

This tutorial is based on Murach's Java SE 6

How to use NetBeans

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.

How to download and install the NetBeans IDE

Once you have installed Java SE, you are ready to install the NetBeans IDE. 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.

If you encounter any problems, you can view the documentation that is 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.
3. Run the install file and respond to the resulting dialog boxes.

How to start NetBeans

Once you have installed NetBeans IDE, you can start it by selecting it from the Start menu. When you start NetBeans for the first time, it should display a Welcome tab. If it does not, you can display this page by selecting the Welcome Screen command from the Help menu. Since you do not need this Welcome page to work with NetBeans, you can close it if you like. To do that, click the X that is displayed to the right of the tab for the Welcome page.

If you are 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 is installed on your system. 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.

How to create a new project with a main class

Figure 1 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.

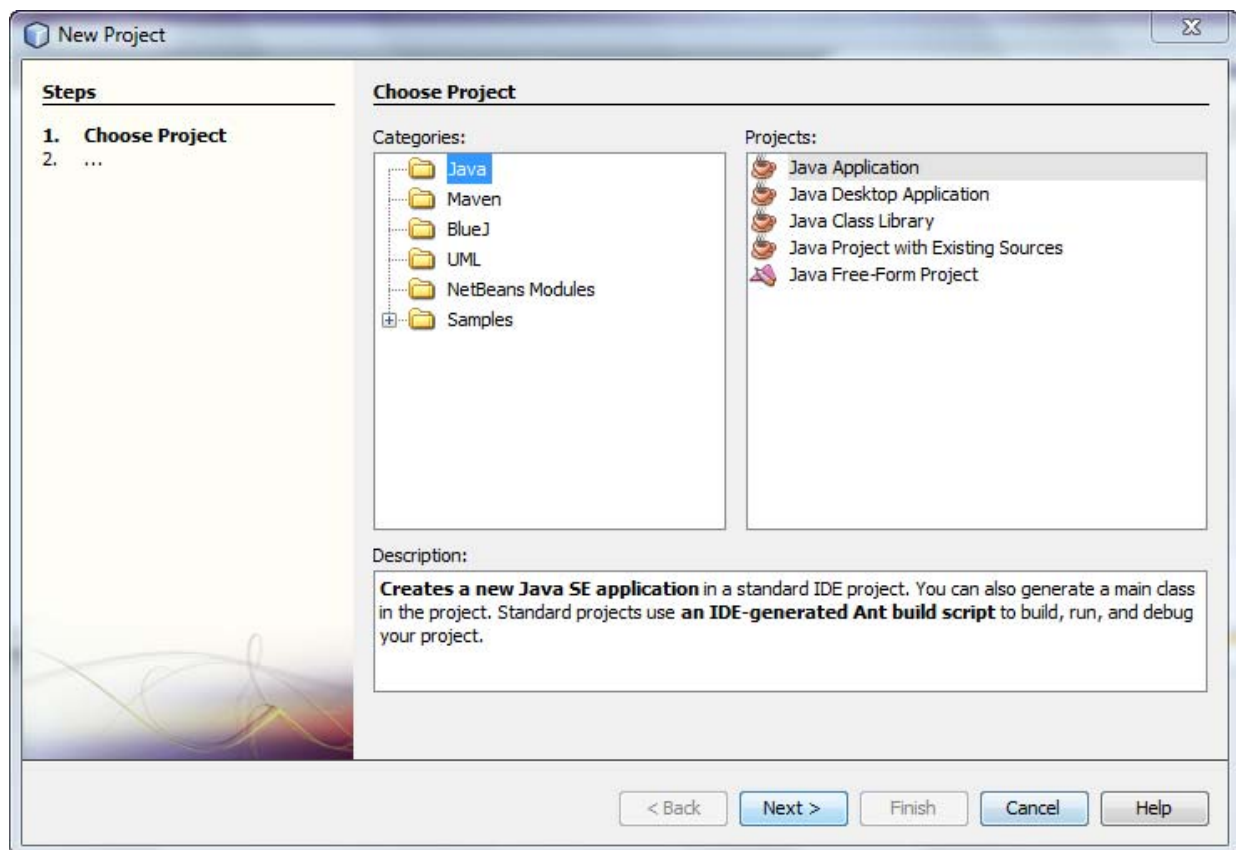


Figure 1. The first dialog box for creating a new project

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.

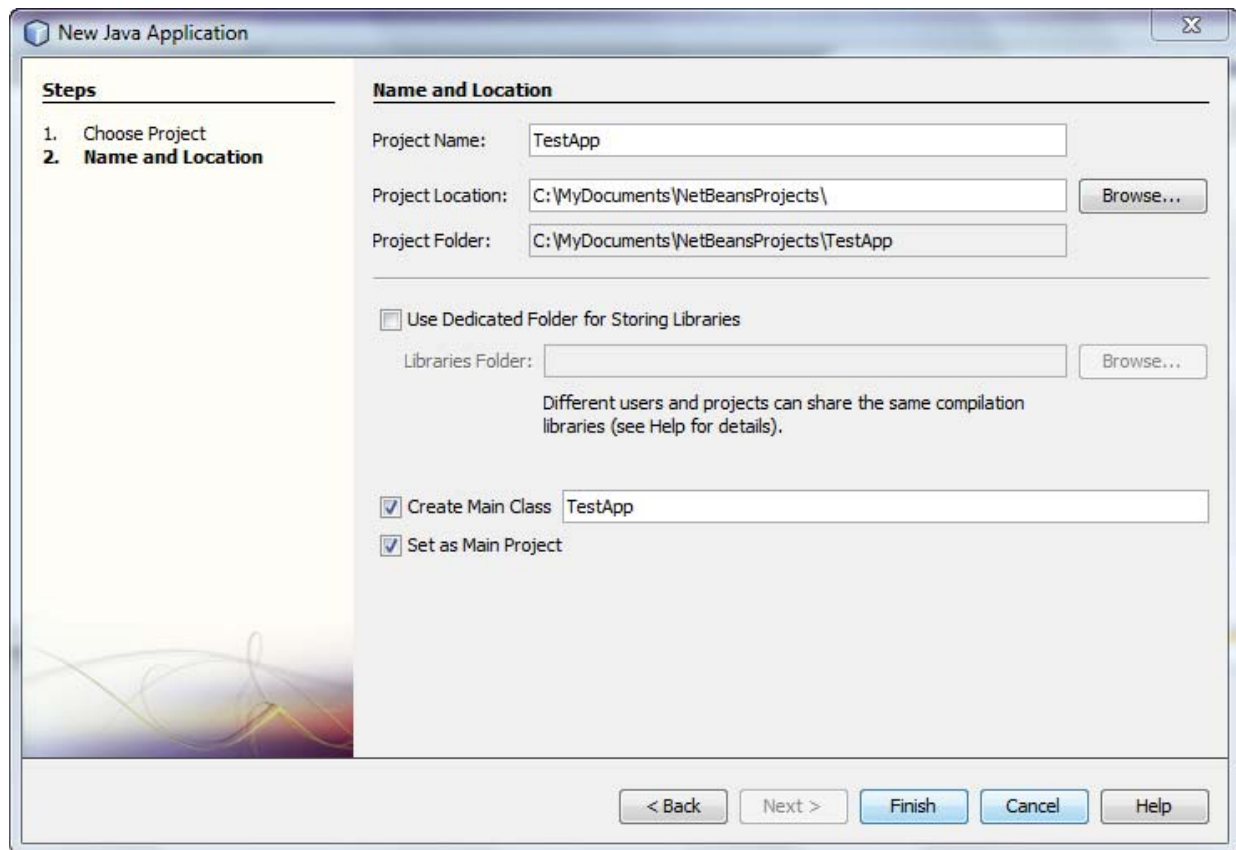


Figure 2. The second dialog box for creating a new project

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:\MyDocuments\NetBeansProjects\

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 is usually what you want. However, you can easily change this later, so this choice is not 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 is used when you do not 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.

