Java Development User Guide

Getting Started

Table of Contents

Preparing the Workbench

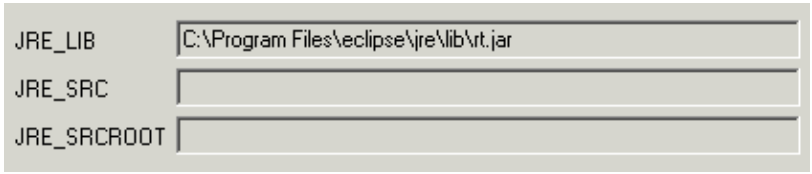
This Getting Started tutorial is designed to help you learn about using the Java development tooling (JDT) in the workbench. This tutorial assumes the following:

- That you are starting with a brand new workbench installation with all the default settings and preferences.
- That you are familiar with the basic workbench mechanisms (e.g., views, text searching, perspectives, etc.)

In this section, you will verify that the workbench is properly set up for Java development.

Verifying JRE Installation & Classpath Variables

1. In the main workbench window, select Window > Preferences > Java > Installed JREs.
2. Make sure that the Standard VM located in the <workbenchRoot>\JRE directory has been detected.
3. The three variables on this page are preset depending on which JRE has been chosen as the default.

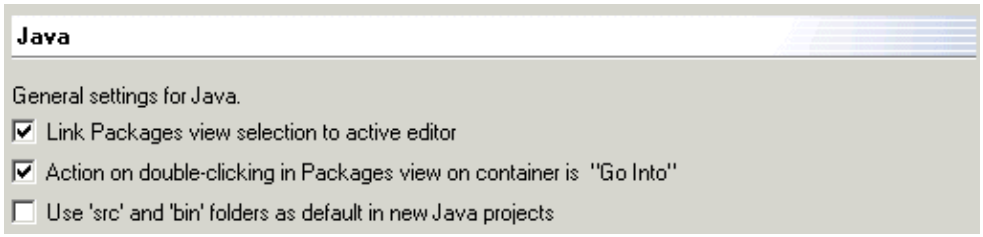


4. On the left, click Workbench to display the Workbench preferences page.

   On this page, verify that the Perform build automatically on resource modification option is checked.

5. On the left, click Java to display the Java preferences page.

   On this page, verify that the Use 'src' and 'bin' folders as defaults for new projects option is checked.



6. Click OK when you have verified that the workbench settings are correct.

Installing the JDT Example

Note: The JDT example project contributes a New Wizard that automatically creates a sample project in your workbench. For this tutorial, however, we will perform these steps manually.

1. Download the ZIP file containing the examples.
2. Extract the contents of the ZIP file to the root directory of your workbench installation.

   Note: The workbench should not be running while the examples are being installed.

   For example, if you installed the Eclipse Project SDK on d:\eclipse−sdk then extract the contents of the examples ZIP file to d:\eclipse−sdk.
3. Start the workbench. The example plug−ins are installed.

Creating Your First Java Project

In this section, you will create a new Java project.

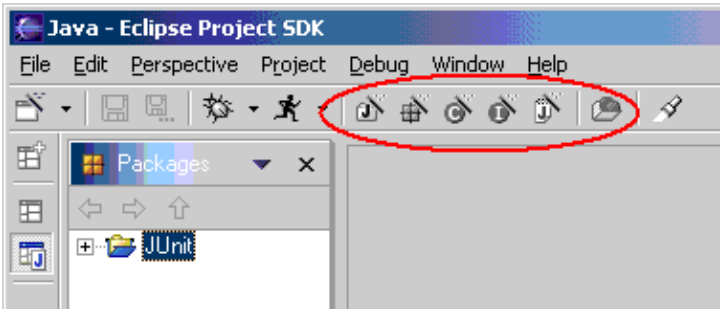
In this tutorial, we will use JUnit as our example project. JUnit is an open source unit testing framework for Java.

Refer to http://www.junit.org for more information.

1. In the main workbench window, select the drop−down menu from the Open New Wizard button and select Project.
2. On the left, select Java, and on the right, select Java Project. Then click Next.

   In the Project name field, type "JUnit", then click Finish. Notice that a new Java perspective opens with the new Java project in the Packages view.

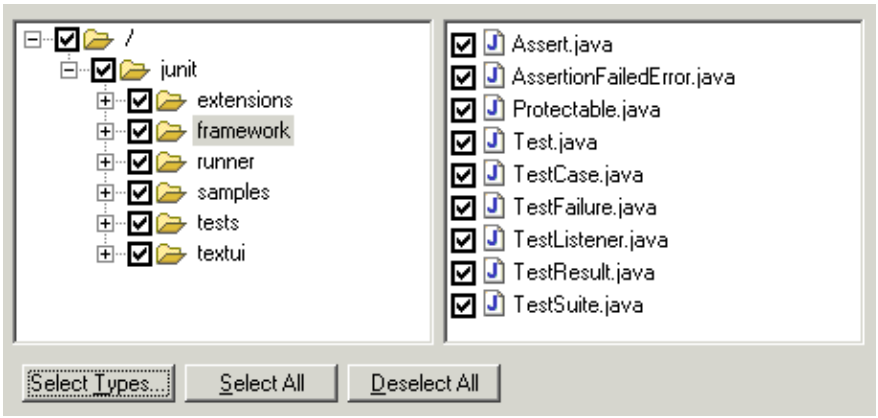
   Notice the new menu options and Java−specific buttons in the workbench toolbar that are loaded along with some of the views in the Java perspective. Depending on which view or editor is active, other buttons and menu options will be available or visible.



3. In the Packages view, make sure that the JUnit project is selected, and from the menu bar, select File > Import.
4. Select Zip file, then click Next.
5. Click the Browse button next to the Zip file field and browse to select <workbenchRoot>/plugins/org.eclipse.jdt.ui.examples.projects/archive/junit/junit37src.jar.
6. Below the import hierarchy list in the Import wizard, click the Select All button.

   You can expand and select elements within the junit directory on the left side to see the individual resources that you are importing displayed on the right side.

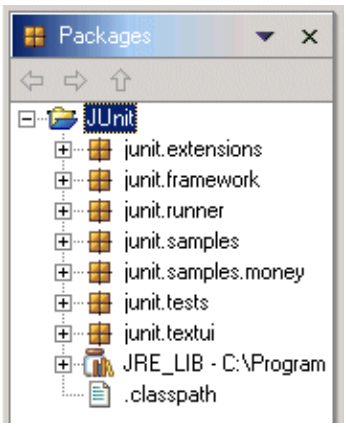
   Note: Do not deselect any of the resources in the junit directory at this time. You will need all of these resources in the tutorial.



7. Make sure that the JUnit project appears in the Where do you want the imported resources to go? field. Then click Finish.

   In the import progress indicator, notice that the imported resources are compiled as they are imported into the workbench. This is because the Perform build automatically on resource modification option is checked on the Workbench preferences page.

8. In the Packages view, expand the JUnit project and notice the JUnit packages.

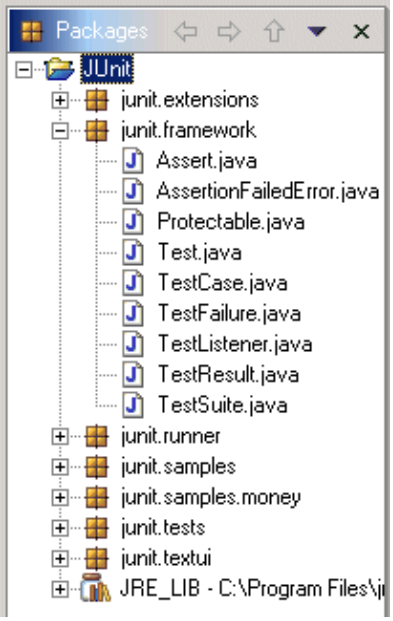


Browsing Packages in the Workbench

In this section, you will browse Java packages in the project.

In the Packages view, expand the JUnit package to inspect the package structure.

Each package contains one or more corresponding JAVA files.



Expanded Packages View

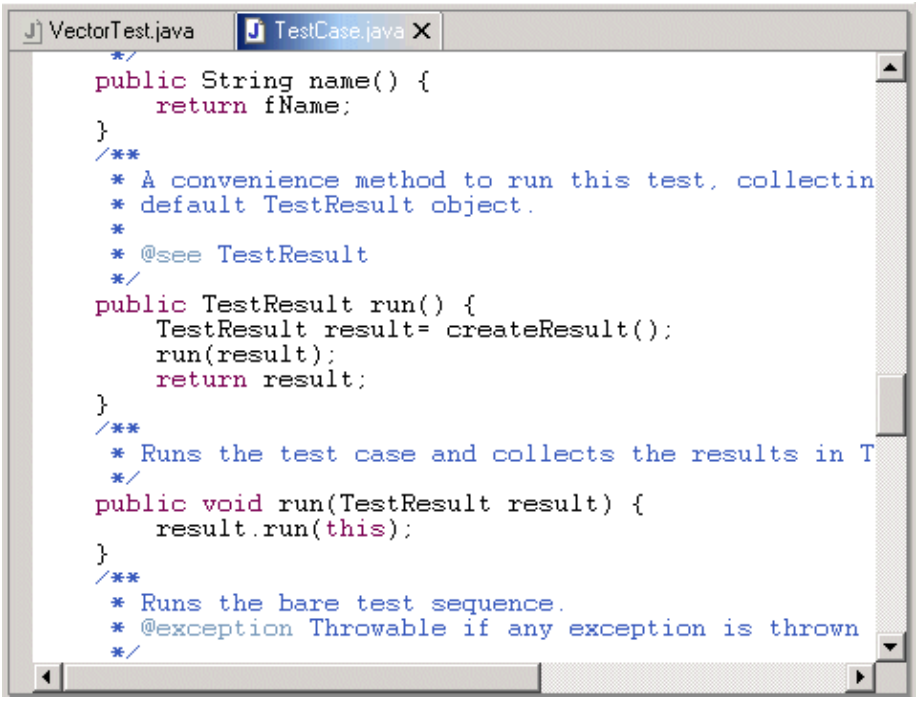
Editing Java Elements

In this section, you will edit Java elements in the workbench.

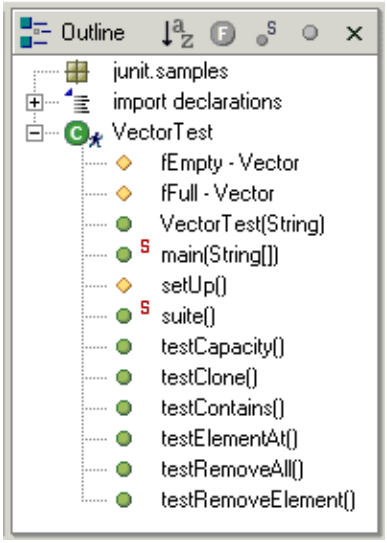
Using the Outline View

1. In the Packages view, find junit.samples.VectorTest.java and double−click it to open this file in the Java editor.
2. In the editor area, notice that this file is shown in the active editor. If it is not the active editor, click its tab to make it so.
3. In the Packages view, open various resources in the editor area and notice the syntax highlighting. For example:

    - Regular comments
    - Javadoc comments
    - Keywords & built−in types
    - Strings

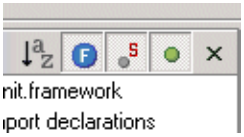


4. Look at the Outline view. Notice how it is populated, displaying a hierarchy outline of the package itself plus import declarations, fields, classes, methods.



5. Notice that the Outline view indicates whether a Java element is abstract, final, etc.

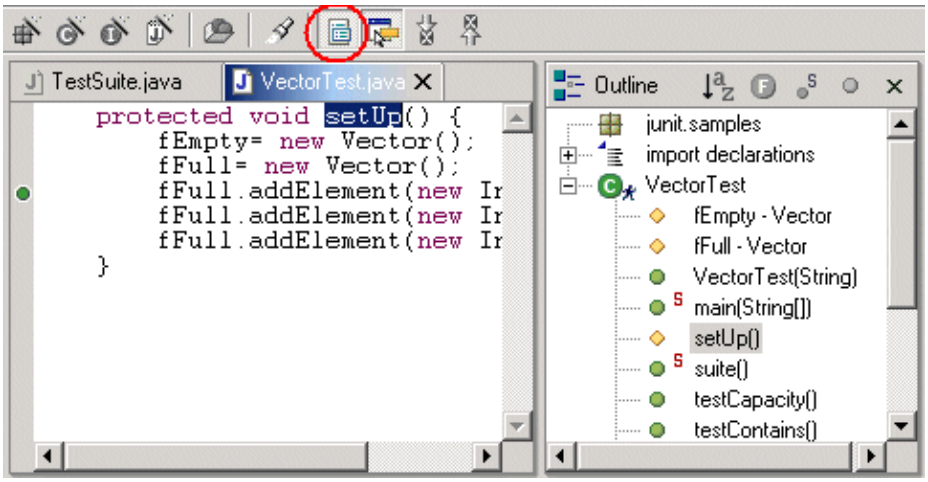
   Toggle the Show/Hide Fields, Show/Hide Non−Public Members, and Show/Hide Static Members buttons in the Outline view toolbar to filter the view's display.



6. Toggle the Sort button in the Outline view to sort the Java elements by either sequential order (as indicated in the compilation unit) or alphabetical order.

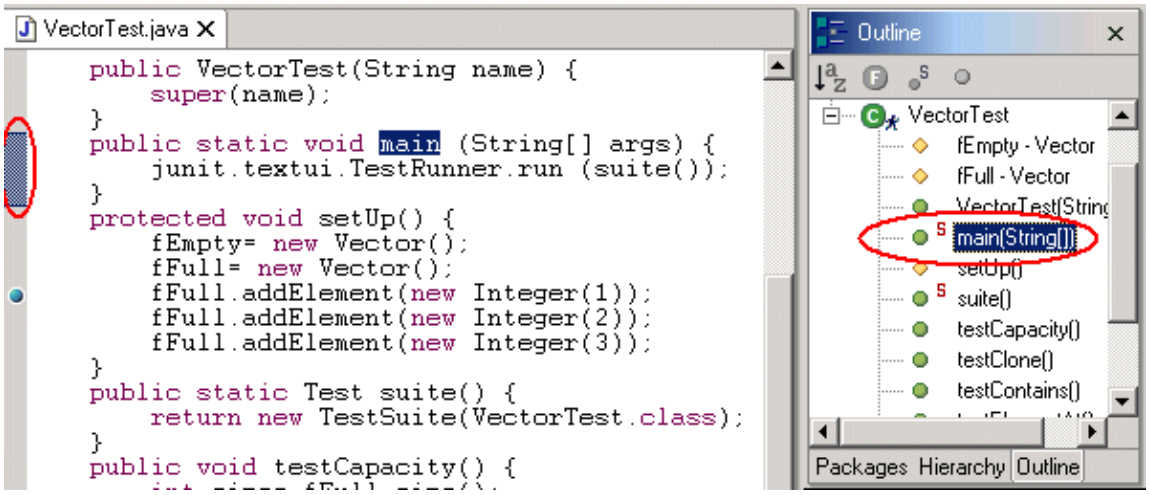
7. You can edit source code by viewing the full source code of the compilation unit, or you can narrow the view to a single element. Click the tab for VectorTest.java, and click the Show Source of Selected Element Only button in the toolbar.

   In the Outline view, select various elements and note that they are displayed individually in a segmented view in the editor.



