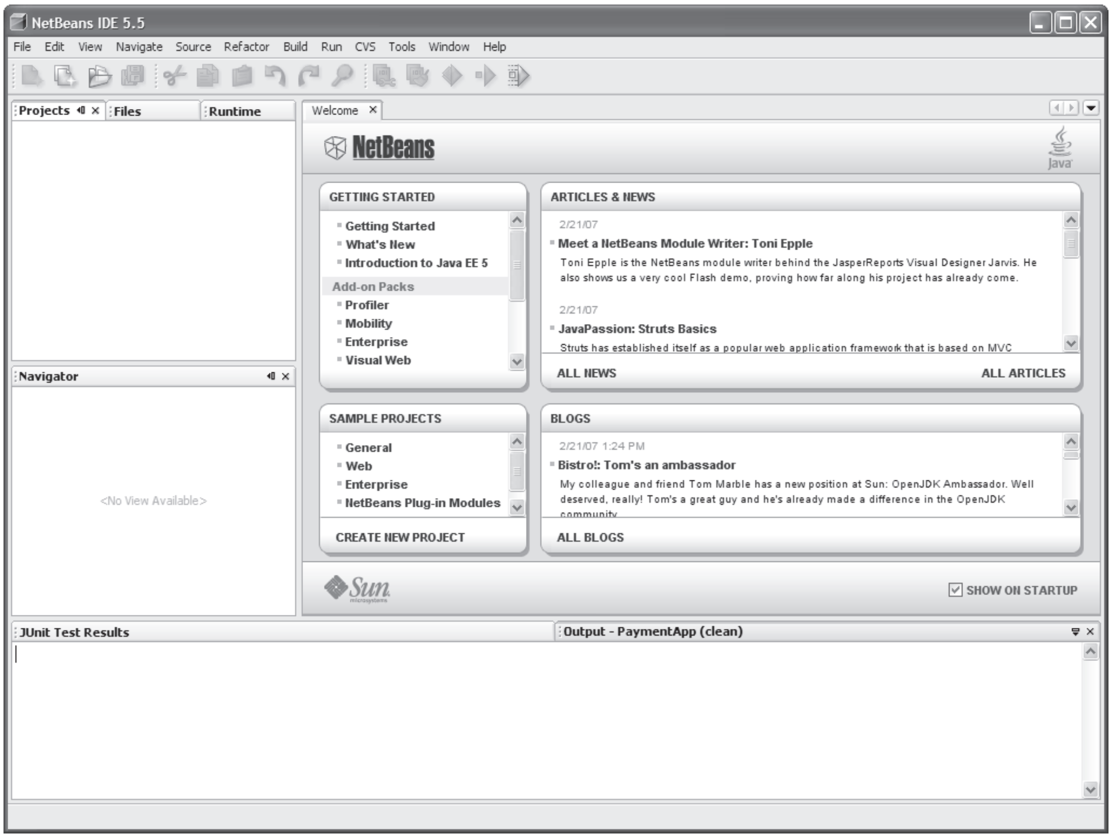
Figure 1 How to download and install NetBeans

How to start NetBeans

Once you've installed the NetBeans IDE, you can start it by selecting it from the Start menu. When start NetBeans for the first time, it should display a Welcome tab like the one shown in figure 2. If it doesn't, you can display this page by selecting the Welcome Screen command from the Help menu. Since you don't need this Welcome page to work with NetBeans, you can close it if you like. To do that, click the X that's displayed to the right of the tab for the Welcome page.

If you're curious to see what version of Java the NetBeans IDE uses by default, you can select the Java Platform Manager command from the Tools menu to display the Java Platform Manager dialog box. By default, NetBeans uses the latest version of Java that's installed on your system, which is usually what you want. However, if you want to use another version of Java, you can install that version of Java on your system and use the Java Platform Manager to specify that version of Java.

The Welcome page



2/21/07
Meet a NetBeans Module Writer: Toni Epple
Toni Epple is the NetBeans module writer behind the JasperReports Visual Designer Jarvis. He also shows us a very cool Flash demo, proving how far along his project has already come.

2/21/07
JavaPassion: Struts Basics
Struts has established itself as a popular web application framework that is based on MVC

2/21/07 1:24 PM
Bistro!: Tom's an ambassador
My colleague and friend Tom Marble has a new position at Sun: OpenJDK Ambassador. Well deserved, really! Tom's a great guy and he's already made a difference in the OpenJDK community

Description

- You can start NetBeans by selecting it from the Start menu just as you would for any other program. When you start NetBeans for the first time, it usually displays a Welcome tab like the one above.
- If the Welcome page isn't displayed on startup, you can display it by selecting the Welcome Screen command from the Help menu.

Figure 2 How to start NetBeans

How to create a new project with a main class

Figure 3 shows how to create a new NetBeans project. Essentially, a *project* is a folder that contains all of the files that make up an application. To create a new project, select the New Project command from the File menu. When you do, NetBeans will display a New Project dialog box like the one in this figure.

In the New Project box, you can select the Java Application option to use a wizard to create a new Java application (as opposed to the other types of projects that are available from NetBeans). Then, you can click on the Next button. When you do, NetBeans will display a dialog box like the second one in this figure.

In the New Java Application dialog box, you can enter a name for the project. In this figure, for example, the project name is "TestApp".

After you enter the name for the project, you can select the folder that the project should be stored in. In this figure, for example, the application is stored in this folder:

```
C:\murach\java6\netbeans\applications
```

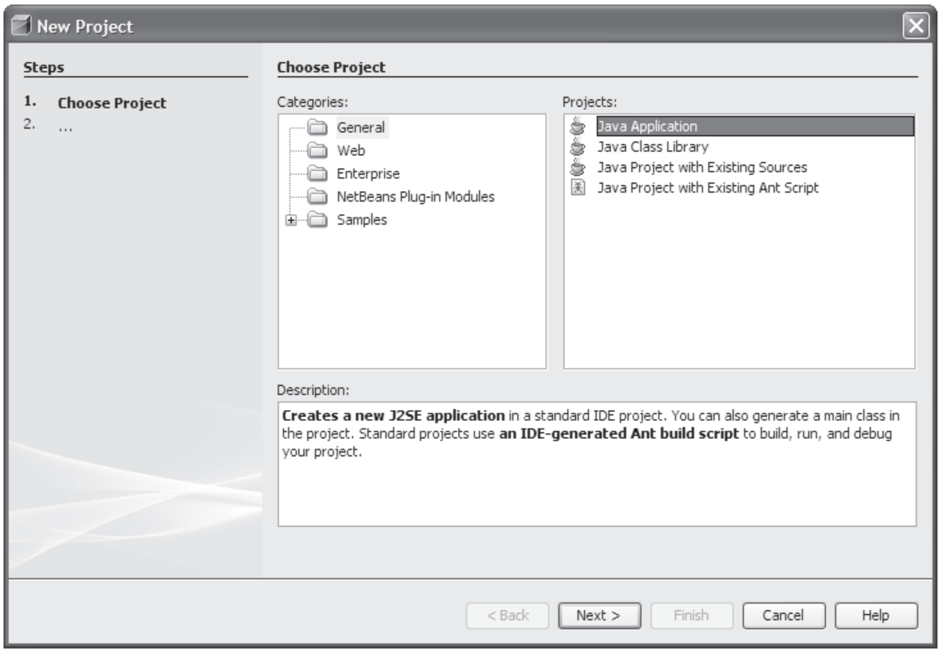
If you download the NetBeans files for *Murach's Java SE 6* and unzip them into the C:\murach\java6 folder, all of the applications presented in this book will be stored in this folder.

After you specify the folder for the project, you can use the first check box to specify whether this project is the main project. By default, this check box is selected, and that's usually what you want. However, you can easily change this later, so this choice isn't critical.

In addition, you can use the second check box to create a class that contains a main method and to specify the name of that class. By default, this dialog box creates a class named Main that contains a main method and it stores this class in a package that has the same name as the project. For example, by default, this dialog box suggests testapp.Main as the name for the class that stores the main method for the TestApp project. However, in this figure, I changed this suggested name to TestApp. As a result, the TestApp project will contain a class named TestApp that contains a main method, and this class will be stored in the default package that's used when you don't specify a package name.

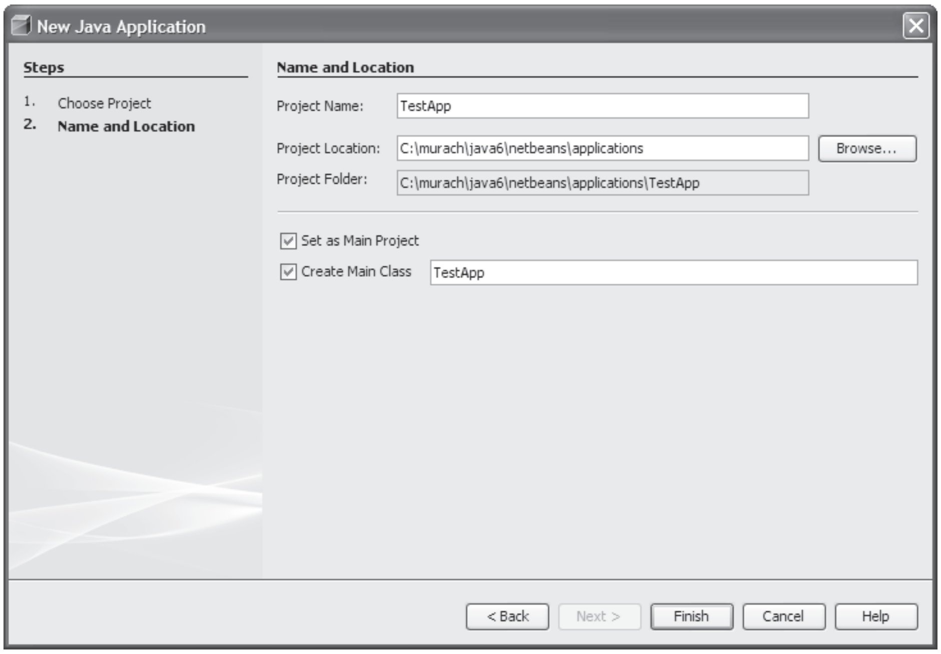
Finally, you can click on the Finish button to create the project and the class that contains the main method. When you do, NetBeans creates a folder that corresponds with the project name and it creates some additional files that it uses to configure the project.

The first dialog box for creating a new project



Creates a new J2SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

The second dialog box for creating a new project



Project Name: TestApp

Project Location: C:\murach\java6\netbeans\applications

Project Folder: C:\murach\java6\netbeans\applications\TestApp

☑ Set as Main Project

☑ Create Main Class TestApp

Description

- To create a new project, select the File➔New Project command from the menu system and respond to the resulting dialog boxes.

Figure 3 How to create a new project with a main class

How to save and edit source code

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 5. To make it easier to for you recognize the Java syntax, the code editor uses different colors for different types of syntax. 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 new class, NetBeans often generates some code for you. In this figure, for example, NetBeans generated the code that declares the class, and it generated the code that declares the main method (since this was specified by the dialog box in figure 3). For more information about declaring a class or coding a main method, you can read chapter 2 of *Murach's Java SE 6*.

If you want, you can delete or modify the generated code. For example, when I created the TestApp class, NetBeans generated a constructor for the class. However, since the TestApp class doesn't need a constructor, I deleted it.

NetBeans also generates some comments that describe the class and its methods. For the TestApp class, I deleted all comments except the javadoc comment that was generated before the main method. Then, I entered a statement that prints text to the console within the main method, and I modified the placement of the braces.