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 3.

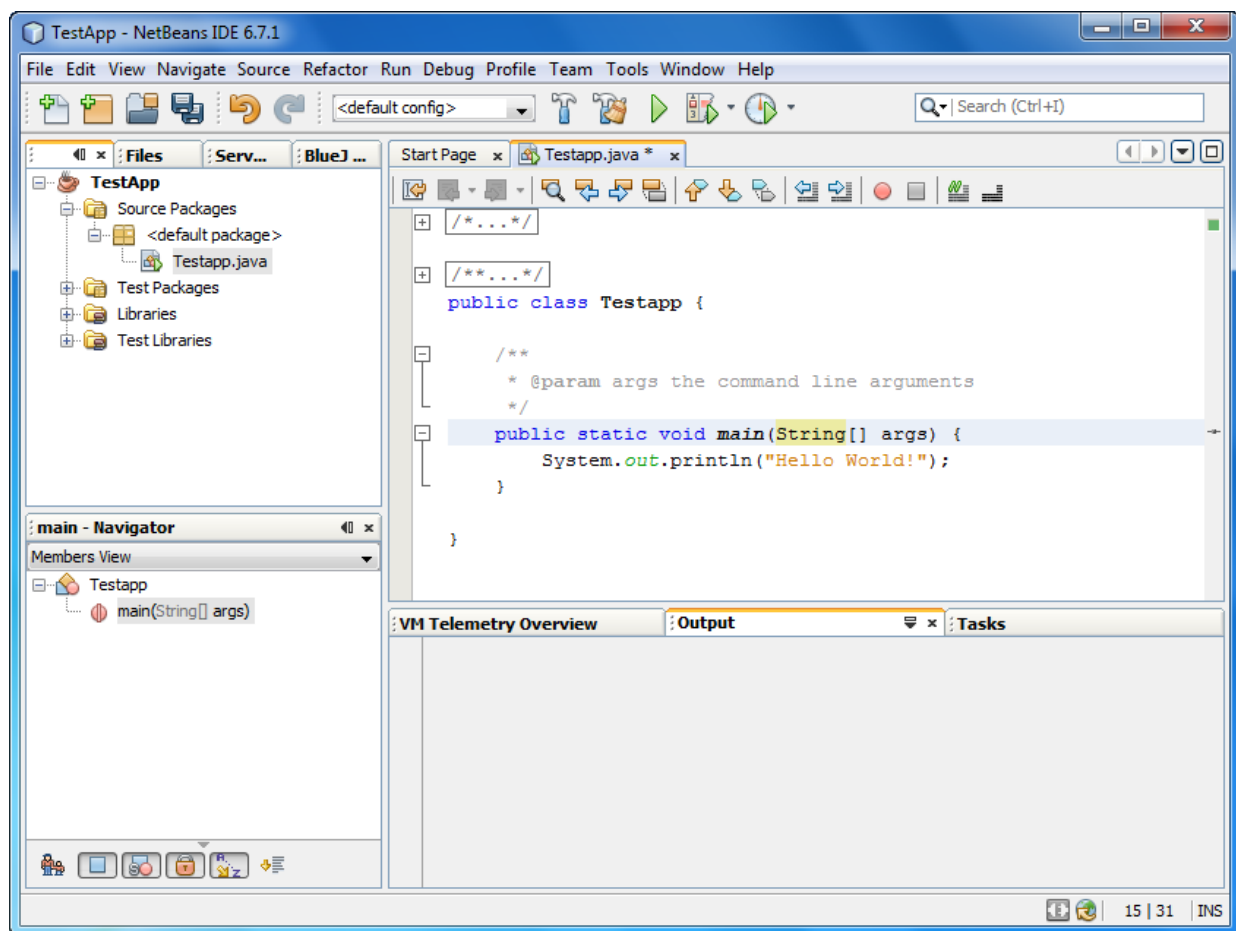


Figure 3. NetBeans's code editor with source code in it

To make it easier to for you recognize the Java syntax, the code editor uses different colours 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 2).

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 does not 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 is 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.

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 will 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 is not 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 figure 3, for example, the `TestApp.java` file that is 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.

How to fix errors

In NetBeans, an error is a line of code that will not compile. As you enter text into the code editor, NetBeans displays errors whenever it detects them. In figure 4, 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 exclamation mark  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.

