

CHAPTER 5

Editing and Refactoring Code



- Opening the Source Editor
- Managing Automatic Insertion of Closing Characters
- Displaying Line Numbers
- Inserting Snippets from Code Templates
- Using Editor Hints to Generate Missing Code
- Matching Other Words in a File
- Generating Methods to Implement and Override
- Creating and Using Macros
- Creating and Customizing File Templates
- Handling Imports
- Displaying Javadoc Documentation While Editing
- Formatting Code
- Text Selection Shortcuts
- Navigating within the Current Java File
- Navigating from the Source Editor
- Searching and Replacing
- Refactoring Commands
- Deleting Code Safely
- Extracting a Superclass to Consolidate Common Methods
- Changing References to Use a Supertype
- Unnesting Classes
- Tracking Notes to Yourself in Your Code
- Comparing Differences Between Two Files
- Splitting the Source Editor
- Maximizing Space for the Source Editor
- Changing Source Editor Keyboard Shortcuts





NETBEANS IDE PROVIDES A WIDE VARIETY OF TOOLS to support Java application development, but it is the Source Editor where you will spend most of your time. Given that fact, a lot of attention has been put into features and subtle touches to make coding faster and more pleasurable.

Code completion and other code generation features help you identify code elements to use and then generate code for you. Refactoring features enable you to easily make complex changes to the structure of your code and have those changes propagated throughout your project. Keyboard shortcuts for these code generation features and for file navigation ensure that your hands rarely have to leave the keyboard.

Architecturally, the Source Editor is a collection of different types of editors, each of which contains features specific to certain kinds of files. For example, when you open a Java file, there is a syntax highlighting scheme specifically for Java files, along with code completion, refactoring, and other features specific to Java files. Likewise, when you open JSP, HTML, XML, `.properties`, deployment descriptor, and other types of files, you get a set of features specific to those files.

Perhaps most importantly, the Source Editor is tightly integrated with other parts of the IDE, which greatly streamlines your workflow. For example, you can specify breakpoints directly in the Source Editor and trace code as it executes. When compilation errors are reported in the Output window, you can jump to the source of those errors by double-clicking the error or pressing F12.

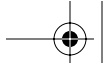
In this chapter, we will demonstrate the ways you can use the IDE's editing features to simplify and speed common coding tasks.

Opening the Source Editor

Before starting to work in the Source Editor, you will typically want to have an IDE project set up. You can then open an existing file or create a new file from a template. See Chapter 3 for basic information on creating projects and files and for a description of the various file templates.

If you would simply like to create a file without setting up a project, you can use the Favorites window. The Favorites window enables you to make arbitrary folders and files on your system accessible through the IDE. The Favorites window is





not designed for full-scale project development, but it can be useful if you just want to open and edit a few files quickly.

To use the Source Editor without creating a project:

1. Choose Window | Favorites to open the Favorites window.
2. Add the folder where you want the file to live (or where it already lives) by right-clicking in the Favorites window, choosing Add to Favorites, and choosing the folder from the file chooser.
3. In the Favorites window, navigate to the file that you want to edit and double-click it to open it in the Source Editor.

If you want to create a new file, right-click a folder node, choose New | Empty File, and enter a filename (including extension).

Managing Automatic Insertion of Closing Characters

When typing in the Source Editor, one of the first things that you will notice is that the closing characters are automatically inserted when you type the opening character. For example, if you type a quote mark, the closing quote mark is inserted at the end of the line. Likewise, parentheses(`()`), brackets (`[]`), and curly braces (`{}`) are completed for you.

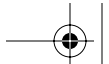
While this might seem annoying at first, the feature was designed to not get in your way. If you type the closing character yourself, the automatically inserted character is overwritten. Also, you can end a line by typing a semicolon (`;`) to finish a statement. The semicolon is inserted at the end of the line after the automatically generated character or characters.

See the following subtopics for information on how to use the insertion of matching closing characters.

Finishing a Statement

When the Source Editor inserts matching characters at the end of the line, this would appear to force you to move the insertion point manually past the closing character before you can type the semicolon. In fact, you can just type the semicolon without moving the insertion point, and it will be placed at the end of the line automatically.





For example, to get the line

```
ArrayList ls = new ArrayList();
```

you would only have to type

```
ArrayList ls = new ArrayList();
```

Splitting a String Between Two Lines

If you have a long string that you want to split between two lines, the Source Editor adds the syntax for concatenating the string when you press Enter.

For example, to get the lines

```
String s = "Though typing can seem tedious, reading long" +  
"and convoluted sentences can be even worse."
```

you could type

```
String s = "Though typing can seem tedious, reading long  
and convoluted sentences can be even worse."
```

The final three quote marks and the plus sign (+) are added for you.

If you want to break the line without creating the concatenation, press Shift-Enter.

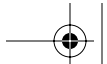
Displaying Line Numbers

By default, line numbers are switched off in the Source Editor to save space and reduce visual clutter. If you need the line numbers, you can turn them on by choosing View | Show Line Numbers. You can also right-click in the left margin of the Source Editor and choose Show Line Numbers.

Generating Code Snippets without Leaving the Keyboard

The Source Editor has several features for reducing the keystrokes needed for typing code. And you can access many of these features without using the mouse, having to use menus, or remembering scores of keyboard shortcuts.





Arguably the most important mechanisms for generating code are the following:

- Ctrl-spacebar keyboard shortcut. This shortcut opens the code completion box, as shown in Figure 5-1. The code completion box contains a context-sensitive list of ways you can complete the statement you are currently typing and of other code snippets you might want to insert in your code.
- Multi-keystroke abbreviations for longer snippets of code called code templates. These abbreviations are expanded into the full code snippet after you press the spacebar.
- Alt-Enter keyboard shortcut. You can use this shortcut to display suggestions the IDE has regarding missing code and then have the IDE insert that code. The IDE notifies you that it has a suggestion by displaying a lightbulb (💡) icon in the left margin of the line you are typing.

In addition to saving keystrokes and use of the mouse, these features might prevent typos and also help you find the right class and method names.

The following several sections illustrate how to get the most out of these features. These topics concentrate on features for Java files, but many of the features (such as code completion and word matching) are also available for other types of files, such as JSP and HTML files.

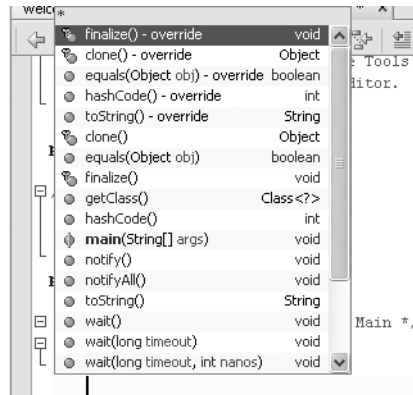
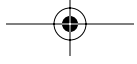
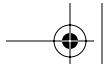


Figure 5-1 Code completion box





Using Code Completion

When you are typing Java identifiers in the Source Editor, you can use the IDE's code completion box to help you finish expressions, as shown in Figure 5-1. In addition, a box with Javadoc documentation appears (as shown in Figure 5-2) and displays documentation for the currently selected item in the code completion box.

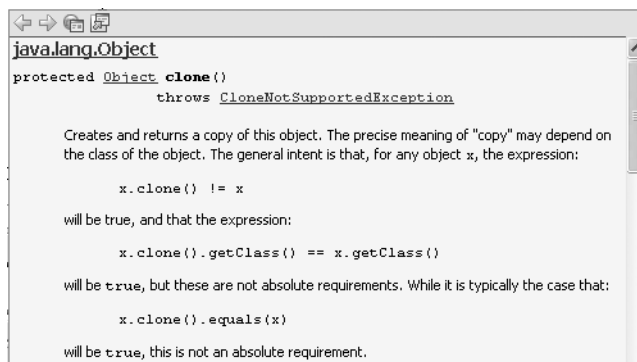
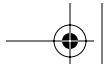


Figure 5-2 Javadoc box that accompanies the code completion box

Beginning with NetBeans IDE 5.0, many types of code generation have been added to the code completion box. Using the code completion box, you can:

- Fill in names of classes and class members. (After you select a class to fill in, an import statement is also filled in, if appropriate.)
- Browse Javadoc documentation of available classes.
- Generate whole snippets of code from dynamic code templates. You can customize code templates and create new ones. See *Inserting Snippets from Code Templates* for more information.
- Generate getter and setter methods.
- Generate skeletons for abstract methods of classes extended by and interfaces implemented by the current class.
- Override inherited methods.
- Generate skeletons of anonymous inner classes.





To open the code completion box, do one of the following:

- Type the first few characters of an expression and then press Ctrl-spacebar (or Ctrl- \backslash).
- Pause after typing a period (.) in an expression.
- Type a space, and then pause for a moment.

The code completion box opens with a selection of possible matches for what you have typed so far.

To narrow the selection in the code completion box, continue typing the expression.

To complete the expression and close the code completion box, do one of the following:

- Continue typing until there is only one option left and then press Enter.
- Scroll through the list, using the arrow keys or your mouse to select a value, and then press Enter.

To close the code completion box without entering any selection, press Esc.

To complete the expression and leave the code completion box open, select a completion and press the period (.) key. This is useful if you are chaining methods. For example, if you want to type

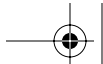
```
getRootPane().setDefaultButton(defaultButtonName)
```

you might do the following:

1. Type `getRo` (which would leave only `getRootPane()` in the code completion box) and press the period (.) key.
2. Type `.setDef` (which should make `setDefaultButton(JButton defaultButton)` the selected method in the code completion box, as shown in Figure 5-3) and press Enter.

`getRootPane().setDefaultButton()` should now be inserted in your code with the insertion point placed between the final parentheses. A tooltip appears with information on the type of parameter for you to enter.





3. Type a name for the parameter.
4. Type a semicolon (;) to finish the statement. The semicolon is automatically placed after the final parenthesis.

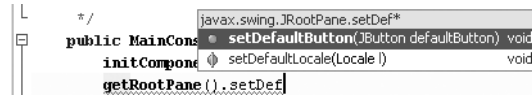


Figure 5-3 Code completion box with `setDefaultButton(JButton defaultButton)` selected

Code Completion Tricks

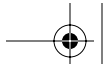
When typing with the code completion box open, there are a few tricks you can use to more quickly narrow the selection and generate the code you are looking for. For example:

- You can use “camel case” when typing class names. For example, if you want to create an instance of `HashSet`, you can type `private HS` and press `Ctrl-spacebar` to display `HashSet` (and other classes that have a capital H and a capital S in their names).
- You can use the comma (,) and semicolon (;) keys to insert the highlighted item from the code completion box into your code. The comma or semicolon is inserted into your code after the chosen item from the code completion box.
- You can fill in text that is common to all of the remaining choices in the list by pressing `Tab`. This can save you several keystrokes (or use of the arrow keys or mouse) when the selection in the code completion box is narrowed to choices with the same prefix. For example, if you are working with a `Hashtable` object `ht`, and you have typed `ht.n`, there will be two methods beginning with `notify` (`notify()` and `notifyAll()`). To more quickly narrow the selection to just `notifyAll()`, press `Tab` to expand `ht.n` to `ht.notify` and then type `A`. You can then press `Enter` to complete the statement with `notifyAll()`.

