



JAVA™



MCourses.com

Les IDEs

Integrated Development Environment

NetBeans (SUN)

Eclipse (IBM)

Jbuilder (Borland)

etc.

Quand ne pas utiliser un IDE ?

- Lorsqu'on apprend le langage
- Pourquoi ?
 - Tous les mécanismes de base du langage peuvent être gérés par l'IDE : classpath, packages, compilation, exécution, génération de la doc.
 - Certaines parties du code peuvent être générées automatiquement : il faut les avoir codés soi-même pour pouvoir les comprendre !
- La connaissance du langage et de ses mécanismes fondamentaux et donc un prérequis à l'utilisation d'un IDE

Quand utiliser un IDE ?

- Une fois les principes de base du langage maîtrisés, il est **impensable** de ne pas utiliser un IDE
- Les IDEs sont des outils très puissants qui améliorent la rapidité et la qualité avec lesquelles le code est produit
- Aucun développeur professionnel ne travaille sans

Avantages d'un IDE

- Ergonomie :
 - Visualisation des sources : packages, organisation d'une classe (import, attribus, méthodes)
 - Opérations de compilation et d'exécution simplifiées
 - Génération de la documentation simplifiée
 - Visualisation suivant des perspectives
 - etc.

Avantages d'un IDE

- Facilités d'édition du code :
 - Complétion (Ctrl + escape).
 - Une grande partie des erreurs est détectée à la volée.
 - Suggestions automatiques de solution pour les erreurs.
 - Refactoring: modifications des sources facilitées.
 - Documentation intégrée et liens vers des API web.
 - Navigation entre les sources (Ctrl + click).
 - Insertion des commentaires facilitée (normaux ou javadoc).
 - Historique des modifications.
 - Templates.
 - Raccourcis clavier pour toutes les fonctions de l'IDE

Avantages d'un IDE

- Fonctionnalités avancées
 - Édition d'interfaces graphiques façon WYSIWYG
 - Gestion de projet (Todo)
 - Outils de débbug intégrés
 - Outils d'analyse des performances
 - Utilisation des dernières technologies :
 - APACHE ANT (Jakarta project)
 - Intégration des outils de gestion de version (CVS)
 - Serveurs Web intégrés

Vue globale - perspective java

The screenshot displays the Eclipse IDE interface for a Java project named 'tamagoshi'. The main editor shows the source code for 'Tamagoshi.java', which includes methods for managing a Tamagoshi character's state and energy.

```
public boolean consommeFun(){ //Exo 6
    fun--;
    if(fun<=0){
        parler("snif : je fais une dépression, ciao!", true);
        parler("", false);
        return false;
    }
    return true;
}

/**
 * @return Returns the age.
 */
public int getAge() {
    return age;
}

public String getName() {
    return name;
}

/**
 * @param lifeTime The lifeTime to set.
 */
public static void setLifeTime(int lifeTime) {
    Tamagoshi.lifeTime = lifeTime;
}

/**
 * @return Returns the lifeTime.
 */
public static int getLifeTime() {
    return lifeTime;
}
```

The Outline view on the right shows the class structure:

- Tamagoshi
 - name : String
 - generateur : Random
 - age : int
 - maxEnergy : int
 - maxFun : int
 - fun : int
 - energy : int
 - lifeTime : int
 - monPanel : TamajPanel
 - Tamagoshi(String)
 - main(String[])
 - parle()
 - parler(String)
 - parler(String, boolean)
 - mange()
 - vieillit()
 - consommeEnergie()
 - consommeFun()
 - getAge()
 - getName()