8. Click in the editor area again and click the same button (Show Complete Source) in the toolbar.

   In the Outline view, select various elements and note that they are once again displayed in a whole file view in the editor. Notice that now, the Outline view selection is marked in the editor with a range indicator in the marker bar.



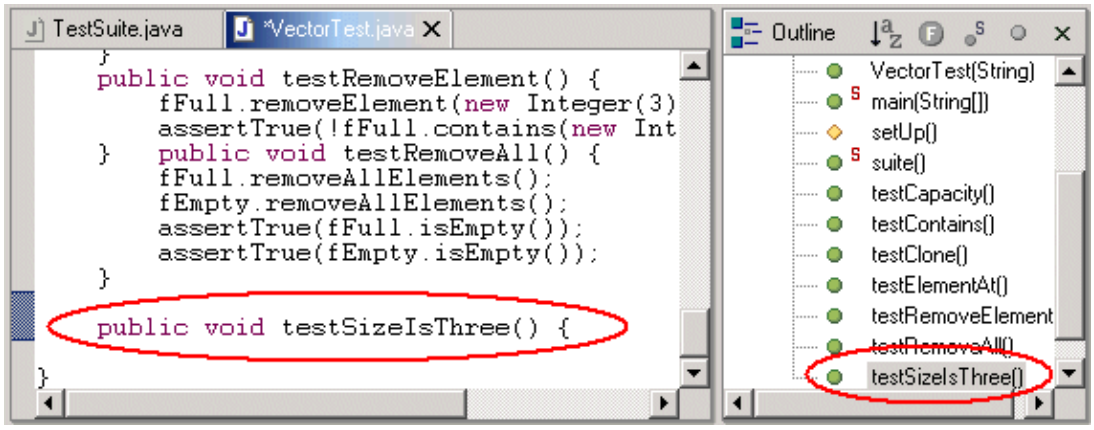
Adding Methods & Using Code Assist

1. Make sure that the Sort button in the toolbar of the Outline view is toggled so that the view is sorted sequentially (instead of alphabetically).

2. In the editor area, type the following at the very end of the VectorTest.java file (but before the closing brackets):

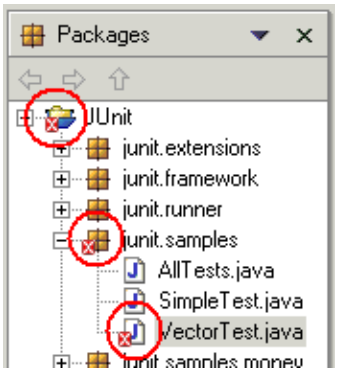
   public void testSizeIsThree (){

   Notice that as soon as you type the method name in the editor area, the new method appears in the Outline view, at the very bottom (since the view is sorted sequentially).



3. Click the Save button.

   Notice that because your workbench builds automatically, errors show up in the Packages view, the Tasks view, and the editor marker bar. Also notice that in the Packages view, the errors propagate up to the project of the compilation unit containing the error.



4. Continue adding the new method by typing the following:

   assertTrue(fFull.size() == 3); }

5. Click the Save button. Notice that the errors disappear.

Deleting & Replacing a Method from the Local History

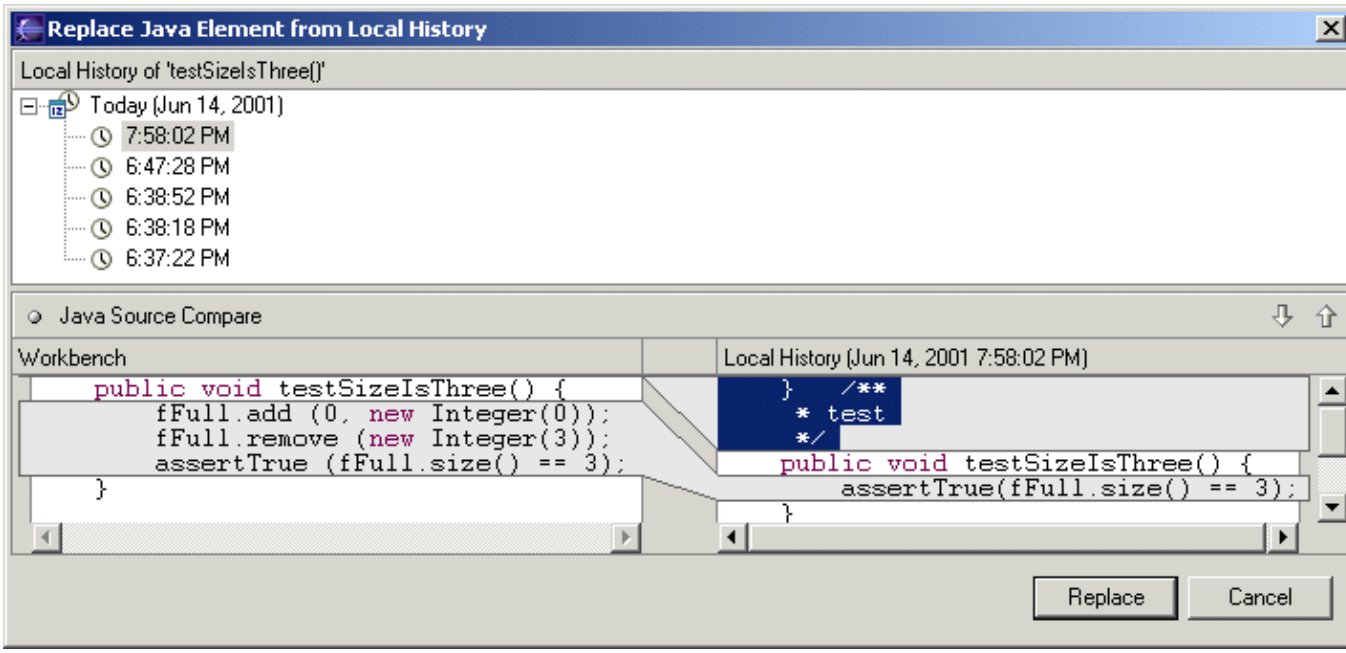
In this section, you will use local history support to easily switch to a previously−saved version of an individual Java element.

1. In the Outline view, select the testSizeIsThree() method that you just created and from its context menu, select Delete.
2. In the editor, at the very end of the VectorTest.java file, add a new testSizeIsThree() method:

   public void testSizeIsThree() { fFull.add(0, new Integer(0)); fFull.remove(new Integer(3)); assertTrue(fFull.size() == 3); }

   Click Save when you are done.

3. In the Outline view, select the testSizeIsThree() method, and from its context menu, select Replace from Local History.
4. In the Replace Java Element from Local History dialog, the Local History list shows the various saved states of that element, and the Java Source Compare pane shows details of the differences between the selected history resource and the existing workbench resource.



5. In the Local History pane, select the version that you deleted, then click Replace.
6. The code in the editor is replaced with the history version.

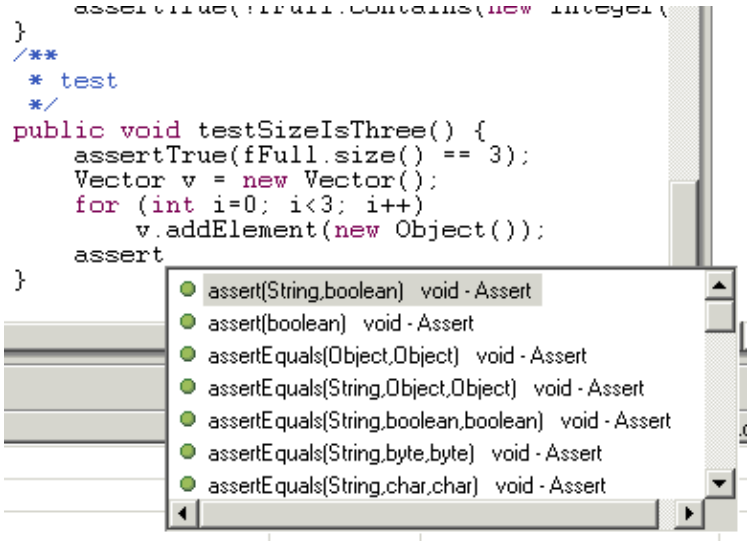
   Click the Save button.

Using Content Assist

1. Double−click junit.samples.VectorTest.java to open it in an editor.
2. In the Outline view, select the testSizeIsThree() method to navigate in the editor to the code for that method.
3. In the editor, add the following lines to the end of the method:

   Vector v = new Vector(); for (int i=0; i<3; i++) v.addElement(new Object()); assert

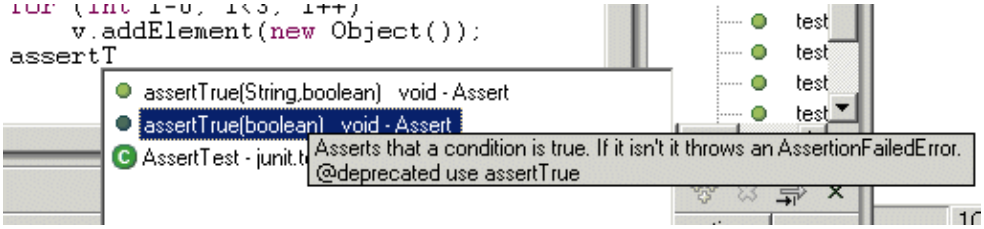
4. With your cursor at the end of the word assert, press Ctrl+Space to activate code assist.



5. Scroll down a bit in the list to see the available choices.

   With the code assist window still active, type the letter t after assert (with no space between) to narrow the list.

6. Select and then hover over various items in the list to view any available Javadoc help for each item.



   Note: You must first select the item, to view the hover help.

7. Select the assertTrue(boolean) option from the lost and press Enter.

8. After the code is inserted, complete the line so that it reads as follows:

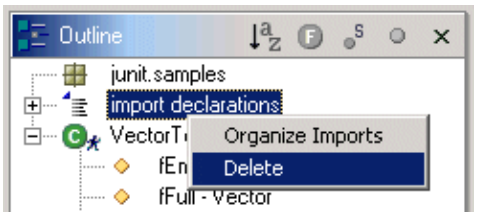
   assertTrue(v.size() == fFull.size());

9. Click Save when you are done.

Using Smart Import Assistance

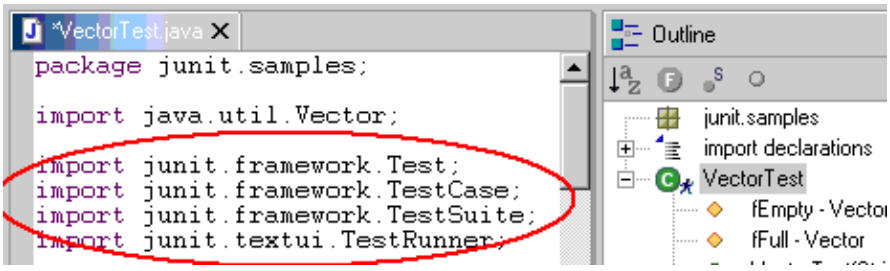
1. If it is not already open, double−click junit.samples.VectorTest.java to open it in an editor.

2. In the Outline view, select the import statements, and from their context menu, select Delete.



3. From the context menu in the editor, select Organize Imports.

4. The required import statements are added to the beginning of your code below the package declaration.



   Note: You can control the order of the import statement in the preferences pages (Window > Preferences > Java > Import Order).

5. Click Save when you are done.

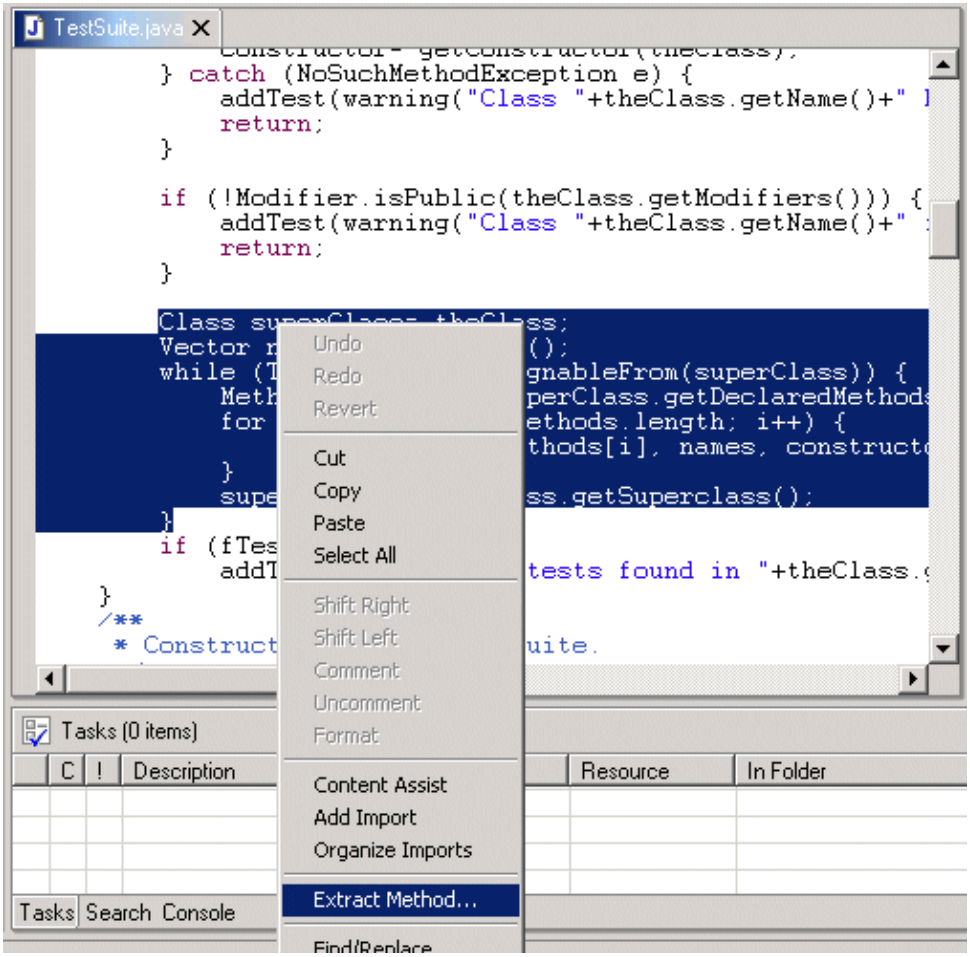
Extract a Method

In this section, you will improve the code of the constructor of junit.framework.TestSuite. To make the intent of the code more clear, you will extract the code that collects test cases from base classes into a new method called collectTestMethods.