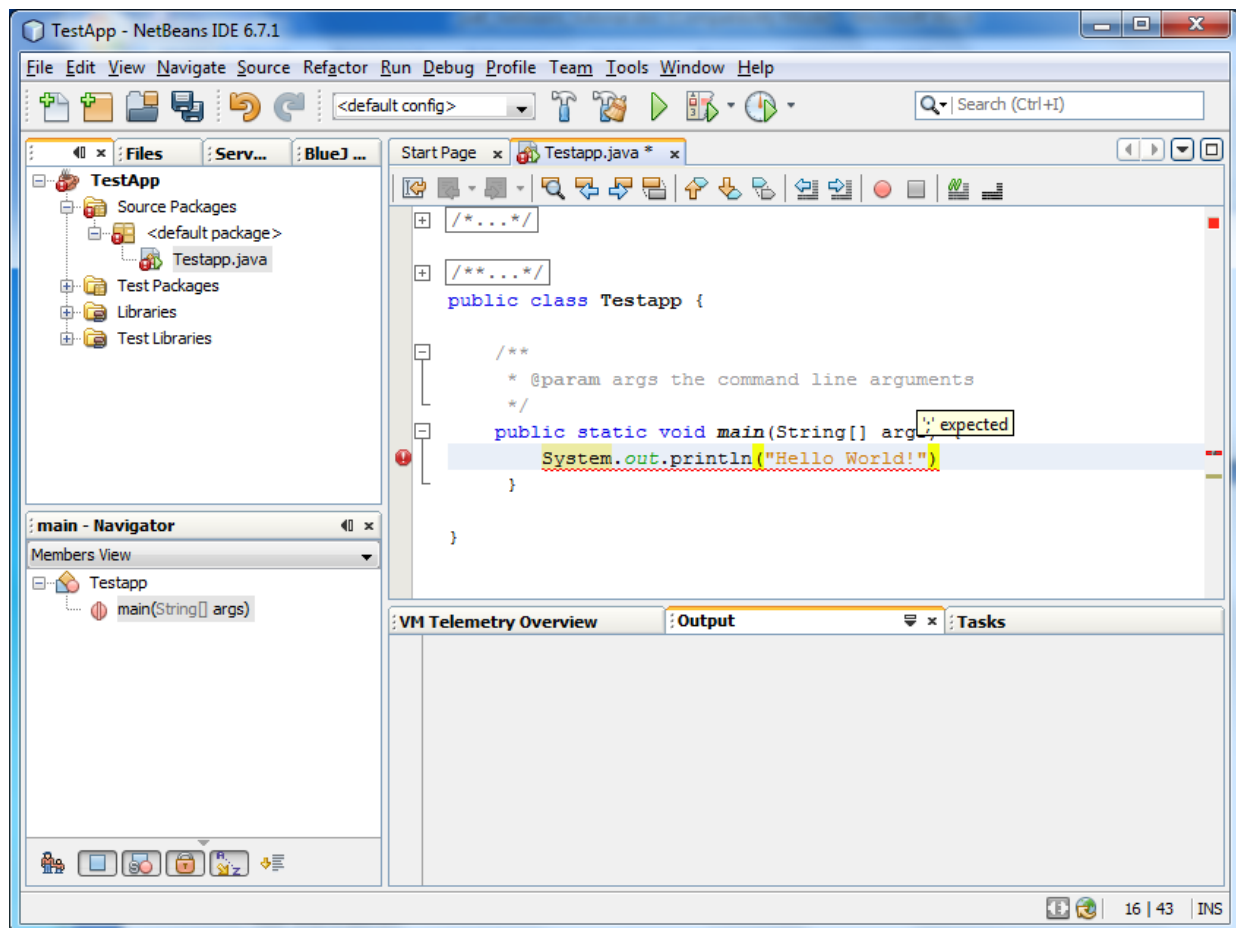


Figure 4. An error that is displayed when you save the source code

If you position the mouse cursor over the red exclamation mark icon or over the statement itself, NetBeans will display a description of the message. In figure 4 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.

How to compile and run an application

By default, NetBeans automatically compiles an application before it runs the application. 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 figure 5, 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 is used by NetBeans. Since this application just prints text to the console, it can be referred to as a console application.

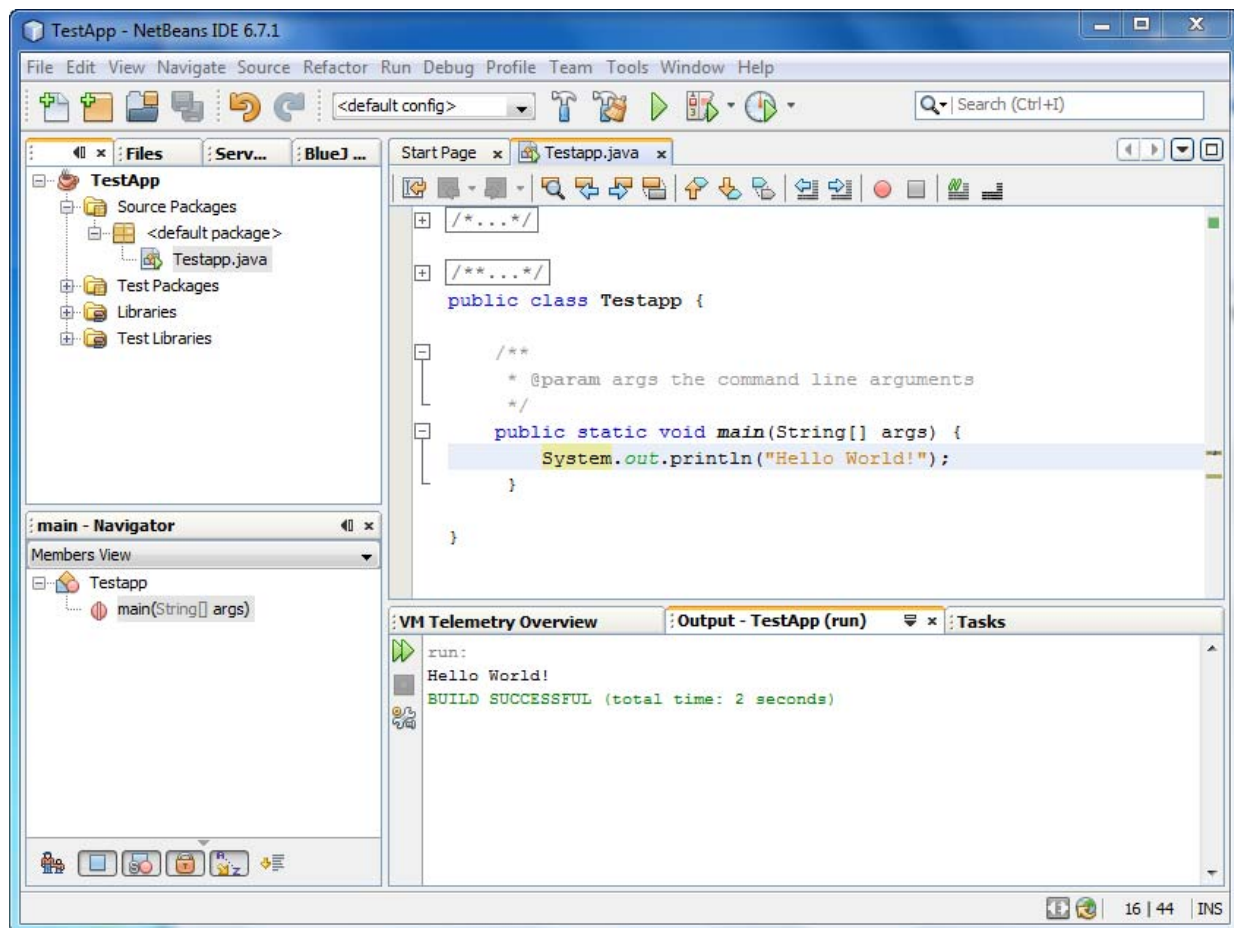


Figure 5. An application that prints text to the console

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 is presented at the end of this tutorial.

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 does not 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 6. To start, you must add the file named `eric.jar` to the Libraries folder for your project.

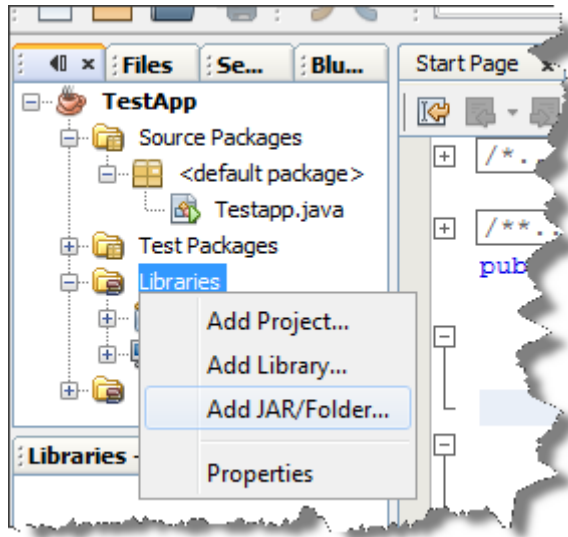


Figure 6. Adding a jar file.

To add the file, right click on Libraries, select Add JAR/Folder..., and follow the prompts. 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-

```
new eric.Console();
```

This line of code creates a new instance of the Console class that is 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 is available from NetBeans.

Once you have added this line of code to the beginning of your console app, you can run the application as you would normally.