Disabling Automatic Appearance of the Java Code Completion Box

If you find the code completion box to be more of a nuisance than a help, you can disable automatic appearance of the code completion popup. Code





completion will still work if you manually activate it by pressing Ctrl-spacebar or Ctrl-\..

You can also leave automatic appearance of the code completion popup enabled but disable the bulkier Javadoc code completion dialog box. The Javadoc popup can be manually invoked with Ctrl-Shift-spacebar.

To disable automatic appearance of the code completion box:

1. Choose Tools | Options, click Editor in the left panel, and select the General tab.
2. Deselect the Auto Popup Completion Window property and click OK.

To disable automatic appearance of the Javadoc popup when you use the code completion box:

1. Choose Tools | Options, click Advanced Options, expand the Editing | Editor Settings node, and select the Java Editor node.
2. In the Expert settings, deselect the Auto Popup Javadoc Window property.



You can also merely adjust the amount of time that elapses before the code completion box appears. To do so, choose Tools | Options, click Advanced Options, expand the Editing | Editor Settings node, and select the Java Editor node. In the Delay of Completion Window Auto Popup property, enter a new value (in milliseconds). By default, the delay is 250 milliseconds.

Changing Shortcuts for Code Completion

If you prefer to use different shortcuts for code completion, you can change those shortcuts in NetBeans IDE:

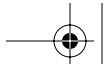
1. Choose Tools | Options, click Editor in the left panel, and select the Keymap tab.
2. Expand the Other folder and select the command for which you want to change the shortcut.

Commands that apply to code completion are Show Code Completion Popup, Show Code Completion Tip Popup, and Show Documentation Popup.

3. Click Add, and type the shortcut that you want to use.

You can remove an already assigned keyboard shortcut by selecting the shortcut and clicking Remove.





Inserting Snippets from Code Templates

As you are typing in the Source Editor, you can use code templates to greatly speed up the entry of commonly used sequences of reserved words and common code patterns, such as for loops and field declarations. The IDE comes with a set of templates, and you can create your own.

Some code templates are composed of segments of commonly used code, such as `private static final int`. Others are more dynamic, generating a skeleton and then letting you easily tab through them to fill in the variable text (without having to use the mouse or arrow keys to move the cursor from blank to blank). Where a code snippet repeats an identifier (such as an Iterator object, as shown in Figure 5-4), you just have to type the identifier name once.

```
for (Iterator it = collection.iterator(); it.hasNext();) {  
    Object elem = (Object) it.next();  
}
```

Figure 5-4 Code template with variable text to be inserted

Here are a few examples:

- You can use the `newo` template to quickly create a new object instance. You type `newo` and a space, the IDE generates `Object name = new Object(args);` and highlights the two occurrences of `Object`. You can then type a class name and press `Tab`. Both occurrences of `Object` are changed to the class name and then `args` is selected. You can then fill in the parameters and press `Enter` to place the insertion point at the end of the inserted code. You can use `Shift-Tab` to move backward through the parameters. You can press `Enter` at any time to skip any parameters and jump straight to the end of the template (or where it is specified that the cursor should rest after the template's parameters are filled in).
- You can use the `fori` template to create a loop for manipulating all of the elements in an array. Initially, the IDE generates the following:

```
for (int i = 0; i < arr.length; i++) {  
}
```




The index is automatically given a name that is unique within the current scope (defaulting to `i`). You can manually change that value (causing the IDE to change the value in all three places) or directly tab to `arr`, to type the array name. If an array is in scope, the IDE will use its name by default. The next time you press Tab, the cursor lands on the next line, where you can type the array processing code.

- You can use the `forc` template to create a skeleton for loop that uses an Iterator object to iterate over a collection as shown in Figure 5-4.

This code template has the additional benefit of generating an import statement for `Iterator`.

You can access code templates in either of the following ways:

- Typing the first few letters of the code, pressing Ctrl-spacebar, and then selecting the template from the list in the code completion box. In the code completion box, templates are indicated with the  icon, as shown in Figure 5-5. The full text of the template is shown in the Javadoc box.
- Typing the abbreviation for the code template directly in the Source Editor and then pressing the spacebar. You can find the abbreviations for the built-in Java code templates in Table 5-1. If you discover a code template in the code completion box, the abbreviation for that template is in the right column of that abbreviation's listing.

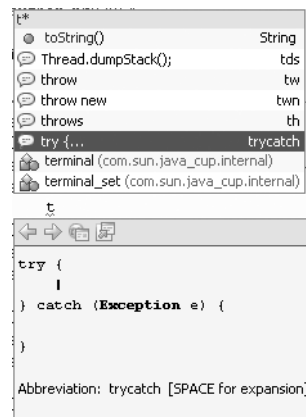
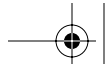


Figure 5-5 Code completion box and Javadoc box, with a code template selected



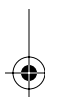
If an abbreviation is the same as the text that you want to type (for example, you do not want it to be expanded into something else), press Shift-spacebar to keep it from expanding.

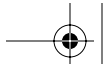
See Table 5-1 for a list of code templates (and their abbreviations) for Java files. The IDE also comes with sets of abbreviations for JSP files (see Chapter 8, Using Code Templates for JSP Files), XML files, and DTD files. You can create your own abbreviations for these file types and for other file types as well (such as for HTML files, SQL files, etc.).

Table 5-1 Java Code Templates in the Source Editor

Abbreviation	Expands To
ab	abstract
bo	boolean
br	break
ca	catch (
cl	class
cn	continue
df	default:
dowhile	do { \${cursor} } while (\${condition});
En	Enumeration
eq	equals
Ex	Exception
ex	extends
fa	false
fi	final
fy	finally
fl	float
forc	for (Iterator it = collection.iterator(); it.hasNext();) { Object elem = (Object) it.next(); }
fore	for (Iterator it = collection.iterator(); it.hasNext();) { Object elem = (Object) it.next(); }

(continued)



**Table 5-1** Java Code Templates in the Source Editor (*Continued*)

Abbreviation	Expands To
fori	<pre>for (int i = 0; i < \${arr array}.length; i++) { \${cursor} }</pre>
ie	interface
ifelse	<pre>if (\${condition}) { \${cursor} } else { }</pre>
im	implements
iof	instanceof
ir	import
le	length
newo	Object name = new Object(args);
Ob	Object
pst	printStackTrace();
pr	private
psf	private static final
psfb	private static final boolean
psfi	private static final int
psfs	private static final String
pe	protected
pu	public
Psf	public static final
Psfb	public static final boolean
Psfi	public static final int
Psfs	public static final String
psvm	<pre>public static void main(String[] args) { \${cursor} }</pre>
re	return
st	static
St	String
serr	System.err.println("\${cursor}");
sout	System.out.println("\${cursor}");
sw	switch {

(continued)

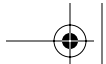


Table 5-1 Java Code Templates in the Source Editor (*Continued*)

Abbreviation	Expands To
sy	synchronized
tds	Thread.dumpStack();
tw	throw
twn	throw new
th	throws
trycatch	try { \${cursor} } catch (Exception e) { }
wh	while (
whilei	while (it.hasNext()) { Object elem = (Object) it.next(); \${cursor} }

Adding, Changing, and Removing Code Templates

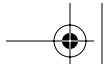
The code templates that come with the IDE are representative of the kind of things you can do with code templates, but they represent only a tiny fraction of the number of potentially useful templates.

You can modify existing code templates and create entirely new ones to suit the patterns that you use frequently in your code.

To create a new code template:

1. Choose Tools | Options, click Editor in the left panel, and select the Code Templates tab.
2. Click New.
3. In the New Code Template dialog box, type an abbreviation for the template and click OK.
4. In the Expanded Text field, insert the text for the template. See Code Template Syntax below for information on how to customize the behavior of your templates.
5. Click OK to save the template and exit the Options dialog box.





To modify a code template:

1. Choose Tools | Options, click Editor in the left panel, and select the Code Templates tab.
2. Select a template from the Templates table and edit its text in the Expanded Text field.
3. Click OK to save the changes and exit the Options dialog box.



In NetBeans IDE 5.0, you cannot directly change the abbreviation for a code template. If you want to assign a different shortcut to an existing template, select that shortcut, copy its expanded text, create a new code template with that text and a different abbreviation, and then remove the template with the undesired abbreviation.

To remove a code template:

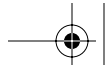
1. Choose Tools | Options, click Editor in the left panel, and select the Code Templates tab.
2. Select a template from the Templates table and click Remove.
3. Click OK to save the changes and exit the Options dialog box.

Code Template Syntax

In code templates, you can set up the variable text to provide the following benefits for the template user:

- Display a descriptive hint for remaining text that needs to be typed in.
- Enable typing of an identifier once and have it generated in multiple places
- Make sure that an import statement is added for a class.
- Specify a type that a parameter of the code template is an instance of in order for the IDE to automatically generate an appropriate value for that parameter when the template is used to insert code.
- Automatically set up a variable name for an iterator, making sure that that variable name is not already used within the current scope.
- Set a location for the cursor to appear within the generated snippet once the static text has been generated and the variable text has been filled in.





For example, you might want to easily generate something like the following code for a class that you instantiate often:

```
FileWriter filewriter = new FileWriter(outputFile);
```

In the definition for such a code template, you could use something like the following:

```
`${fw type = "java.io.FileWriter"  
  editable="false"} ${filewriter} = new ${fw}(${outputFile});
```

When the template is inserted into the code, the following things happen:

- ``${fw type = "java.io.FileWriter" editable="false"}`` is converted to `FileWriter` in the inserted code.
- ``${fw}`` is also converted to `Filewriter` (as it is essentially shorthand for the previously defined ``${fw type = "java.io.FileWriter" editable="false"}``).
- ``${filewriter}`` and ``${outputFile}`` generate text (`filewriter` and `outputFile`, respectively).
- `filewriter` is selected. You can type a new name for the field and then press `Tab` to select `outputFile` and type a name for that parameter. Then you can press `Tab` or `Enter` to place the cursor after the whole generated snippet.

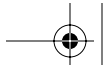
You could further refine the code template by defining an `instanceof` attribute for ``${outputFile}`` (e.g. `OutputFile instanceof = "java.io.File"`). This would enable the IDE to detect an instance of that class and dynamically insert the name of the instance variable in the generated snippet instead of merely `outputFile`.

See Table 5-2 for a description of examples of the syntax that you can use in the creation of code templates.

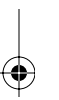
Changing the Expander Shortcut for Code Templates

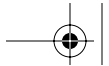
If you find that the code templates get in your way because they inadvertently get invoked when you type certain strings, you can configure the IDE to activate the templates with a different key or key combination. This enables you to continue using the code template feature without having to individually change any templates that get in your way.