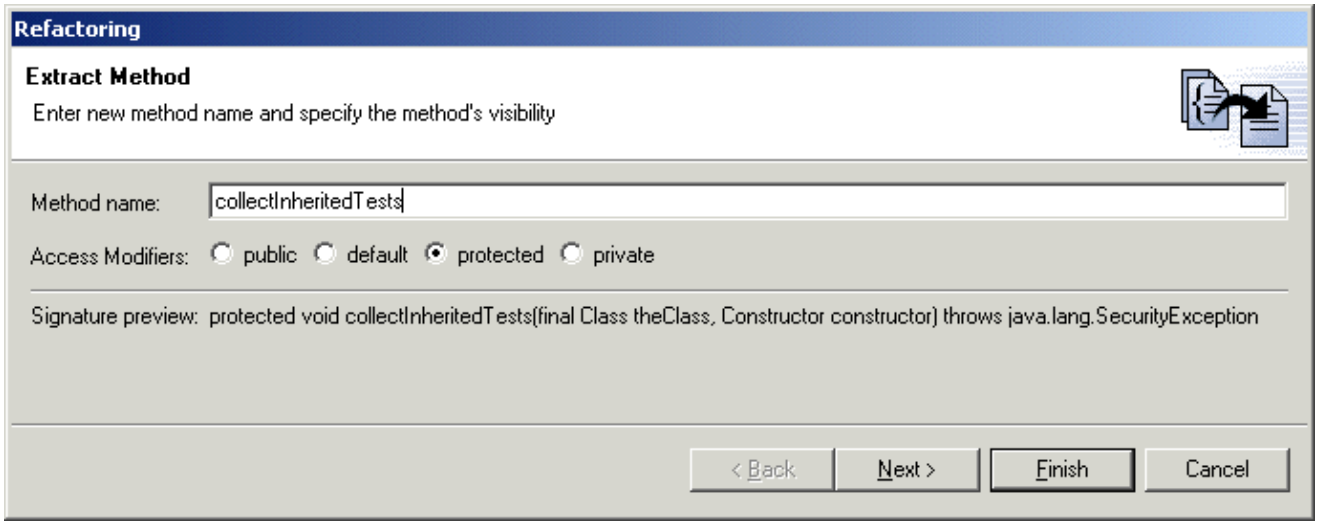
1. In the junit.framework.TestSuite.java file, select the following range of code:

   Class superClass= theClass; Vector names= new Vector(); while (Test.class.isAssignableFrom(superClass)) { Method[] methods= superClass.getDeclaredMethods(); for (int i= 0; i < methods.length; i++) { addTestMethod(methods[i], names, constructor); } superClass= superClass.getSuperclass(); }

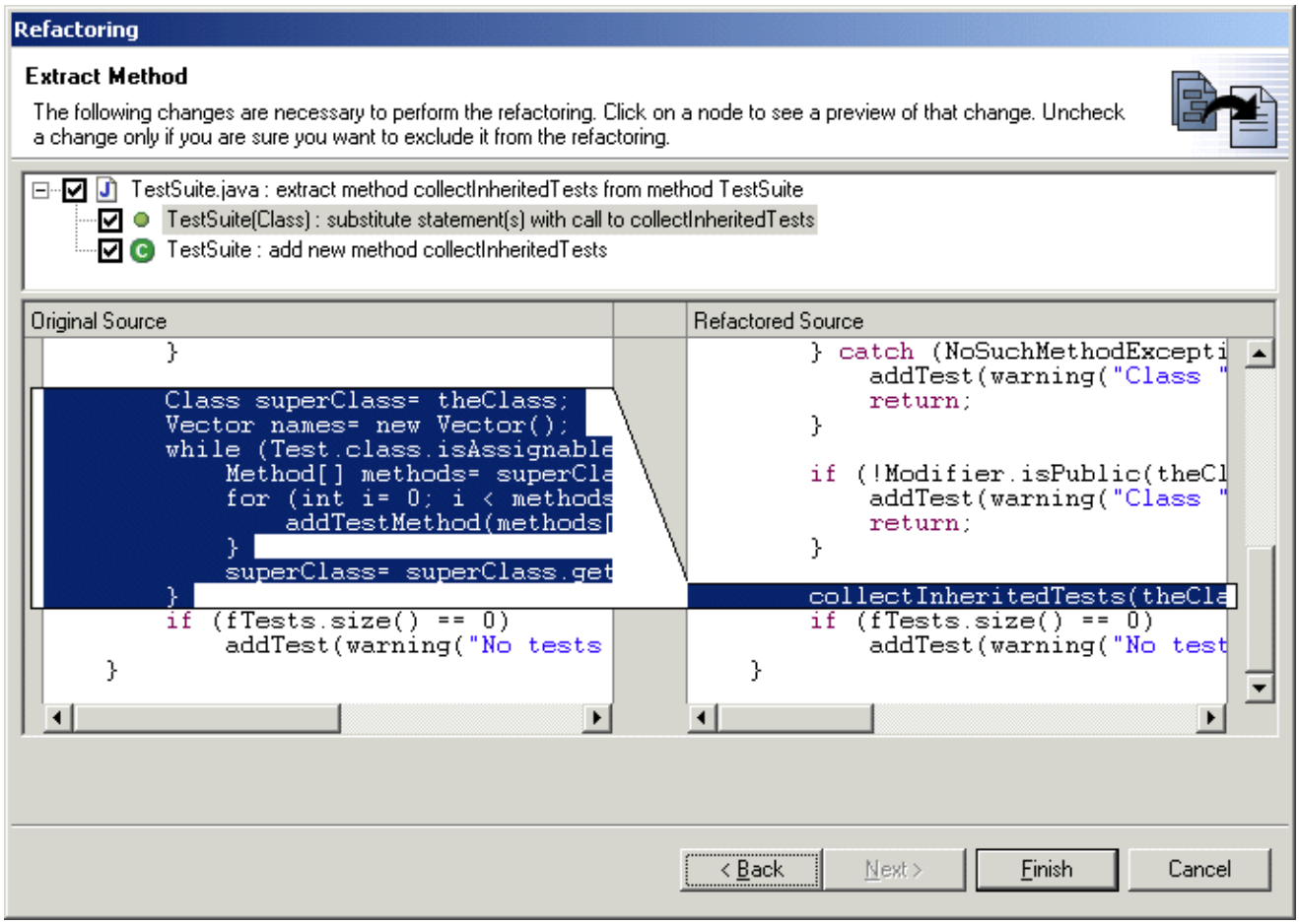
2. From the selection's context menu in the editor, select Extract Method.



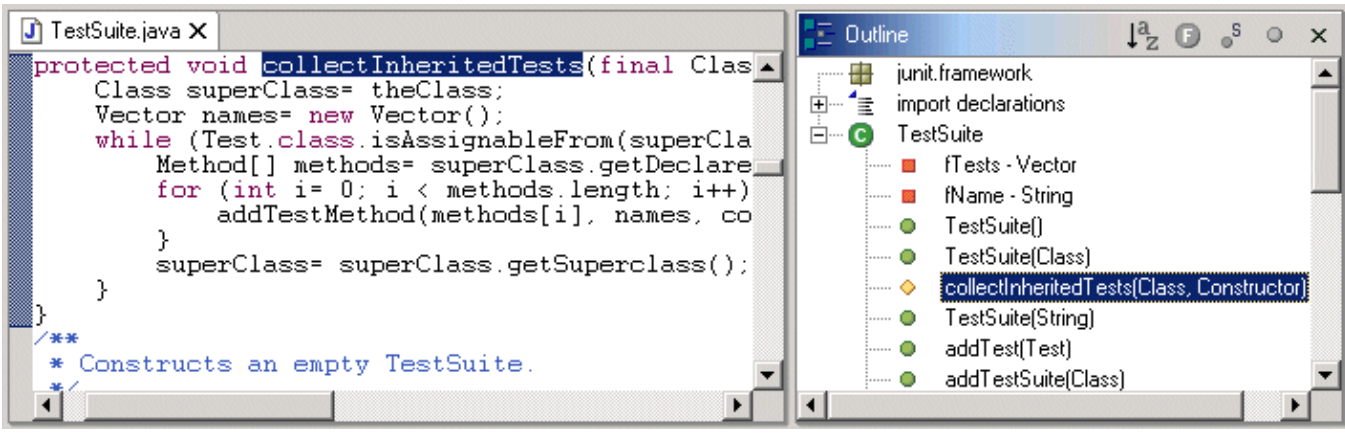
3. In the Method Name field, type collectInheritedTests, then click Next.



4. The refactoring preview page displays the changes that will be made.



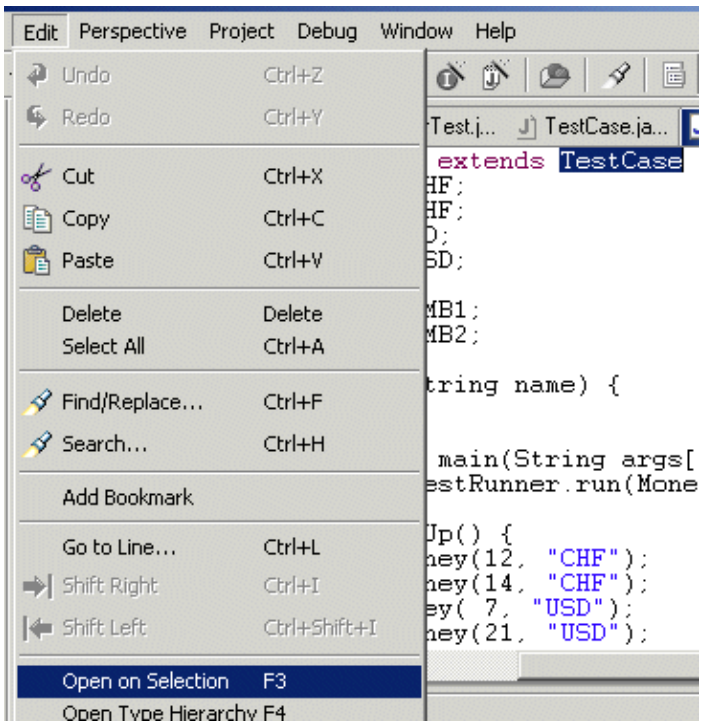
5. The method is extracted. Select it in the Outline view to navigate to it in the editor.



Using Open on Selection & Open on Type Hierarchy

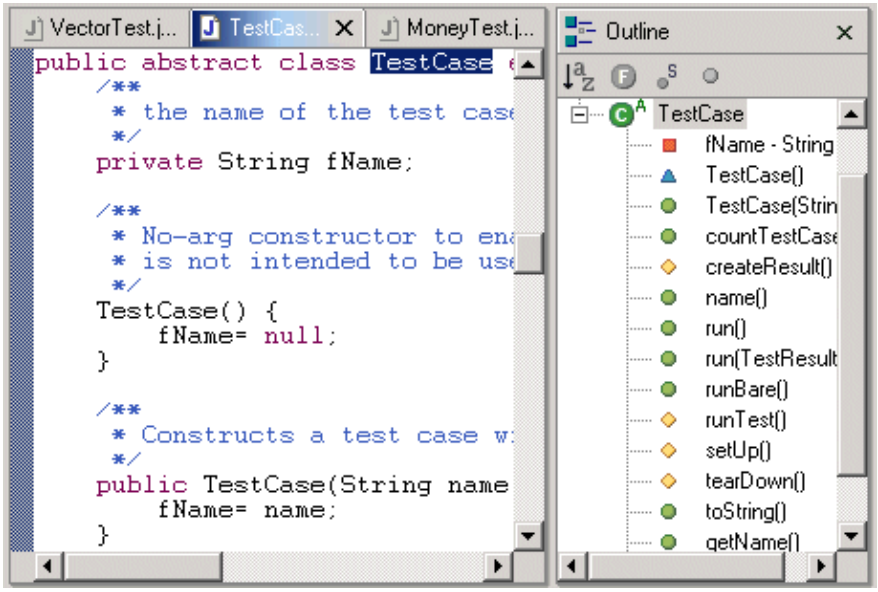
1. In the Packages view, double−click junit.samples.money.MoneyTest.java to open it in an editor.
2. In first line of the MoneyTest class declaration in the editor, select the TestCase superclass specification and either

   - select from the menu bar Edit > Open on Selection or
   - press F3.



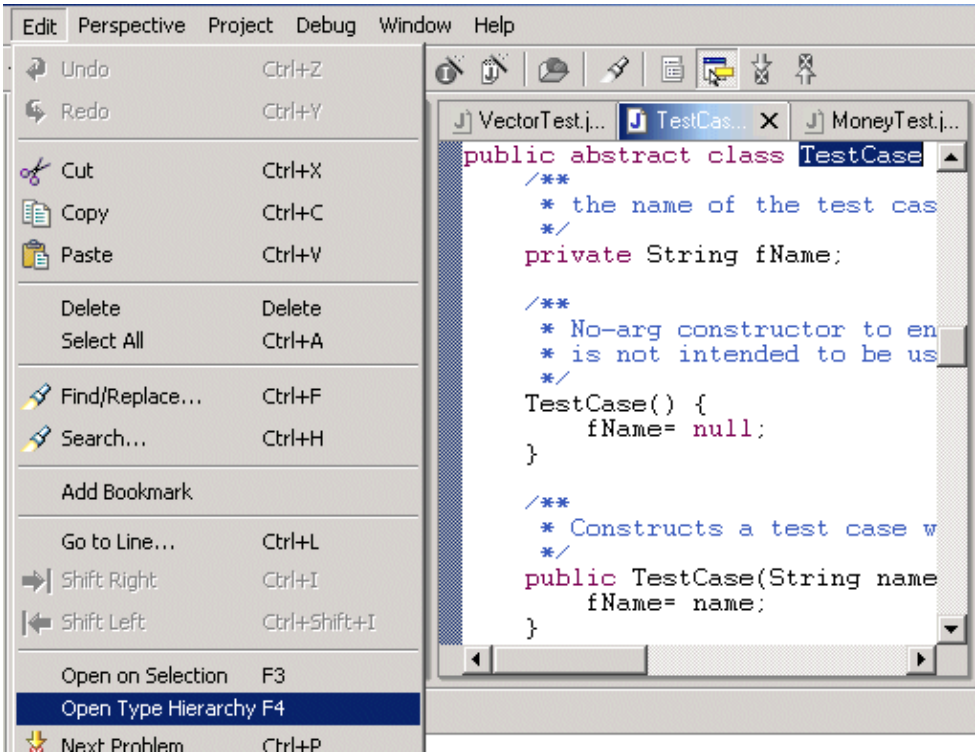
Note: This command also works on methods and fields.

3. The TestCase superclass opens in the editor area and is also represented in the Outline view.

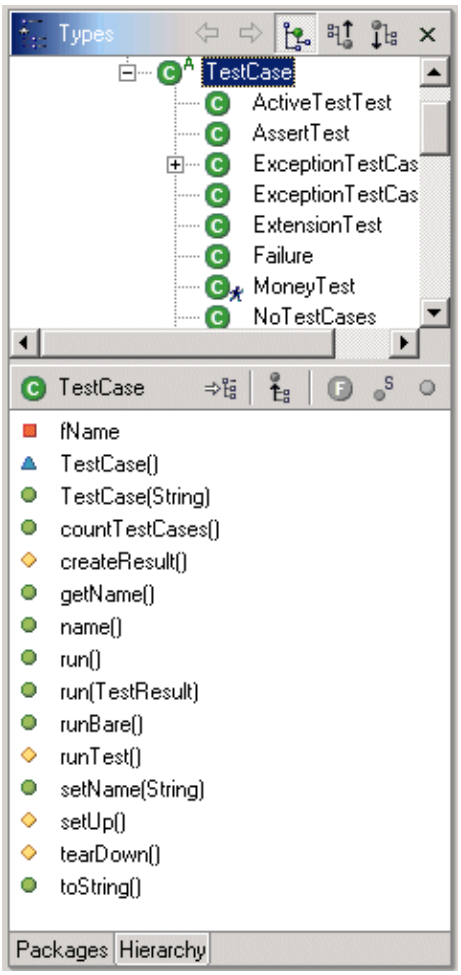


4. Click the TestCase.java editor tab to make it the active editor. Make sure that the class declaration is still selected, and:

   - select from the menu bar Edit > Open Type Hierarchy or
   - press F4.