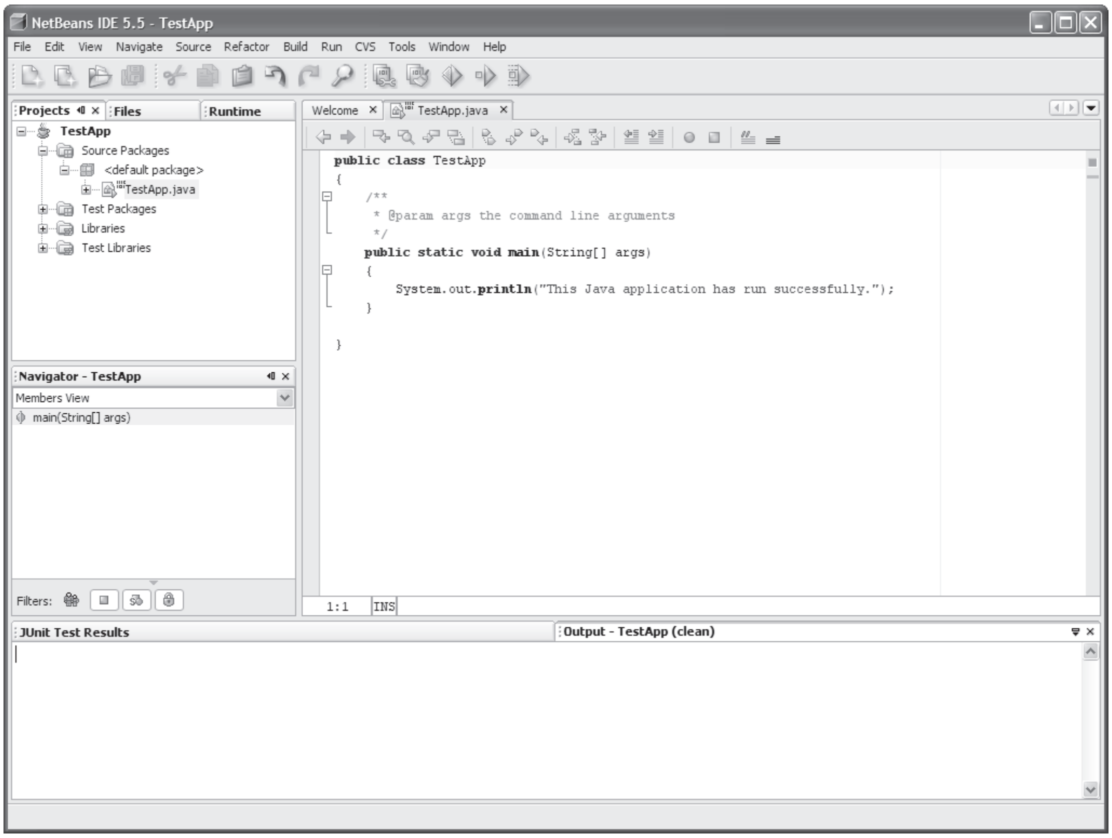
When you enter code, you can use the *code completion* feature that's available from NetBeans to complete your code. This feature prevents you from making typing mistakes, and it allows you to discover what methods are available from various classes and objects. In this figure, for example, I used code completion when entering the statement that prints text to the console. To start, I entered "sys" and pressed Ctrl+Spacebar (both keys at the same time). This displayed a list of possible options. Then, I selected an option from the list that looked like this:

```
System.out.println("|");
```

When I did, NetBeans entered this code into the editor for me and placed the cursor between the two quotation marks. Finally, I typed the text that should be printed to the console between the quotation marks. If you experiment with the code completion feature, you'll quickly figure out when it helps you enter code more quickly and when it makes sense to enter the code yourself.

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 double-click on it to open it in a code editor window. In this figure, for example, the TestApp.java file that's open in the code editor is also shown in the Projects window. As you can see, this file is stored in the default package of the Source Packages folder of the TestApp project.

NetBeans's code editor with source code in it



```
public class TestApp
{
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        System.out.println("This Java application has run successfully.");
    }

}
```

Description

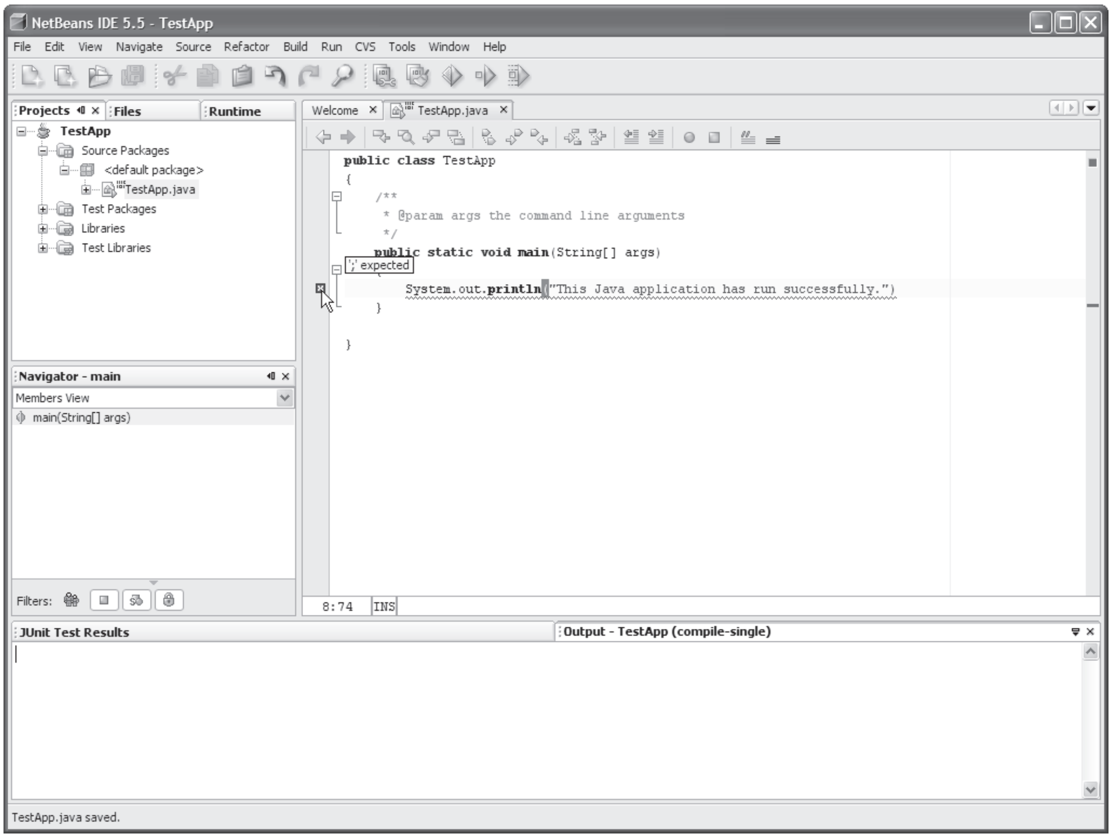
- To open a code editor window for a file, double-click on the .java file in the Projects window.
- To enter and edit source code, you can use the same techniques that you use with any text editor.
- To activate the code completion feature, press Ctrl+Spacebar after entering the first few letters of a class or object. Or, enter the period after a class or object. Then, you can select an item from a list of possibilities.
- To save the source code, select the Save command (Ctrl+S) from the File menu.
- By default, NetBeans may generate some code. You can delete this code or modify it for use in your application.

How to fix errors

In NetBeans, an *error* is a line of code that won't compile. As you enter text into the code editor, NetBeans displays errors whenever it detects them. In figure 5, for example, NetBeans has displayed an error that indicates that a semicolon needs to be entered to complete the statement. This error is marked with a red icon that has an X on it just to the left of the statement that contains the error. In addition, the statement that contains the error is marked with a wavy red underlining.

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

An error that's displayed when you save the source code



```
public class TestApp
{
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        System.out.println("This Java application has run successfully.")
    }

}
```

';' expected

TestApp.java saved.

Description

- NetBeans often displays errors before you attempt to compile or run an application.
- NetBeans marks errors with a red square with an X in it in the code editor window.
- If you position the mouse pointer over the error, NetBeans will display a description of the error. This usually provides enough information for you to fix the error.

Figure 5 How to fix errors

How to compile and run an application

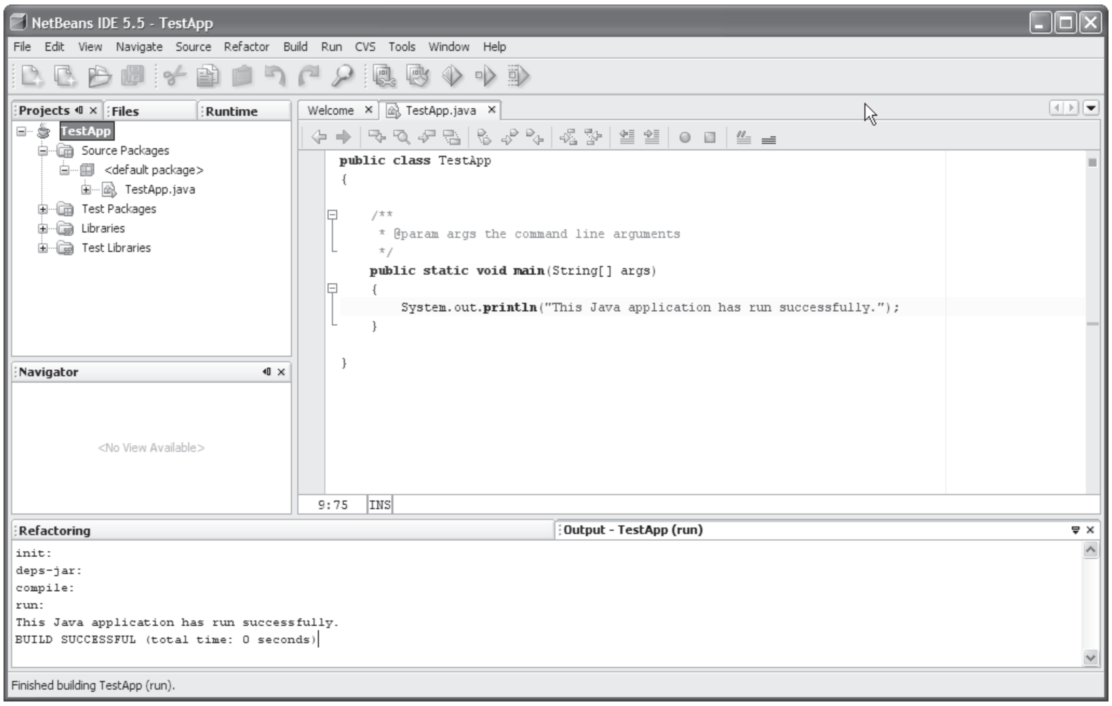
By default, NetBeans automatically compiles an application before it runs the application. Since this saves a step in the development process, this is usually what you want.

An easy way to run an application for the first time is to press F6. Then, NetBeans will run the main class in the main project. In this figure, for example, you only have one project and one class. As a result, NetBeans runs the TestApp class in the TestApp project. This prints a line of text to the Output window, which is the default *console* that's used by NetBeans. Since this application just prints text to the console, it can be referred to as a *console application*.

If you have several projects open at the same time, you can specify the *main project* by right-clicking on the project and selecting the Set As Main command. Similarly, if a project contains several classes that have a main method, you can specify the *main class* by right-clicking on the project, selecting the Properties command, clicking on the Run category, and using the dialog box to specify the class. Or, if you just want to run the class, you can use the Projects window to navigate to the .java file for the class, right-click on it, and select the Run File command.

Now that you know how to create a simple project, you have all the skills you need to complete the first exercise that's presented at the end of this tutorial. If you're working through *Murach's Java SE 6*, you can complete this exercise when you finish chapter 1.