**Table 5-2** Java Code Template Syntax

Syntax Element Example	Description
<code>\${}</code>	Used to enclose dynamic parts in the template.
<code>\${ElementName}</code>	Text you put within the braces will appear in the generated code snippet as highlighted text that you can type over. For example, if you use <code>\${Object}</code> , that would be a hint for the template user to type in an object name. If a given <i>ElementName</i> appears multiple times in the code template definition, you only have to replace the variable text once when you create a code snippet from that template.
<code>\${ElementName type="FULLY-QUALIFIED-CLASS-NAME" editable="false"}</code>	Includes a class name as part of the inserted code and has the IDE automatically insert an import statement for the class, if necessary. Here, <i>ElementName</i> can be any unique identifier, but it is only used within the template syntax. The class referred to in <i>FULLY-QUALIFIED-CLASS-NAME</i> is inserted when you use the template. Specifying <code>editable="false"</code> merely ensures that the inserted class name is not highlighted for editing. For example, the <code>for</code> code template uses <code>\${iterator type="java.util.Iterator" editable=false}</code> to enter <code>Iterator</code> into the generated code and add an import statement for that class.
<code>\${ElementName instanceof "FULLY-QUALIFIED-CLASS-NAME"}</code>	Declares that, if possible, the variable text that is initially generated should be the name of an instance variable of <i>FULLY-QUALIFIED-CLASS-NAME</i> that has been declared in the class. If there is no instance of that class available, <i>ElementName</i> is inserted. This construct is used in the <code>for</code> and <code>fore</code> templates.
<code>\${ElementName array}</code>	Declares that, if possible, the variable text that is initially generated should be the name of an array that is used in the class. If there is no array available within the current scope, <i>ElementName</i> is inserted. This construct is used in the <code>fori</code> template.
<code>\${ElementName index}</code>	Generates a variable that is unused in the current scope, the default being <i>i</i> . If <i>i</i> is already used, then <i>j</i> is attempted, and then <i>k</i> , etc.
<code>\${cursor}</code>	Determines where the cursor will end up after the code snippet has been inserted and all of the variable text has been filled in.





To change the code template expander key:

1. Choose Tools | Options, click Editor in the left panel, and select the Code Templates tab.
2. Select the preferred key or key combination from the Expand Template On drop-down list.
3. Click OK to save the change and exit the Options dialog box.

Using Editor Hints to Generate Missing Code

When the IDE detects an error for which it has identified a possible fix, a lightbulb (💡) icon appears in the left margin of that line. You can click the lightbulb or press Alt-Enter to display a list of possible fixes. If one of those fixes suits you, you can select it and press Enter to have the fix generated in your code.

Often, the “error” is not a coding mistake but a reflection of the fact that you have not gotten around to filling in the missing code. In those cases, the editor hints simply automate the entry of certain types of code.

For example, assume you have just typed the following code, but `x` is not defined anywhere in the class.

```
int newIntegerTransformer () {  
    return x;  
}
```

If your cursor is still resting on the line of the return statement, the 💡 icon will appear. If you click the icon or press Alt-Enter, you will be offered three possible solutions as shown in Figure 5-6. You can select one of those hints to generate the code.

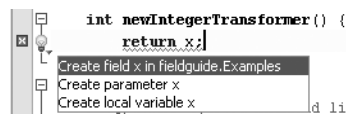
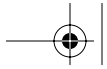


Figure 5-6 Display of editor hints





In NetBeans IDE 5.0, editor hints only appear when you are in the line where the error is detected. In subsequent releases that is likely to change.

The IDE is able to provide hints for and generate the following solutions to common coding errors:

- Add a missing `import` statement
- Insert abstract methods that are declared in a class' implemented interfaces and abstract superclasses
- Insert a method parameter
- Create a missing method
- Create a missing field
- Create a missing local variable
- Initialize a variable
- Insert a cast
- Add a `throws` clause with the appropriate exception
- Surround the code with a `try-catch` block including the appropriate exception

Matching Other Words in a File

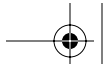
If you are typing a word that appears elsewhere in your file, you can use a keyboard shortcut to complete that word according to the first word found in the Source Editor that matches the characters you have typed. This word-match feature works for any text in the file. It also searches through files that you have been recently working in (in the order that you have last accessed the files).

To search backward from the cursor for a match, press `Ctrl-K`.

To search forward from the cursor for a match, press `Ctrl-L`.

For example, if you have defined the method `refreshCustomerInfo` on line 100 and now want to call that method from line 50, you can type `ref` and then press `Ctrl-L`. If there are no other words that start with `ref` between lines 50 and 100, the rest of the word `refreshCustomerInfo` will be filled in. If a different match is found, keep pressing `Ctrl-L` until the match that you want is filled in.





For typing variable names, you might find that the word match feature is preferable to code completion, since the IDE only has to search a few files for a text string (as opposed to code completion feature, where the IDE searches the whole classpath).

Generating Methods to Implement and Override

When you extend a class or implement an interface, you have abstract methods that you need to implement and possibly non-abstract methods that you can override. The IDE has several tools that help you generate these methods in your class:

- **Editor hints.** When you add the `implements` or `extends` clause, a lightbulb (💡) icon appears in the left margin. You can click this icon or press Alt-Enter to view a hint to implement abstract methods. If you select the hint and press Enter, the IDE generates the methods for you. This hint only is available when your cursor is in the line of the class declaration.
- **Code completion.** You can generate methods to implement and override individually by pressing Ctrl-Space and choosing the methods from the code completion box. As shown in Figure 5-7, methods to implement or override are marked `implement` and `override`, respectively.

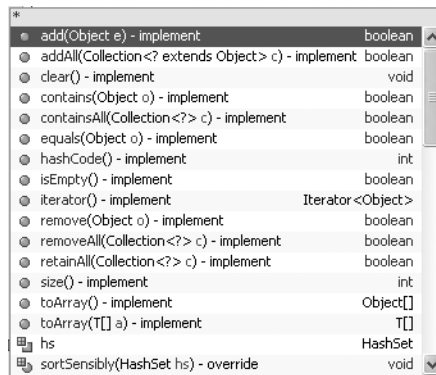


Figure 5-7 Code completion box showing methods to implement and methods available to be overridden

- **Override and Implement Methods dialog box.** You can use this dialog box (shown in Figure 5-8) for generating any combination of the available implementable or overridable methods. This feature also enables you to



generate calls to the super implementation of the methods within the body of the generated methods. To open this dialog box, choose Source | Override Methods or press Ctrl-I. To select multiple methods, use Ctrl-click.

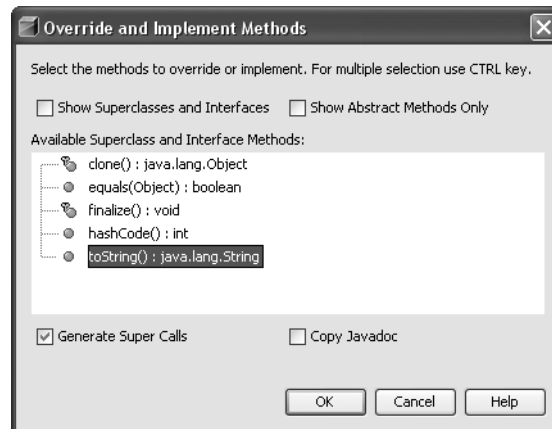


Figure 5-8 Override and Implement Methods dialog box

Generating JavaBeans Component Code

The IDE has a few levels of support for creating JavaBeans components. You can use the following features:

- **Code completion.** When you have a field in your class without a corresponding get or set method, you can generate that method by pressing Ctrl-Space and choosing the method from the code completion box.
- **Refactor | Encapsulate Fields command.** You can use this command to generate get and set methods, change the field's access modifier, and update code that directly accesses the field to use the getters and setters instead.
- **Bean Patterns node.** In the Projects window, each class has a subnode called Bean Patterns as shown in Figure 5-9. You can right-click this node and choose from a variety of commands that enable you to generate code for bean properties, property change support, and event listening. In addition, you can generate BeanInfo classes.

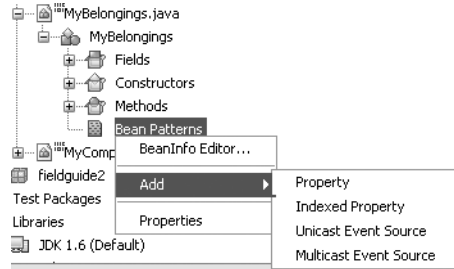
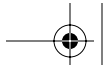


Figure 5-9 Bean Patterns node and menu

Creating and Using Macros

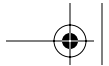
You can record macros in the IDE to reduce what would normally involve a long set of keystrokes to one keyboard shortcut. In macros, you can combine the typing of characters in the Source Editor and the typing of other keyboard shortcuts.

To record a macro:

1. Put the insertion point in the part of a file in the Source Editor where you want to record the macro.
2. Click the ● button in the Source Editor's toolbar (or press Ctrl-J and then type s) to begin recording.
3. Record the macro using any sequence of keystrokes, whether it is the typing of characters or using keyboard shortcuts. Mouse movements and clicks (such as menu selections) are not recorded.
4. Click the ■ in the Source Editor's toolbar (or press Ctrl-J and then type e) to finish recording.
5. In the Macro field of the Recorded Macro dialog box that appears, fine-tune the macro, if necessary.
6. Click Add to assign a keyboard shortcut to the macro.
7. In the Add Keybinding dialog box, press the keys that you want to use for the keyboard shortcut. (For example, if you want the shortcut Alt-Shift-Z, press the Alt, Shift, and Z keys.) If you press a wrong key, click the Clear button to start over.

Be careful not to use a shortcut that is already assigned. If the shortcut you enter is an editor shortcut, a warning appears in the dialog box. However, if





the key combination is a shortcut that applies outside of the Source Editor, you will not be warned.

You can assign a new shortcut in the Options window. Choose Tools | Options, click the Editor panel, select the Macros tab, and then click the Set Shortcut button.

Creating and Customizing File Templates

You can customize the templates that you create files from in the IDE and create your own templates. This might be useful if you need to add standard elements in all of your files (such as copyright notices) or want to change the way other elements are generated.

You can also create your own templates and make them available in the New File wizard.

There are several macros available for use in templates to generate text dynamically in the created files. These macros are identifiable by the double underscores that appear both before and after the macro name. See Table 5-3 for a list of the macros available.

To edit a template:

1. Choose Tools | Template Manager.
2. Expand the appropriate category node and select the template that you want to edit.
3. Click Open in Editor.
4. Edit the template and then save it.



Not all of the templates listed in the Template Manager can be modified at the user level. In some cases, the templates are available in the New File wizard but do not represent file constructs (such as those in the Enterprise and Sun Resources categories).

To create a new file template based on another template:

1. Choose Tools | Template Manager.
2. Navigate to and select the template on which you want to model the new template and click Duplicate.





A new node appears for the copied template. `_1` is appended to the template's name.


3. Click Open in Editor.
4. Edit the file, incorporating any of the template macros that you want to use (see Table 5-3), and save it.

If the template is for a Java class, you can use the filename for the class name and constructor name. These are automatically adjusted in the files you create from the template.

To import a file template:

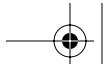
1. Choose Tools | Template Manager.
2. Select the category folder for the template.
3. Click Add to open the Add Template dialog box.
4. Navigate to and select the file that you want to import as a template. Then click Add.