5. The Hierarchy view opens with the TestCase class displayed.

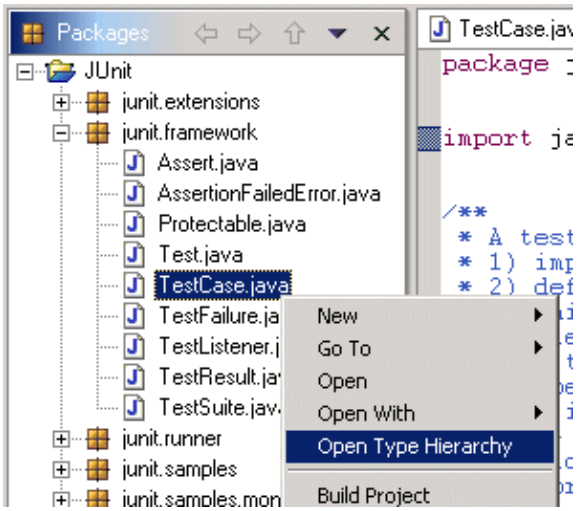


You can also open editors on the types and methods in the Hierarchy view.

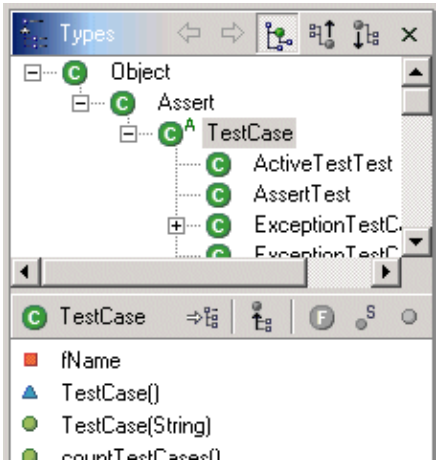
Viewing the Type Hierarchy

In this section, you will learn about using the Hierarchy view.

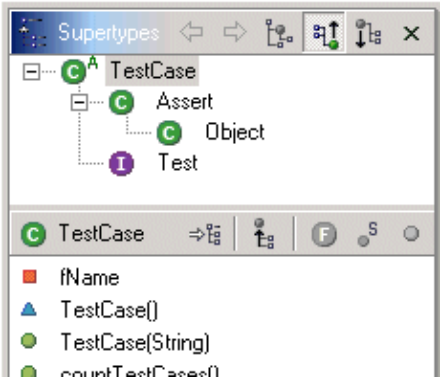
1. In the Packages view, find junit.framework.TestCase.java and from its context menu, select Open Type Hierarchy (or select from the menu bar Edit > Open Type Hierarchy).



2. In the Hierarchy view, click the Show the Type Hierarchy button to see the class hierarchy, including the base classes and subclasses.

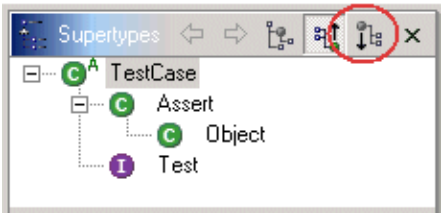


3. In the Hierarchy view, click the Show the Supertype Hierarchy view button to see a hierarchy showing the type's parent elements including implemented interfaces (i.e., the results of going up the type hierarchy).



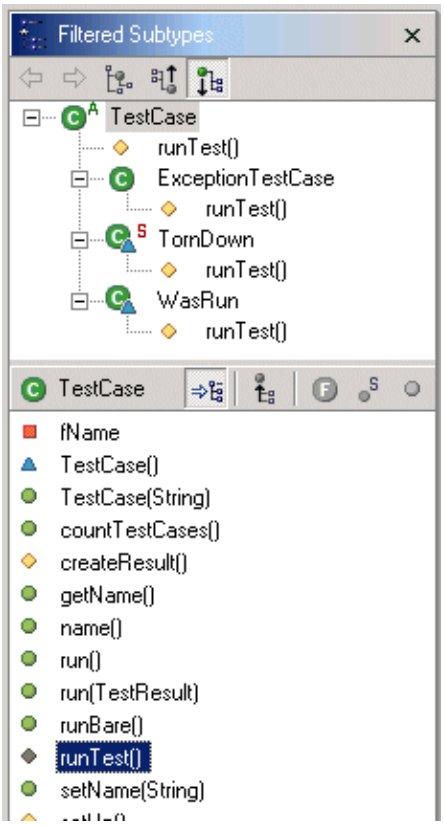
Notice that in this "upside−down" display, you can now see that TestCase implements the Test interface.

4. Click the Show the Subtype Hierarchy button in the view toolbar.

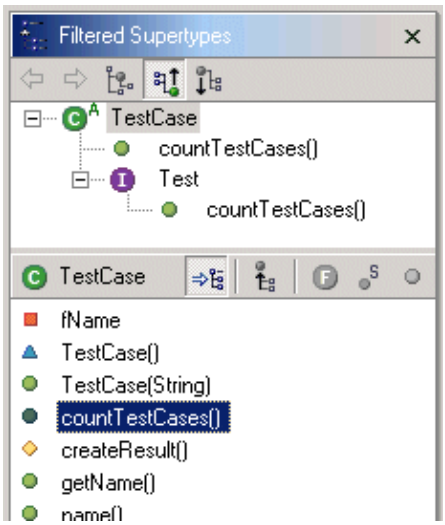


5. Click the Lock View button in the method pane (the bottom pane) in the Hierarchy view, then select the runTest() method in the method pane.

Notice that the view now shows all the types implementing runTest().



6. In the Hierarchy view, click the Show the Supertype Hierarchy view button. Then in the list pane, select countTestCases() to display the places where this method is overridden.



7. In the Hierarchy view, select the Test element, and select Open Type Hierarchy from its context menu.

   The resource containing the selected element is represented in the Hierarchy view and also opens in an active editor.

8. Select from the menu bar Window > Preferences > Java.

9. In the Open a new type hierarchy inside area, click the Hierarchy Perspective. Then click OK.

10. In the Hierarchy view, select the Test element again, and once again select Open Type Hierarchy from its context menu.

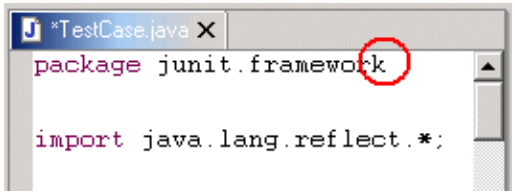
    The resource containing the selected type is shown in a new perspective (the Hierarchy perspective), and its source is shown in the Java editor. If the preference is set to this option for viewing new type hierarchies, you can have more than one type hierarchy in your workbench and switch between them as needed.

    Note: You can also choose to open the Hierarchy perspective in a new window by using the hot key specified on the Window > Preferences > Workbench page for opening perspectives in new windows.

Building & Fixing Problems in Your Code

In this section, you will build Java projects and fix problems.

1. Verify that there are currently no problems in the workbench by looking in the Tasks view, where build problems are listed if they exist.
2. If it is not already open, then in the Packages view, double−click junit.framework.TestCase.java to open it in an editor.
3. Add a syntax error by deleting the semicolon at the end of the first line in the file (package junit.framework;).

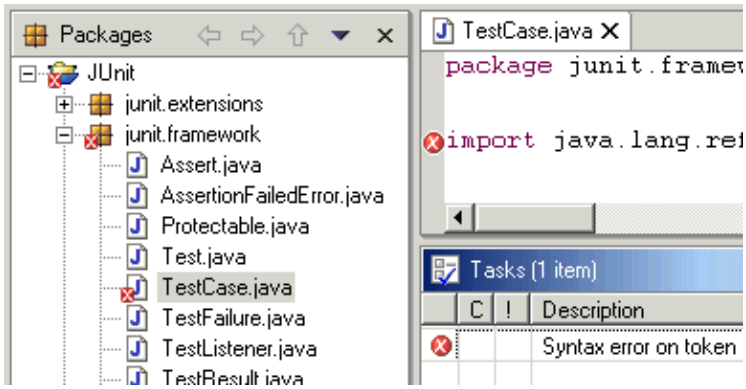


4. Click the Save button when you are done. The project is incrementally built, and the problem is discovered.

   Note: Only the changed file and the file depending on it are recompiled.

5. The problem is indicated as follows:

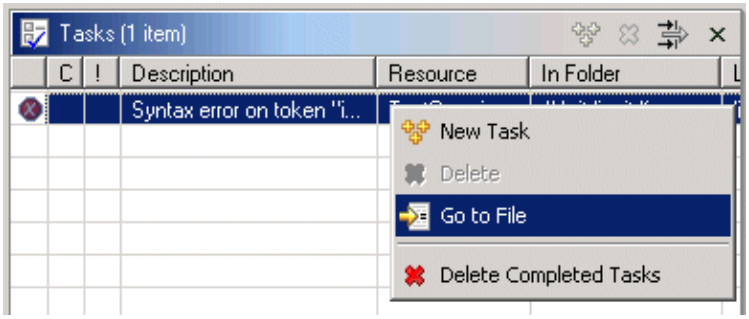
   - In the Tasks view, the problems are listed.
   - In the Packages view, problem ticks show up on affected Java elements and their parent elements.
   - In the editor, a problem marker is displayed near the affected line.



6. Hover over the problem marker in the marker bar in the editor area and view the information in the hover help.

7. Click the Close button on the editor's tab to close the editor.

8. In the Tasks view, select the problem in the list, and from its context menu, select Go to File. The file where the problem is detected opens in the editor area, where the problem is shown.



9. Correct the problem in the editor by retyping the semicolon, then click the Save button.

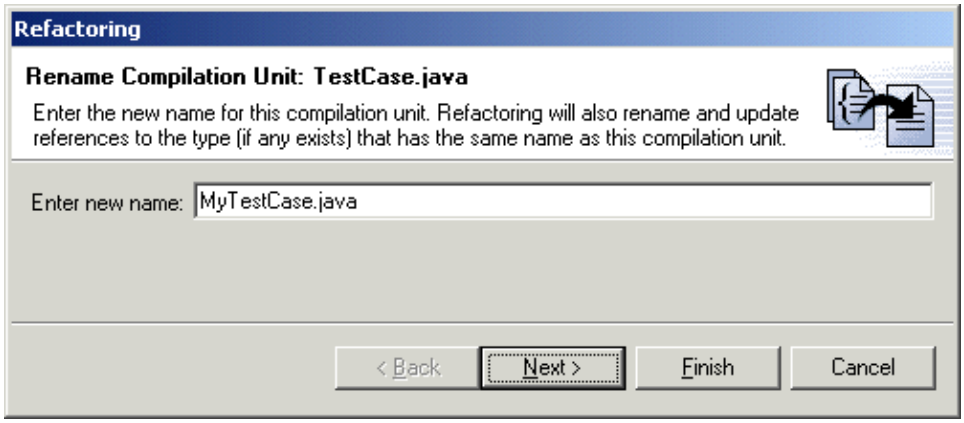
   The project is built automatically on the save, and all the problem indicators in the workbench disappear.

Renaming Java Elements

In this section, you will rename a Java element using refactoring.

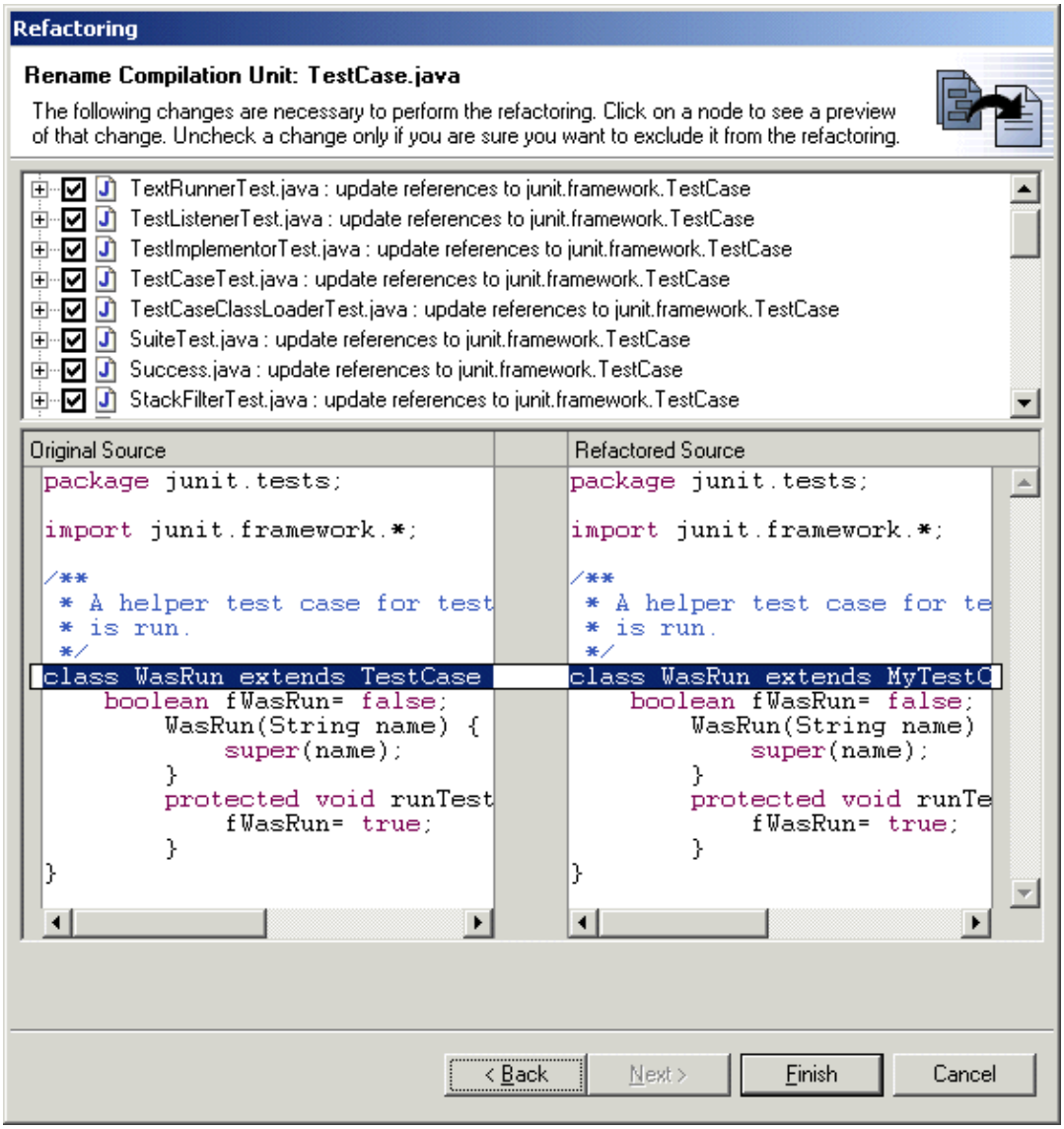
1. In the Packages view, select junit.framework.TestCase.java.
2. From its context menu, select Rename.
3. Renaming is a refactoring action. Refactoring supports changing the structure of your code without changing its semantic behavior.

   In the Enter New Name field in the Rename Compilation Unit dialog, type " MyTestCase.java ", then click Next.



4. The workbench analyzes the proposed change and presents you with a preview of the changes that would take place if you actually choose to rename this resource.

   Renaming a compilation unit invalidates import statements in other compilation units; therefore, a simple rename operation is not sufficient in this case and would result in compile errors.