An application that prints text to the console



```
public class TestApp
{

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        System.out.println("This Java application has run successfully.");
    }

}
```

```
init:
deps-jar:
compile:
run:
This Java application has run successfully.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Description

- When you run an application, NetBeans automatically compiles the application. As a result, you don't need to compile an application separately. If you want to compile an application without running it, though, you can select one of the commands from the Build menu.
- To run the main class from the main project, press F6 or select the Run Main Project command from the Run menu or from the toolbar.
- If necessary, you can set the *main project*, by right-clicking on the project and selecting the Set Main Project command.
- If necessary, you can set the *main class*, by right-clicking on the project, selecting the Properties command, selecting the Run category, and using the dialog box to specify the main class for the project.
- To run a class that has a main method, right-click on the .java file and select the Run File command.
- When the application prints to the *console*, NetBeans displays an Output window like the one shown above.

Figure 6 How to compile and run an application

How to run a console application that gets user input

In addition to printing data to the console, a console app can also get input from a user. Unfortunately, the Output window that NetBeans uses as the default console doesn't act like a typical console window. Worse, NetBeans has a bug that prevents you from using the print method of the System.out object when working with console applications that use the Output window. As a result, when developing console apps that get user input, we recommend using an interactive console that was developed by Eric G. Berkowitz, an Associate Professor at Roosevelt University in Chicago.

To use this interactive console instead of the Output window, you can follow the procedure shown in figure 7. To start, you must add the file named eric.jar to the Libraries folder for your project. This file is included in the NetBeans versions of the applications that you can download from our web site (www.murach.com).

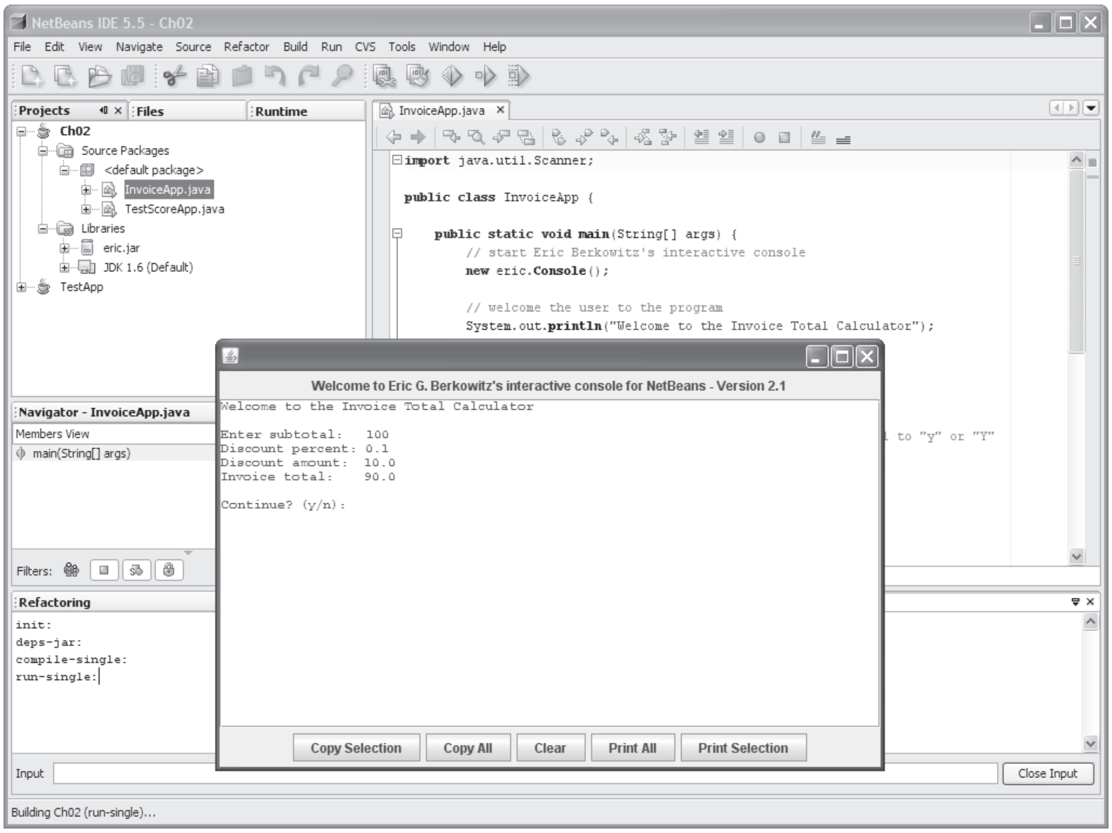
After you add the eric.jar file to your project, you must add a line of code to the beginning of your application that starts the interactive console. This line of code creates a new instance of the Console class that's stored in the eric package of the eric.jar file. As a result, the applications use the interactive console instead of the Output window that's available from NetBeans.

Once you've added this line of code to the beginning of your console app, you can run the application as you would normally. When you do, the interactive console should appear and the application should print some text to the console that prompts you to enter data. Then, you can type the input into the console and press Enter. When you do, the application will continue until it finishes or until it prompts you for more information.

In this figure, for example, the application prompted me to enter a subtotal, and I typed "100" and pressed Enter. Then, the application asked me if I wanted to continue. At this point, the application is still running, and I can enter "y" to continue or "n" to stop the application. However, even if I enter "n" to stop the application, NetBeans doesn't finish running the application until you close the interactive console window. As a result, you should always close the console window after you're done running an application. Or, if you don't want to finish running the application, you can stop the application at any point during its execution by closing the interactive console window.

When you're learning Java, it's common to create applications that use the console to get input from the user and to display output to the user. You'll learn how to create this type of application in chapter 2 of our book, and this type of application is used in chapters 2 through 14.

A console app that uses an interactive console to get input from a user



```
import java.util.Scanner;

public class InvoiceApp {

    public static void main(String[] args) {
        // start Eric Berkowitz's interactive console
        new eric.Console();

        // welcome the user to the program
        System.out.println("Welcome to the Invoice Total Calculator");
```

```
Welcome to the Invoice Total Calculator

Enter subtotal:   100
Discount percent: 0.1
Discount amount:  10.0
Invoice total:    90.0