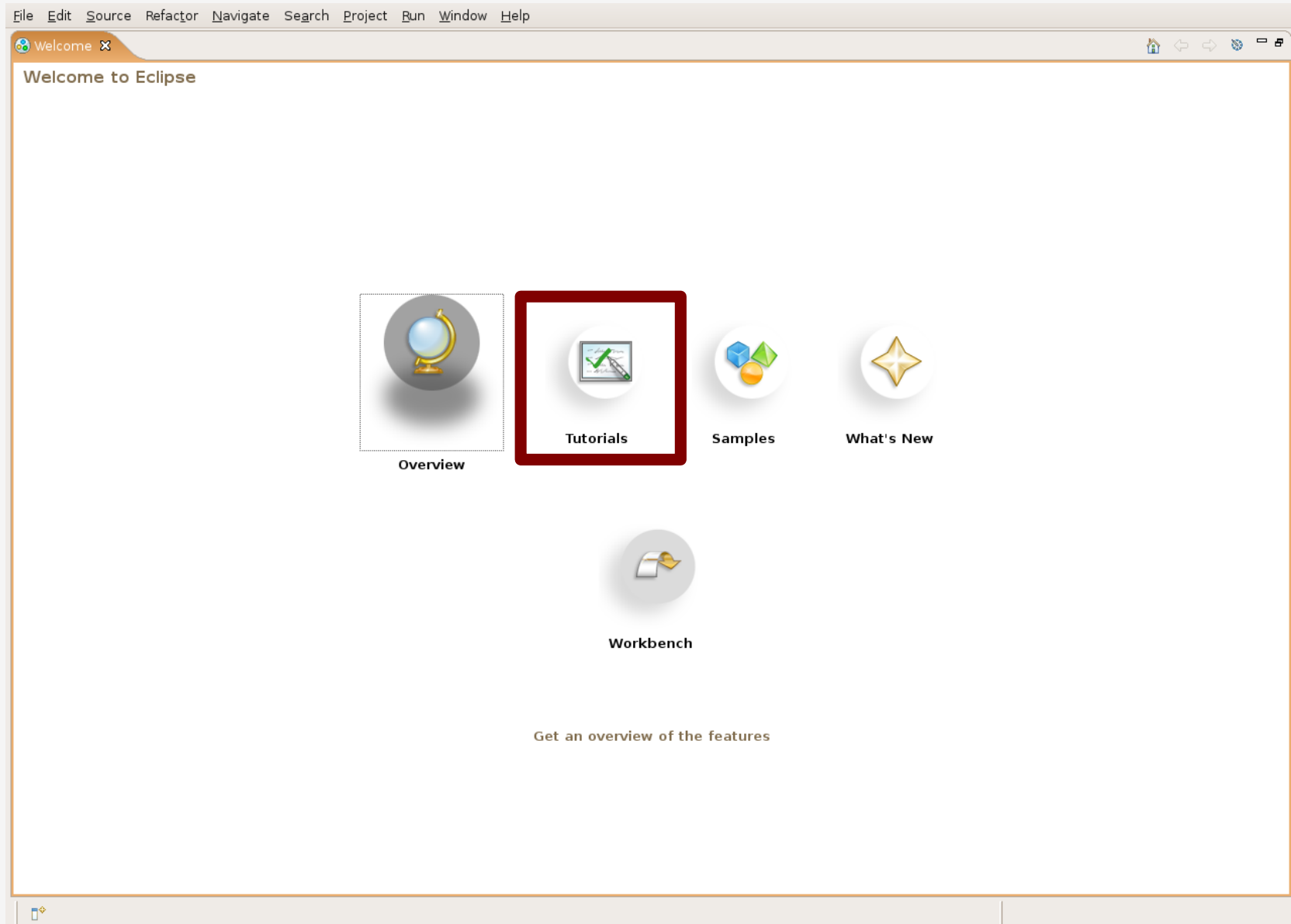
The Console view at the bottom shows the output of the application:

```
<terminated> Tamagoshi [Java Application] /usr/lib/jvm/java-1.5.0-sun-1.5.0.08/bin/java (25 janv. 07 11:13:46)

-----Cycle n01-----

Neo : "je m'ennuie à mourrir !"
Jacques : "Tout va bien !"
Zizou : "Tout va bien !"
```


eclipse - démarrage



eclipse - HelloWorld

Create a Java project

Create a Java project in the workspace or in an external location.



Project name: HelloWorld

Contents

- Create new project in workspace
- Create project from existing source

Directory: /home/fab/workspace/HelloWorld

[Browse...](#)

JRE

- Use default JRE (Currently 'java-1.5.0-sun-1.5.0.08') [Configure JREs...](#)
- Use a project specific JRE: java-1.5.0-sun-1.5.0.08

Project layout

- Use project folder as root for sources and class files
- Create separate source and output folders [Configure default...](#)