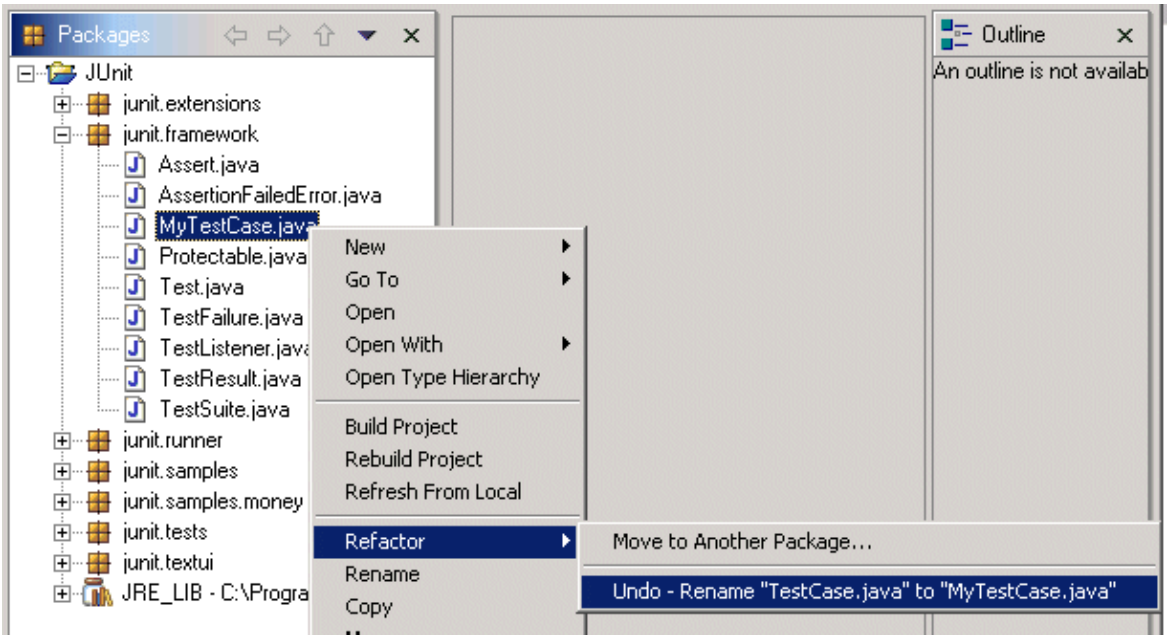


5. In the Refactoring preview dialog, you can scroll through the various proposed changes and select or deselect changes, if necessary.

   Note: Typically, you will accept all the proposed changes.

6. Click Finish to accept all proposed changes.

7. In the Packages view, select the newly−renamed MyTestCase.java file, and from its context menu, select Refactor > Undo Rename TestCase.java to MyTestCase.java.



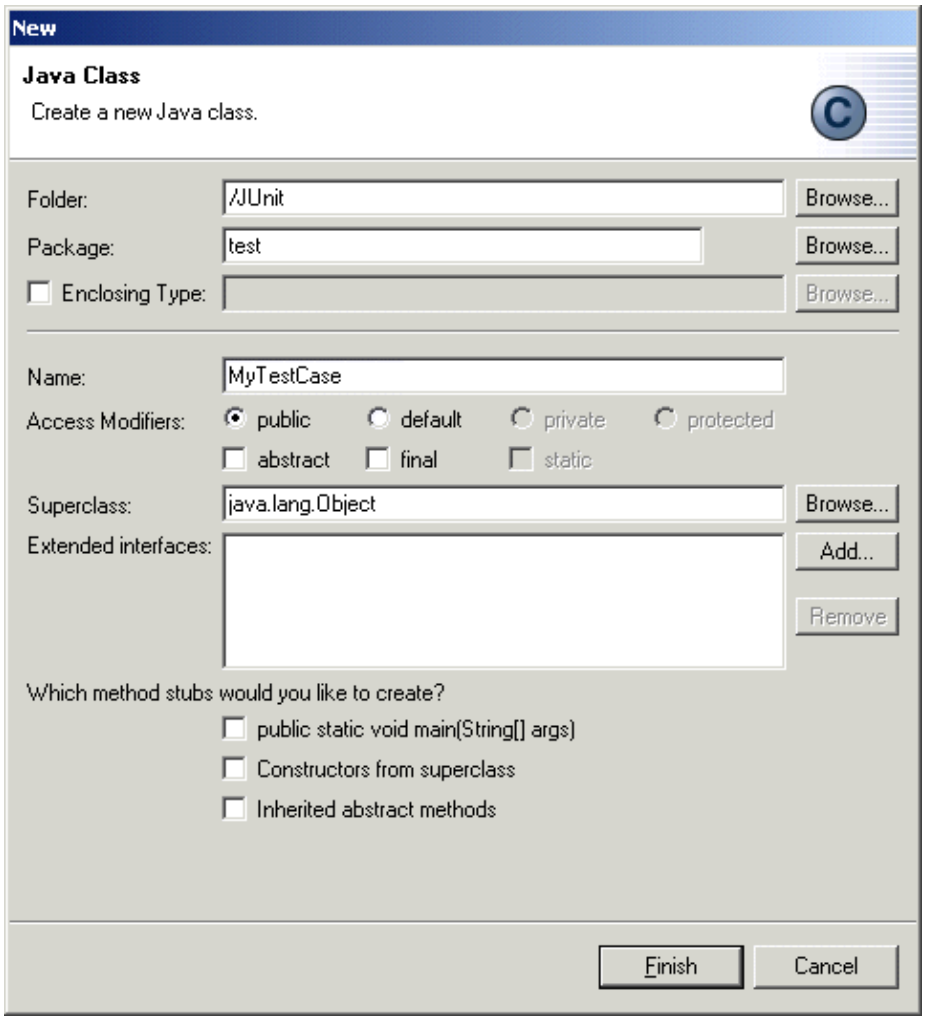
8. The refactoring changes are undone, and the workbench returns to its previous state. You can undo refactoring actions right up until you change and save a compilation unit, at which time the refactoring undo stack is cleared.

Creating a Java Class

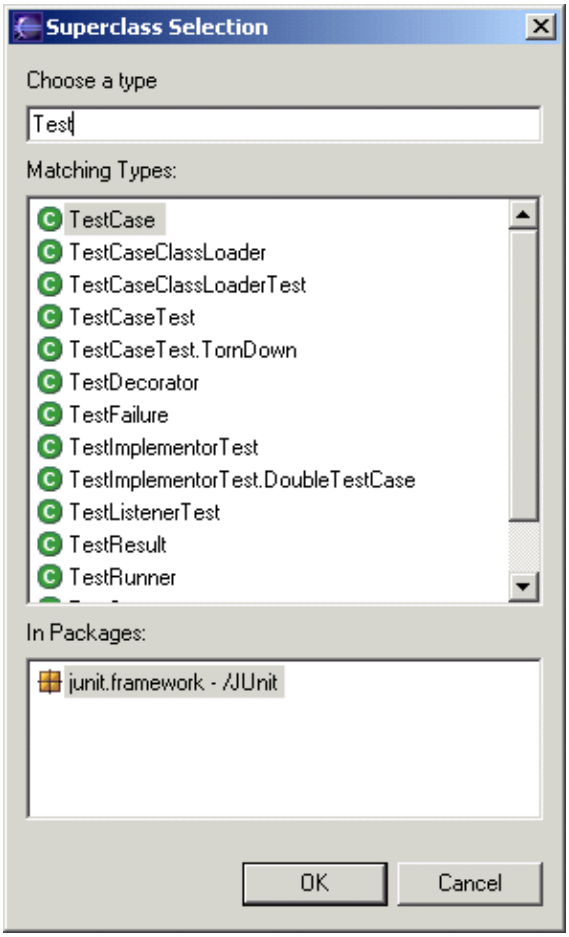
In this section, you will create a new Java CLASS file.

1. In the Packages view, select the JUnit project. From the project's context menu, select New > Package.
2. In the Package field, type test as the name for the new package. Then click Finish.
3. In the Packages view, select the new test package and click the New Class button.
4. Make sure that /JUnit appears in the Folder field and that test appears in the Package field.

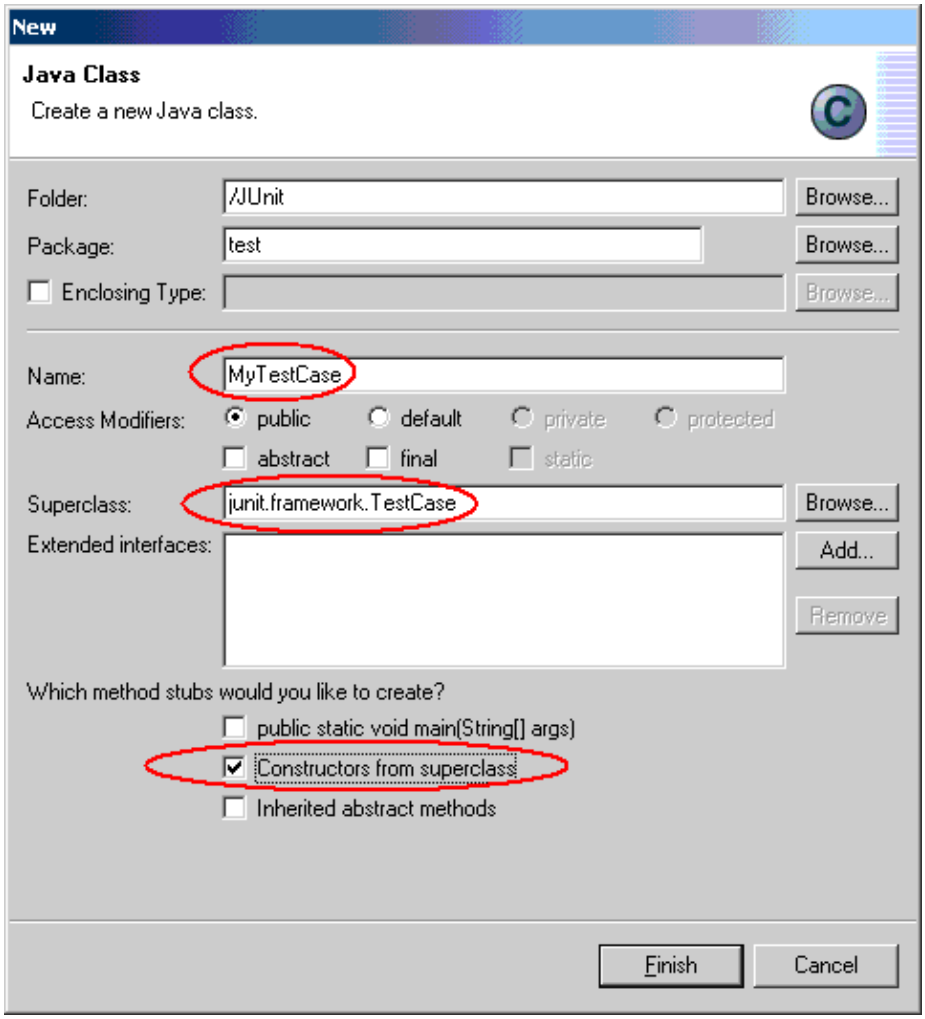
   In the Name field, type MyTestCase.



5. Click the Browse button next to the Superclass field.
6. In the Choose a type field in the Superclass Selection dialog, type Test to narrow the list of available superclasses for the new class.



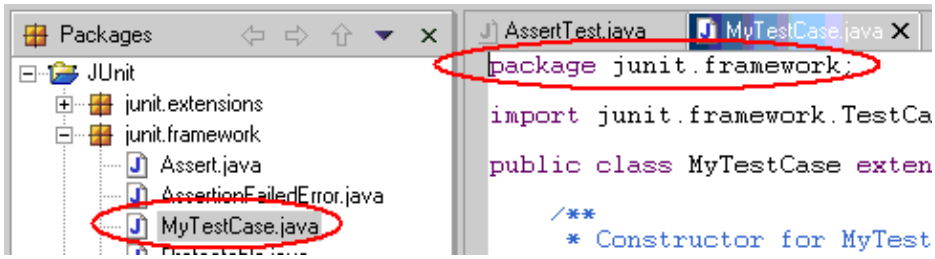
7. Select the TestCase class and click OK.
8. Check the Constructors from Superclass checkbox.
9. Click Finish to create the new class.



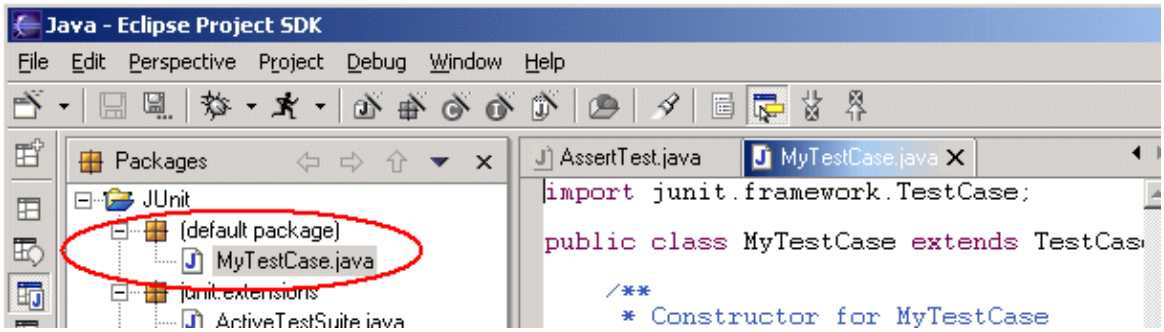
Moving & Copying Java Elements

In this section, you will create a new package and classes and move resources between Java packages.

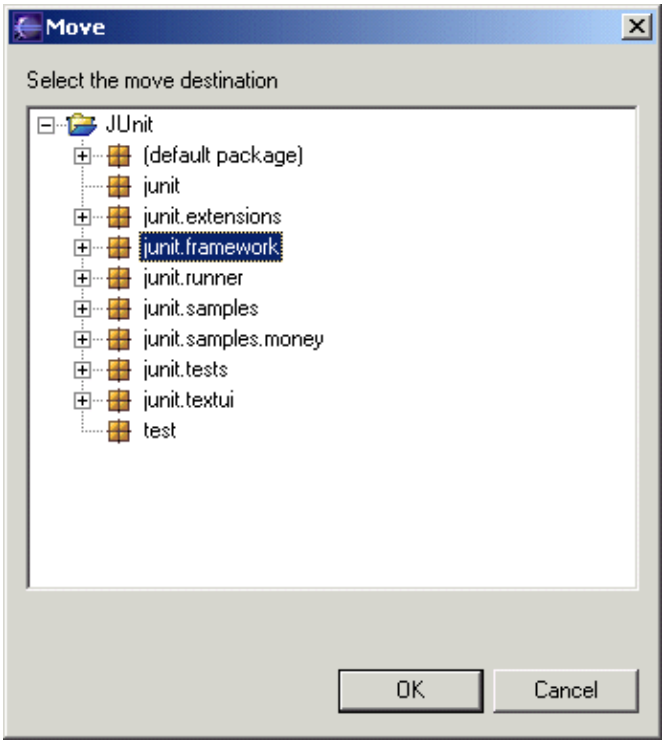
1. In the Packages view, select the MyTestCase.java file and drag it into to the junit.framework package. The class is moved, and its declaration changes to reflect its new location.