Continue? (y/n):
```

```
init:
deps-jar:
compile-single:
run-single:
```

Building Ch02 (run-single)...

How to use Eric G. Berkowitz's interactive console

- Add the eric.jar file to the Libraries folder for your project. To do that, right-click on the Libraries folder, select the Add Jar/Folder, and use the resulting dialog box to select the jar file (which should be in c:\murach\java6\netbeans).
- Add the following line of code to the beginning of your console application:

  `new eric.Console();`
- When you run the application, the interactive console window will appear and you can enter text by typing text and pressing the Enter key.
- When you're done running the application, close the console window.

Description

- NetBeans has a bug that prevents it from working properly with the print method of the System.out object. As a result, we recommend using the interactive console that was developed by Eric G. Berkowitz.
- The eric.jar file is included with the NetBeans versions of the applications that are available from our web site, and these applications have already been configured to use the interactive console.

Figure 7 How to run a console application that gets user input

How to open and close projects and import source code

Once you understand how to use NetBeans to run simple applications like the ones presented in chapters 1 and 2, you may want to add more projects to the Projects window. For example, if you have an existing NetBeans project that isn't displayed in the Projects window, you may want to add that project to the project window by opening it. Or, if you have some code that was created with a text editor or another IDE, you may want to add a new project to the Projects window and import this code into that project so you can work with it.

Then, if the Projects window becomes cluttered with multiple projects, you may want to remove one or more of these projects from the Projects window. For example, if you download the NetBeans version of the source code for *Murach's Java SE 6*, you'll find that the applications folder contains more than 10 projects. Similarly, the exercises folder contains over 40 projects. As a result, you won't want to have all of these projects open at the same time. Instead, you can open the ones you want to work with and close them when you're done with them.

How to open a project

To add an existing project to the Projects window, you can select the Open command from the File menu and use the Open Project dialog box as shown in figure 8. To start, you can navigate to the folder that contains the project or projects that you want to add. When you do, all of the possible projects will be displayed in the Open Project dialog box so you can select the projects that you want to add. In this figure, for example, the Open Project dialog box shows all of the existing NetBeans projects for the applications presented in *Murach's Java SE 6*.

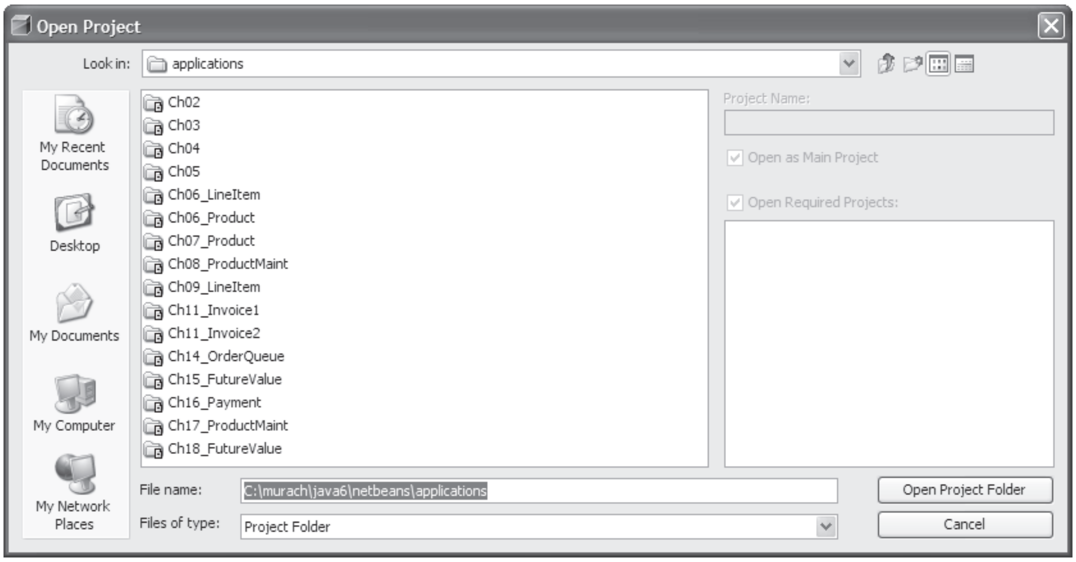
To clearly indicate when a folder is a project folder, the Open Project dialog box displays a small green icon in the lower right corner of the folder icon. In this figure, for example, all of the folder icons have this small green icon. As a result, you can add any of these projects to the Projects window by selecting the folder and clicking on the Open Project Folder button.

How to close a project

To remove a project from the Projects window, you can right-click on the project and select the Close Project command. For example, to remove the project named TestApp that's shown in figure 7, you can right-click on the project in the Projects window and select the Close Project command. Since this doesn't delete the files for the project, you can easily open the project later.

Alternately, you can remove the project from the Projects window and delete its files by right-clicking on the project and selecting the Delete Project

The dialog box for opening an existing NetBeans project



Description

- To add a project to the Projects window, right-click on the project in the Projects window and select the Open Project command. Then, use the Open Project dialog box to locate the folder for the project and click on the Open Project Folder button.
- The Open Project dialog box identifies a NetBeans project by displaying a small icon in the lower right corner of the icon for its project folder.
- To remove a project from the Projects window, right-click on the project in the Projects window and select the Close command.
- You can download a zip file from www.murach.com that contains the NetBeans projects for all of the applications described in *Murach's Java SE 6*.

command. When you do, NetBeans will prompt you to confirm the deletion. By default, NetBeans deletes most of the files for the project but does not delete the source code. However, if you select the "Also Delete Sources" option, NetBeans will delete all folders and files for the project. Of course, if you delete the files from a project, it's no longer easy to open the project with NetBeans. As a result, you'll only want to use this option if you don't plan on using NetBeans to work with the project anymore.

Now that you know how to run the interactive console and how to open existing projects, you have all the skills you need to complete the second exercise that's presented at the end of this tutorial. If you're working through *Murach's Java SE 6*, you should do this exercise before you do the exercises for chapter 2 of the book.

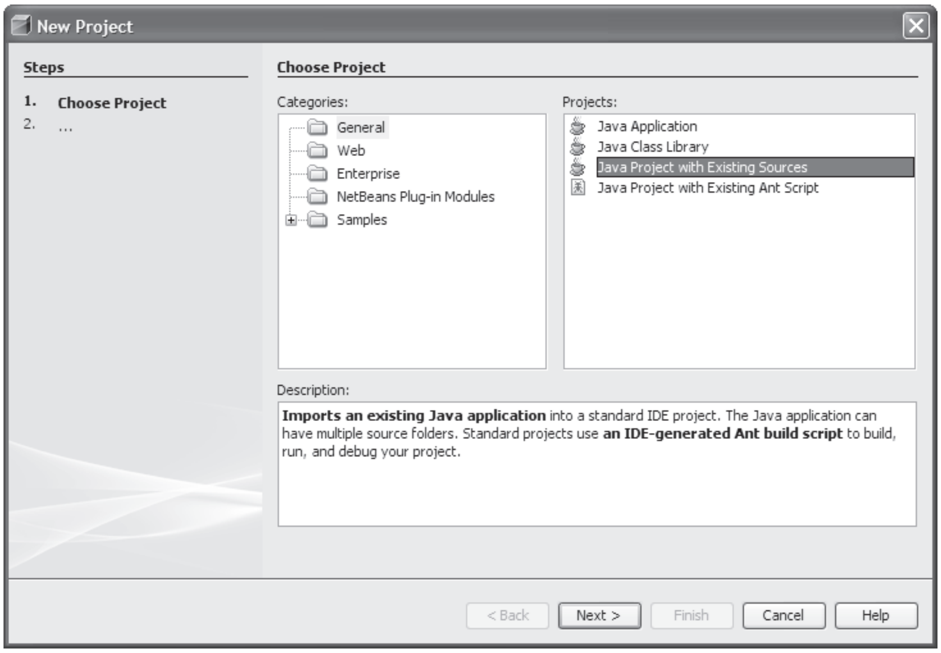
How to import existing Java files into a new project

Before you import existing Java files into a NetBeans project, you should move or copy the .java files into the right folder for storing your source code. Often, that means moving the .java files into a subfolder of the folder for the project. In this figure, for example, the .java files are stored in the src subfolder of the C:\murach\java6\netbeans\applications\Ch02 folder.

Once you have stored the .java files in the correct folder, you can use the New Project dialog box to create a new project and import these files into the project. To start, you can select the New Project command from the File menu to display the first dialog box shown in figure 9. In this dialog box, you can select the "Java Project with Existing Sources" option to create a new project that imports existing .java files.

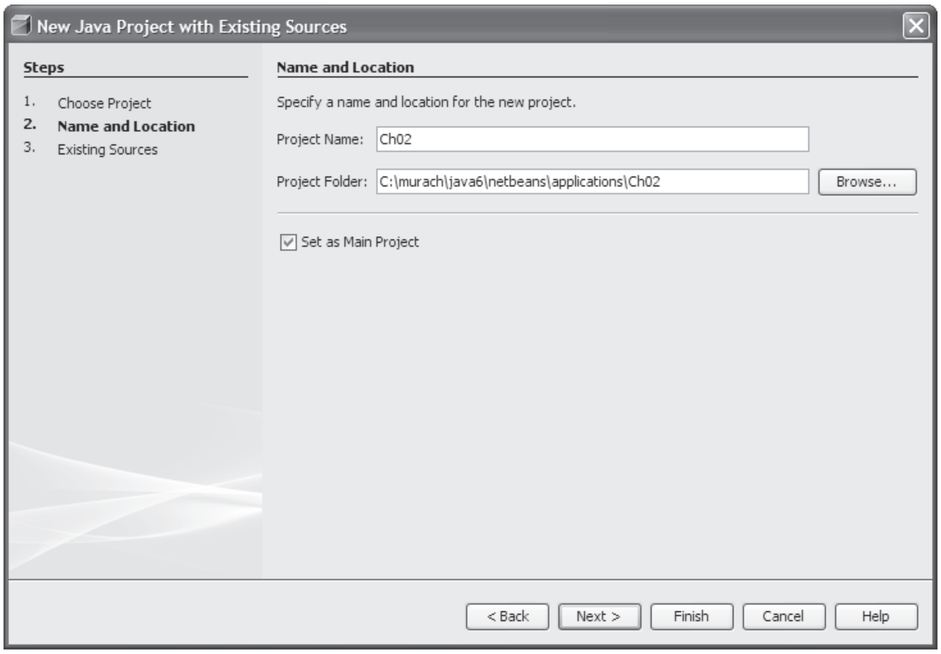
Then, you can use the second dialog box shown to specify the name and location for the project. In this figure, for example, the second dialog box creates a project named Ch02 in the C:\murach\java6\netbeans\applications\Ch02 folder. Note that this folder already exists since it had to be created to store the src subfolder that contains the .java files.

The first dialog box for importing Java files into a NetBeans project



Imports an existing Java application into a standard IDE project. The Java application can have multiple source folders. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

The second dialog box for importing Java files into a NetBeans project



Specify a name and location for the new project.

Project Name: Ch02

Project Folder: C:\murach\java6\netbeans\applications\Ch02

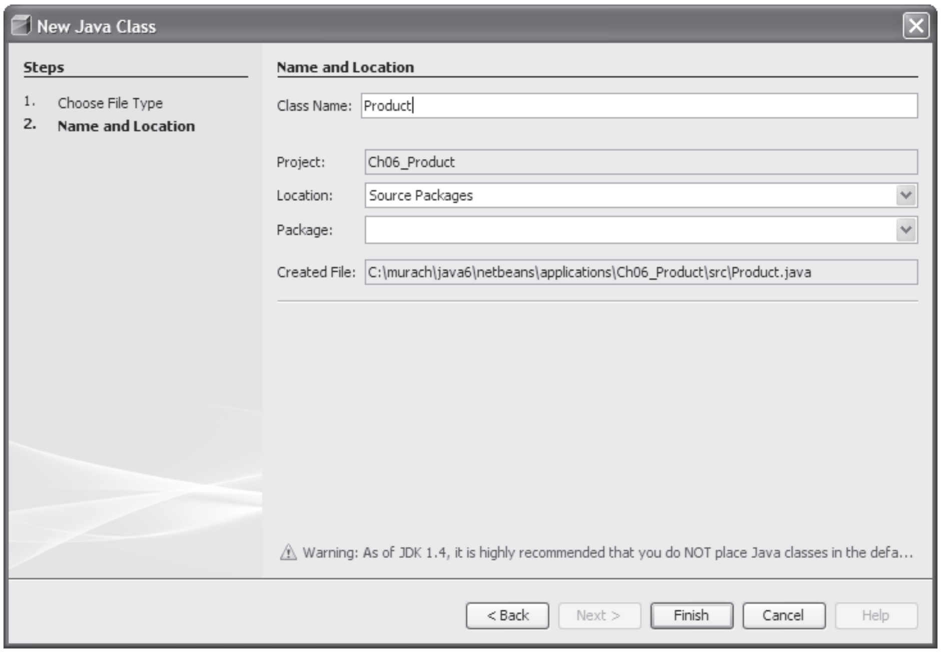
[✓] Set as Main Project

Figure 9 How to import existing Java files into a new project (part 1 of 2)

Finally, in the third dialog box, you can select the src folder that contains the .java files. When you click on the Finish button to complete the process NetBeans creates the Ch02 project and imports all .java files in the src subfolder of this project.

When you use this technique to import .java files, you need to pay attention to the packages that contain these files. If you use the third dialog box to select the root folder for the application you want to import, NetBeans usually places the .java files in the appropriate folders. If, for example, the src folder contains a subfolder named murach/business that corresponded with a package named murach.business, NetBeans will place any classes within this subfolder in the corresponding package within the Ch02 project. Usually, that's what you want when you're importing .java files. If it isn't, you can use the third dialog box to import the files in the murach/business subfolder into the default package. Or, you can use the skills described in figure 14 to reorganize these packages.

The third dialog box for importing Java files into a NetBeans project



Class Name: Product

Project: Ch06_Product

Location: Source Packages

Package:

Created File: C:\murach\java6\netbeans\applications\Ch06_Product\src\Product.java

Warning: As of JDK 1.4, it is highly recommended that you do NOT place Java classes in the defa...

Description

- Before you import existing Java files into a new NetBeans project, you should move or copy your .java files into the folder where you want to store your source code. In this figure, for example, the source files are stored in the Ch02\src folder.
- To import existing Java files into a new NetBeans project, select the New Project command from the File menu, select the "Java Project with Existing Sources" option and respond to the resulting dialog boxes.

Object-oriented development with NetBeans

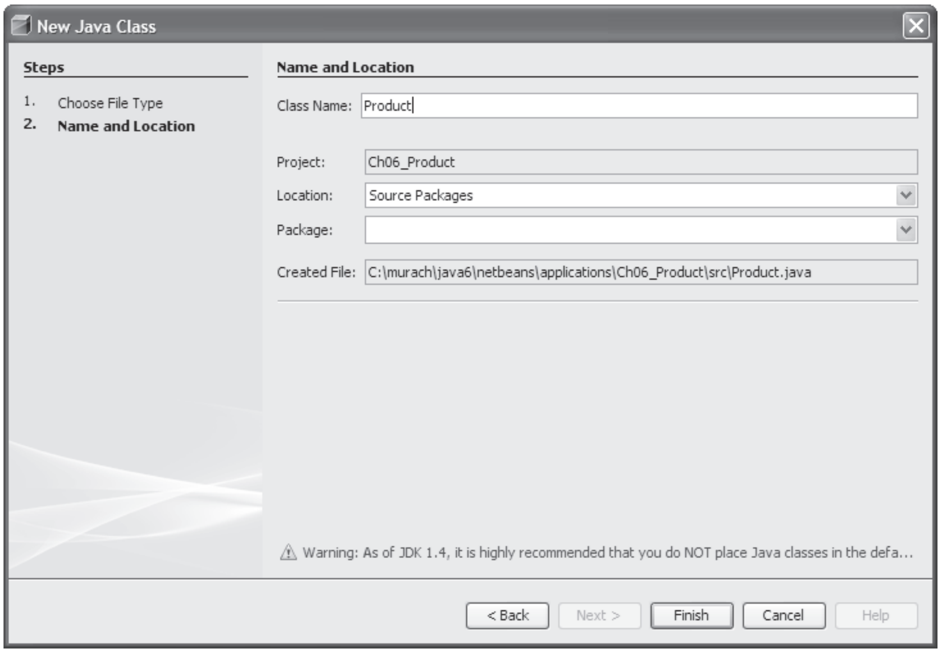
NetBeans has many features that make it easier to work with the object-oriented programming techniques that are presented in chapters 6 through 9 of *Murach's Java SE 6*. For example, NetBeans makes it easy to create get and set methods for a class, to begin coding a class that implements an interface, and to store classes and interfaces in packages. However, these skills don't make sense until you understand object-oriented programming. As a result, you may want to read chapters 6 through 9 of *Murach's Java SE 6* before you go through the topics that follow.

How to create a new class

In figure 3, you learned how to create a project that contains a single class that contains a main method. However, when you start developing object-oriented applications, you'll need to add other classes to your project. To do that, you can display the New Java Class dialog box shown in figure 10. Then, you can use this dialog box to add new classes to your project. In this figure, for example, the New Java Class dialog specifies a public class named Product.

Although the New Java Class dialog box encourages you to enter a package for the class, this isn't required. If you don't enter a package for the class, NetBeans will use the default package. However, if you enter a package for the class, NetBeans will automatically create a folder for the package and store the class within that package. For more information about working with packages, see figure 14 of this tutorial and chapter 9 of *Murach's Java SE 6*.

The dialog box for creating a new class



Class Name: Product

Project: Ch06_Product

Location: Source Packages

Package:

Created File: C:\murach\java6\netbeans\applications\Ch06_Product\src\Product.java

Warning: As of JDK 1.4, it is highly recommended that you do NOT place Java classes in the defa...

Description

- To create a new class, right-click on the package where you want to add the class, select the New→Java Class command, and respond to the resulting dialog boxes.
- You must enter a name for the class in the Class Name text box.
- Although this dialog box encourages you to enter a package for the class, this isn't required. If you don't enter a package for the class, NetBeans will use the default package.

Figure 10 How to create a new class

How to work with classes

Figure 11 describes a few skills that are useful for working with classes. To start, after you enter the private fields for a class, you can generate the get and set methods for those fields. In this figure, for example, the Encapsulate Fields dialog box will generate one get and one set method for each of the private fields for the class.

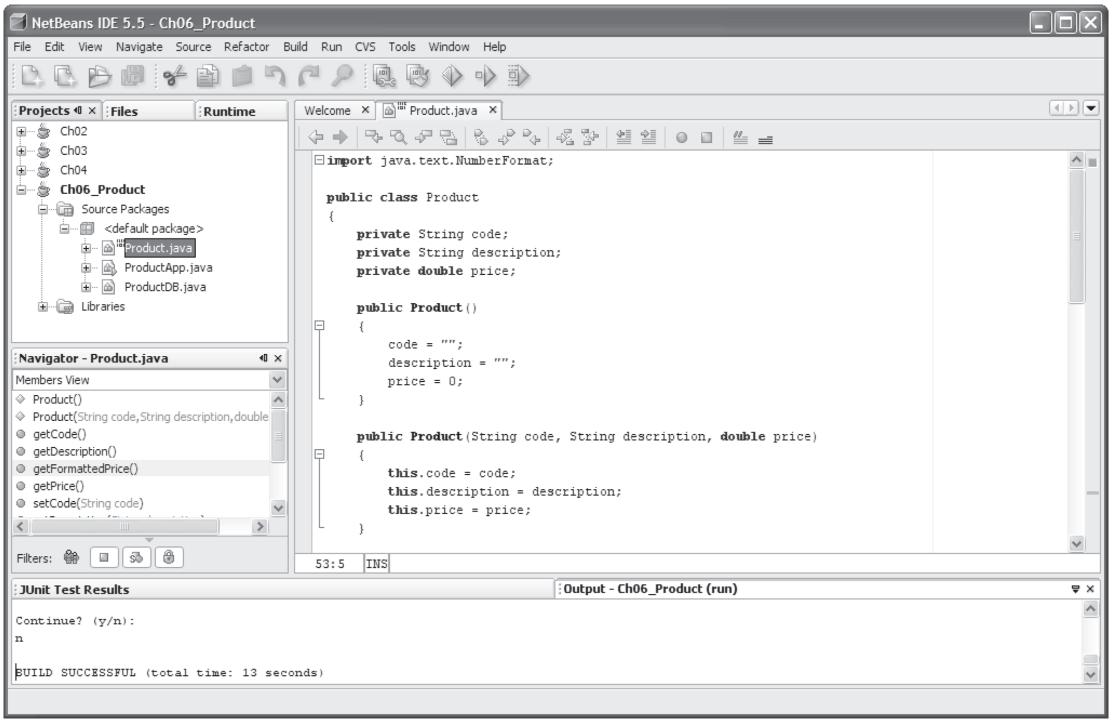
By default, the get and set methods for a field are formatted like this:

```
public String getCode() {
    return code;
}

public void setCode(String code) {
    this.code = code;
}
```

However, if you have another coding style that you prefer, you can select the Tools➔Options command, click on the Editor tab, and use it change the coding style. For example, you can select the "Add New Line Before Brace" option to add a new line character before the opening braces.

Once you've created a class that contains multiple methods, you can use the Navigator window to navigate to that method. To do that, just double-click on the method in the Navigator window. In this figure, for example, you could double-click on the getFormattedPrice method to display that method in the code editor. Although the usefulness of this feature isn't obvious for a short class like the Product class shown in this figure, it's very helpful for longer classes.

The NetBeans window for the Product application in chapter 6

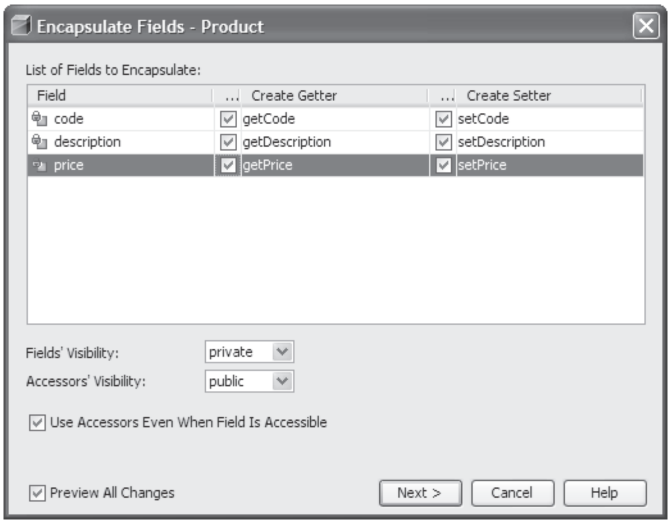


```
Product()
Product(String code,String description,double
getCode()
getDescription()
getFormattedPrice()
getPrice()
setCode(String code)
```

```
import java.text.NumberFormat;

public class Product
{
    private String code;
    private String description;
    private double price;

    public Product()
    {
        code = "";
        description = "";
        price = 0;
    }

    public Product(String code, String description, double price)
    {
        this.code = code;
        this.description = description;
        this.price = price;
    }
```

```
Continue? (y/n):
n

BUILD SUCCESSFUL (total time: 13 seconds)
```

The dialog box for generating the get and set methods for a field



Field	Create Getter	Create Setter
code	☑ getCode	☑ setCode
description	☑ getDescription	☑ setDescription
price	☑ getPrice	☑ setPrice

Fields' Visibility: private
Accessors' Visibility: public
☑ Use Accessors Even When Field Is Accessible
☑ Preview All Changes

Description

- To generate the get and set methods for a field, select the Refactor➔Encapsulate Fields command from the menu system and respond to the resulting dialog boxes.
- To jump to a method, double-click on the method in the Navigator window.

Figure 11 How to work with classes

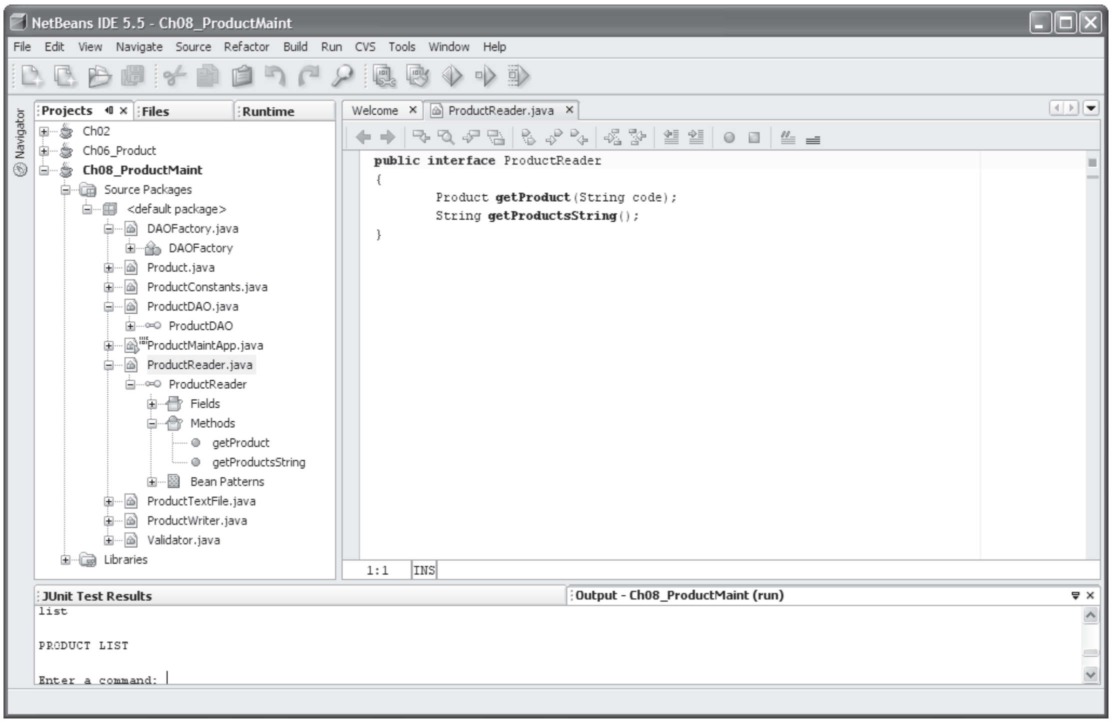
How to work with interfaces

Figure 12 shows the NetBeans window for the Product Maintenance application presented in chapter 8. If you look at the Projects window, you can see that the project for this application contains source code files for several classes and interfaces. If you expand the nodes for the .java files that contain the source code, you can see that NetBeans uses different icons to identify the files as classes or interfaces. In this figure, for example, the DAOFactory.java file contains the source code for a class while the ProductDAO.java and ProductReader.java files contain the source code for interfaces.

To add an interface, you can right-click on the package where you want to add the interface and select the New➔Java Interface command. Then, you can use the resulting dialog box to enter a name for the interface. This dialog box works like the one for creating a new class that's shown in figure 10.

Once you've added an interface, you shouldn't have any trouble entering and editing the code for the interface. In general, you can use many of the same skills that you use for entering and editing the code for a class.

The NetBeans window for the Product Maintenance application in chapter 8



```
public interface ProductReader
{
        Product getProduct(String code);
        String getProductsString();
}
```

```
list

PRODUCT LIST

Enter a command:
```

Description

- The Projects window identifies interfaces by using the interface icon, which is a small circle and a large circle connected by two lines.
- To add an interface to a project, right-click on the package you want to add the interface to, select the New→Java Interface command, and use the resulting dialog box to enter a name for the interface.

Note

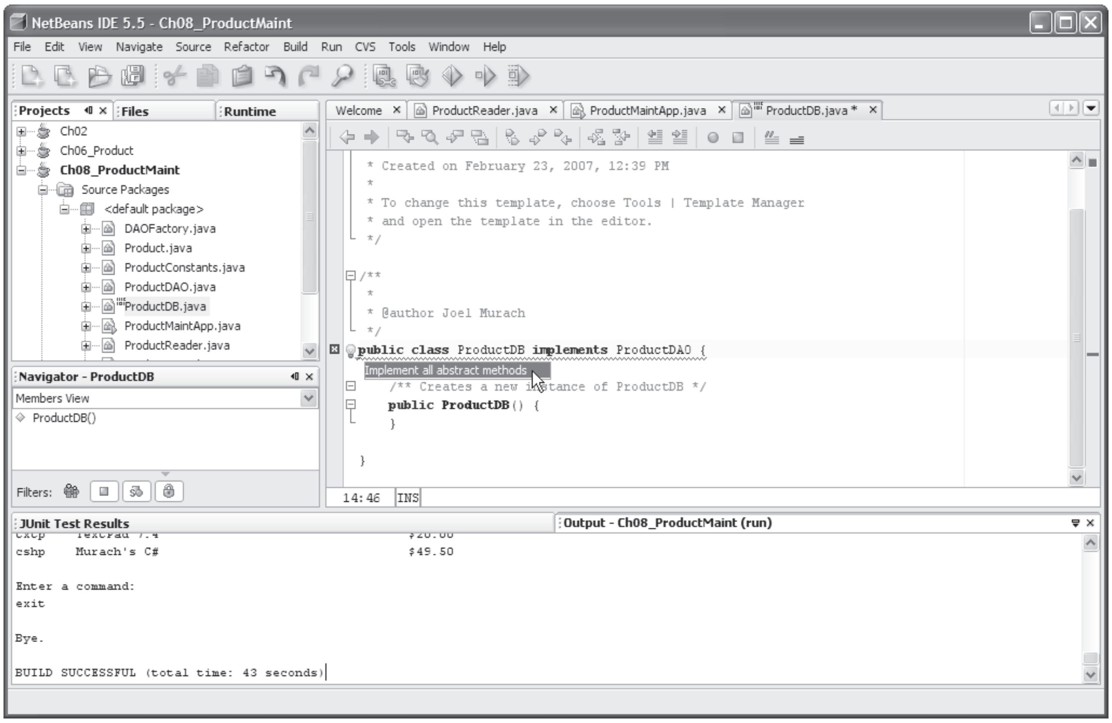
- The products.txt file that's used by this application is stored in the netbeans\Ch08_ProductMaint folder. In other words, it's stored in the root folder for the project.

How to start a class that implements an interface

When coding a class that implements an interface, you can automatically generate all the method stubs for the interface. To do that, you can begin by creating a new class. Then, you enter the implements keyword followed by the interface or interfaces that the class implements. When you do, a yellow light bulb icon will be displayed to the left of the declaration for the class.

If you click on this icon, you'll get a menu with the "Implement all abstract methods" command as shown in figure 13. To generate all method stubs for the interfaces specified by the class declaration, you can select this command. Then, NetBeans will generate the method stubs for all of the methods specified by the interface or interfaces that are implemented by the class.

A class that implements the ProductDAO interface in chapter 8



```
 * Created on February 23, 2007, 12:39 PM
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

/**
 *
 * @author Joel Murach
 */
public class ProductDB implements ProductDAO {

    /** Creates a new instance of ProductDB */
    public ProductDB() {
    }

}
```

```
cxcp    TextPad 7.4                          $20.00
cshp    Murach's C#                          $49.50

Enter a command:
exit

Bye.

BUILD SUCCESSFUL (total time: 43 seconds)
```

Description

- When coding a class that implements an interface, you can automatically generate all the method stubs for the interface. To do that, create a new class as described in figure 10, use the implements keyword to identify the interface, click on the light bulb icon, and select the "Implement all abstract methods" command as shown above.

Figure 13 How to start a class that implements an interface

How to work with packages

Often, the classes for an application are organized into *packages*. Compared to using a text editor, NetBeans makes it easy to create packages and to store your classes in packages.

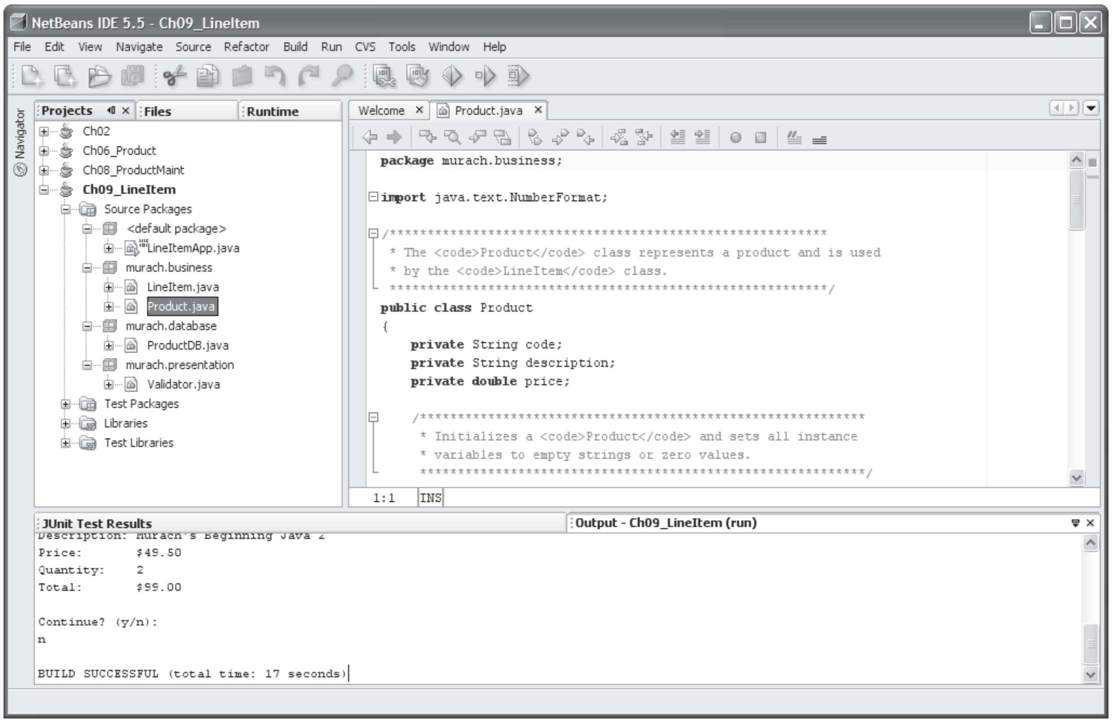
When a project contains packages, you can use the Projects window to navigate through the project's packages. To do that, you can click on the plus and minus signs to the left of the packages to expand or collapse them. In figure 14, for example, the Projects window displays the four packages that are used for the Line Item application that's presented in chapter 9 of *Murach's Java SE 6*.

To get started with packages, you can add a new package to a project by right-clicking on the project and selecting the New➔Java Package command. When you do, you'll get a dialog box that allows you to enter a name for the package. As you create packages, remember that packages correspond to the folders and subfolders that are used to store the source code. In this figure, for example, the murach.business package is stored in the murach/business subfolder of the Ch09 folder.

Once you've created some packages for your application, NetBeans can automatically add the necessary package statements when you create a new class or interface. For example, if you right-click on the murach.business package and select the New➔Java Class command, the murach.business package will automatically be added to the New Java Class dialog box. Then, when you complete this dialog box, NetBeans will automatically add the necessary package statement at the beginning of the class.

If you need to delete a package, you can right-click on the package and select the Delete command from the resulting menu. This will delete the folder for the package and all subfolders and classes within that folder.

The NetBeans window for the Line Item application in chapter 9