Table 5-3 Java File Template Macros

Macro	Substituted Information
<code>__USER__</code>	Your username. If you would like to change the value of <code>__USER__</code> , choose Tools Options, click Advanced Options, and select the Editing Java Sources node. Then click the  button in the Strings Table property and change the value of USER.
<code>__DATE__</code>	The date the new file is created.
<code>__TIME__</code>	The time the new file is created.
<code>__NAME__</code>	The name of the class (without the file extension). It is best not to use this macro for the class and constructor name in the file (instead, use the filename).
<code>__PACKAGE__</code>	The name of the package where the class is created.
<code>__PACKAGE_SLASHES__</code>	The name of the class' package with slash (/) delimiters instead of periods (.).

(continued)



**Table 5-3** Java File Template Macros (*Continued*)

Macro	Substituted Information
<code>__PACKAGE_AND_NAME__</code>	The fully qualified name of the file (such as <code>com.mydomain.mypackage.MyClass</code>).
<code>__PACKAGE_AND_NAME_SLASHES__</code>	The fully qualified name of the file with slash (/) delimiters instead of periods (.).
<code>__QUOTES__</code>	A double quote mark ("). Use this macro if you want the substituted text to appear in quotes in the generated file. (If you place a macro within quote marks in the template, text is not substituted for the macro name in the created file.)

Handling Imports

When you use the IDE's code completion and editor hints features, import statements are generated for you automatically.

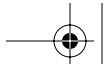
For example, if you have the code completion box open and you start typing a simple class name instead of its fully-qualified class name (e.g., you type `Con` and then select `ConcurrentHashMap` from the code completion box), the following import statement will be added to the beginning of the file:

```
import java.util.concurrent.ConcurrentHashMap;
```

For cases where these mechanisms are not sufficient for the management of import statements, you can use the following commands:

- **Fix Imports (Alt-Shift-F)**, which automatically inserts any missing import statements for the whole file. Import statements are generated by class (rather than by package). For rapid management of your imports, use this command.
- **Fast Import (Alt-Shift-I)**, which enables you to add an import statement or generate the fully qualified class name for the currently selected identifier. This command is useful if you want to generate an import statement for a whole package or if you want to use a fully qualified class name inline instead of an import statement.





Displaying Javadoc Documentation While Editing

The IDE gives you a few ways to access documentation for JDK and library classes.

To glance at documentation for the currently selected class in the Source Editor, press Ctrl-Shift-spacebar. A popup window appears with the Javadoc documentation for the class. This popup also appears when you use code completion. You can dismiss the popup by clicking outside of the popup.

To open a web browser on documentation for the selected class, right-click the class and choose Show Javadoc (or press Alt-F1).

To open the index page for a library's documentation in a web browser, choose View | Documentation Indices and choose the index from the submenu.



Documentation for some libraries is bundled with the IDE. However, you might need to register the documentation for other libraries in the IDE for the Javadoc features to work. See Making External Sources and Javadoc Available in the IDE in Chapter 3 for more information.

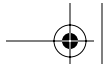
Paradoxically, JDK documentation is available through a popup in the Source Editor but not through a browser by default. This is because the Javadoc popup in the Source Editor picks up the documentation from the sources that are included with the JDK. However, the browser view of the documentation requires compiled Javadoc documentation, which you have to download separately from the JDK. See Referencing JDK Documentation (Javadoc) from the Project in Chapter 3.

Formatting Code

When you type or have code generated in the Source Editor, your Java code is automatically formatted in the following ways by default:

- Members of classes are indented four spaces.
- Continued statements are indented eight spaces.
- Any tabs that you enter are converted to spaces.
- When you are in a block comment (starting with `/**`), an asterisk is automatically added to the new line when you press Enter.
- The opening curly brace is put on the same line as the declaration of the class or method.
- No space is put before an opening parenthesis.





If your file loses correct formatting, you can reformat the whole file by selecting Source | Reformat Code (Ctrl-Shift-F). If you have any lines selected, the reformatting applies only to those lines.

If you have copied code, you can have it inserted with correct formatting by pasting with the Ctrl-Shift-V shortcut.

Indenting Blocks of Code Manually

You can select multiple lines of code and then indent all those lines by pressing Tab or Ctrl-T.

You can reverse indentation of lines by selecting those lines and then pressing Shift-Tab or Ctrl-D.

Changing Formatting Rules

For various file types, you can adjust formatting settings, such as for number of spaces per tab, placement of curly braces, and so on.

To adjust formatting rules for Java files:

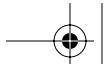
1. Choose Tools | Options.
2. Click Editor in the left panel and select the Indentation tab.
3. Adjust the properties for the indentation engine to your taste.
4. Reformat each file to the new rules by opening the file and pressing Ctrl-Shift-F (with no text selected).

For other file types, you can change formatting properties in the Advanced Options part of the Options Window and by adjusting that file type's indentation engine.

To change formatting rules for non-Java file types:

1. Choose Tools | Options and click Advanced Options.
2. Expand Editing | Indentation Engines, and select the indentation engine for the file type for which you want to modify formatting rules.





If there is no specific indentation engine for your file type, find out which indentation is being used for that file type by expanding Editing | Editor Settings, selecting the editor type, and looking at the value of its Indentation Engine property.

3. Adjust the properties for the indentation engine to your taste.
4. Reformat each file to the new rules by opening the file and pressing Ctrl-Shift-F (with no text selected).

There are other preset indentation engines available (the “simple” and “line wrapping” indentation engines) you might prefer to use.

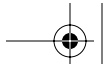
To change the indentation engine that is used for a file type:

1. Choose Tools | Options and click Advanced Options.
2. Expand Editing | Editor Settings and select the node for the editor type for which you want to change the indentation engine.
3. Select the indentation engine from the Indentation Engine property’s combo box.
4. Reformat each file to the new rules by opening the file and pressing Ctrl-Shift-F (with no text selected).

To create a new indentation engine:

1. Choose Tools | Options and click Advanced Options.
2. Expand Editing | Indentation Engines, right-click the node for the indentation engine on which you want to base your new indentation engine, and choose Copy.
3. Right-click the Indentation Engines node and choose Paste | Copy.
4. Modify the name of the indentation inline and adjust the properties to your taste.
5. In the Advanced Options part of the Options dialog box, expand Editing | Editor Settings and select the node for the editor (such as Java Editor or HTML Editor) that you want the indentation engine to apply to.





Changing Fonts and Colors

You can adjust the fonts that are used in the Source Editor and the way colors and background highlighting are used to represent syntactic elements of your code. You can make changes that apply to all file types and changes that apply to specific file types.

To make changes in fonts and colors that are reflected throughout all editor types:

1. Choose Tools | Options and click the Fonts & Colors panel.
2. In the Languages drop-down list, select All Languages.
3. In the Category list, select Default.
4. Use the Font, Foreground, Background, Effects, and Effect Color fields to change the appearance of that code element.

These changes should be reflected in the syntax categories for all the languages, since other categories are essentially designed as customizations of this one.

To change fonts and colors for a specific code syntax element:

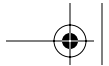
1. Choose Tools | Options and click the Fonts & Colors panel.
2. Select the editor type from the Languages drop-down list.
If the syntax element applies to multiple languages, select All Languages.
3. Select a syntax element in the Category list.
4. Use the Font, Foreground, Background, Effects, and Effect Color fields to change the appearance of that code element.



You can also create profiles of fonts and colors. The IDE comes with two profiles built in – NetBeans (the default profile) and City Lights (which is based on a black background).

You can choose the profile from the Profile combo box at the top of the Fonts & Colors panel. You can create a new profile by clicking Duplicate to start a new profile based on the selected profile.





Text Selection Shortcuts

To enable you to keep both hands on the keyboard, a number of shortcuts allow you to select text, deselect text, and change the text that is selected. See Table 5-4 for a selection of these shortcuts.

Table 5-4 Text Selection Shortcuts

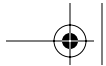
Description	Shortcut
Selects the current identifier or other word that the insertion point is on.	Alt-J
Selects all the text between a set of parentheses, brackets, or curly braces. The insertion point must be resting immediately after either the opening or closing parenthesis/bracket/brace.	Ctrl-Shift-[
Selects the current code element. Upon subsequent pressings, incrementally increases the size of the selection to include surrounding code elements. For example, if you press Alt-Shift-S once, the current word is selected. If you press it again, the rest of the expression might be selected. Pressing a third time might select the whole statement. Pressing a fourth time might select the whole method.	Alt-Shift-S
Selects the next (previous) character or extends the selection one character.	Shift-Right (Shift-Left)
Selects the next (previous) word or extends the selection one word.	Ctrl-Shift-Right (Ctrl-Shift-Left)
Creates or extends the text selection one line down (up).	Shift-Down (Shift-Up)
Creates or extends the text selection to the end (beginning) of the line.	Shift-End (Shift-Home)
Creates or extends the text selection to the end (beginning) of the document.	Ctrl-Shift-End (Ctrl-Shift-Home)
Creates or extends the text selection one page down (up).	Shift-Page Down (Shift-Page Up)



Navigating within the Current Java File

The IDE provides several mechanisms to make it easier to view and navigate a given Java file:





- The Navigator window, which appears below the Projects window and provides a list of members (for example, constructors, fields, and methods) in the currently selected Java file.
- Bookmarks, which enable you to easily jump back to specific places in the file.
- The Alt-K and Alt-L “jump list” shortcuts, mentioned in Jumping Between Areas Where You Have Been Working later in this chapter.
- Keyboard shortcuts to scroll the window. See Table 5-5 in the following section.
- The code folding feature, which enables you to collapse sections of code (such as method bodies, Javadoc comments, and blocks of import statements), thus making a broader section of your class visible in the window at a given time.

Viewing and Navigating Members of a Class

The IDE’s Navigator window (shown in Figure 5-10) provides a list of all “members” (constructors, methods, and fields) of your class. You can double-click a member in this list to jump to its source code in the Source Editor. Alternatively, instead of using the mouse, press Ctrl-7 to give focus to the Navigator window. Then begin typing the identifier until the Navigator locates it and press Enter to select that identifier in the Source Editor.

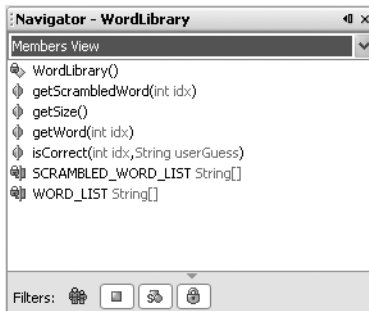
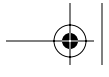


Figure 5-10 Navigator window

You can use the filter buttons at the bottom of the window to hide non-public members, static members, fields, and/or inherited members.





Moving the Insertion Point and Scrolling the Window

There is a wide range of shortcuts that you can use for moving the insertion point around and scrolling the Source Editor without moving the insertion point. See Table 5-5 for a list of some of the most useful file navigation shortcuts.

Table 5-5 Cursor and Scrolling Shortcuts

Description	Shortcut
Moves the insertion point to the next word (previous word).	Ctrl-Right (Ctrl-Left)
Moves the insertion point to the top (bottom) of the file.	Ctrl-Home (Ctrl-End)
Scrolls up (down) without moving the insertion point.	Ctrl-Up (Ctrl-Down)
Scrolls the window so that the current line moves to the top of the window.	Alt-U, then T
Scrolls the window so that the current line moves to the middle of the window.	Alt-U, then M
Scrolls the window so that the current line moves to the bottom of the window.	Alt-U, then B
Moves the insertion point to the parenthesis, bracket, or curly brace that matches the one directly before your insertion point.	Ctrl-[



Bookmarking Lines of Code

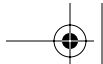
You can set bookmarks in files to make it easy to find an area of the file that you are working with frequently. You can then cycle through the file's bookmarks by pressing F2 (next bookmark) or Shift-F2 (previous bookmark).