2. Select the MyTestCase.java file and drag it into to the root directory of the JUnit project. A default package is created to contain the class, and the package declaration is removed to reflect its new location.



3. Select the MyTestCase.java file and from its context menu, select Move. In the Move dialog, expand the hierarchy to browse the possible new locations for the resource.



Using the context menu option is simply an alternative to dragging and dropping.

Select the junit.framework package, then click OK. The class is moved, and the package declaration changes to reflect its new location.

Note: This example did not result in compile errors. However, moving a compilation unit can lead to compile errors because import statements are not updated with a compilation unit is moved.

Searching the Workbench

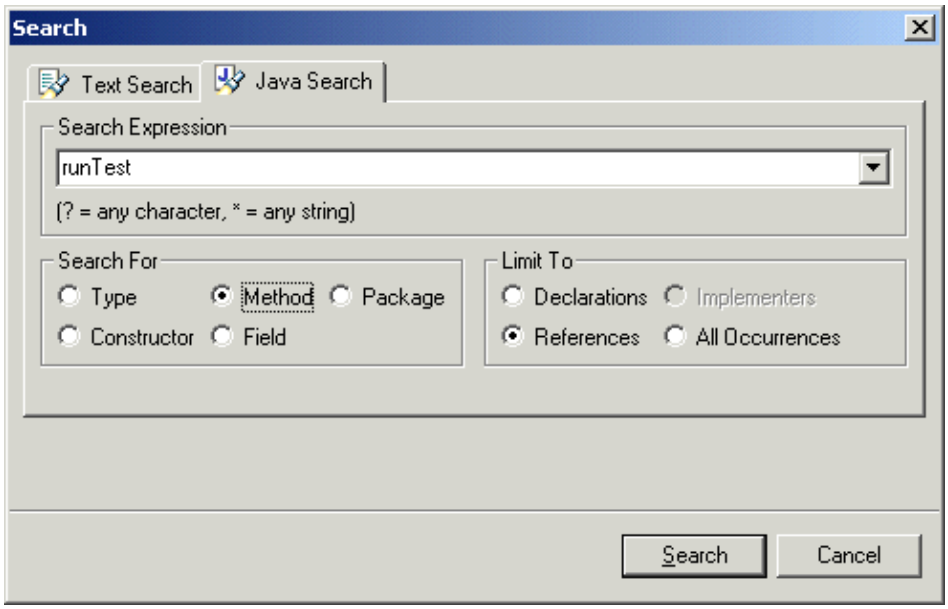
In this section, you will search the workbench for Java elements.

In the Search dialog, you can perform text searches and Java searches.

- Java searching is more precise and faster than text searching.
- Java searches are indexed.
- Text searches are less precise and operate on simple pattern matching.
- Text searches can find matches inside comments, for example.

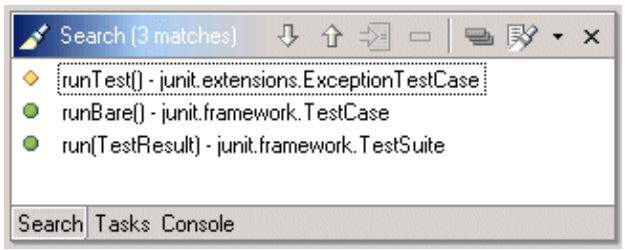
Performing a JDT Search

1. In the Java perspective, click the Search button in the workbench toolbar.
2. Select the Java Search tab (if it is not already selected).
3. In the Search Expression field, type runTest . In the Search For area, select Method, and in the Limit To area, select References.



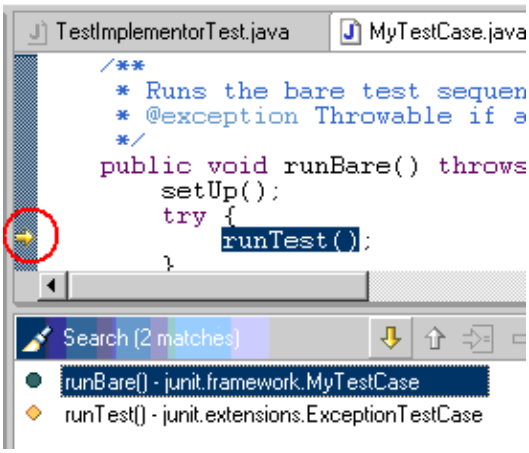
Then click Search. Notice that in the search progress dialog, you can click Cancel at any point while the search is being conducted to stop the search.

4. In the Java perspective, the Search view displays.



Use the Show Next Match and Show Previous Match buttons to view each match. If the file in which the match was found is not currently open, it is opened in an editor.

5. Note that when you navigate to a search match using the Search view buttons, the file opens in the editor at the position of the search match, which is tagged with a search marker in the marker bar.

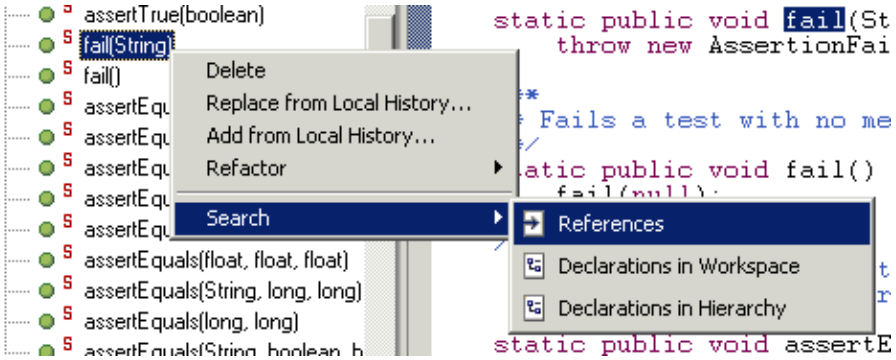


Notice that the search results list displays the method containing the match.

From a Java View

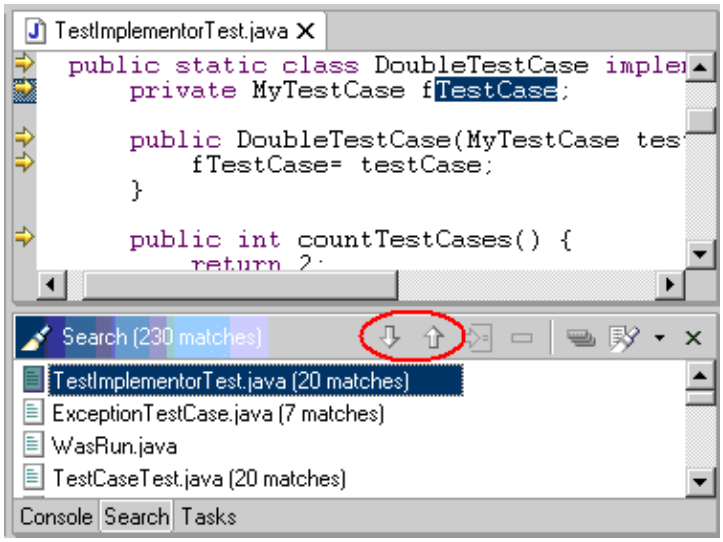
You can conduct a Java search from many views, such as the Outline, Hierarchy, or Packages view.

1. In the Packages view, double−click junit.framework.Assert.java to open it in an editor.
2. In the Outline view, select the fail(String) method, and from its context menu, select Search > References.



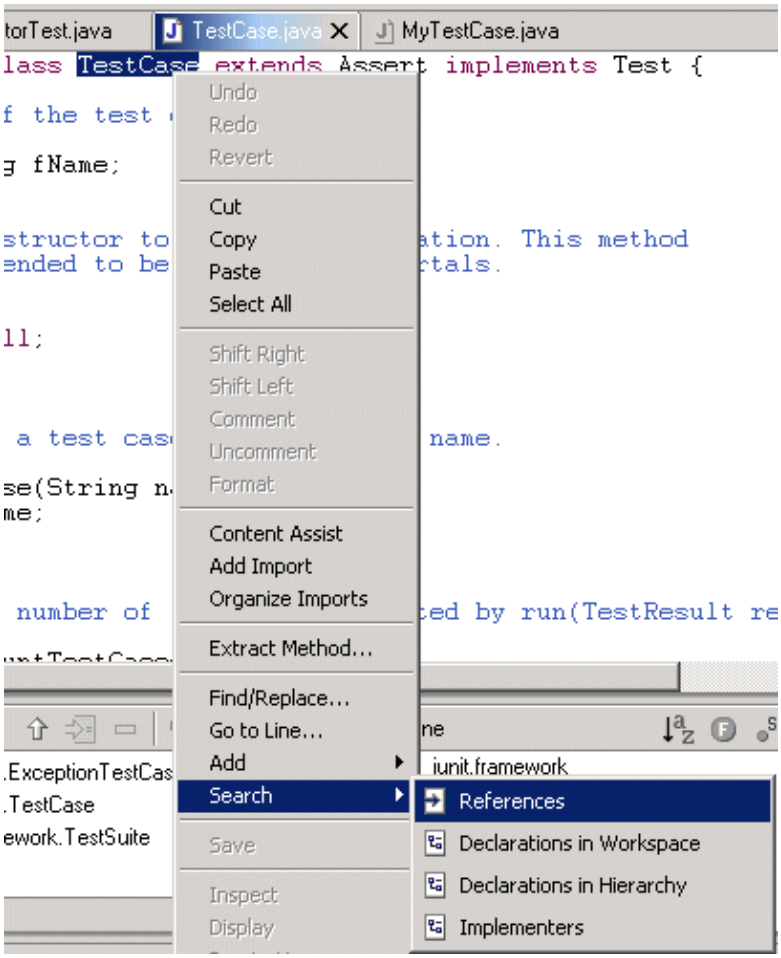
3. Notice that in the search progress dialog, you can click Cancel at any point while the search is being conducted to stop the search.
4. In the Java perspective, the Search Results view displays.

   Use the Show Next Match and Show Previous Match buttons to view each match. If the file in which the match was found is not currently open, it is opened automatically in an editor.



From an Editor

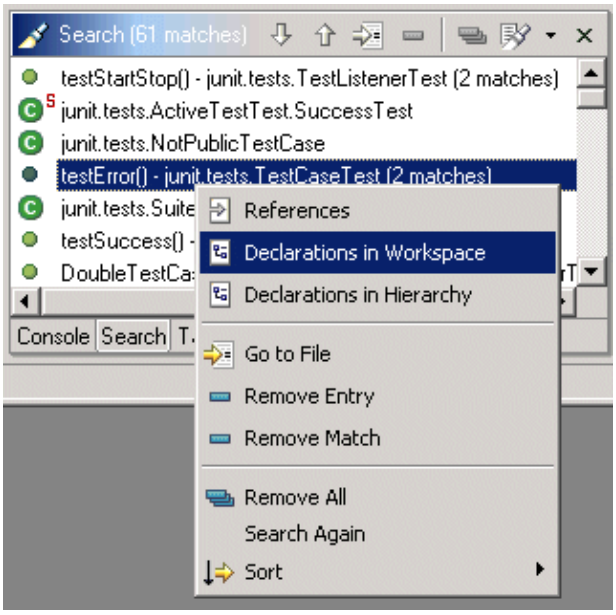
In the Packages view, double−click junit.framework.TestCase.java to open it in an editor. In the editor, select the word TestCase (in its initial declaration line, at the top of the file), and from its context menu, select Search > References.



Searching from an Editor

From the Search View

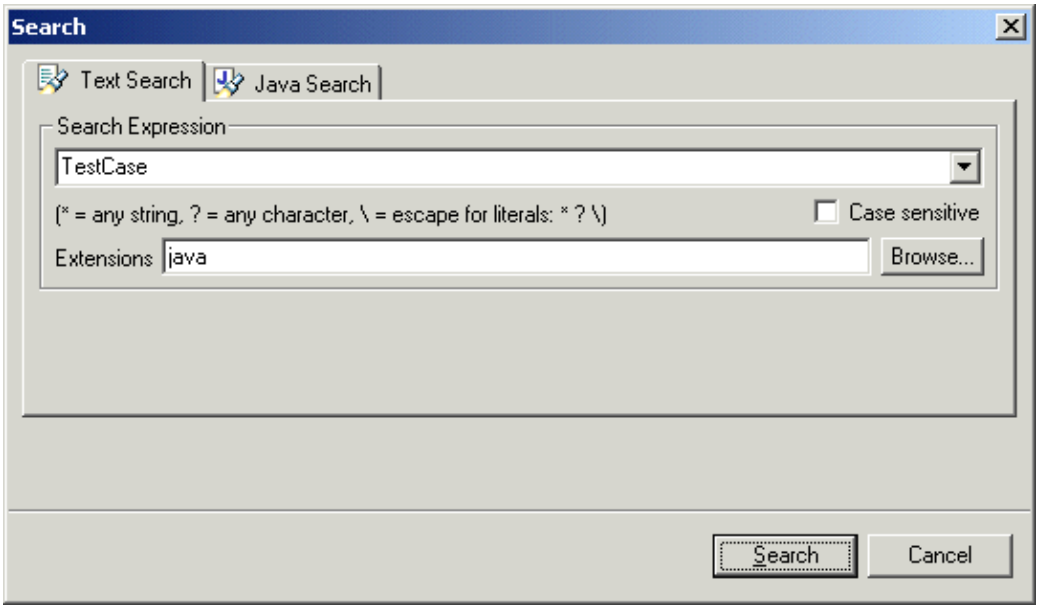
In the Search Results view showing the results for the TestCase search, select any search result. From the context menu of this result, select either References, Declarations in Workspace, or Declarations in Hierarchy.



Searching from the Search View

Performing a Text Search

1. In the Java perspective, click the Search button in the workbench toolbar.
2. Select the Text Search tab (if it is not already selected).
3. In the Search Expression field, type TestCase. In the Extensions field, make sure that java is the only extension.

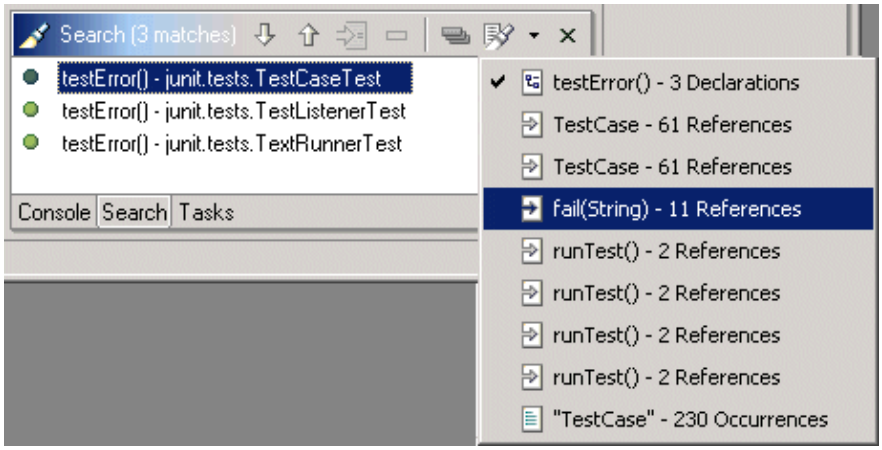


Then click Search.