```
package murach.business;

import java.text.NumberFormat;

/*************************************************************
 * The <code>Product</code> class represents a product and is used
 * by the <code>LineItem</code> class.
 *************************************************************/
public class Product
{
    private String code;
    private String description;
    private double price;

    /*************************************************************
     * Initializes a <code>Product</code> and sets all instance
     * variables to empty strings or zero values.
     *************************************************************/
```

```
Description:  Murach's Beginning Java 2
Price:        $49.50
Quantity:     2
Total:        $99.00

Continue? (y/n):
n

BUILD SUCCESSFUL (total time: 17 seconds)
```

Description

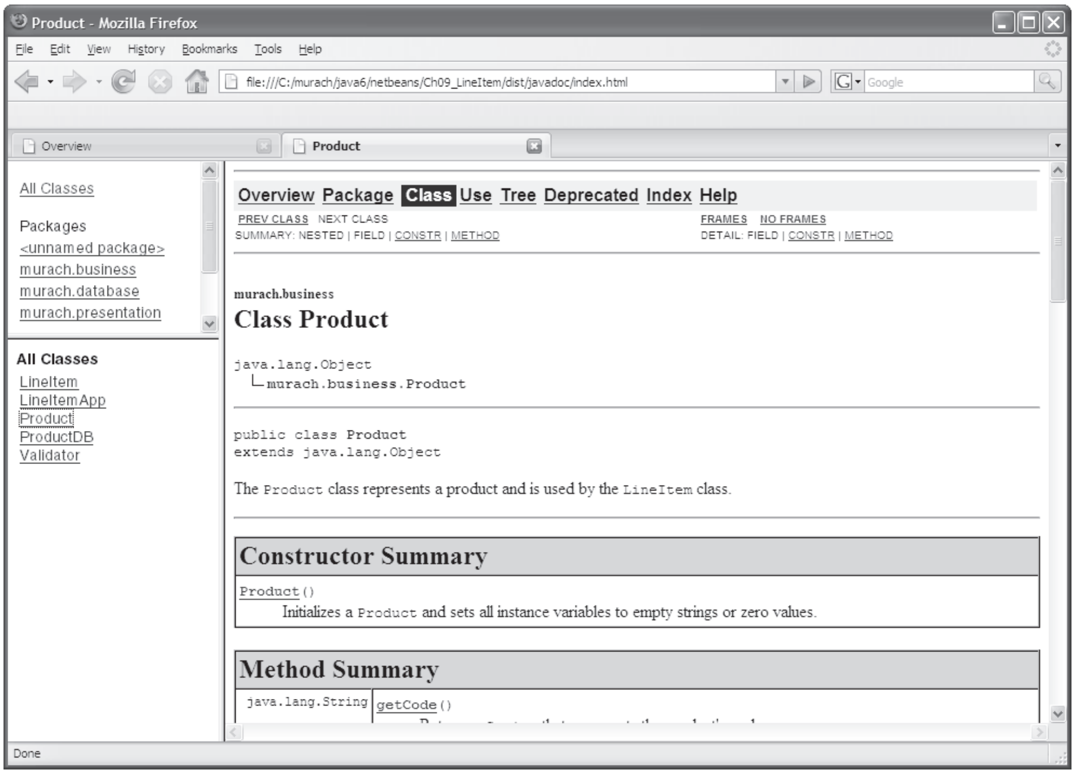
- To navigate through existing packages, use the Projects window to expand or collapse the packages within a project.
- To add a new *package* to a project, right-click on the Source Packages folder in the Projects window, select the New➔Java Package command, and respond to the resulting dialog box. This creates a subfolder within the current folder.
- If you add a new class or interface to a package, NetBeans automatically adds the necessary package statement to the class or interface.
- To remove a package from a project, right-click on the package and select the Delete command from the resulting menu. This will delete the folder for the package and all subfolders and files within that folder.

How to generate documentation

NetBeans also makes it easy to generate the documentation for your classes. First, you make sure that your document has javadoc comments that describe its constructors and methods, and you make sure that your classes are stored in the appropriate packages. (Both of these skills are described in chapter 9 of our book.) Then, you can generate the documentation by right-clicking on the project in the Projects window and selecting the Generate Javadoc for Project command. When you do, NetBeans will generate the Java documentation for the project and display it in the default web browser.

By default, NetBeans stores the documentation for a project in a subfolder named dist\javadoc that’s subordinate to the project’s root folder. If this folder already contains documentation when you generate the documentation, NetBeans will overwrite these files with the new ones, which is usually what you want.

The documentation for the application in chapter 9


Product - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/murach/java6/netbeans/Ch09_LineItem/dist/javadoc/index.html
Google
Overview
Product
Done

All Classes

Packages
<unnamed package>
murach.business
murach.database
murach.presentation

All Classes
LineItem
LineItemApp
Product
ProductDB
Validator

Overview Package **Class** Use Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

murach.business
Class Product

```
java.lang.Object
  └murach.business.Product
```

```
public class Product
extends java.lang.Object
```

The `Product` class represents a product and is used by the `LineItem` class.

Constructor Summary

`Product()`
Initializes a `Product` and sets all instance variables to empty strings or zero values.

Method Summary

`java.lang.String`	`getCode()`

Description

- To generate or view the documentation for a project, you can right-click on the project folder in the Projects window and select the Generate Javadoc for Project command. When you do, NetBeans will generate the Java documentation for the project and display it in the default web browser.
- By default, NetBeans stores the documentation for a project in a subfolder named dist\javadoc that's subordinate to the project's root folder.
- If the project already contains documentation, NetBeans will overwrite existing files without any warning prompts.

Debugging with NetBeans

As you test applications, you will encounter errors that are commonly referred to as *bugs*. When that happens, you must find and fix those errors. This process is commonly known as *debugging*. Fortunately, NetBeans includes a powerful tool known as a *debugger* that can help you identify and fix these errors.

How to set and remove breakpoints

The first step in debugging an application is to figure out what is causing the bug. To do that, it's often helpful to view the values of the variables at different points in the application as it is executing. This will help you determine the cause of the bug, which is critical to debugging the application.

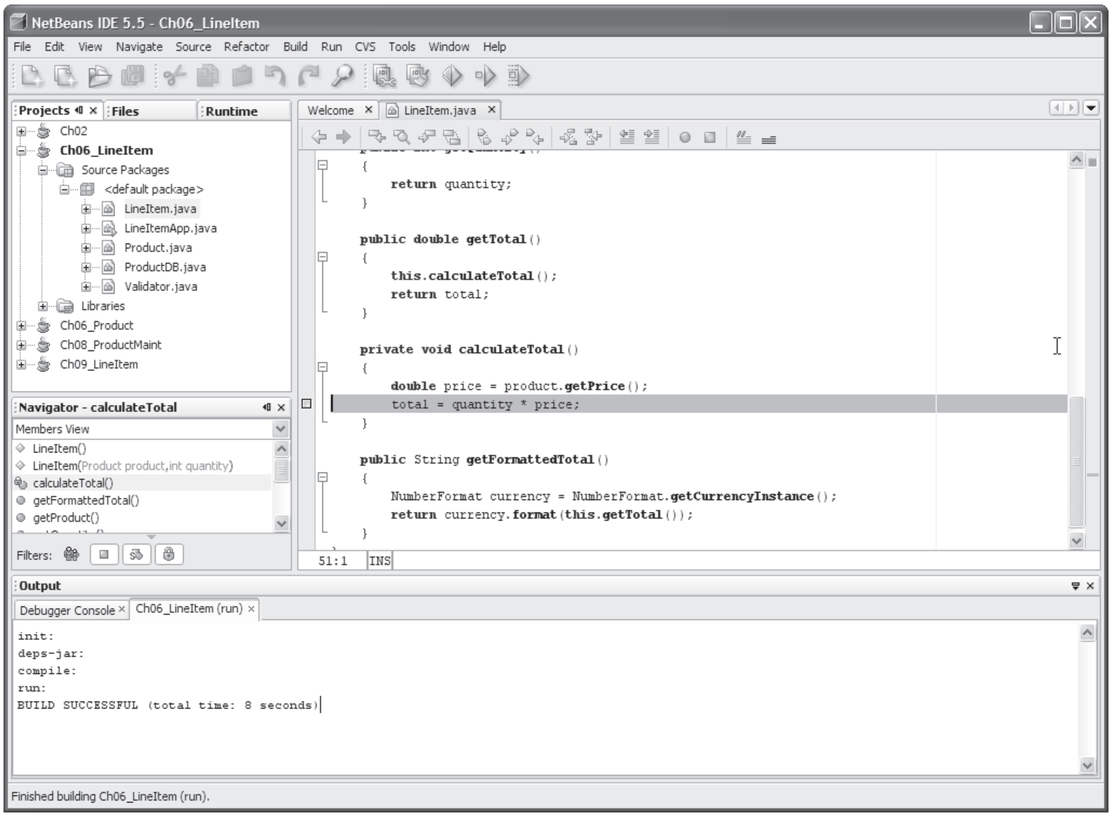
The easiest way to view the variables at a particular point in an application is to set a *breakpoint* as shown in figure 16. To do that, you click to the left of the line of code on the vertical bar on the left side of the code editor window. Then, the breakpoint is marked by a red square to the left of the line of code. Later, when you run the application with the debugger, execution will stop just prior to the statement at the breakpoint, and you will be able to view the variables that are in scope at that point in the application.

When debugging, it's important to set the breakpoint before the line in the application that's causing the bug. Often, you can figure out where to set a breakpoint by reading the runtime exception that's printed to the Console window when your application crashes. However, there are times when you will have to experiment a little before finding a good location to set a breakpoint.

After you set the breakpoint, you need to run the application with the debugger attached. To do this, you can use the Debug Main Project button that's available from the toolbar (just to the right of the Run Main Project button). If you encounter any problems, try right-clicking on the .java file that contains the main method and selecting the Debug File command to run the application with the debugger.

When you run the application with the debugger, NetBeans will display the debugging windows and buttons that are shown in figure 17. After you run the application with the debugger, the breakpoints will remain where you set them. If you want to remove a breakpoint, you can do that by clicking on the red square for the breakpoint.

A code editor window with a breakpoint



```
    {
        return quantity;
    }

    public double getTotal()
    {
        this.calculateTotal();
        return total;
    }

    private void calculateTotal()
    {
        double price = product.getPrice();
        total = quantity * price;
    }

    public String getFormattedTotal()
    {
        NumberFormat currency = NumberFormat.getCurrencyInstance();
        return currency.format(this.getTotal());
    }
```