< Back

Next >

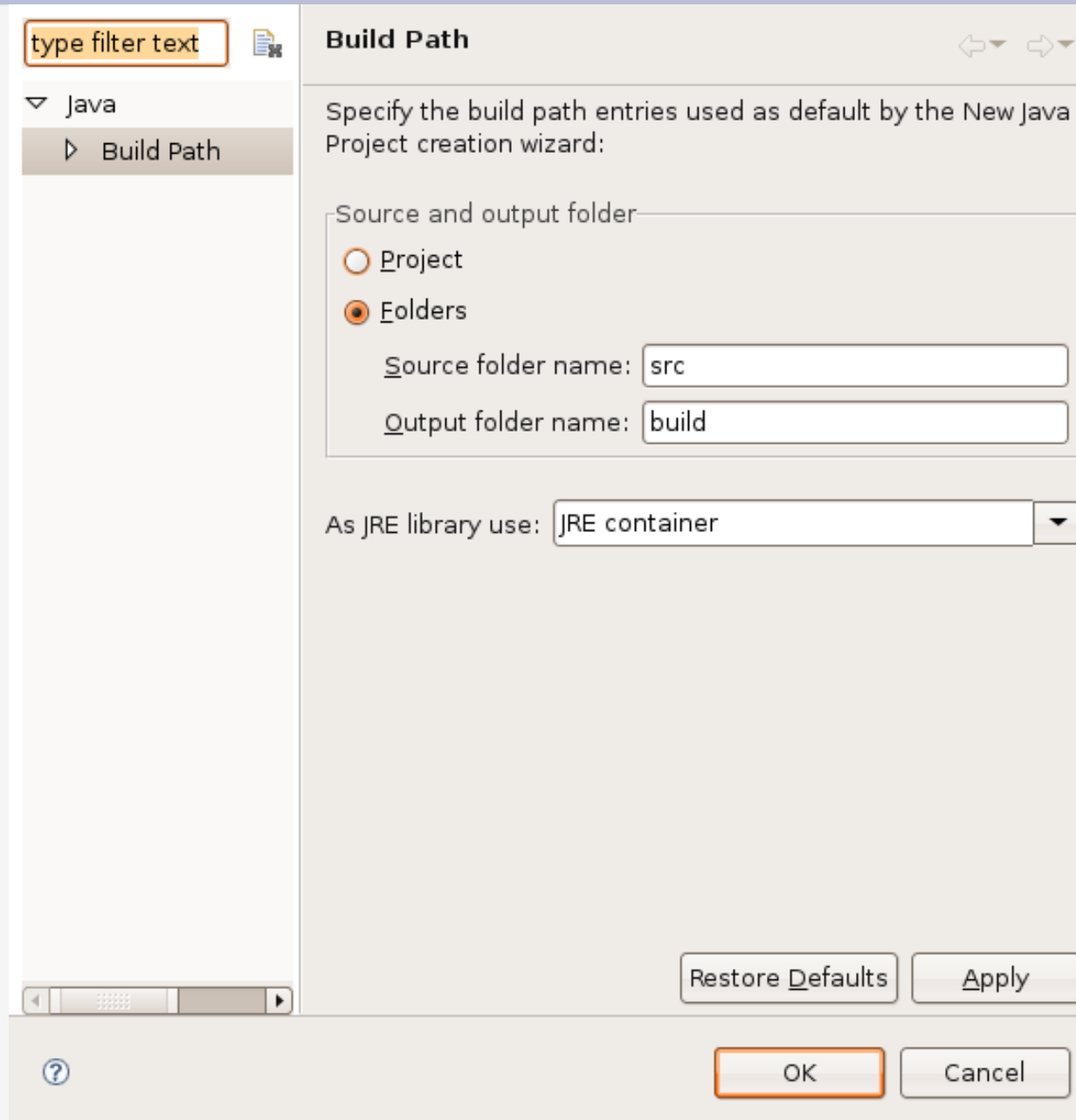
Finish

Cancel

Create a Hello World application

- ✓ ▶ Introduction ?
- ✓ ▶ Open the Java perspective ?
- ▼ **Create a Java project** ?
 - Before creating a class, we need a project to put it in. In the main toolbar, click on the **New Java Project** button, or click on the link below. Enter **HelloWorld** for the project name, then click **Finish**.
 - Click to Complete
- ▶ Create your HelloWorld class ?
- ▶ Add a print statement ?
- ▶ Run your Java ?


Distinguer les *.java des *.class



Java settings

Java Settings




Define the Java build settings.



Source Projects Libraries Order and Export

HelloWorld
src

Details





-  [Create new source folder](#): use this if you want to add a new source folder to your project.
-  [Link additional source](#): use this if you have a folder in the file system that should be used as additional source folder.
-  [Add project 'HelloWorld' to build path](#): Add the project to the build path if the project is the root of packages and source files. Entries on the build

Allow output folders for source folders

Default output folder:




HelloWorld/build

Create a Hello World application

- ✓ Introduction 
- ✓ Open the Java perspective 
- ✓ Create a Java project 
- ▼ **Create your HelloWorld class** 

The next step is to create a new class. In the main toolbar again, click on the **New Java Class** button (or the link below). Enter **HelloWorld** for the class name, select the checkbox to create the **main()** method, then click **Finish**.

The Java editor will automatically open showing your new class.

 [Click to Complete](#)
- ▶ Add a print statement 
- ▶ Run your Java 

Résultat

The screenshot shows an IDE interface with the following components:

- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Includes icons for file operations, search, and running. The **Run** button (a green play icon) is highlighted with a red box.
- Package Explorer (Left):** Shows a project named 'HelloWorld' with sub-packages: src, JRE System Library [java-1.5.], iut, madkit [madkit.cvs.sourceforge], madkitkernel, tamagoshi, and Web.
- Outline (Middle-Right):** Displays the message 'An outline is not available.'
- Welcome (Right):** Contains a 'Return to Welcome' link and a section titled 'Create a Hello World application'. It features a checklist of steps:
 - Introduction
 - Open the Java perspective
 - Create a Java project
 - Create your HelloWorld class
 - Add a print statement
 - Run your JavaThe 'Run your Java' step is currently selected.
- Console (Bottom):** Shows 'No consoles to display at this time.'
- Bottom Bar:** Displays the current project name 'HelloWorld'.

Créer une classe

Java Class

Create a new Java class.



Source folder:

Package:

Enclosing type:

Name:

Modifiers: public default private protected
 abstract final static

Superclass:

Interfaces:

Which method stubs would you like to

- public static void main(String[] args)
- Constructors from superclass
- Inherited abstract methods

Do you want to add comments as configured in the [properties](#) of the current project?