For an interactive console where the user can add input, an import statement has to be added before the class declaration –

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

The application should print some text to the console that prompts the user to enter data. The user can type input into the console and press Enter. When the user has done this, the application will continue until it finishes or until it prompts you for more information.

In figure 7, the application prompts the user to type in their name. When you are learning Java, it is common to create applications that use the console to get input from the user and to display output to the user.

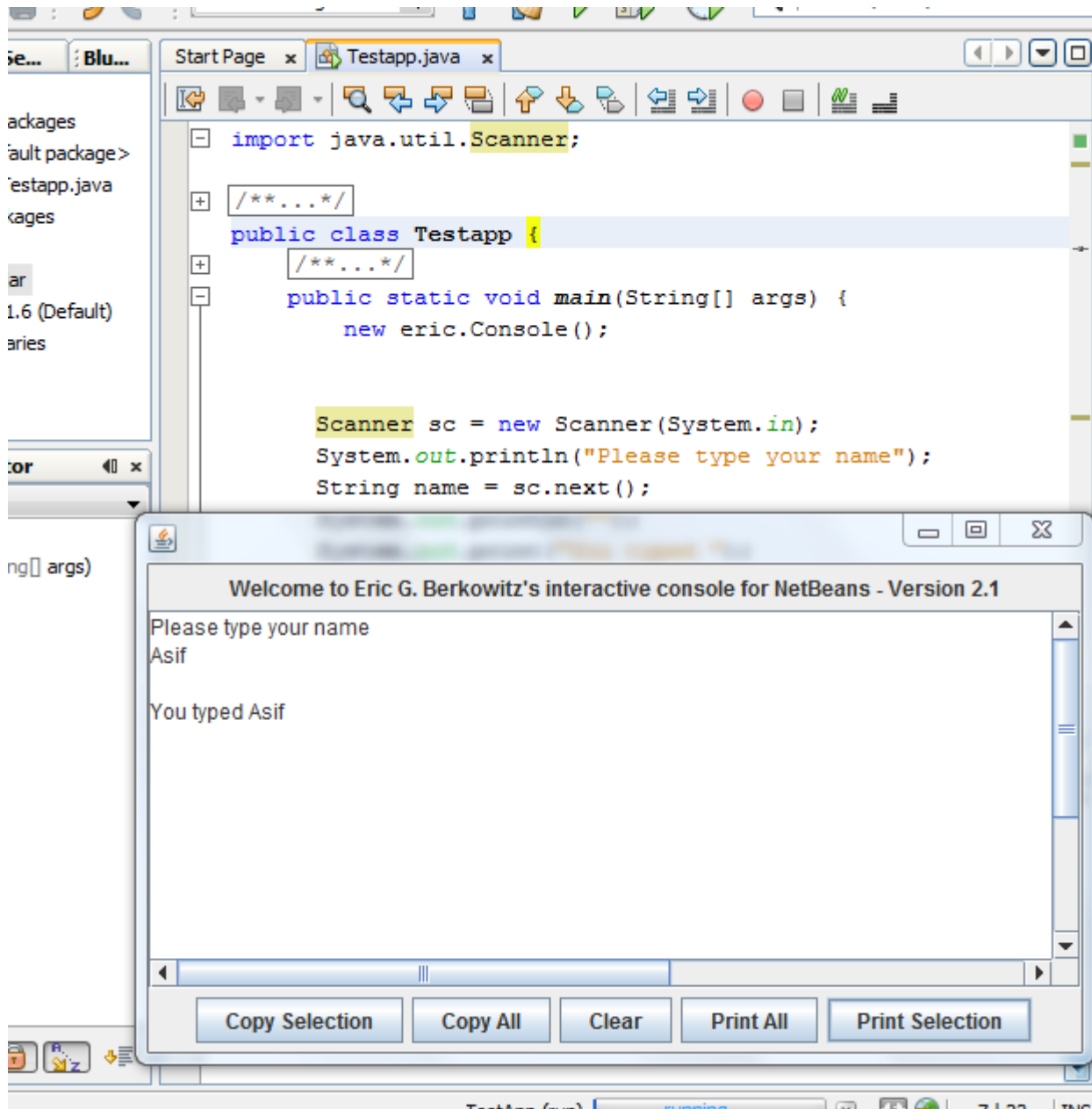


Figure 7. A console app that uses an interactive console to get input from a user

How to open and close projects

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.

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, you can right-click on the project in the Projects window and select the Close Project command. Since this does not 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 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 is no longer easy to open the project with NetBeans. As a result, you will only want to use this option if you do not plan on using NetBeans to work with the project anymore.

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.

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 8. In this dialog box, you can select the “Java Project with Existing Sources” option to create a new project that imports existing .java files.