```
init:
deps-jar:
compile:
run:
BUILD SUCCESSFUL (total time: 8 seconds)
```

Finished building Ch06_LineItem (run).

Description

- A *breakpoint* is indicated by a small red square icon that's placed to the left of the line of code.
- To set a breakpoint for a line, open the code editor for the class and click on the vertical bar to the left of the line number.
- To remove a breakpoint, click on it.
- Once you set a breakpoint, you can use the Debug Main Project button on the toolbar to begin debugging. This works much like the Run Main Project button described in figure 6, except that it allows you to debug the application.
- Alternatively, you can right-click on the file that contains the main method you want to run and select the Debug File command.

How to step through code

When you run application with the debugger and it encounters a breakpoint, execution will stop just prior to the statement at the breakpoint. Once execution is stopped, a green arrow marks the next statement to be executed. In addition, NetBeans opens several new windows, including the Local Variables, Watches, and Call Stack windows shown in figure 17. Of these windows, the Local Variables window shows the values of the variables that are in scope at the current point of execution.

NetBeans also displays some extra toolbar buttons while you're debugging. For instance, you can use the Step Over and Step Into buttons to step through the statements in an application, one statement at a time. This lets you observe exactly how and when the variable values change as the application executes, and that can help you determine the cause of a bug. Once you have stepped through the code that you're interested in, you can use the Continue button to continue execution until the next breakpoint is encountered. Or, you can use the Finish Debugger Session button to end the application's execution.

How to inspect variables

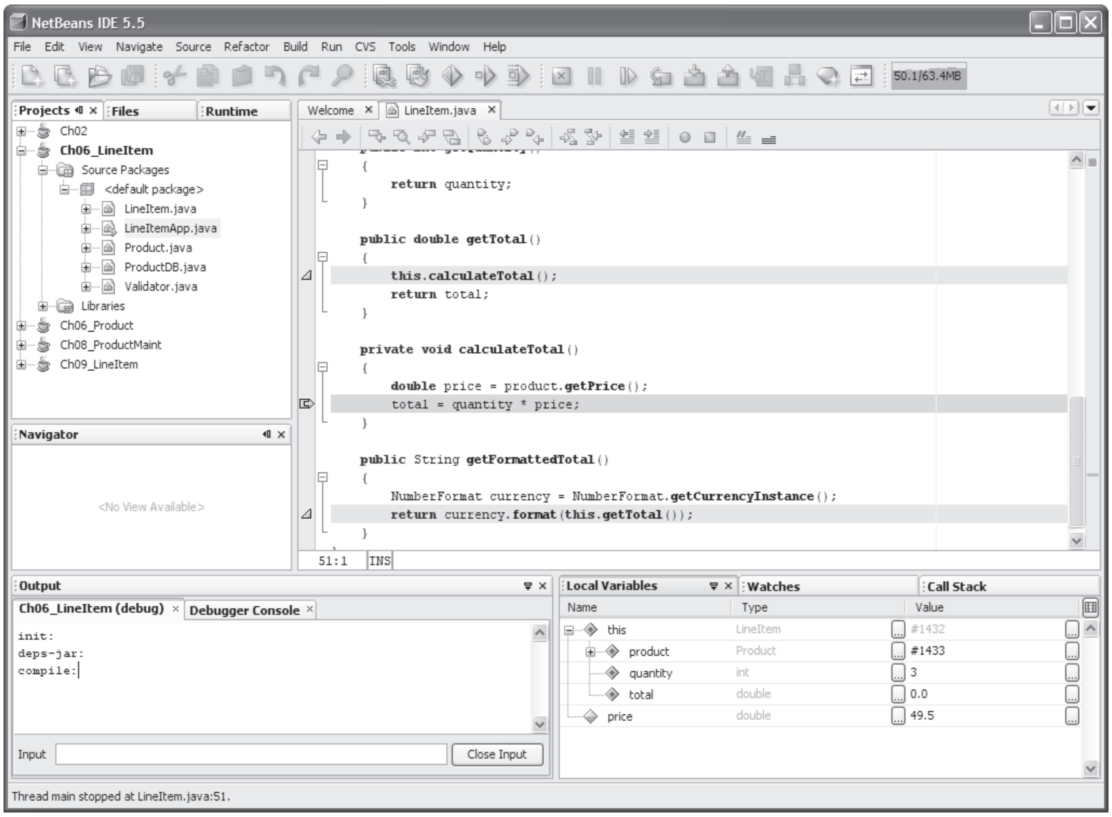
When you set breakpoints and step through code, the Local Variables window will automatically display the variables that are in scope. In figure 17, the execution point is in the calculateTotal method of the LineItem class. Here, the price variable is a local variable that's declared to store the price for the product. In addition, the quantity, total, and product instance variables of the LineItem object are also in scope. To view these variables, expand the variable named *this*, which is a standard variable name that's used to refer to the current object (in this case, the LineItem object).

For numeric variables and strings, the value of the variable is shown in the Local Variables window. However, you can also view the values for an object by expanding the variable that refers to the object. In this figure, for example, you could expand the product variable by clicking on the plus sign to its left to view the values of its code, description, and price variables.

How to inspect the stack trace

When you're debugging, the Call Stack window shows the *stack trace*, which is a list of methods in the reverse order in which they were called. You can click on any of these methods to display the method and highlight the line of code that called the next method. This opens a new code editor window if necessary. Although the Call Stack window isn't shown in this figure, you could click on the Call Stack tab to display this window. If you experiment with is, you'll find that it can help you locate the origin of a bug.

A debugging session



```
    {
        return quantity;
    }

    public double getTotal()
    {
        this.calculateTotal();
        return total;
    }

    private void calculateTotal()
    {
        double price = product.getPrice();
        total = quantity * price;
    }

    public String getFormattedTotal()
    {
        NumberFormat currency = NumberFormat.getCurrencyInstance();
        return currency.format(this.getTotal());
    }
```

```
init:
deps-jar:
compile:
```

Name	Type	Value
this	LineItem	#1432
product	Product	#1433
quantity	int	3
total	double	0.0
price	double	49.5

Thread main stopped at LineItem.java:51.

Description

- When a breakpoint is reached, program execution is stopped before the line is executed.
- In the code editor window, the arrow shows the line that will be executed next.
- The Local Variables window shows the values of the variables that are in scope for the current method. This includes static variables, instance variables, and local variables. If a variable refers to an object, you can view the values for that object by expanding the object to drill down through its variables.
- The Call Stack window shows the *stack trace*, which is a list of methods in the reverse order in which they were called. You can click on any of these methods to display the code for the method in a code editor window and to display the variables for that method in the Variables window.
- To step through the code one statement at a time, you can click on the Step Over or Step Into buttons that are available from the debugging toolbar. These buttons work the same, except that the Step Over button skips over called methods while the Step Into button steps through all of the statements in a called method.
- To continue executing code until the next breakpoint, select the Continue button.
- To end the application's execution, select the Finish Debugger Session button.

More NetBeans skills

So far, this tutorial has presented all of the skills that you need for using NetBeans to develop the object-oriented Java applications described in *Murach's Java SE 6*. However, if you want to develop applets as described in chapter 18, you can use NetBeans to test an applet. Or, if you want to access classes that are stored in a JAR file, such as the eric.jar file described in figure 7 of this tutorial or the derby.jar file described in chapter 22 of the book, you need to learn how to add a JAR file to the build path.

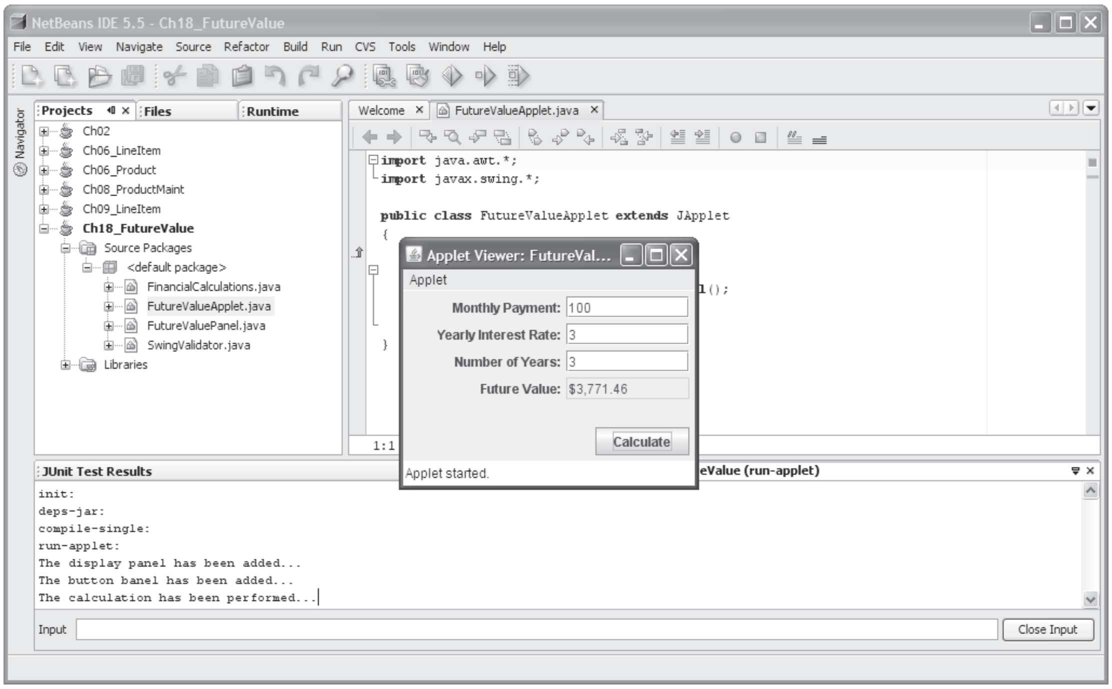
How to run an applet

An *applet* is a special type of class that can be downloaded from an Internet or intranet server and run on a client computer within a web browser. To add an applet to a project, you can add a new class to the project as described in figure 10. Then, you can enter the code for the applet. For more information about coding applets, see chapter 18 of our book.

To run an applet in the Applet Viewer, you can find the .java file for the applet in the Projects window, right-click on it, and select the Run File command. When you do, NetBeans will generate a temporary HTML page for the applet and display the applet in an Applet Viewer dialog box like the one in this figure. However, you may need to resize this dialog box to get the applet to display correctly.

If you don't want to manually resize this dialog box, you can code an HTML page for the applet as described in chapter 18 of *Murach's Java SE 6*. Within this HTML page, you can specify the height and width for the applet. Then, you can run the applet by viewing this HTML page in a web browser.

The dialog box for running the applet in chapter 18



```
import java.awt.*;
import javax.swing.*;

public class FutureValueApplet extends JApplet
{
```

```
Monthly Payment: 100
Yearly Interest Rate: 3
Number of Years: 3
Future Value: $3,771.46
```

```
init:
deps-jar:
compile-single:
run-applet:
The display panel has been added...
The button banel has been added...
The calculation has been performed...
```

Description

- To run an applet in the Applet Viewer, right-click on the source code file for the applet and select the Run File command. If necessary, resize the Applet Viewer window so it is the correct size for the applet.
- If you want to run an applet in an HTML page instead of running it in the Applet Viewer, see chapter 18 of *Murach's Java SE 6*.

How to add a JAR file to the libraries for a project

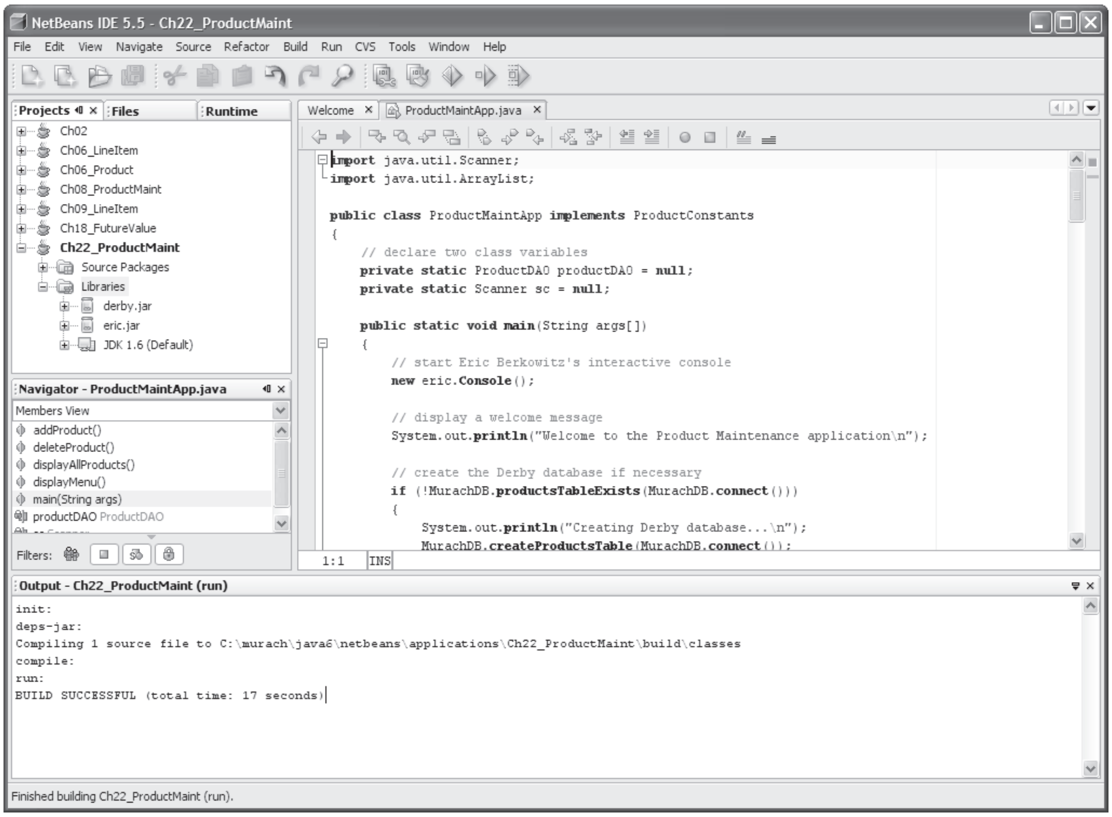
Most of the projects in *Murach's Java SE 6* only use classes that are available from the standard JDK 1.6 libraries, which are available to all projects by default. However, to use classes that are stored in other libraries, you can add the JAR file (Java Archive file) for that library to the build path. In figure 7, for example, you learned how to add the library for Eric G. Berkowitz's interactive console to a project.

Once you add a JAR file to a project, your project can use any classes within the JAR file when it compiles and runs. If, for example, you want to use a database driver other than the standard JDBC-ODBC bridge driver described in chapter 21, you need to add the JAR file for the database driver to the libraries of your project. Similarly, if you want to work with the Derby database described in chapter 22, you need to add the appropriate JAR file or files to the libraries for your project.

To add a JAR file to the libraries for a project, you can begin by right-clicking on the Libraries folder for the project and selecting the Add JAR/Folder command. Then, you can use the resulting dialog box to select the JAR file. In figure 19, for example, I used this dialog box to select the derby.jar file that's needed to run the Product Maintenance application described in chapter 22.

In this figure, you can see that the derby.jar file has been added to the Libraries folder for the project, above the eric.jar file and the JDK 1.6 libraries. As a result, any classes in this project will be able to use any of the classes that are stored within any of these libraries.

A JAR file that has been added to the application in chapter 22