To bookmark a line in a file, click in the line and press Ctrl-F2. To remove a bookmark, also use Ctrl-F2.

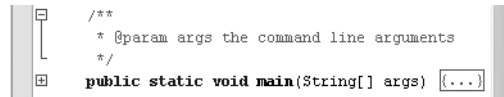
Hiding Sections of Code

You can collapse (or *fold*) low-level details of code so that only one line of that block is visible in the Source Editor, leaving more room to view other lines. Methods, inner classes, import blocks, and Javadoc comments are all foldable.







Collapsible blocks of code are marked with the  icon in the left margin next to the first line of the block. The rest of the block is marked with a vertical line that extends down from the  icon. Collapsed blocks are marked with the  icon. You can click one of these icons to fold or expand the particular block it represents. See Figure 5-11 for an example.



```

 /**
 * @param args the command line arguments
 */
 public static void main(String[] args) {...}

```

Figure 5-11 Examples of expanded and folded code in the Source Editor

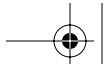
You can also collapse and expand single or multiple blocks of code with keyboard shortcuts and menu items in the Edit | Code Folds menu and the Code Folds submenu in the Source Editor. See Table 5-6 for a list of these commands and shortcuts.

Table 5-6 Code Folding Commands

Command	Shortcut
Collapse Fold	Ctrl-NumPad-Minus
Expand Fold	Ctrl-NumPad-Plus
Collapse All	Ctrl-Shift-NumPad-Minus
Expand All	Ctrl-Shift-NumPad-Plus
Collapse All Javadoc	none
Expand All Javadoc	none
Collapse All Java Code (collapses everything except Javadoc documentation)	none
Expand All Java Code (expands everything except Javadoc documentation)	none

By default, none of the code that you write is folded. You can configure the Source Editor to fold Java code by default when you create a file or open a previously unopened file.





To configure the IDE to fold certain elements of Java code automatically:

1. Choose Tools | Options, click the Editor panel, and select the General tab.
2. Where it says Collapse By Default, select the checkboxes for the elements that you want folded by default. You can choose from methods, inner classes, imports, Javadoc comments, and the initial comment.

Navigating from the Source Editor

The IDE includes handy shortcuts for navigating among files, different bits of code, and different windows. The more of these shortcuts you can incorporate into your workflow, the less your fingers will have to stray from your keyboard to your mouse.

Switching Between Open Files

Besides using the Source Editor's tabs and drop-down list, you can switch between open files using the keyboard shortcuts shown in Table 5-7.

Table 5-7 Shortcuts for Navigating Among Open Files

Shortcut	Description
Alt-Left and Alt-Right	Select files in order of tab position.
Ctrl-Tab	Opens a popup box showing all open files. Hold down the Ctrl key and press the Tab key multiple times until the file that you want to view is selected. Then release both keys to close the box and display the file.
Shift-F4	Opens a dialog box that lists all open files. You can use the mouse or the arrow keys to select the file that you want to view and press Enter to close the dialog box and display the file.

Jumping to Related Code and Documentation

The shortcuts in Table 5-8 enable you to jump to parts of the current file or other files that are relevant to the selected identifier. The first six of these shortcuts are available from the Navigate menu and the Go To submenu of the Source Editor's contextual (right-click) menu. The Show Javadoc command is available straight from the Source Editor's contextual menu.



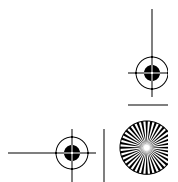
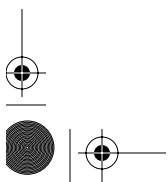
Table 5-8 Java Class Navigation Shortcuts

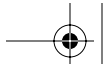
Command	Shortcut	Description
Go to Source	Alt-O (or Ctrl-click)	Jumps to the source code for the currently selected class, method, or field, if the source is available. You can achieve this either by pressing Alt-O with the identifier selected or by holding down the Ctrl key, hovering the mouse over the identifier until it is underlined in blue, and then clicking it.
Go to Declaration	Alt-G	Jumps to the declaration of the currently selected class, method, or field.
Go to Super Implementation	Ctrl-B	Jumps to the super implementation of the currently selected method (if the selected method overrides a method from another class or is an implementation of a method defined in an interface).
Go to Line	Ctrl-G	Jumps to a specific line number in the current file.
Go to Class	Alt-Shift-O	Enables you to type a class name and then jumps to the source code for that class if it is available to the IDE.
Go To Test	Alt-Shift-E	Jumps to the unit test for the selected class.
Show Javadoc	Alt-F1	Displays documentation for the selected class in a web browser. For this command to work, Javadoc for the class must be made available to the IDE through the Java Platform Manager (for JDK documentation) or the Library Manager (for documentation for other libraries). See Referencing JDK Documentation (Javadoc) from the Project in Chapter 3 and Making External Sources and Javadoc Available in the IDE, also in Chapter 3.

Jumping Between Areas Where You Have Been Working

When you are working on multiple files at once or in different areas of the same file, you can use the “jump list” shortcuts to navigate directly to areas where you have been working instead of scrolling and/or switching windows. The “jump list” is essentially a history of lines where you have done work in the Source Editor.

You can navigate back and forth between jump list locations with the Alt-K (back) and Alt-L (forward) shortcuts. Use Alt-Shift-K and Alt-Shift-L to navigate files in the jump list without stopping at multiple places in a file.





Jumping from the Source Editor to a File's Node

When you are typing in the Source Editor, you can jump to the node that represents the current file in other windows. This can be useful, for example, if you want to navigate quickly to another file in the same package or you want to browse versioning information for the current file.

See Table 5-9 for a list of available shortcuts.

Table 5-9 Shortcuts for Selecting the Current File in a Different Window

Command	Shortcut
Select the node for the current file in the Projects window.	Ctrl-Shift-1
Select the node for the current file in the Files window.	Ctrl-Shift-2
Select the node for the current file in the Favorites window.	Ctrl-Shift-3

Searching and Replacing

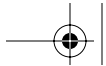
There are several types of searches in the IDE for different needs. You can

- Find occurrences of an identifier for a class, method, or field in your project using the Find Usages command
- Rename a class, method, or field throughout your project by using the Rename command
- Find and replace specific character combinations in an open file by pressing Ctrl-F in the file
- Find files that match search criteria based on characters in the file, characters in the filename, file type, and/or date by right-clicking a folder or project node in the Projects window and choosing Find (or by pressing Ctrl-F)

Finding Occurrences of the Currently Selected Class, Method, or Field Name

When you are working in the Source Editor, you can quickly find out where a given Java identifier is used in your project using the Find Usages command.





Find Usages improves upon a typical Find command by being sensitive to the relevance of text in the Java language context.

Depending on what kind of identifier you have selected and which options you have selected in the Find Usages dialog box, the Find Usages command output displays lines in your project that contain a combination of the following items:

- (For classes and interfaces) a declaration of a method or variable of the class or interface
- (For classes and interfaces) a usage of the type, such as at the creation of a new instance, importing a class, extending a class, implementing an interface, casting a type, or throwing an exception
- (For classes and interfaces) a usage of the type's methods or fields
- (For classes and interfaces) subtypes
- (For fields) the getting or setting of the field's value
- (For methods) the calling of the method
- (For methods) any overriding methods
- Comments that reference the identifier

The Find Usages command does not match

- Parts of words
- Words that differ in case

To find occurrences of a specific identifier in your code:

1. In the Source Editor, move the insertion point to the class, method, or field name that you want to find occurrences of.
2. Choose **Edit | Find Usages**, right-click and choose **Find Usages**, or press **Alt-F7**.
3. In the Find Usages dialog box, select the types of occurrences that you want displayed and click **Next**.

The results are displayed in the Usages window (shown in Figure 5-12), which appears at the bottom of the IDE.



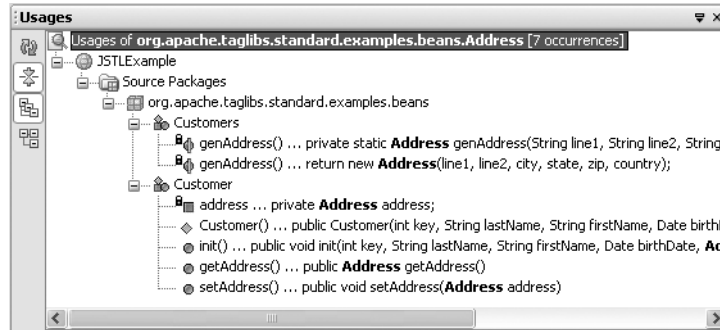
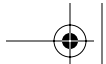


Figure 5-12 Usages window

You can navigate to a given occurrence of a class, method, or field name by double-clicking the occurrences line in the Usages window.

Renaming All Occurrences of the Currently Selected Class, Method, or Field Name

If you want to rename a class, method, or field, you can use the Refactor | Rename command to update all occurrences of the identifier in the Project to the new name. Unlike standard search and replace operations, the Rename command is sensitive to the Java context of your code, which makes it much more easy and reliable to use when reworking code. In addition, with the Rename command, you get a preview of the changes to be made and can prevent renaming of specific occurrences.

To rename a class, method, or field name:

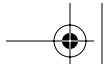
1. In the Source Editor, move the insertion point to an occurrence in the code of the class, method, or field name that you want to rename.
2. Right-click and choose Refactor | Rename or press Alt-Shift-R.
3. In the Rename dialog box, type the new name for the element.

If you want occurrences of the name in comments to also be changed, check the Apply Name on Comments checkbox.

4. In the Rename dialog box, click Next.

If you have deselected the Preview All Changes checkbox, the changes are applied immediately.





If you leave the Preview All Changes checkbox selected, the Refactoring window appears with a preview of the changes.

5. In the Refactoring window (shown in Figure 5-13), which appears at the bottom of the IDE, verify the occurrences that are set to change. If there is an occurrence that you do not want to change, deselect that line's checkbox.
6. Click Do Refactoring to apply the changes.

If you later find that the refactoring has had some consequences that you would like to reverse, you can choose Refactor | Undo.

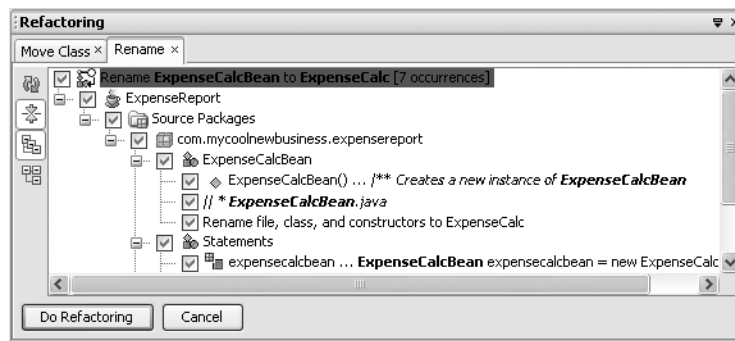


Figure 5-13 Refactoring preview window



You can initiate the renaming of a class or interface by renaming it inline in the Projects window. If you rename a node but do not want that change to be reflected in other places, select the Rename Without Refactoring checkbox.