4. Notice that the search results list displays the resource that contains each match.

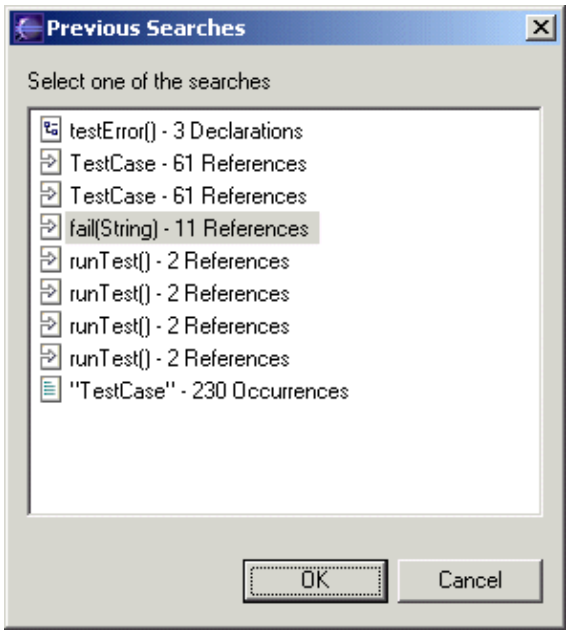
Viewing Previous Search Results

In the Search Results view, click the Previous Search Results button's drop−down menu (on the view toolbar) to see a list of most recently−conducted searches.



Repeating a Previous Search

You can also click the Previous Search Results button to bring up a dialog displaying the list of all previous searches.



Previous Searches Dialog

Select any one of these previous searches either in the menu or in the dialog to review the results of the selected search.

Clearing Previous Search Results

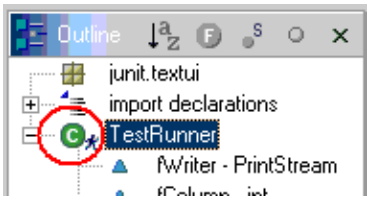
In the Search Results view, click the Remove All Searches button on the view toolbar to clear the list of previously−conducted searches. After this, all search results are no longer available for you to review.

Remove All Searches Button

Launching Your Programs

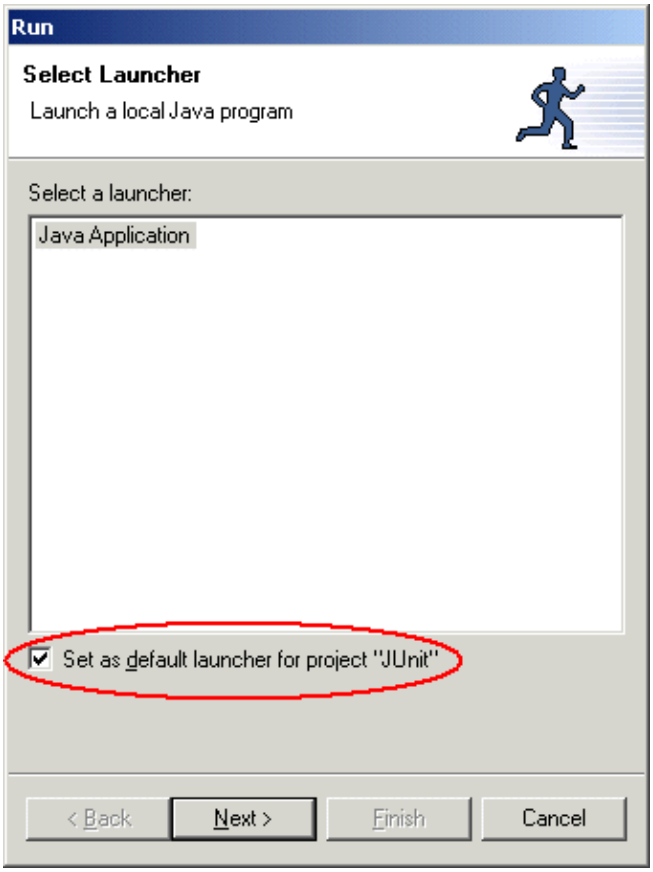
In this section, you will learn more about launching the Java executables in the workbench.

1. In the Packages view, find junit.textui.TestRunner.java and double−click it to open it in an editor.
2. In the Outline view, notice that the TestRunner class has a "runnable" icon.



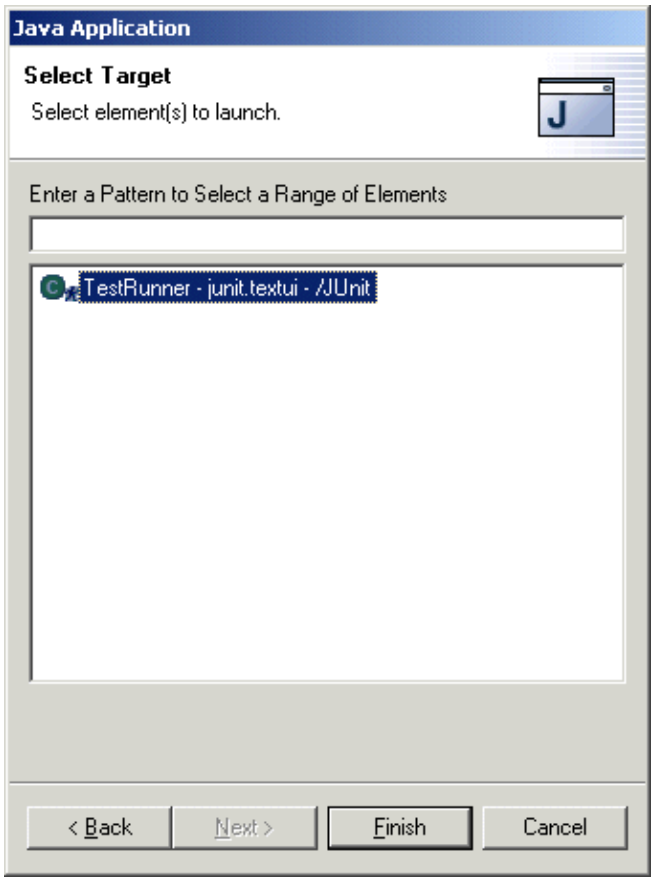
3. In the Packages view, select the junit.textui package and click the Run button in the toolbar.

4. In the Run dialog, make sure that the Set as default launcher for project JUnit box is checked, then click Next.



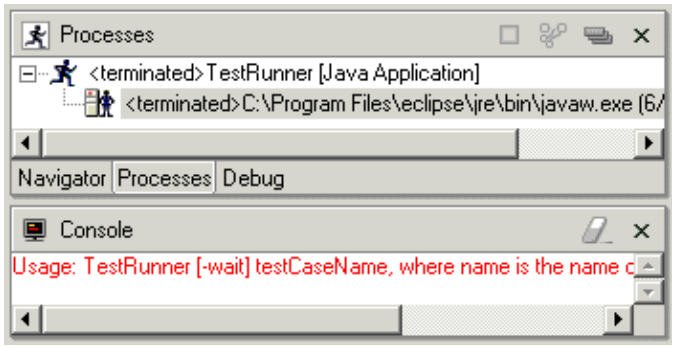
Note: This option "remembers" that the Java Application Launcher should be used for running programs in this project.

5. In the Select Target dialog, select TestRunner− junit.textui, and click Finish.

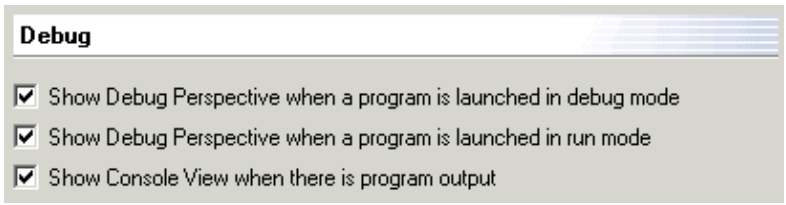


6. The Debug perspective opens, and the TestRunner program runs with a message in the Console view telling you that the program needs an execution argument.

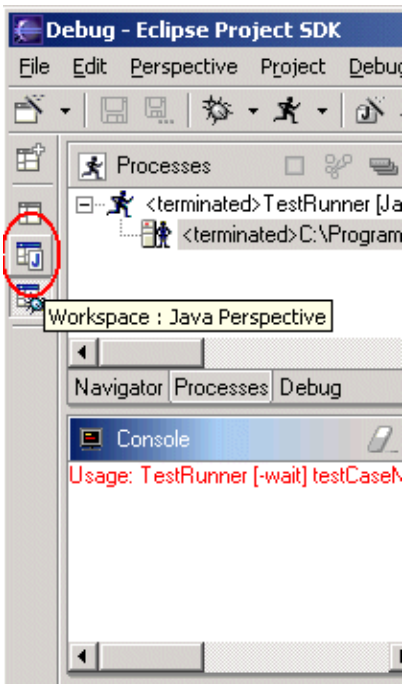
   Notice that in the Processes view, the JUnit launch is represented.



   Note: You can choose whether to automatically switch to the Debug perspective when launching a program on the Debug preferences page (Window > Preferences > Debug).

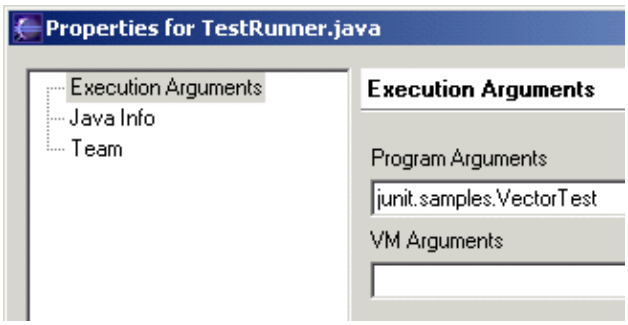


7. Return to the Java perspective by clicking the corresponding perspective button in the shortcut bar, along the left edge of the main workbench window.



8. In the Packages view, select the junit.textui.TestRunner.java resource and from its context menu, select Properties.

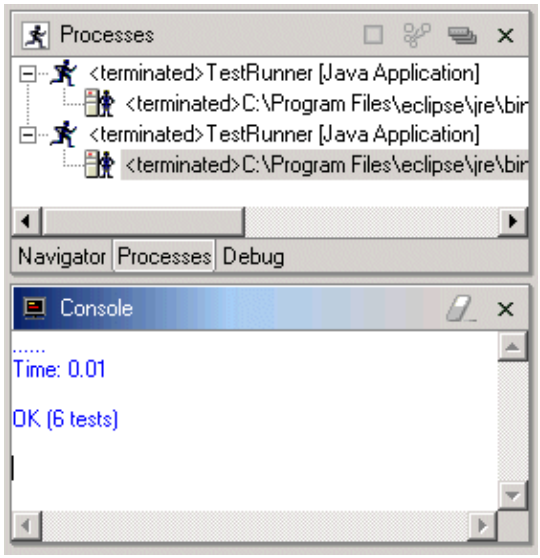
9. In the Program Arguments field on the Execution Arguments properties page, type junit.samples.VectorTest. Then click OK.



10. Select TestRunner from the drop−down menu from the Run button in the workbench toolbar. This list contains the previously−launched programs.

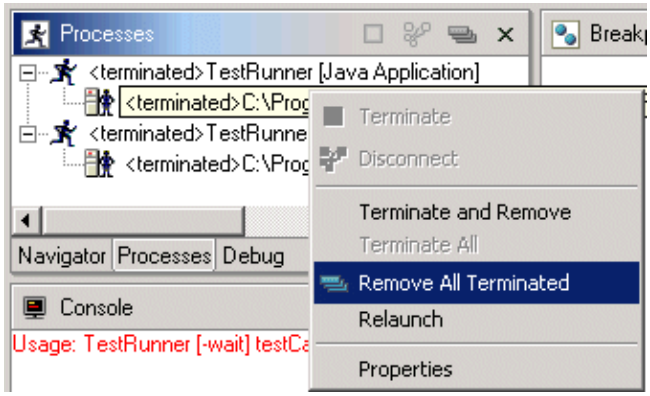
11. The Debug perspective opens, and the TestRunner program runs correctly this time, indicating the number of tests that were run.

    Notice that in the Processes view, a different JUnit launch is represented.



Note: You can relaunch any of these processes by selecting Relaunch from its context menu.

12. From the context menu in the Processes view, select Remove All Terminated to clear the view of terminated launches.



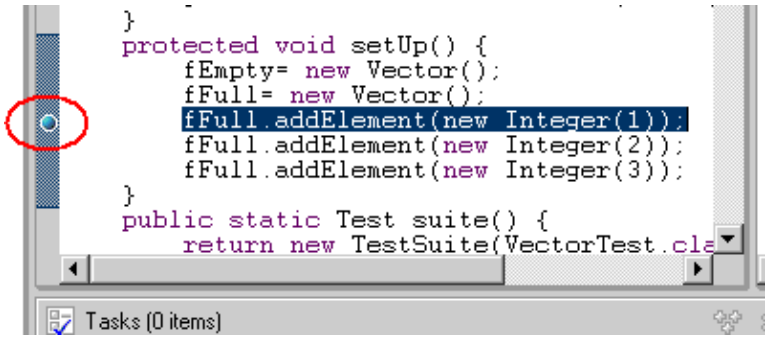
Debugging Your Programs

In this section, you will debug a Java program.

1. In the Packages view in the Java perspective, double−click junit.samples.VectorTest.java to open it in an editor.
2. Place your cursor on the marker bar (along the left edge of the editor area) on the following line in the setUp() method:

   fFull.addElement (new Integer(1));

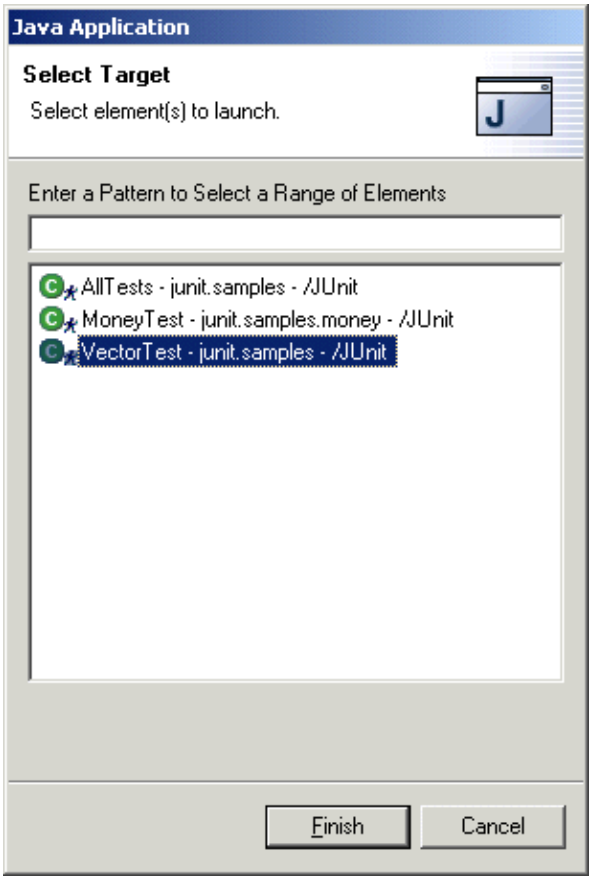
   and double−click to set a breakpoint.