```
import java.util.Scanner;
import java.util.ArrayList;

public class ProductMaintApp implements ProductConstants
{
    // declare two class variables
    private static ProductDAO productDAO = null;
    private static Scanner sc = null;

    public static void main(String args[])
    {
        // start Eric Berkowitz's interactive console
        new eric.Console();

        // display a welcome message
        System.out.println("Welcome to the Product Maintenance application\n");

        // create the Derby database if necessary
        if (!MurachDB.productsTableExists(MurachDB.connect()))
        {
            System.out.println("Creating Derby database...\n");
            MurachDB.createProductsTable(MurachDB.connect());
```

```
init:
deps-jar:
Compiling 1 source file to C:\murach\java6\netbeans\applications\Ch22_ProductMaint\build\classes
compile:
run:
BUILD SUCCESSFUL (total time: 17 seconds)
```

Finished building Ch22_ProductMaint (run).

Description

- If you add a JAR file to the libraries for a project, the JRE will be able to find and run any of the classes within the JAR file that are needed by the project.
- To add a JAR file to the libraries for a project, right-click on the Libraries folder for the project, select the Add JAR/Folder command, and use the resulting dialog box to select the JAR file.
- To remove a JAR file from the libraries for a project, right-click on the JAR file and select the Remove command.

Perspective

In this tutorial, you learned all of the NetBeans skills that you need for developing the applications that are described in *Murach's Java SE 6*. Keep in mind, though, there's much more to learn about NetBeans. To start, you may want to review the documentation that's available from the NetBeans Help menu and from the NetBeans web site. Or, you may want to experiment with the commands in the Source and Refactor menus to see how they can help you enter and modify code.

Summary

- NetBeans is a software framework for developing *Integrated Development Environments* (*IDEs*). The NetBeans IDE for Java is built on this framework. NetBeans is open-source, available for free from the NetBeans web site (www.netbeans.org), and runs on all modern operating systems.
- A *project* is a folder that contains all of the files that make up an application.
- The *code completion* feature can help you enter code.
- In NetBeans, an *error* is a line of code that won't compile.
- If you have multiple projects open, you can specify one project as the *main project*. If this project contains multiple classes that have a main method, you can specify one class as the *main class*. Then, you can use the Run Main Project command to run the main class in the main project.
- When you run an application from NetBeans, the default *console* is the Output window. However, this console has a bug that prevents it from working with the print method of the System.out object. As a result, to use NetBeans to test console applications, you may want to use a third-party console such as Eric G. Berkowitz's interactive console.
- NetBeans includes a *debugger* that can help you find and fix any *bugs* in your applications. To use the debugger, you can set a *breakpoint* to stop program execution. Then, you can *step through* the statements in your applications and view the values of variables and the *stack trace*.

Before you do any of the exercises that follow…

Before you do the exercises that follow, you should use the procedure shown in chapter 1 (figures 1-4) of *Murach's Java SE 6* to install the JDK. Then, you should do the procedure in figure 1 of this tutorial to install NetBeans. In addition, you should download the NetBeans versions of the folders and files for our book from our web site (www.murach.com) and unzip them into the C:\murach\java6 folder. If this folder doesn't already exist, you should create it.

Exercise 1 Use NetBeans to develop an application

Please read chapter 1 in *Murach's Java SE 6* before you do this exercise. This exercise will guide you through the process of using NetBeans to enter, save, compile, and run a simple application.

Enter and save the source code

1. Start NetBeans.
2. Select the File➔New Project command from the NetBeans menu system. Then, use the resulting dialog box to create a project named TestApp that contains a class named TestApp that has a main method and store this project in this folder:

   ```
   C:\murach\java6\netbeans\exercises
   ```
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. Enter the statement that starts with System.out again, right after the first statement. This time, type sys and press Ctrl+Spacebar. Then, select system.out.println by double-clicking on it, and complete the statement.
5. Enter that 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.
6. Use the Save command (Ctrl+S) in the File menu to save your changes.

Run the application

7. Click on the Run Main Project button in the toolbar to compile and run the class. This should display "This Java application has run successfully" three times in a row in the Output tab.
8. Press F6 to run the application a second time.

Introduce and correct a compile-time error

9. 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.
10. Correct the error, and NetBeans will remove the error icon.
11. Save the files and press F6 to run the application again.
12. Exit NetBeans by selecting the Exit command from the File menu.

Exercise 2 Use NetBeans to open and run an existing application

Please read chapter 2 in *Murach's Java SE 6* before you do this exercise. This exercise will guide you through the process of using NetBeans to open and run a simple console application that gets input from a user.

Open and run the Invoice application

1. Start NetBeans, and open the project named Ch02 as described in figure 8. This project folder should be stored in this directory:
   ```
   C:\murach\java6\netbeans\exercises
   ```
2. Check to make sure that the eric.jar file has been added to the Libraries folder. If it hasn't, add this JAR file to the Libraries folder as described in figure 7.
3. Open the InvoiceApp.java file in the text editor. Note that the main method for this class begins with the line of code that starts the interactive console.
4. Run the Invoice application. To do that, right-click on the InvoiceApp.java file and select the Run File command. Respond to the prompts on the interactive console until you finish running the application.
5. Close the interactive console. Note that this causes the Output window to display a message that indicates that the application has finished running.

Open and run the Test Scores application

6. Follow steps 3 through 5 for the TestScoreApp.java file.
7. Close the project, and exit from NetBeans.

Exercise 3 Test and debug an application

Please read chapter 2 in *Murach's Java SE 6* before you do this exercise. This exercise will guide you through the process of using NetBeans to test and debug an application.

Test the Invoice application with invalid data

1. Open the ch02 project, and test the Invoice application with an invalid subtotal like $1000 (enter the dollar sign too). This time, the application will crash with a run-time error, and an error message will be displayed in the Console window.
2. Study the error message, and note the line number of the statement that caused the crash. Then, open the InvoiceApp.java file in the text editor and select the View➔Show Line Numbers command to display the line numbers to the left of each line of code. Based on this information, you should be able to figure out that the application crashed because $1000 isn't a valid double value.

Set a breakpoint and step through the application

3. Set a breakpoint on this line of code:

   ```
   double discountPercent = 0.0;
   ```
4. Right-click on the InvoiceApp.java file in the Projects tab and select Debug File. This will run the application with the debugger on.
5. When the application prompts you for a subtotal entry, enter 100. Then, when the application reaches the breakpoint and stops, click on the Local Variables tab and note that the choice and subtotal variables have been assigned values.
6. Click on the Step Into button in the toolbar to step through the application one statement at a time. After each step, review the variables in the Local Variables tab to see how they have changed. Note too how the application steps through the if/else statement based on the subtotal value.
7. Click on the Continue button in the toolbar to continue the execution of the application at full speed. When prompted, enter the required values until you reach the breakpoint the second time. Then, click the Finish Debugger Session button to end the application. This should give you some idea of how useful the NetBeans debugging tools can be.
8. End the application, and exit from NetBeans.

What else you need to know before you do the exercises for *Murach's Java SE 6…*

At this point, you have all the NetBeans skills you need for doing the exercises in *Murach's Java SE 6*. However, you need to know (1) how the names we used for the NetBeans exercise starts relate to those in the book exercises, and (2) how to copy and rename files.

In addition, you need to use common sense as you follow the directions in the book. For example, when you use NetBeans to do these exercises, you'll find that NetBeans automatically performs some operations that are done manually in the exercises. That's particularly true when you apply the object-oriented skills described in chapters 6 through 9 to the exercises for those chapters.

How the NetBeans project names relate to the book exercises

- By now, you should have unzipped all of the NetBeans exercise starts into:

  ```
  C:\murach\java6\netbeans\exercises\
  ```

- Some of the projects stored in this directory have short names like ch02 and ch03 that map directly to the directory names that are used by the exercises in the book.
- Other projects have longer names that don't map directly to the directory names that are used in the exercises in the book. Here's how project names used by this workspace map to the directory names that are used by the exercises in the book:

Project name	**Directory name**
ch06_LineItem	ch06\LineItem
ch08_DisplayableTest	ch08\DisplayableText

- As you do the exercises in the book, you need to convert the directory names that are used in the book to the project names that are available from the NetBeans directories.

How to use NetBeans to copy and rename files

- As you do the exercises in the book, you are often asked to copy an existing application or to save one with a new name. Most of the time, the easiest way to do these tasks is to use the Projects tab.
- To copy a file, right-click on the .java file in the Projects tab, select the Copy command, right-click on the package in the Projects tab, and select the Paste command. If you're copying a file within the same project, NetBeans will automatically modify the name of the file so it doesn't conflict with the original file.

- To rename a file, right-click on the .java file in the Projects tab, select the Refactor➔Rename command, and respond to the resulting dialog boxes. To do that, click on the Next button in the first dialog box. Then, after previewing the changes in the Refactoring window, click on the Do Refactoring button to complete the refactoring.
- If you change the name of a file, NetBeans automatically changes the name of the class, which is usually what you want.

Examples of converting the book exercises to NetBeans

- Exercise 2-2 in the book asks you to save the TestScoreApp.java file as ModifiedTestScoreApp.java in the same directory. To do that, open the Ch02 project and use the Projects tab to display the TestScoreApp.java file that's stored in the default package. Then, right-click on the file, select the Copy command, right-click on the default package, and select the Paste command. When you do, NetBeans will create a new file named TestScoreApp_1.java. Finally, right-click on the TestScoreApp_1.java file, select the Refactor➔Rename command, click on the Next button in the first dialog box, and click on the Do Refactoring button in the Refactoring window.
- Exercise 3-2 in the book asks you to open a file named ModifiedTestScoreApp.java in the ch02 directory and save it in the ch03 directory as EnhancedTestScoreApp.java. To get the same result, you can use the Projects tab to open the ch02 and ch03 projects. Then, you can copy the file from the default package of the ch02 project to the default package of the ch03 project. Finally, you can rename the file by right-clicking on it, selecting the Refactor➔Rename command, clicking on the Next button in the first dialog box, and clicking on the Do Refactoring button in the Refactoring window.
- Exercise 6-1 in the book asks you to open the classes that are in the ch06\LineItem directory. To do that, open the existing project named ch06_LineItem. Then, use the Projects tab to open the classes in this project.
- Exercise 9-1 in the books asks you to use the Windows Explorer to create and work with the subdirectories that correspond with the packages for the application. With NetBeans, you can use the Projects tab to create packages and work with packages as described in figure 14 of this tutorial. This automates many aspects of working with packages.