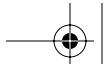
Searching and Replacing Combinations of Characters in a File

If you merely want to find a combination of characters in your file, click in the file that you want to search, choose Edit | Find (Ctrl-F), and type the text that you want to find in the Find dialog box (as shown in Figure 5-14).

In the Find dialog box, you can use a regular expression as your search criterion by selecting the Regular Expressions checkbox.

Unlike the Find Usages command, the Find command allows you to search for parts of words, do case-insensitive searches, and highlight matches in the current file.





Once you have dismissed the Find dialog box, you can jump between occurrences of the search string by pressing F3 (next occurrence) and Shift-F3 (previous occurrence).

To select the word in which the cursor is resting and start searching for other occurrences of that word, press Ctrl-F3.

To search and replace text, click in the file that you want to replace text, press Ctrl-H, and fill in the Find What and Replace With fields.

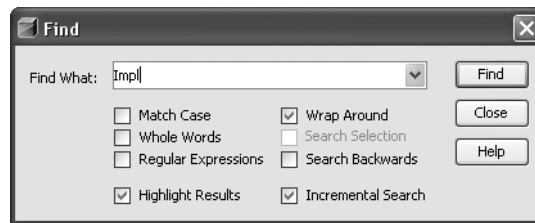


Figure 5-14 Find window for the Source Editor



By default, matches to a Find command remain highlighted in the Source Editor after you have dismissed the Find dialog box. To turn off the highlighting, press Alt-Shift-H.

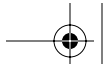
Other File Searches

If you want to do a search on multiple files for something other than an occurrence of a specific Java identifier, you can use the Find and Find in Projects commands. These commands enable you to search files within a folder, project, or all projects.

You can base these commands on any combination of the following types of criteria:

- Matches to a substring or regular expression on text in the file
- Matches to a substring or regular expression on the filename
- Dates the files were modified
- File type





To initiate such a file search, do one of the following:

- Choose Edit | Find in Projects to search all files in all open projects (including project metadata files).
- In the Projects window, right-click the node for the folder or project that you want to search in and choose Find (or press Ctrl-F). If you choose Find this way, the project metadata, including the build script and the contents of the nbproject folder, are not searched.
- Right-click a folder in the Files window and choose Find. If you choose Find this way, the project metadata, including the build script and the contents of the nbproject folder, are also searched.

After you initiate the search, fill as many search criteria as you would like. When you fill in a criterion on a given tab, the Use This Criterion for Search checkbox is selected. Deselect this checkbox if you decide to search according to a different type of criterion and you do not want the criterion on the currently selected tab to be used.

After you enter the criteria in the Find dialog box or the Find in Projects dialog box (shown in Figure 5-15) and click Search, the results are displayed in the Search Results window with nodes for each matched file. For full-text searches, these nodes can be expanded to reveal the individual lines where matched text occurs. You can double-click a match to open that file in the Source Editor (and, in the case of full-text matches, jump to the line of the match).



The dialog box that appears when you press Ctrl-F or choose Edit | Find (or Edit | Find in Projects) depends on which IDE window has focus. If you have the Source Editor selected, the Find dialog box for an individual file appears. If you have a node selected in the Projects window (or one of the other tree-view windows), the dialog box for searching in multiple files is opened.

Deleting Code Safely

Over time, your code might gather elements that have limited or no usefulness. To make the code easier to maintain, it is desirable to remove as much of this code as possible. However, it might be hard to immediately determine whether you can delete such code without causing errors elsewhere.



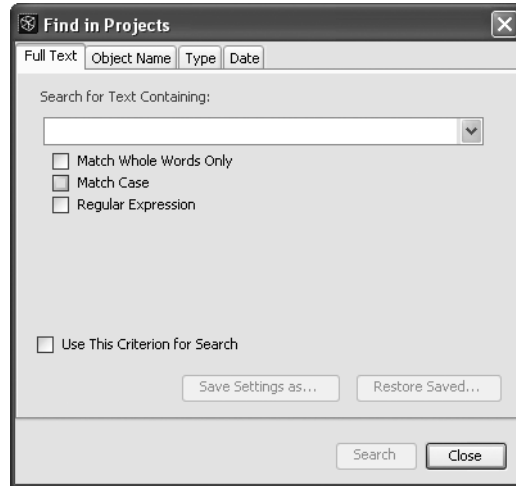
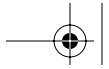


Figure 5-15 Find in Projects dialog box

Refactoring Commands

NetBeans IDE has special support for refactoring code. The term *refactoring* refers to renaming and rearranging code without changing what the code does. Reasons for refactoring include things such as the need to separate API from implementation, making code easier to read, and making code easier to reuse.

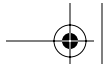
The IDE's refactoring support makes refactoring easier by enabling you to update all of the code in your project automatically to reflect changes that you make in other parts of your project.

For example, if you rename a class, references to that class in other classes are also updated.

You can access most refactoring commands from the Refactor menu on the main menu bar or by right-clicking in the Source Editor or on a class node in the Projects window and choosing from the Refactor submenu. The Find Usages command is in the Edit menu and the contextual (right-click) menu for the Source Editor and the Projects window.

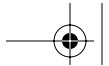
Typically, the currently selected identifier is filled in as the code element to be refactored.

Table 5-10 provides a summary of the refactoring commands that are available. These commands are explained more thoroughly in task-specific topics throughout this chapter.

**Table 5-10** Refactoring Commands

Command	Description
Find Usages	Displays all occurrences of the name of a given class, method, or field. See Finding Occurrences of the Currently Selected Class, Method, or Field Name earlier in this chapter.
Rename	Renames all occurrences of the selected class, interface, method, or field name. See Renaming All Occurrences of the Currently Selected Class, Method, or Field Name earlier in this chapter.
Safely Delete	Deletes a code element after making sure that no other code references that element. See Deleting Code Safely earlier in this chapter.
Change Method Parameters	Enables you to change the parameters and the access modifier for the given method. See Changing a Method's Signature later in this chapter.
Encapsulate Fields	Generates accessor methods (getters and setters) for a field and changes code that accesses the field directly so that it uses those new accessor methods instead. See Encapsulating a Field later in this chapter.
Move Class	Moves a class to a different package and updates all references to that class with the new package name. See Moving a Class to a Different Package later in this chapter.
Pull Up	Moves a method, inner class, or field to a class' superclass. You can also use this command to declare the method in the superclass and keep the method definition in the current class. See Moving Class Members to Other Classes later in this chapter.
Push Down	Moves a method, inner class, or field to a class' direct subclasses. You can also use this command to keep the method declaration in the current class and move the method definition to the subclasses. See Moving Class Members to Other Classes later in this chapter.
Extract Method	Creates a new method based on a selection of code in the selected class and replaces the extracted statements with a call to the new method. See Creating a Method from Existing Statements later in this chapter.
Extract Interface	Creates a new interface based on a selection of methods in the selected class and adds the new interface to the class' implements clause. See Creating an Interface from Existing Methods later in this chapter.

(continued)

**Table 5-10** Refactoring Commands (*Continued*)

Command	Description
Extract Superclass	Creates a new superclass based on a selection of methods in the selected class. You can have the class created with just method declarations, or you can have whole method definitions moved into the new class. See <i>Extracting a Superclass to Consolidate Common Methods</i> later in this chapter.
Use Supertype Where Possible	Change code to reference objects of a superclass (or other type) instead of objects of a subclass. See <i>Changing References to Use a Supertype</i> later in this chapter.
Move Inner to Outer Level	Moves a class up one level. If the class is a top-level inner class, it is made into an outer class and moved into its own source file. If the class is nested within the scope of an inner class, method, or variable, it is moved up to the same level as that scope. See <i>Unnesting Classes</i> later in this chapter.
Convert Anonymous Class to Inner	Converts an anonymous inner class to a named inner class. See <i>Unnesting Classes</i> later in this chapter.



The IDE's **Safely Delete** command can help you with the process of removing unused code, saving you cycles of manual searches and compilation attempts. When you use this command, the IDE checks to see if the selected code element is referenced elsewhere. If the code element is not used, the IDE deletes it. If the code element is used, the IDE displays where the code is used. You can then resolve references to the code you want to delete and then try the **Safely Delete** operation again.



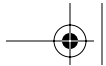
You can use the **Safely Delete** command on any type (such as a class, interface, and or enumeration), method, field, or local variable.

To safely delete a code element:

1. In the Source Editor or Projects window, right-click the code element that you want to delete and choose **Refactor | Safely Delete**.
2. In the Safe Delete dialog box, make sure that the item to be deleted is listed.
3. If you want the IDE to look inside comments for mentions of the code element, select the **Search in Comments** checkbox.

If this checkbox is not selected, comments referring to the code element are not affected if you delete the code element.





If this checkbox is selected, any found references to the code element in comments will prevent the code element from being immediately deleted, even if the code element is not used.

4. Click Next.

If no references to the code element are found, the Safe Delete dialog box closes and the code element is deleted.

If references to the code element are found, no code is deleted and the Safely Delete dialog box remains open. If you click the Show Usages button, the Usages window opens and displays the references to that code element.

Double-click an item in the list to jump to the line of code that it represents. If you remove the cited references, you can click the Rerun Safe Delete button to repeat the attempt to safely delete the code element.



To undo the Safely Delete command, choose Refactor | Undo.

Changing a Method's Signature

If you want to change a method's signature, you can use the IDE's Refactor | Change Method Parameters command to update other code in your project that uses that method. Specifically, you can

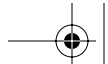
- Add parameters.
- Change the order of parameters.
- Change the access modifier for the method.
- Remove unused parameters.

You cannot use the Change Method Parameters command to remove a parameter from a method if the parameter is used in your code.

To change a method's signature:

1. Right-click the method in the Source Editor or the Projects window and choose Refactor | Change Method Parameters.
2. Click Add if you want to add parameters to the method. Then edit the Name, Type, and (optionally) the Default Value cells for the parameter. You have to double-click a cell to make it editable.





3. To switch the order of parameters, select a parameter in the Parameters table and click Move Up or Move Down.
4. Select the preferred access modifier from the Access Modifier combo box.
5. Click Next.

If you have deselected the Preview All Changes checkbox, the changes are applied immediately.

If you leave the Preview All Changes checkbox selected, the Refactoring window appears with a preview of the changes.

6. In the Refactoring window, look at the preview of the code to be changed. If there is a modification that you do not want to be made, deselect the checkbox next to the line for that change.
7. Click Do Refactoring.

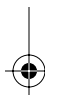
If you later find that the refactoring has had some consequences that you would like to reverse, you can choose Refactor | Undo.

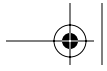
Encapsulating a Field

One common design pattern in Java programs is to make fields accessible and changeable only by methods in the defining class. In the convention used by JavaBeans components, the field is given private access and accessor methods are written for the field with broader access privileges. The names of the accessor methods are created by prefixing get and set to the field's name.