- Generate comments

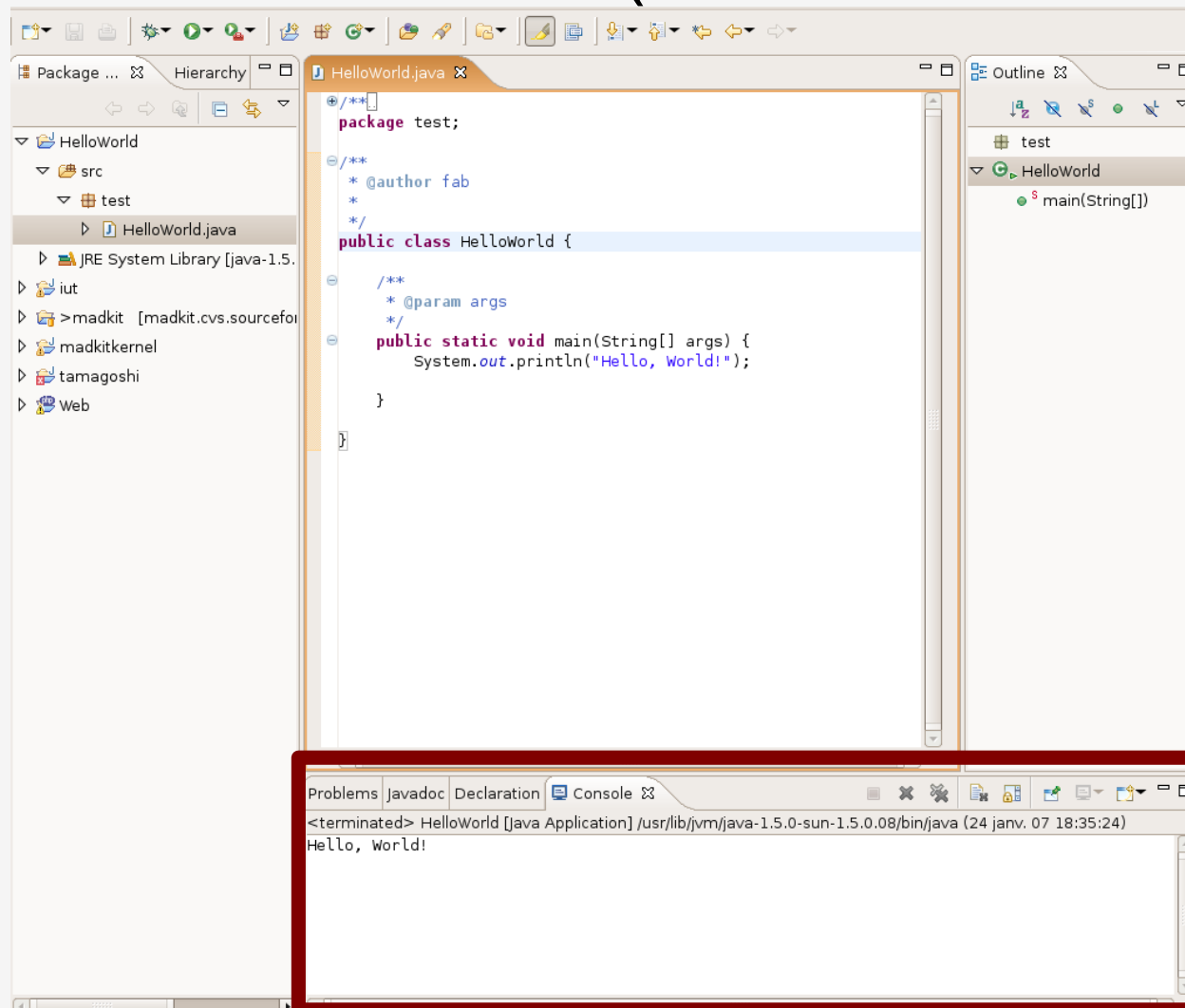


Finish

Cancel

Exécuter

- Run -> Run As -> Java Application
- ou bien : Alt+Shift+x -> (ouverture d'un menu) -> j





Package ... Hierarchy

- HelloWorld
 - src
 - test
 - HelloWorld.java**
- JRE System Library [java-1.5.0_08]
- itut
- >madkit [madkit.cvs.sourceforge]
- madkitkernel
- tamagoshi
- Web

```
package test;

/**
 * @author fab
 */
public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Outline

- test
- HelloWorld**
 - main(String[])

Problems Javadoc Declaration Console

```
<terminated> HelloWorld [Java Application] /usr/lib/jvm/java-1.5.0-sun-1.5.0.08/bin/java (24 janv. 07 18:35:24)
Hello, World!
```

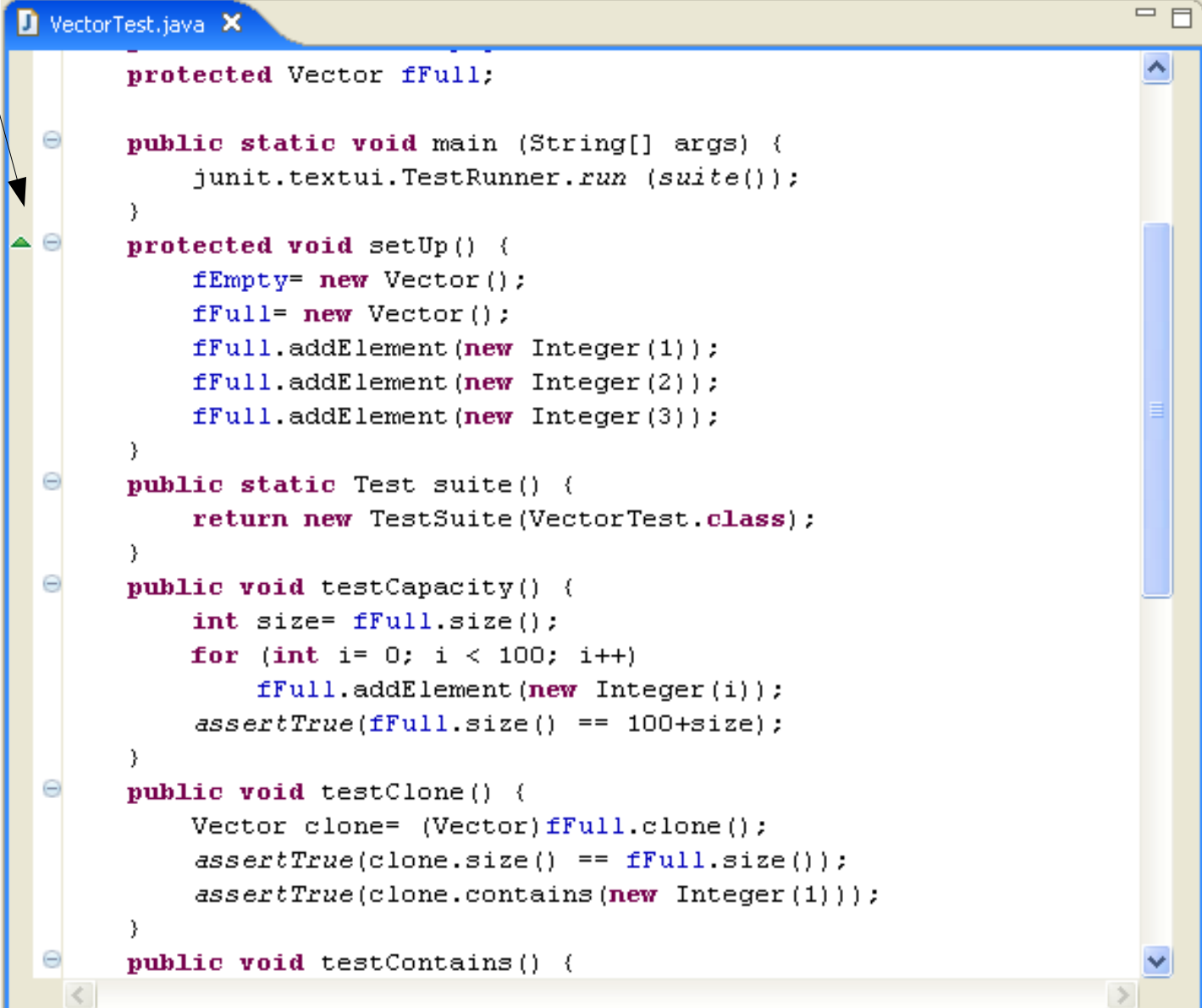

Le Package Explorer



La partie édition

Informations :

- erreurs
- surcharge d'une fonction
- etc.