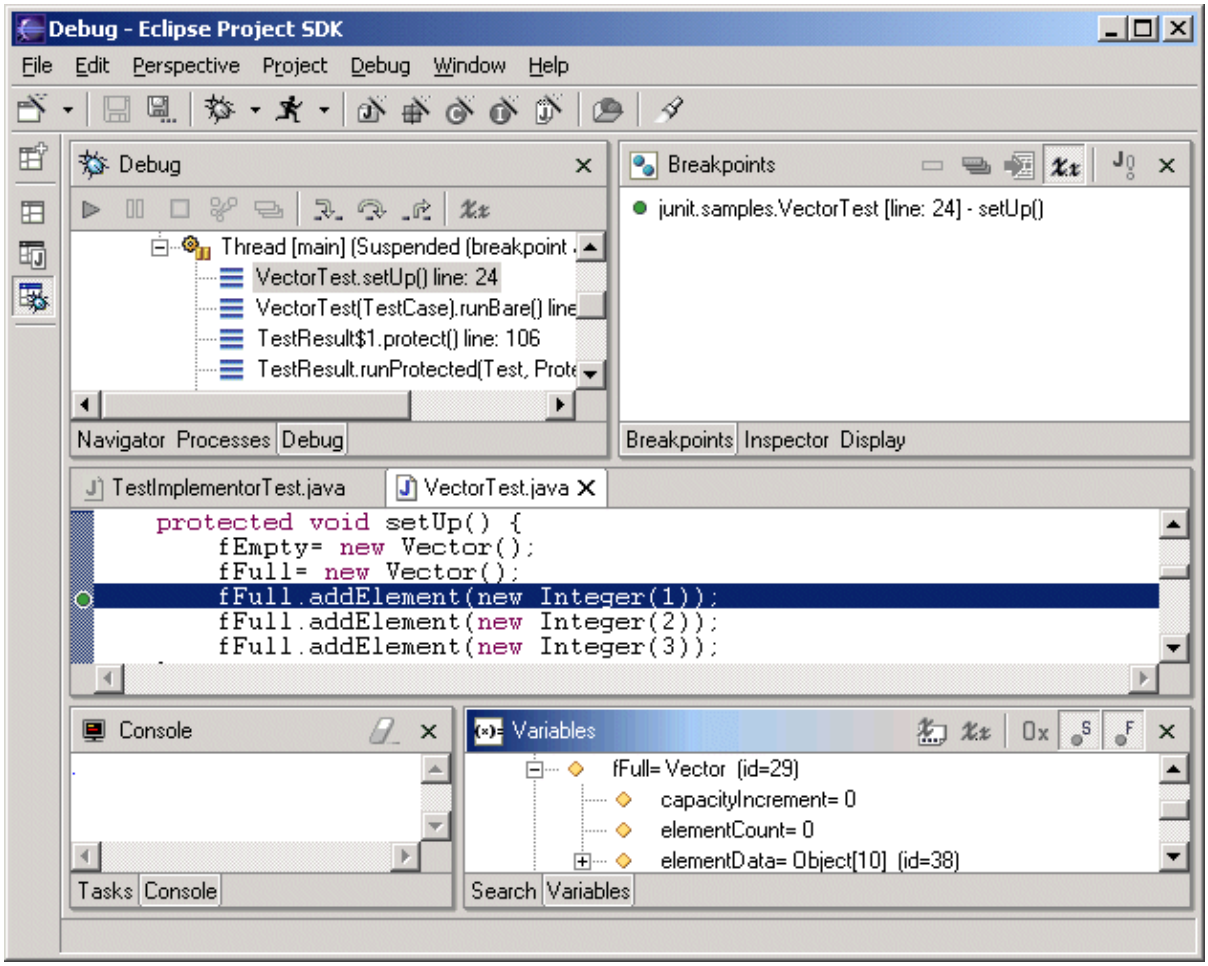
   Adding a Breakpoint

   Note: The breakpoint is blue because it is unverified, meaning that the containing class has not yet been loaded by the Java VM.

3. In the Packages view, select the junit.samples package and click the Debug button in the toolbar.
4. Select the VectorTest − junit.samples − /JUnit item in the dialog, then click Finish.

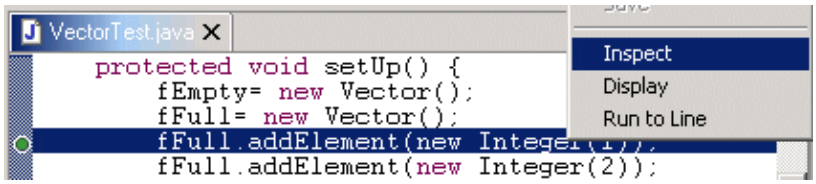


5. As soon as the breakpoint is hit, the Debug perspective opens, and execution is suspended. Notice that the process is still active (not terminated) in the Processes view. Other threads might still be running.



Note: The breakpoint is green because is it now verified.

6. In the editor in the Debug perspective, select the entire line where the breakpoint is set, and from its context menu, select Inspect.

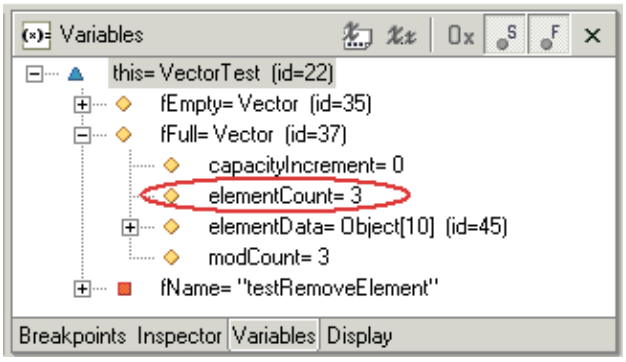


7. The expression is evaluated in the context of the current stack frame, and the results are displayed in the Inspector view.

Select the expression in the Inspector view, and from its context menu, select Remove.

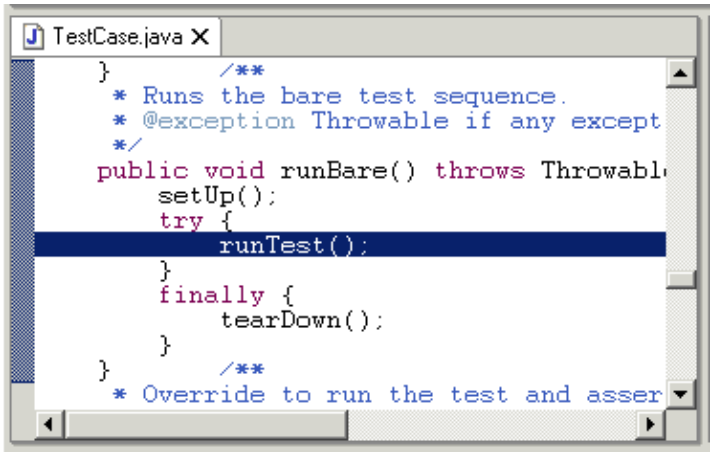
8. The Variables view displays the values of the variables in the selected stack frame.

Expand the fFull tree in the Variables view until you can see elementCount.



9. Watch the variables (e.g., elementCount) in the Variables view as you do the following in the Debug view to step through VectorTest:

Click the Step Over button to step over the highlighted line of code. Execution will continue at the next line in the same method (or, if you are at the end of a method, it will continue in the method from which the current method was called).



10. If the program has not executed fully when you are done debugging, select Terminate from the context menu of the program's launch item in either the Processes view or the Debug view.

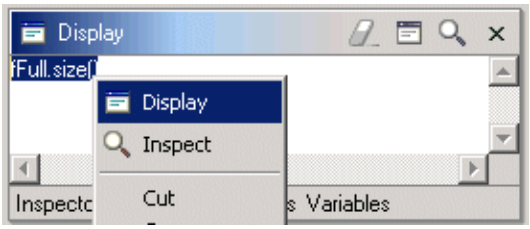
Evaluating Expressions

In this section, you will evaluate expressions from your Java code.

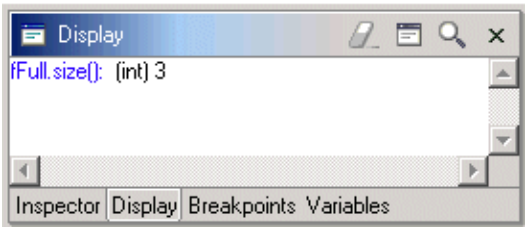
1. In the Display view in the Debug perspective, type the following line:

   fFull.size()

2. Select the text you just typed, and from its context menu, select Display.



3. The expression is evaluated.

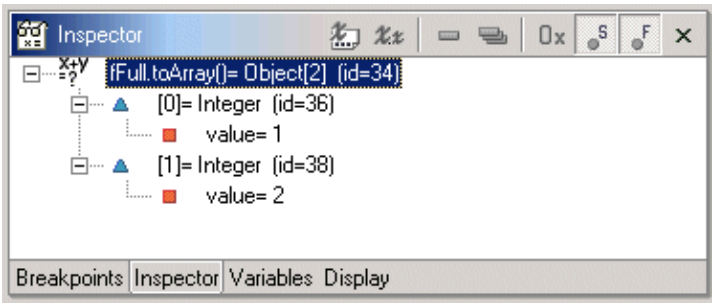


4. On a new line in the Display view, type the following line:

   fFull.toArray()

5. Select this line, and select Inspect from its context menu.

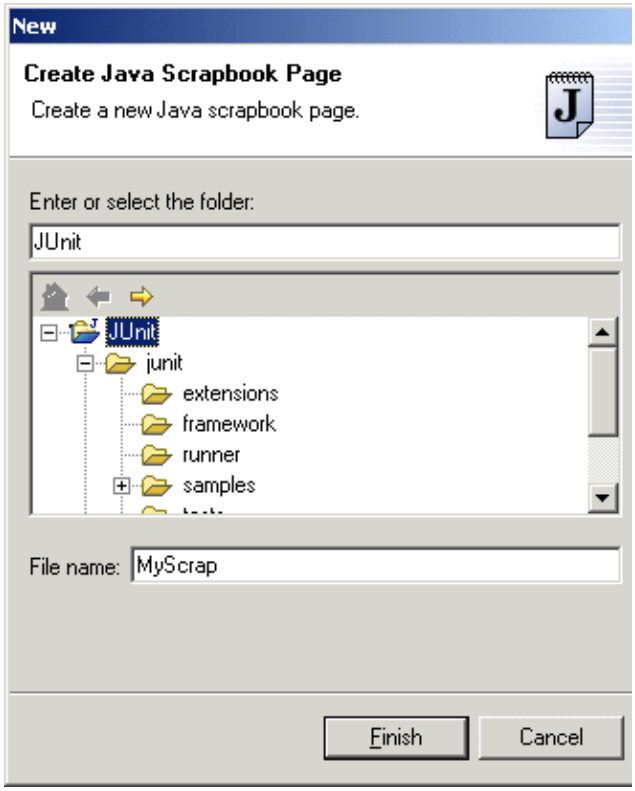
6. The Inspector view opens with the value of the evaluated expression.



Evaluating Snippets

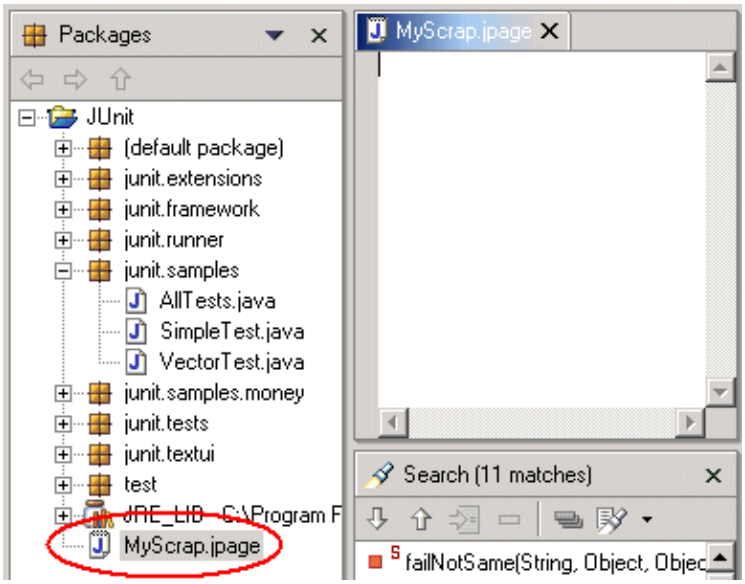
In this section, you will evaluate Java expressions using the Java scrapbook. The scrapbook can be used to experiment with Java code fragments before putting them into your program.

1. In the Java perspective, click the Create a Scrapbook Page button in the workbench toolbar.
2. In Enter or select the folder field, type or browse below to select the JUnit project root directory (JUnit).
3. In the File name field, type MyScrap. The JPAGE file extension will be added automatically, if you do not enter it yourself.

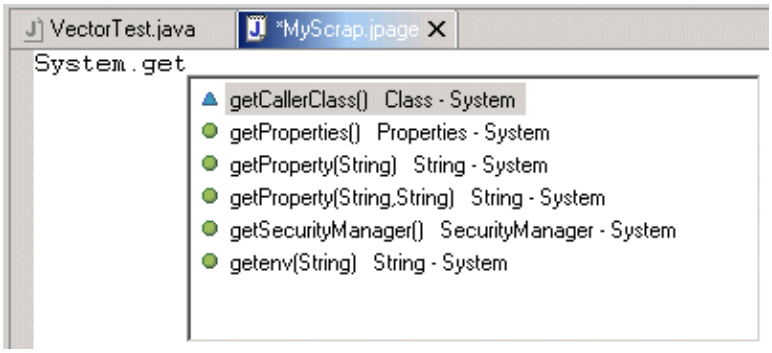


Click Finish when you are done.

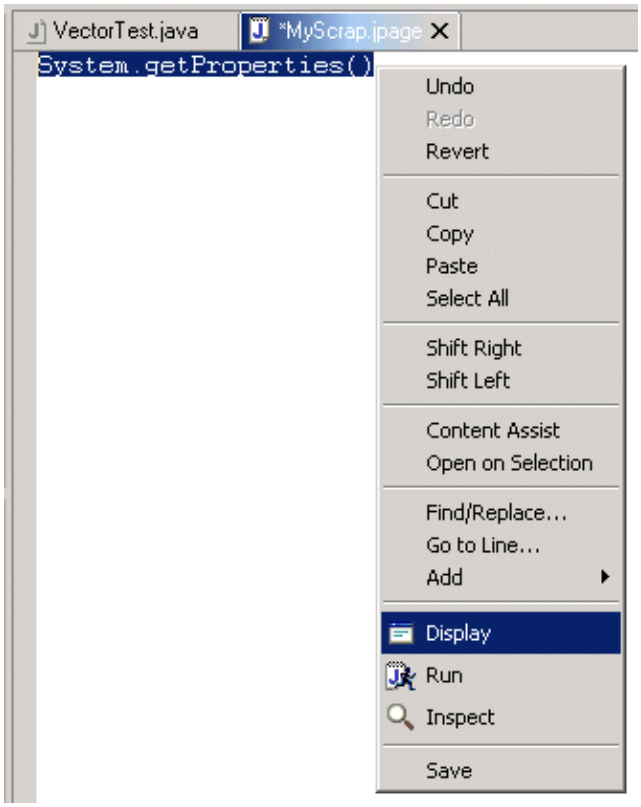
4. The scrapbook page opens in an editor in the editor area.



5. In the editor, type System.get and then use content assist (Ctrl+Space) to complete the snippet as System.getProperties().



6. Select the entire line you just typed and select Display from its context menu to evaluate the expression.



7. The result of the evaluation is highlighted in the scrapbook page.

8. Save and close the snippet file by clicking the Close button in the view toolbar and then selecting Yes in the Save dialog.