If you have fields that are visible to other classes and would like to better control access to those fields, you can use the IDE's Encapsulate Fields command to automate the necessary code modifications. The Encapsulate Fields command does the following things:

- Generates getter and setter methods for the desired fields.
- Enables you to change the access modifier for the fields and accessor methods.
- Changes code elsewhere in your project that accesses the fields directly to instead use the newly generated accessor methods.





To encapsulate fields in a class:

1. Right-click the field or the whole class in the Source Editor or the Projects window and choose Refactor | Encapsulate Fields.
2. In the Encapsulate Fields dialog box, select the Create Getter and Create Setter checkboxes for each field that you want to have encapsulated.

If you have selected a specific field, the checkboxes for just that field should be selected by default.

If you have selected the whole class, the checkboxes for all of the class' fields should be selected by default.

3. In the Fields' Visibility drop-down list, set the access modifier to use for the fields that you are encapsulating.

Typically, you would select `private` here. If you select a different visibility level, other classes will still have direct access to the fields for which you are generating accessor methods.

4. In the Accessors' Visibility drop-down list, set the access modifier to use for the generated getters and setters.

5. If you decide to leave the fields visible to other classes but you want to have current references to the field replaced with references to the accessor methods, select the Use Accessors Even When Field Is Accessible checkbox.

Otherwise, those direct references to the field will remain in the code.

This checkbox is only relevant if you decide to leave the fields accessible to other classes and there is code in those classes that accesses the fields directly.

6. Click Next.

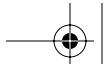
If you have deselected the Preview All Changes checkbox, the changes are applied immediately.

If you leave the Preview All Changes checkbox selected, the Refactoring window appears with a preview of the changes.

7. In the Refactoring window, verify the changes that are about to be made and click Do Refactoring.

If you later find that the refactoring has had some consequences that you would like to reverse, you can choose Refactor | Undo.





Moving a Class to a Different Package

If you want to place a class in a different package, you can use the IDE's refactoring features to move the class and then update references to that class automatically throughout your project.

To move a class:

1. In the Projects window, drag the class from its current package to the package you want to place it in. (You can also use the Cut and Paste commands in the contextual menus or the corresponding keyboard shortcuts.)
2. In the Move Class or Move Classes dialog box (shown in Figure 5-16), click Next after verifying that the To Package and This Class fields reflect the destination package and the class you are moving. (If you move multiple classes, a List of Classes text area is shown instead of the This Class field.)

If you have deselected the Preview All Changes checkbox, the changes are applied immediately.

If you leave the Preview All Changes checkbox selected, the Refactoring window appears with a preview of the changes.

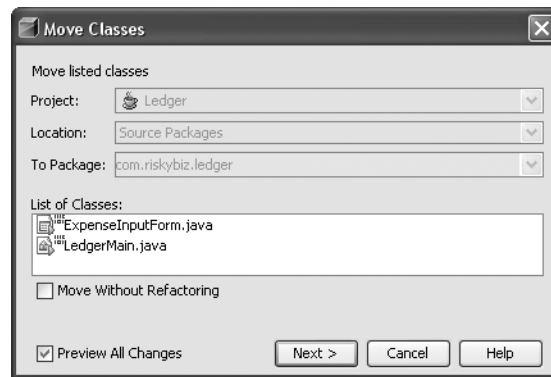


Figure 5-16 Move Classes dialog box

3. In the Refactoring window (shown in Figure 5-17), look at the preview of the code to be changed. If there is a modification that you do not want to be made, deselect the checkbox next to the line for that change.



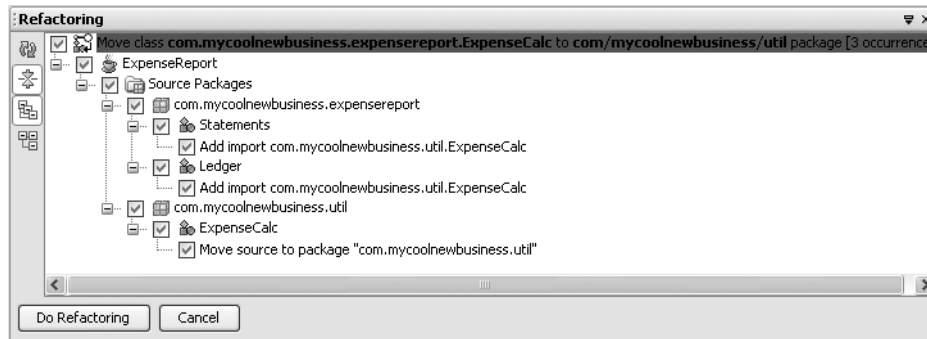
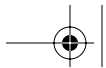


Figure 5-17 Refactoring window

4. Click Do Refactoring.

If you later find that the refactoring has had some consequences that you would like to reverse, you can choose Refactor | Undo.



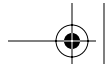
If you want to create a new package and move all of the classes in the old package to the new package, you can do an in-place rename of a package in the Projects window (or of a folder in the Files window).

Moving Class Members to Other Classes

The IDE has the Pull Up and Push Down refactoring commands for moving methods and fields to other classes and interfaces. When you use these commands to move class members, the IDE updates references to those members throughout your project. These commands are useful for improving the inheritance structure of your code. You can:

- Move methods and fields to a superclass or super-interface.
- Leave method implementations in the current class but create abstract declarations for those methods in a superclass.
- Move methods and fields to the class's subclasses or sub-interfaces.
- Move the implementations of methods to subclasses while leaving abstract method declarations in the current class.
- Move the interface name from the `implements` clause of a class to the `implements` clause of another class.





Moving Code to a Superclass

To move a member, a method's declaration, or an `implements` clause from the current class to a superclass:

1. In the Source Editor or the Projects window, select the class or interface that contains the member or members that you want to move.
2. Choose Refactor | Pull Up to open the Pull Up dialog box.
3. In the Destination Supertype drop-down list, select the superclass or interface that you want to move the members to.
4. Select the checkbox for each member that you want to move.

If you want to leave the method implementation in the current class and create an abstract declaration for the method in the superclass, select the Make Abstract checkbox for the method.

If the class from which you are moving members implements an interface, a checkbox for that interface is included in the dialog box. If you select the checkbox for that interface, the interface is removed from the `implements` clause of the current class and moved to the `implements` clause of the class to which you are moving members. (If you select a checkbox for an interface, be sure that all the checkboxes for the methods declared in that interface are also selected.)

5. Click Next.

If you have deselected the Preview All Changes checkbox, the changes are applied immediately.

If you leave the Preview All Changes checkbox selected, the Refactoring window appears with a preview of the changes.

6. In the Refactoring window, look at the preview of the code to be changed. If there is a modification that you do not want to be made, deselect the checkbox next to the line for that change.
7. Click Do Refactoring.

If you later find that the refactoring has had some consequences that you would like to reverse, you can choose Refactor | Undo.

Moving Code to Subclasses

To move a member, a method's implementation, or an `implements` clause from the current class to that class' subclasses:





1. In the Source Editor or the Projects window, select the class or interface that contains the member or members that you want to move.
2. Choose Refactor | Push Down to open the Push Down dialog box.
3. Select the checkbox for each member you want to move.

If you want to leave an abstract declaration for the method in the current class and move the implementation to the subclasses, select the Keep Abstract checkbox for the method.

If the class from which you are moving members implements an interface, a checkbox for that interface is included in the dialog box. If you select the checkbox for that interface, the interface is removed from the `implements` clause of the current class and moved to the `implements` clause of the class to which you are moving members.

4. Click Next.

If you have deselected the Preview All Changes checkbox, the changes are applied immediately.

If you leave the Preview All Changes checkbox selected, the Refactoring window appears with a preview of the changes.

5. In the Refactoring window, look at the preview of the code to be changed. If there is a modification that you do not want to be made, deselect the checkbox next to the line for that change.

6. Click Do Refactoring.

If you later find that the refactoring has had some consequences that you would like to reverse, you can choose Refactor | Undo.

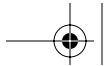
Creating a Method from Existing Statements

As your code evolves, you might find it desirable to break some methods up into multiple methods. You can use the Extract Method command to simplify this process. The Extract Method command does the following:

- Creates a new method and moves the selected statements to that method.
- Adds a call to the new method in the location from where the statements were moved.

To extract a method from existing statements:





1. In the Source Editor, select the statements that you want to be extracted into the new method.
2. Right-click the selection and choose Refactor | Extract Method.
3. In the Extract Method dialog box, enter a name for the method and select an access level.
4. If you want the method to be static, select the Static checkbox.
5. Click Next.

If you left the Preview All Changes checkbox clear, the method is immediately extracted.

If you have selected the Preview All Changes checkbox, the changes to be made to your code are listed in the Refactoring window. After verifying the changes, click Do Refactoring to complete the method extraction.

If you later find that the refactoring has had some consequences that you would like to reverse, you can choose Refactor | Undo.

Creating an Interface from Existing Methods

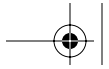
If you decide to divide your code into API and implementation layers, you can get started on that conversion by using the Extract Interface command to create an interface from methods in an existing class. The Extract Interface command does the following:

- Creates a new interface containing declarations for selected public methods.
- Adds the name of the created interface to the `implements` clause of the class from which the interface is extracted. (If the interface is extracted from another interface, the name of the newly created interface is added to the `extends` clause of the other interface.)

To extract an interface from existing methods:

1. In the Source Editor or the Projects window, select the class that contains the methods that you want to be extracted into the new interface.
2. Choose Refactor | Extract Interface.
3. In the Extract Interface dialog box, select the checkbox for each method that you want to be declared in the new interface.





If the class from which you are extracting an interface already implements an interface, a checkbox for that interface is included in the Extract Interface dialog box. If you select the checkbox for that interface, the interface is removed from the `implements` clause of the previously implementing interface and moved to the `extends` clause of the new interface.

4. Click Next.

If you have deselected the Preview All Changes checkbox, the changes are applied immediately.

If you leave the Preview All Changes checkbox selected, the Refactoring window appears with a preview of the changes.

5. In the Refactoring window, look at the preview of the code to be changed. If there is a modification that you do not want to be made, deselect the checkbox next to the line for that change.

6. Click Do Refactoring.

If you later find that the refactoring has had some consequences that you would like to reverse, you can choose Refactor | Undo.



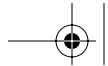
When you use the Extract Interface command, the interface is always created in the same package as the class from which it was extracted. If you want to move the interface to another package, you can use the Refactor | Move Class command to do so.

Extracting a Superclass to Consolidate Common Methods

As a project evolves, you might need to add levels to your inheritance hierarchy. For example, if you have two or more classes with essentially duplicate methods that are not formally related, you might want to create a superclass to hold these common methods. Doing so will make your code easier to read, modify, and extend, whether now or in the future.

You can use the Extract Superclass command to create such a superclass based on methods in one of the classes that you want to turn into a subclass. For each method that you add to the superclass, the Extract Superclass command enables you to choose between the following two options:

- Moving the whole method to the superclass
- Creating an abstract declaration for the method in the superclass and leaving the implementation in the original class



To extract a new superclass:

1. In the Source Editor or the Projects window, select the class that contains the methods that you want to be extracted into the new superclass.
2. Choose Refactor | Extract Superclass.
3. In the Extract Superclass dialog box, select the checkbox for each method and field that you want to be moved to the new superclass. Private methods and private fields are not included.