```
protected Vector fFull;

public static void main (String[] args) {
    junit.textui.TestRunner.run (suite());
}

protected void setUp() {
    fEmpty= new Vector();
    fFull= new Vector();
    fFull.addElement(new Integer(1));
    fFull.addElement(new Integer(2));
    fFull.addElement(new Integer(3));
}

public static Test suite() {
    return new TestSuite(VectorTest.class);
}

public void testCapacity() {
    int size= fFull.size();
    for (int i= 0; i < 100; i++)
        fFull.addElement(new Integer(i));
    assertTrue(fFull.size() == 100+size);
}

public void testClone() {
    Vector clone= (Vector)fFull.clone();
    assertTrue(clone.size() == fFull.size());
    assertTrue(clone.contains(new Integer(1)));
}

public void testContains() {
```

Outline : Résumé de la classe

Outline window showing the structure of the `junit.samples` package. The `VectorTest` class is expanded, showing its methods and fields:

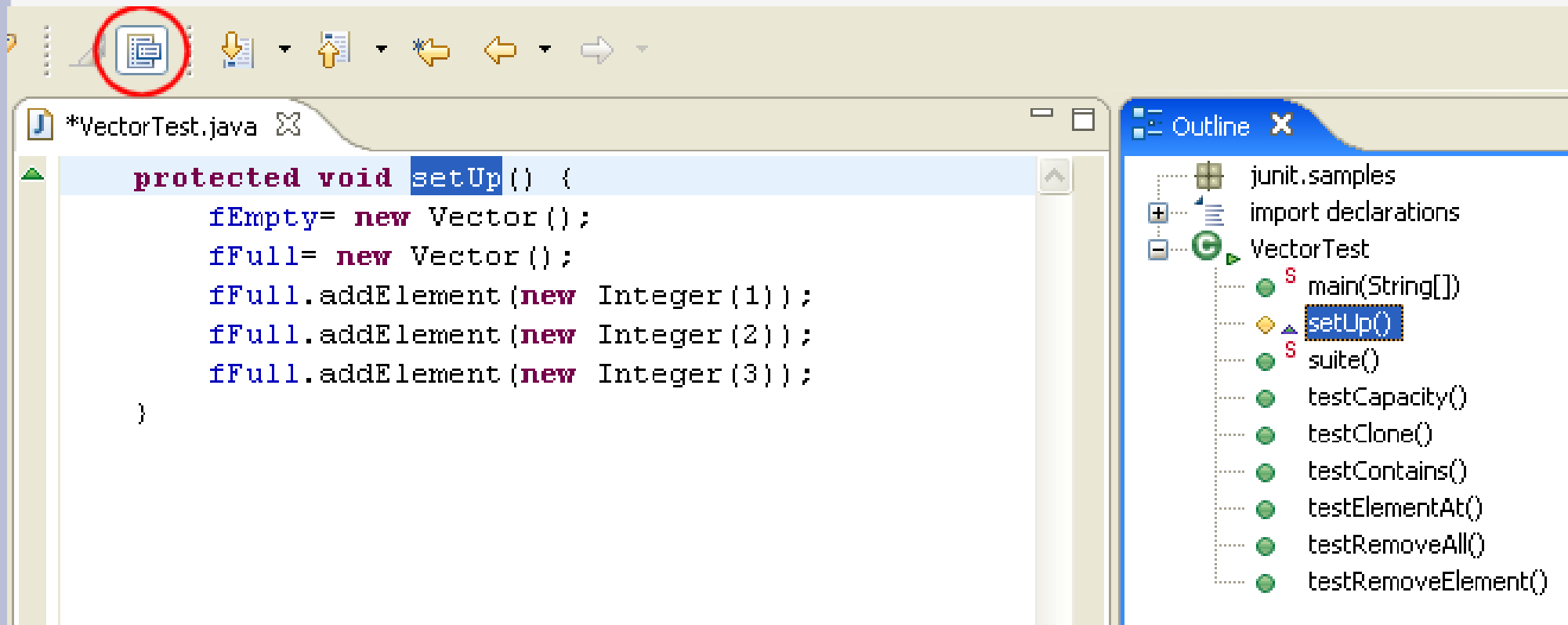
- import declarations
- VectorTest
 - fEmpty : Vector
 - fFull : Vector
 - main(String[])
 - setUp()
 - suite()
 - testCapacity()
 - testClone()
 - testContains()
 - testElementAt()
 - testRemoveAll()
 - testRemoveElement()



Outline window showing the structure of the `junit.samples` package. The `VectorTest` class is expanded, showing its methods and fields:

- import declarations
- VectorTest
 - setUp()
 - testCapacity()
 - testClone()
 - testContains()
 - testElementAt()
 - testRemoveAll()
 - testRemoveElement()

Edition par éléments



The screenshot displays an IDE interface with the following components:

- Toolbar:** Located at the top, it includes icons for file operations (copy, paste, undo, redo) and navigation (back, forward). The copy icon is circled in red.
- Editor Window:** Titled `*VectorTest.java`, it shows the following code:

```
protected void setUp() {  
    fEmpty= new Vector();  
    fFull= new Vector();  
    fFull.addElement(new Integer(1));  
    fFull.addElement(new Integer(2));  
    fFull.addElement(new Integer(3));  
}
```

The `setUp()` method signature is highlighted in blue.
- Outline Window:** Titled `Outline`, it shows a project structure with the following elements:
 - `junit.samples`
 - `import declarations`
 - `VectorTest` (expanded)
 - `main(String[])`
 - `setUp()` (highlighted in blue)
 - `suite()`
 - `testCapacity()`
 - `testClone()`
 - `testContains()`
 - `testElementAt()`
 - `testRemoveAll()`
 - `testRemoveElement()`

Navigation

The image shows an IDE window with two panes. The left pane displays the source code of `VectorTest.java`. The right pane shows the Outline view of the project.

Left Pane (Code Editor):

```
package junit.samples;

import junit.framework.*;

/**
 * A sample test case, testing <code>java.util.Vect
 *
 */
public class VectorTest extends TestCase {
    protected Vector fEmpty;
    protected Vector fFull;

    public static void main (String[] args) {
        junit.textui.TestRunner.run (suite());
    }

    protected void setUp() {
```

The `main` method is highlighted in blue. A red circle highlights the vertical scrollbar on the left side of the editor.

Right Pane (Outline View):

- junit.samples
 - import declarations
 - VectorTest
 - main(String[])** (highlighted with a red circle)
 - suite()
 - setUp()
 - testCapacity()
 - testClone()
 - testContains()
 - testElementAt()
 - testRemoveAll()
 - testRemoveElement()

Quick outline : ctrl+o (menu navigate)

The image shows an IDE window with two tabs: `TestCase.java` and `VectorTest.java`. The `VectorTest.java` tab is active, displaying the following code:

```
package junit.samples;

import junit.framework.*;