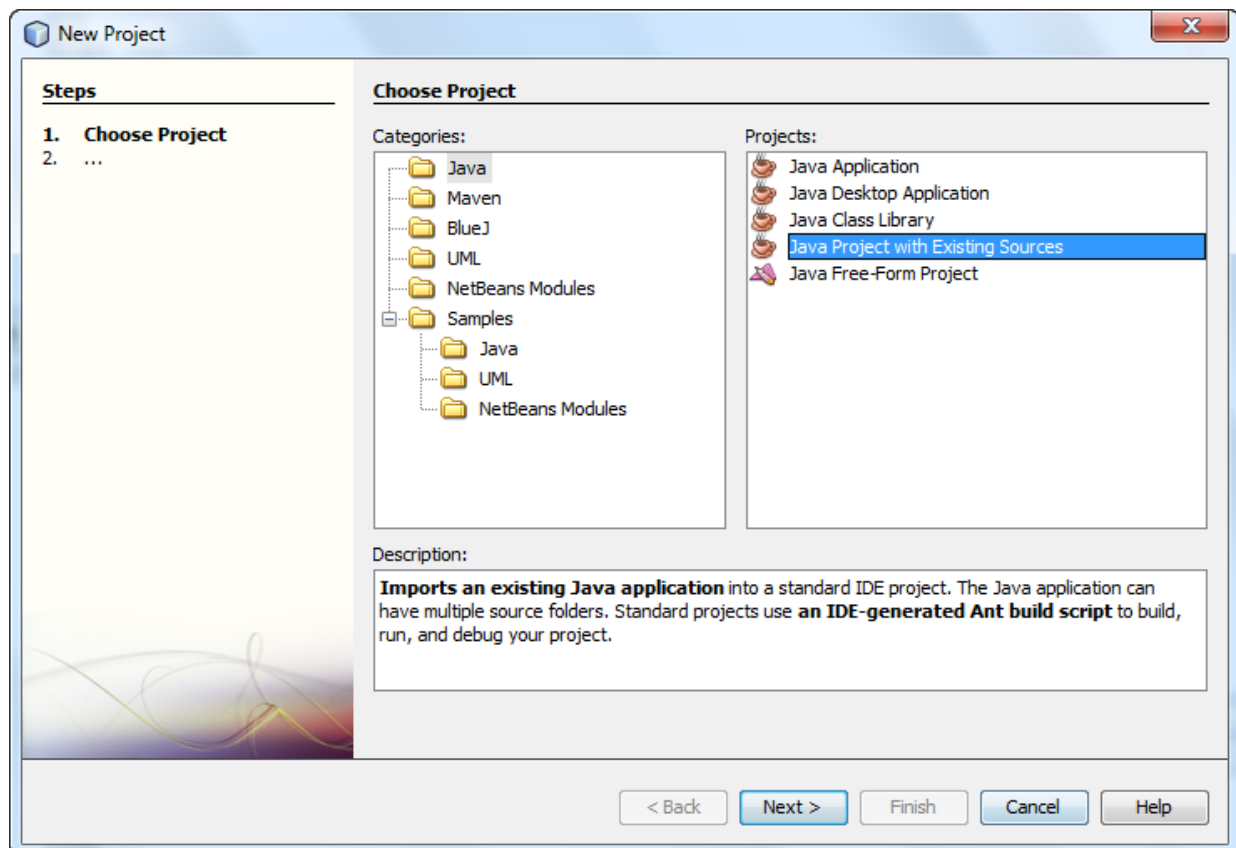


Figure 8. The first dialog box for importing Java files into a NetBeans project

Then, you can use the second dialog box shown to specify the name and location for the project. In figure 9, for example, the second dialog box creates a project named MyCoursework in the C:\MyDocuments\NetBeansProjects\MyCoursework folder.

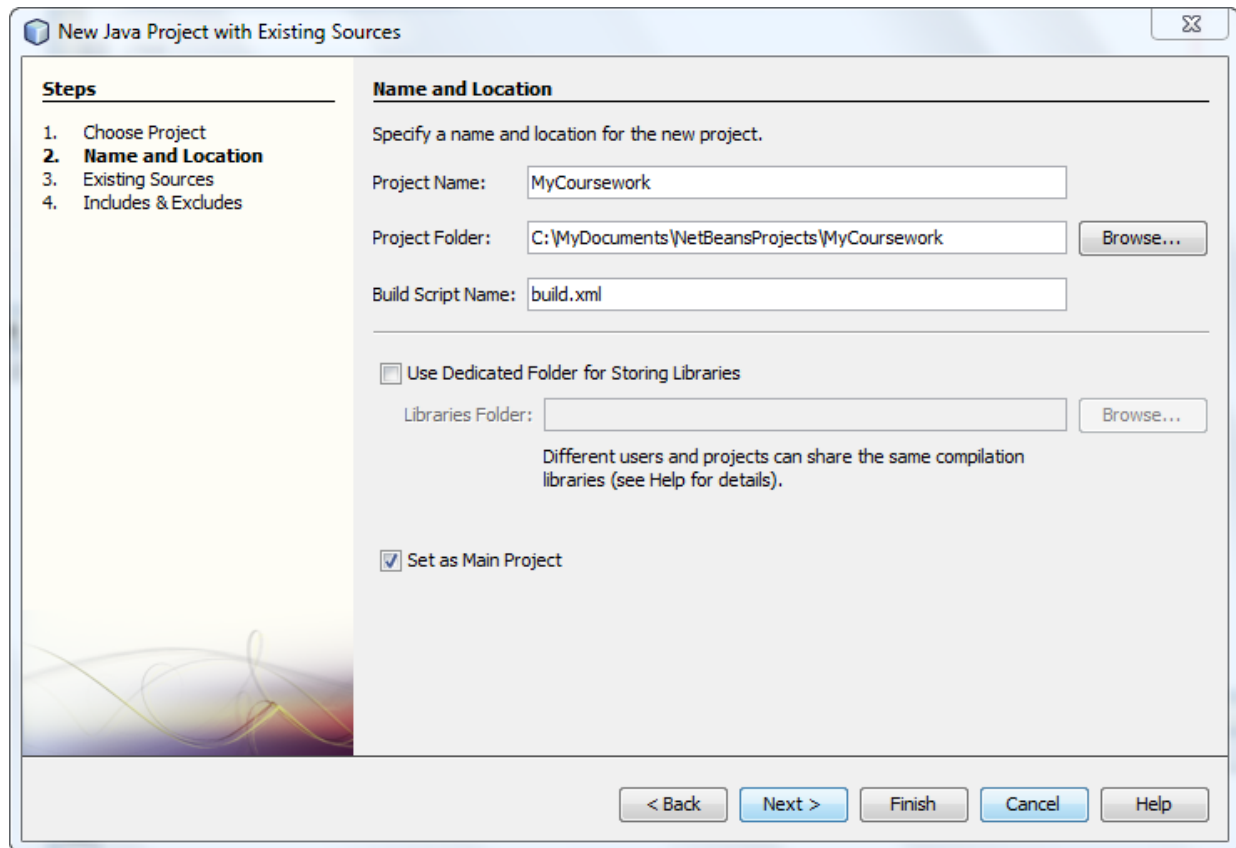


Figure 9. The second dialog box for importing Java files into a NetBeans project

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 MyCoursework 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.