If you want to leave a method implementation in the current class and create an abstract declaration for the method in the superclass, select the Make Abstract checkbox for the method.

If the class from which you are extracting a superclass implements an interface, a checkbox for that interface is included in the Extract Superclass dialog box. If you select the checkbox for that interface, the interface is removed from the `implements` clause of the class that you are extracting from and moved to the `implements` clause of the new superclass.

4. Click Next.

If you have deselected the Preview All Changes checkbox, the changes are applied immediately.

If you leave the Preview All Changes checkbox selected, the Refactoring window appears with a preview of the changes.

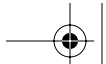
5. In the Refactoring window, look at the preview of the code to be changed. If there is a modification that you do not want to be made, deselect the checkbox next to the line for that change.
6. Click Do Refactoring.

If you later find that the refactoring has had some consequences you would like to reverse, you can choose Refactor | Undo.

After you have extracted the superclass, you can use a combination of the following techniques to complete the code reorganization:

- Add the name of the new superclass to the `extends` clause of any other classes that you want to extend the new superclass.
- Use the Pull Up command to move methods from other classes to the new superclass. As with the Extract Superclass command, you can move whole





methods or merely create abstract declarations for the methods in the new superclass. See *Moving Class Members to Other Classes* earlier in this chapter.

- For other classes that you want to extend the new superclass, use the *Override and Implement Methods* feature (Ctrl-I) to add methods from the superclass to those classes. See *Generating Methods to Override from Extended Classes* earlier in this chapter.
- Use the *Use Supertype Where Possible* command to change references in your code to the original class to the just created superclass. See *Changing References to Use a Supertype* below.

Changing References to Use a Supertype

You can use the *Use Supertype Where Possible* refactoring command to change code to reference objects of a superclass (or other type) instead of objects of a subclass. The operation only changes the reference in places where your code can accommodate such upcasting.

Typically you would use this refactoring operation to enable a single method to take as an argument different types of objects (all deriving from the same superclass).

This operation might be particularly useful after you have used the *Extract Superclass* command.

To change references to a supertype:

1. Select the class to which you want to replace references and choose *Refactor | Use Supertype Where Possible*.
2. In the *Select Supertype to Use* list, select the class or other type that should be referenced instead of the type currently referenced and click *Next*.
If you have deselected the *Preview All Changes* checkbox, the changes are applied immediately.
If you leave the *Preview All Changes* checkbox selected, the *Refactoring* window appears with a preview of the changes.
3. In the *Refactoring* window, look at the preview of the code to be changed. If there is a modification that you do not want to be made, deselect the checkbox next to the line for that change.





4. Click Do Refactoring.

If you later find that the refactoring has had some consequences that you would like to reverse, you can choose Refactor | Undo.

Unnesting Classes

As a project grows, you might find that some classes become tangled with a dense structure of inner classes that are hard to read and which cannot be elegantly modified or extended. If this is the case, you might want to simplify the nesting structure and move some classes into their own source files.

The IDE has a pair of useful commands for simplifying your code's nesting structure:

- **Move Inner to Outer Level.** This command moves a class up one level. If the class is a top-level inner class, it is made an outer class and moved into its own source file. If the class is nested within an inner class, method, or variable scope, it is moved up one level.
- **Convert Anonymous Class to Inner.** This command converts an anonymous inner class (i.e., a class that is unnamed and has no constructor) into a named inner class (inner class that has a name and a constructor). This also makes it possible for other code to reference this class.

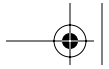


Moving an Inner Class up One Level

To move an inner class up one level:

1. In the Source Editor, right-click the inner class that you want to move and choose Refactor | Move Inner to Outer Level.
2. In the Class Name field of the Move Inner to Outer Level dialog box, set the name of the class.
3. Select the Declare Field for the Current Outer Class checkbox if you want to generate a field in the moved inner class to hold the outer class instance and include a reference to that instance as a parameter in the moved class' constructor.





If you select this option, fill in the Field Name text field with a name for the the outer class' instance field.

4. Click Next.

If you have deselected the Preview All Changes checkbox, the changes are applied immediately.

If you leave the Preview All Changes checkbox selected, the Refactoring window appears with a preview of the changes.

5. In the Refactoring window, look at the preview of the code to be changed. If there is a modification that you do not want to be made, deselect the checkbox next to the line for that change.

6. Click Do Refactoring.

If you later find that the refactoring has had some consequences that you would like to reverse, you can choose Refactor | Undo.

Unless you have selected the Preview All Changes box, the inner class is immediately moved up one level.

If you have selected the Preview All Changes box, the changes to be made are shown in the Refactoring window. You can then apply the changes by clicking Do Refactoring.

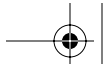
If the result is different from what you expected, you can reverse the command by choosing Refactor | Undo.

Converting an Anonymous Inner Class to a Named Inner Class

To convert an anonymous inner class to a named inner class:

1. In the Source Editor, right-click the anonymous inner class that you want to convert and choose Refactor | Convert Anonymous Class to Inner.
2. In the Inner Class Name field of the Convert Anonymous Class to Inner dialog box, enter a name for the class.
3. In the Access field, select the access modifier for the class.
4. Select the Declare Static checkbox if you want the class to be static.
5. In the Constructor Parameters list, use the Move Up and Move Down buttons to set the order of the parameters.
6. Click Next.





Unless you have selected the Preview All Changes box, the anonymous class is converted to the named inner class.

If you have selected the Preview All Changes box, the changes to be made are shown in the Refactoring window. You can then apply the changes by clicking Do Refactoring.

If the result is different from than expected, you can reverse the command by choosing Refactor | Undo.

Tracking Notes to Yourself in Your Code

The IDE has a task list feature that provides a way for you to write notes in your code and then view all of these notes in a single task (or “to do”) list. You can use the task list as the center of operations when cleaning up loose ends in your code.

A line is displayed in the task list if it is “tagged” with (contains) any of the following text:

- @todo
- TODO
- FIXME
- XXX
- PENDING
- <<<<<<<<



When you type a tag in your code, it must be typed as a whole word for the IDE to recognize it. For example, if you do not put a space between the tag and the note, the note will not appear in the task list.

To view the task list, choose Window | To Do (or press Ctrl-6).

Once you have displayed the To Do window (shown in Figure 5-18), you can view tasks for the current file, for all open files, or for a specific folder by clicking the corresponding button at the top of the To Do window.



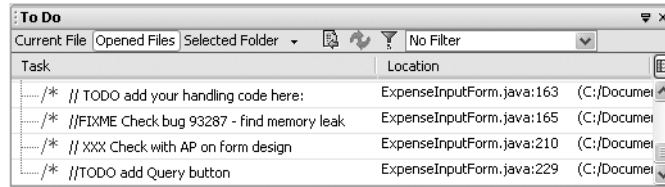
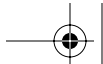



Figure 5-18 To Do window

You can sort task-list items by task, location, or priority by clicking the corresponding column titles. See *Displaying Tasks by Priority* later in this chapter for information on displaying the Priority column.

You can jump from an entry in the task list straight to the line in the code where you wrote the note by double-clicking the entry.


Adding, Removing, and Changing Task-List Tags

To change the tags that are used for the task list:

1. Choose Tools | Options, click Advanced Options, and select the Editing | To Do Settings node.
2. Click the  button in the Task Tags property.
3. In the To Do Settings dialog box, use the Add, Change, and Delete buttons to modify the contents of the Task List table.

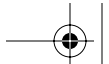
Displaying Tasks by Priority

You can also display priorities for each task-list item. The available priorities are High, Medium-High, Medium, Medium-Low, and Low.


By default, the Priority column is not displayed. You can display the Priority column by clicking the  icon and selecting the Priority checkbox in the Change Visible Columns dialog box.

The priority values can be assigned by tag. By default, all tags are assigned Medium priority except the <<<<<< tag, which is given High priority.






To change a priority value for a tag:

1. Choose Tools | Options, click Advanced Options, and select the Editing | To Do Settings node.
2. Click the  button in the Task Tags property.
3. In the To Do Settings dialog box, select the new priority in the combo box in the Priority column for the tag that you want to change.

Filtering Task-List Entries

You can further limit the entries displayed in the task list by creating and using filters. When you use a filter, only entries that match criteria specified by the filter are displayed. Criteria include text that needs to appear in the note, the priority of the task, and/or the filename.

To create a filter:

1. Click the  icon in the To Do window's toolbar.
2. In the Edit Filters dialog box, click the New button and then type a name for the filter in the Name field.
3. Fill in the details for the criterion.
4. Optionally, add additional criteria by clicking the More button and then filling in the details for the filters. You can select to have the filter match all or any of the criteria using the radio buttons at the top of the dialog box.

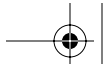
An entry for the newly defined filter appears in a combo box in the To Do Window toolbar.

Comparing Differences Between Two Files

You can generate a side-by-side comparison of two files with the differing lines highlighted. To compare two files, select the nodes for the two files in the Projects window and choose Tools | Diff.

The “diff” appears as a tab in the Source Editor.





The Diff command appears in the Tools menu only when two (and no more than two) files are selected in the Projects, Files, or Favorites window.

Splitting the Source Editor

You can split the Source Editor to view two files simultaneously or to view different parts of the same file.

To split the Source Editor window:



1. Make sure at least two files are already open.
2. Click a tab on one file; hold down the mouse button; and drag the tab to the far left, far right, or bottom of the Source Editor window.
3. Release the mouse button when the red outline that appeared around the tab when you started dragging changes to a rectangle indicating the placement of the split window.

To view different parts of the same file simultaneously:

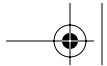
1. Click the file's tab in the Source Editor and choose Clone Document to create a second tab for the same document.
2. Drag and drop one of the file tabs to create a split Source Editor area. (See the procedure above for info on dragging and dropping Source Editor tabs.)

Maximizing Space for the Source Editor

There are a number of things you can do to make more space for your code in the IDE, such as:

- Maximize a file in the Source Editor within the IDE by double-clicking that file's tab. When you do this, the file takes the entire space of the IDE except for the main menu and row of toolbars. You can make the other windows reappear as they were by double-clicking the tab again.
- Make other windows "sliding" so that they appear only when you click or mouse over a button representing that window on one of the edges of the IDE. You can make a window sliding by clicking its  icon. You can return the window to its normal display by clicking the  button within the sliding





window. See Managing IDE Windows in Chapter 2 for information on working with windows in the IDE.

- Hide the IDE's toolbars. You can toggle the display of the main toolbars by choosing View | Toolbars and then individually choosing the toolbars that you want to hide (or display). You can toggle the display of the Source Editor's toolbar by choosing View | Show Editor Toolbar.

Changing Source Editor Keyboard Shortcuts

You can change existing keyboard shortcuts or map other available commands to shortcuts.

To add a keyboard shortcut for a command:

1. Choose Tools | Options and click the Keymap panel.
2. In the Actions panel, navigate to a command that you want to change, and click Add.
3. In the Add Shortcut dialog box, type in the key combination that you want to use and click OK.



The IDE also comes with keyboard shortcut profiles for the Eclipse IDE and Emacs, either of which you can select from the Profiles drop-down box in the Keymap panel. You can also create your own profiles.