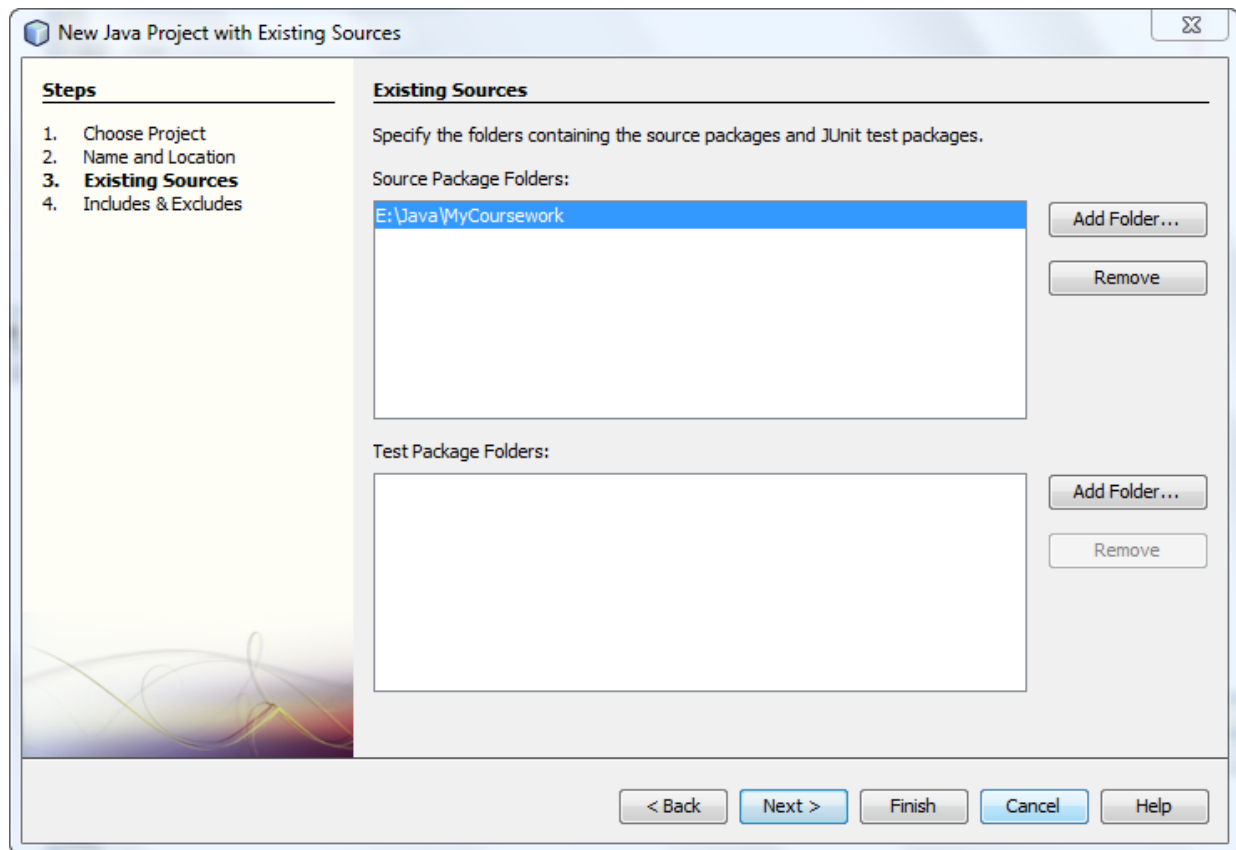


Figure 10. The third dialog box for importing Java files into a NetBeans project

How to create a new class

In figure 1, 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 will need to add other classes to your project. To do that, right click on the project name in the project window, select New, Java Class, figure 11. You can use the New Java Class dialog box to add new classes to your project, figure 12. In this figure, for example, the New Java Class dialog specifies a public class named Detail.

Although the New Java Class dialog box encourages you to enter a package for the class, this is not required. If you do not 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.

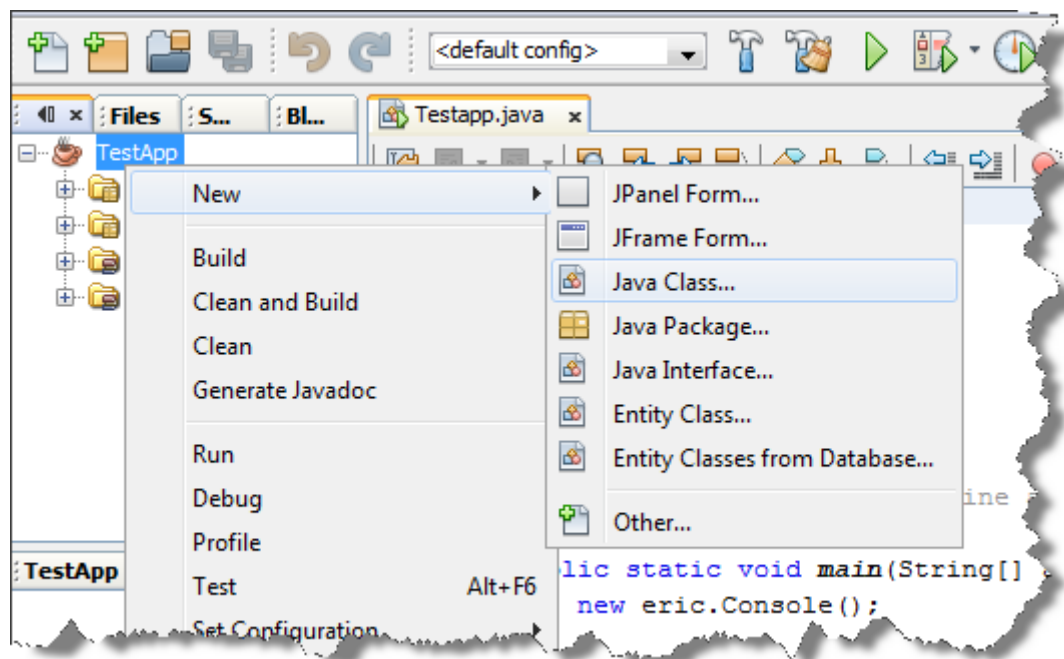


Figure 11. The dialog box for creating a new class

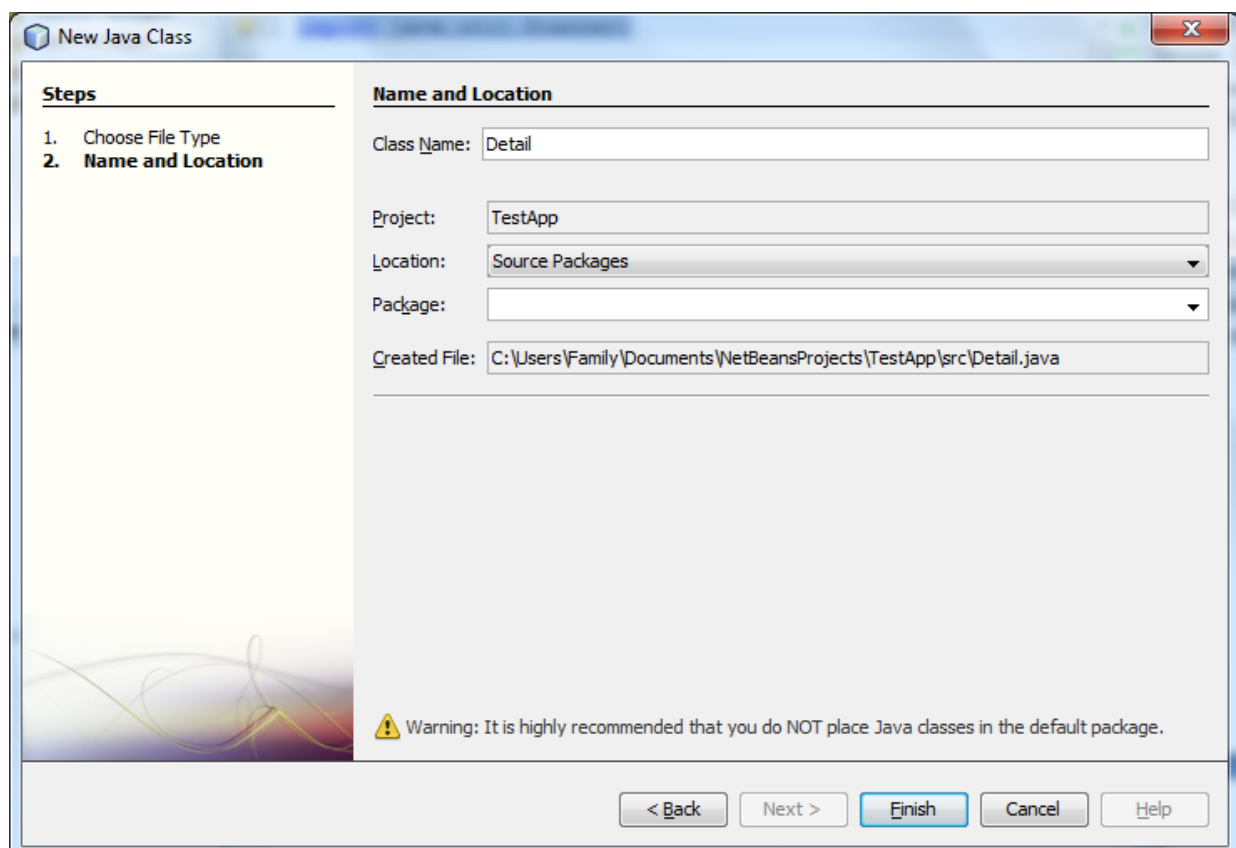


Figure 12. The dialog box for naming a new class

How to work with classes

Figure 13 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 figure 14 shows how to start this process off. The Encapsulate Fields dialog box, figure 15, will generate one get and one set method for each of the private fields for the class.

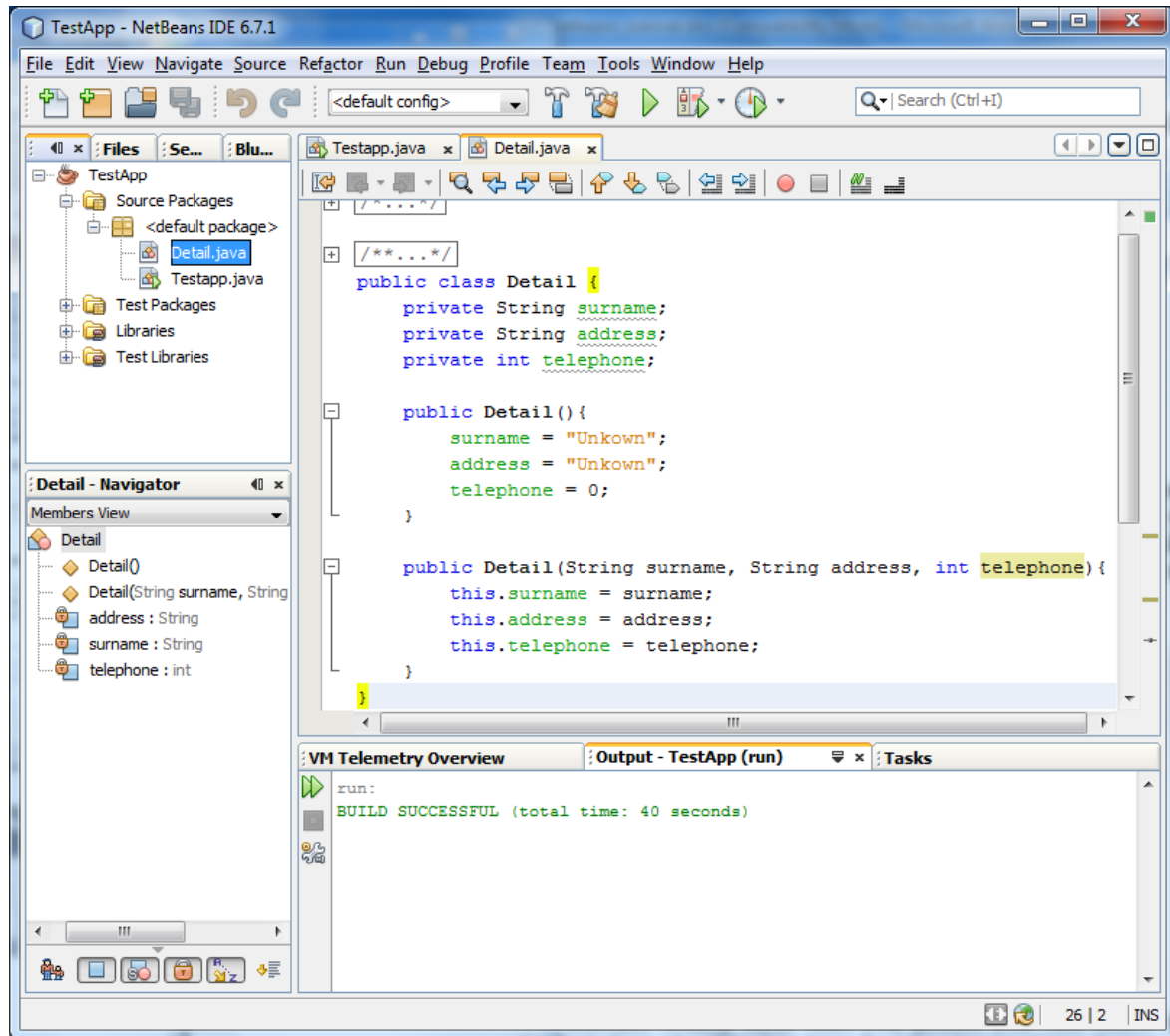


Figure 13. The NetBeans window for the Detail class

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

```
public String getSurname() {  
    return surname;  
}  
  
public void setSurname(String surname) {  
    this.surname = surname;  
}
```

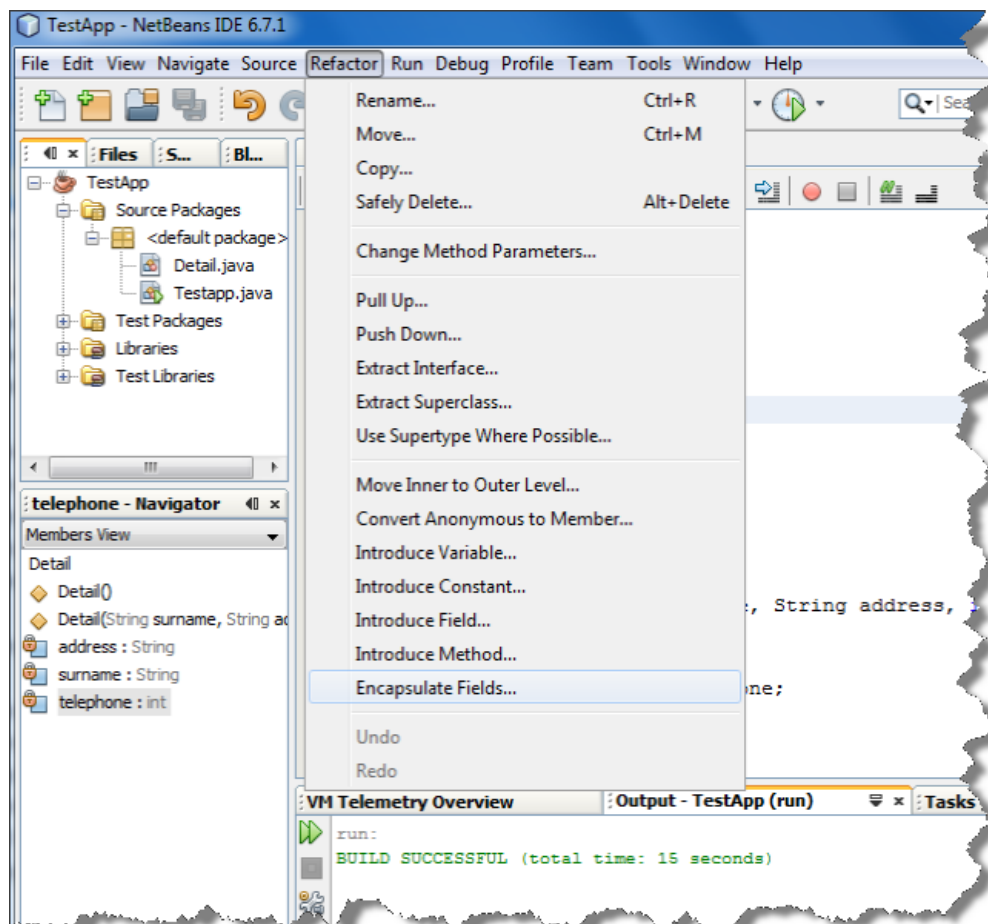


Figure 14. How to access the Encapsulate Fields dialog box

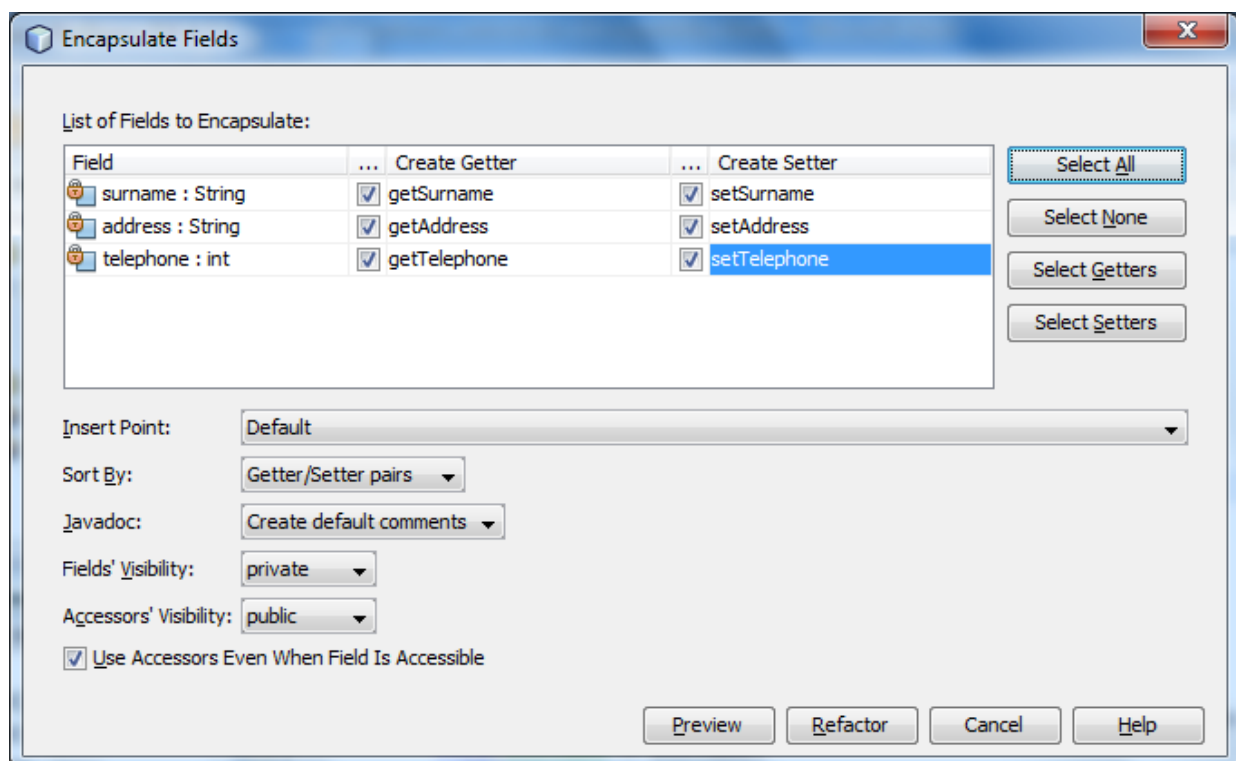


Figure 15. The dialog box for generating the get and set methods for a field

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 have 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.

How to work with 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 is shown in figure 11.

Once you have added an interface, you should not 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.

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.

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 will 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.

Once you have created some packages for your application, NetBeans can automatically add the necessary package statements when you create a new class or interface.

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.

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. 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 is 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.

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 is 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 is important to set the breakpoint before the line in the application that is causing the bug. Often, you can figure out where to set a breakpoint by reading the runtime exception that is 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 is available from the toolbar (just to the right of the Run Main Project button).

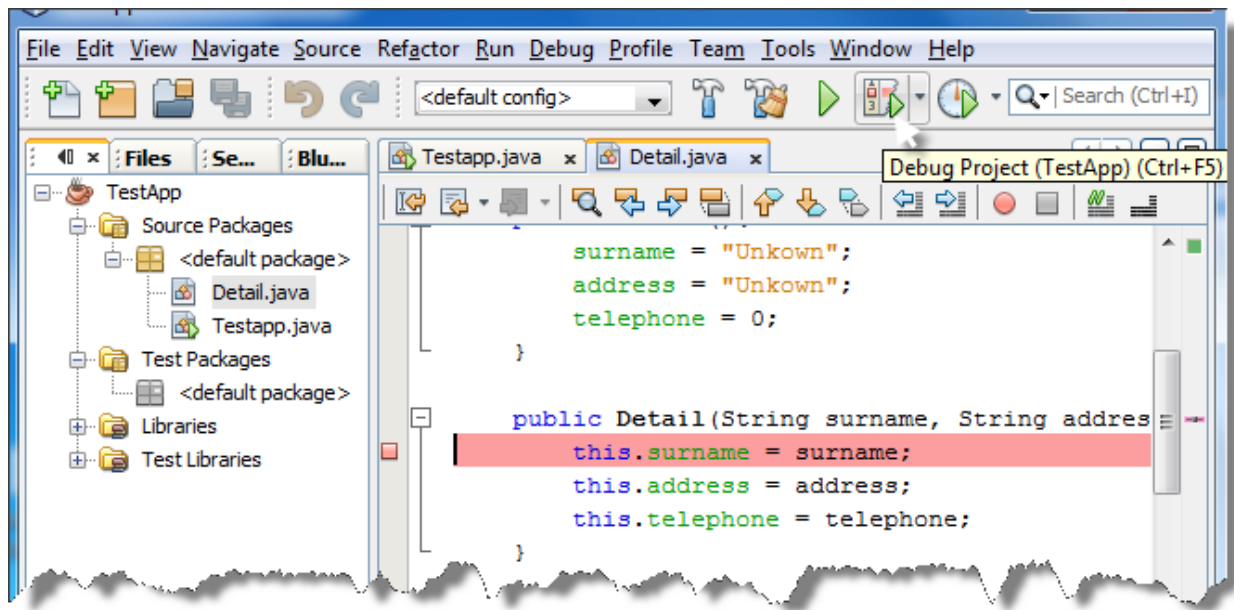


Figure 16. A code editor window with a breakpoint

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. 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.

How to step through code

When you run your 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. 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 are 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 are 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. The execution point is in the `calculateTotal` method of the `LineItem` class. Here, the `price` variable is a local variable that is 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 is 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 are 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.

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 11. Then, you can enter the code for the applet.

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 do not want to manually resize this dialog box, you can code an HTML page for the applet. 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.

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

Most of the projects you will do use classes that are available from the standard JDK 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 6, 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.

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.

How to use NetBeans to copy and rename files

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.

Exercise 1 Use NetBeans to develop an 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.
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, 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.
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.
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

1. Start NetBeans, and open the project named Ex2.
2. Check to make sure that the eric.jar file has been added to the Libraries folder. If it has not, add this JAR file to the Libraries folder as described in figure 6.
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.
6. Follow steps 3 through 5 for the TestScoreApp.java file.

Exercise 3 Test and debug an application

Test the Invoice application with invalid data

1. Open the Ex2 project, and test the Invoice application with an invalid subtotal like £1000 (enter the pound 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 is not 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.