/**
 * A sample t
 *
 */
public class
protected
protected

public st
    junit
}

protected
    fEmpt
    fFull
    fFull
    fFull
    fFull
}

public st
    retur
}

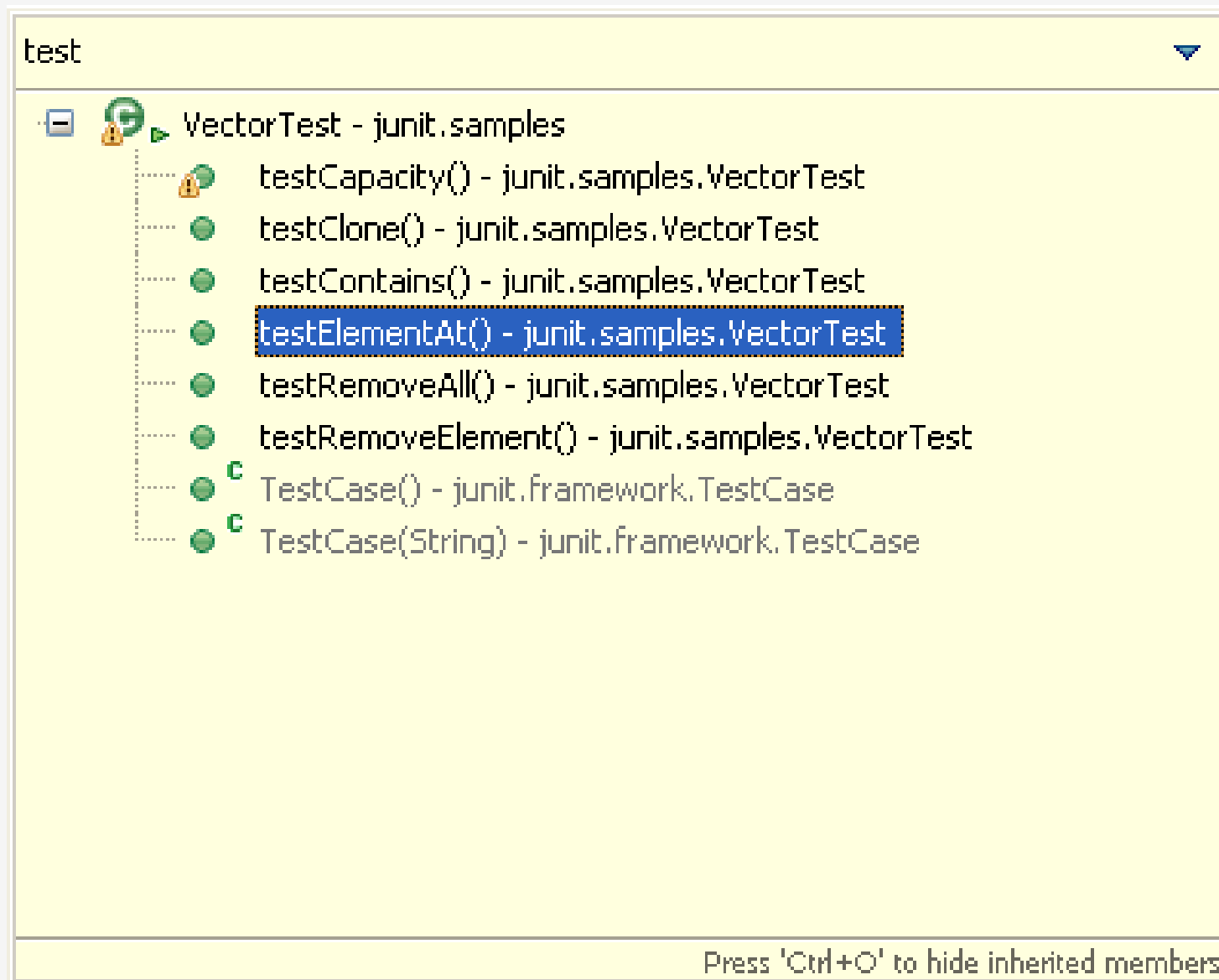
public void testCapacity() {
    int size= fFull.size();
    for (int i= 0; i < 100; i++)
```

A class outline window is overlaid on the code, showing the structure of the `junit.samples` package and the `VectorTest` class. The outline lists the following members:

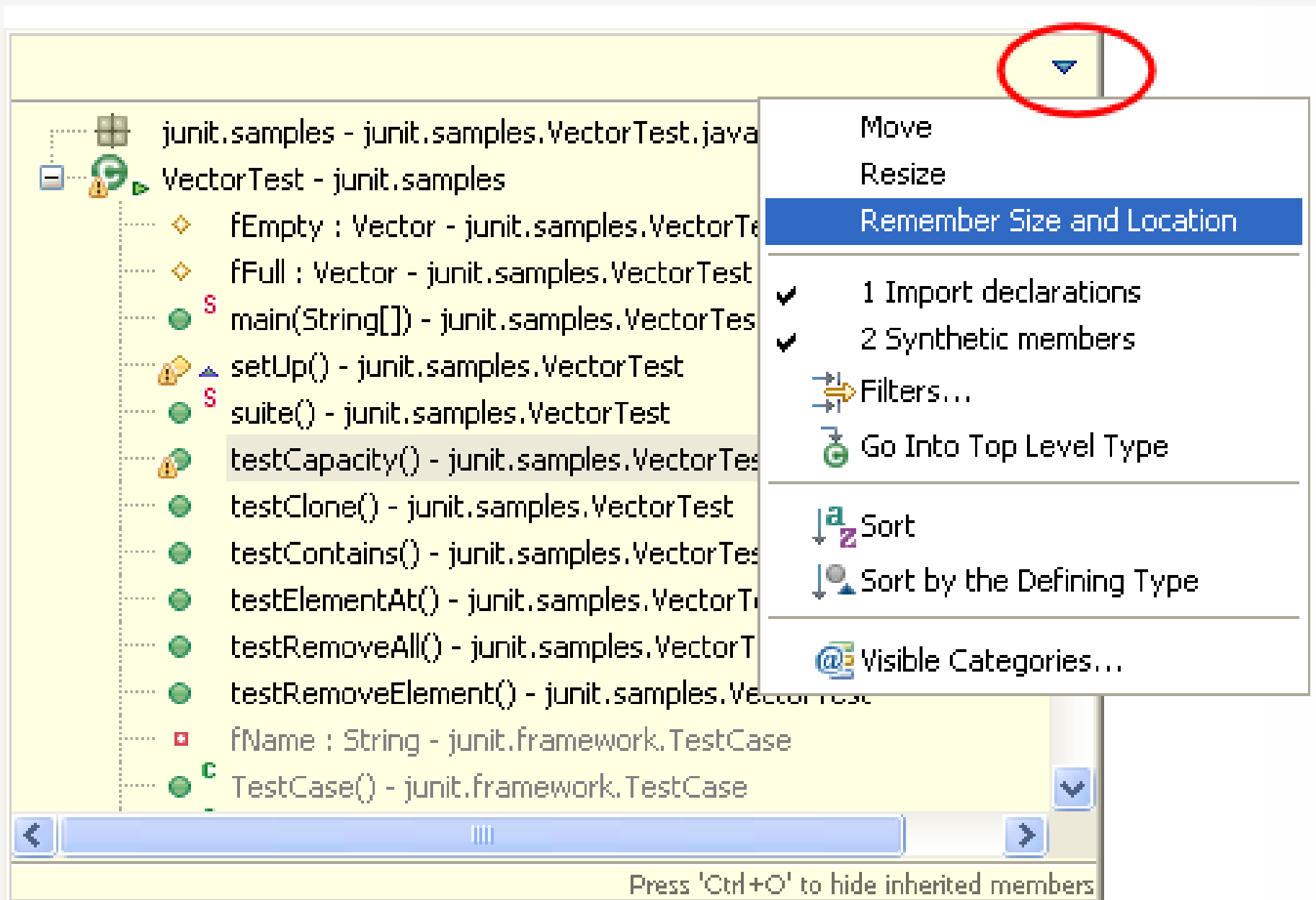
- `fEmpty : Vector`
- `fFull : Vector`
- `main(String[])`
- `setUp()`
- `suite()`
- `testCapacity()`
- `testClone()`
- `testContains()`
- `testElementAt()`
- `testRemoveAll()`
- `testRemoveElement()`

A red circle highlights a tooltip at the bottom of the outline window that reads: "Press 'Ctrl+O' to show inherited members".

Quick outline : recherche par frappe



Quick outline : menu préférences



Outils pour travailler sur le code

Rajout d'une méthode

The screenshot shows an IDE window titled `*VectorTest.java` with the following Java code:

```
public void testElementAt() {
    Integer i = (Integer)fFull.elementAt(0);
    assertTrue(i.intValue() == 1);

    try {
        fFull.elementAt(fFull.size());
    } catch (ArrayIndexOutOfBoundsException e) {
        return;
    }
    fail("Should raise an ArrayIndexOutOfBoundsException");
}

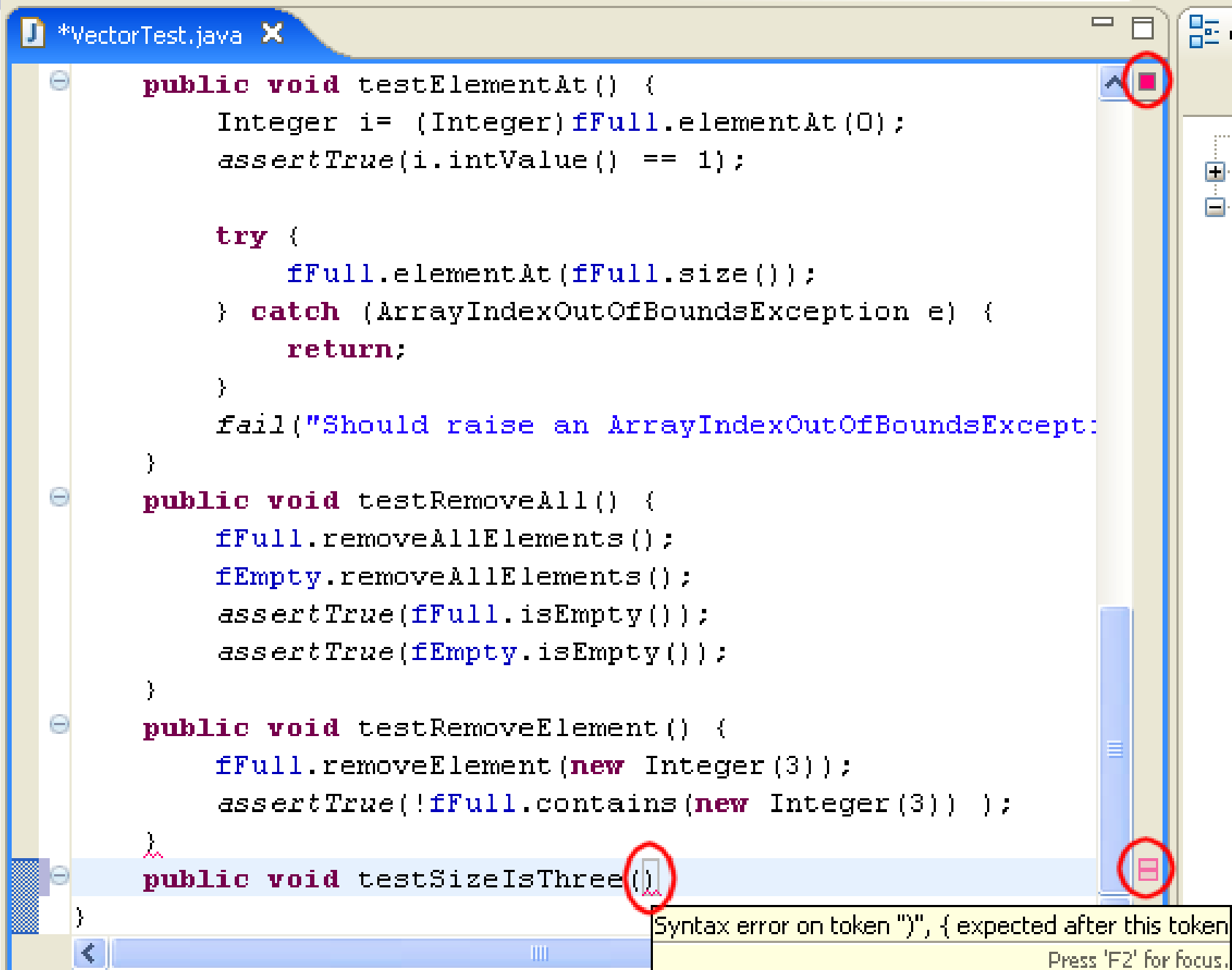
public void testRemoveAll() {
    fFull.removeAllElements();
    fEmpty.removeAllElements();
    assertTrue(fFull.isEmpty());
    assertTrue(fEmpty.isEmpty());
}

public void testRemoveElement() {
    fFull.removeElement(new Integer(3));
    assertTrue(!fFull.contains(new Integer(3)));
}

public void testSizeIsThree()
```

The `testSizeIsThree()` method signature is circled in red. In the Outline window on the right, the `testSizeIsThree()` method is also circled in red, indicating it has been added to the class's public API.

Les erreurs en temps réel



```
*VectorTest.java X  
  
public void testElementAt() {  
    Integer i= (Integer)fFull.elementAt(0);  
    assertTrue(i.intValue() == 1);  
  
    try {  
        fFull.elementAt(fFull.size());  
    } catch (ArrayIndexOutOfBoundsException e) {  
        return;  
    }  
    fail("Should raise an ArrayIndexOutOfBoundsException");  
}  
  
public void testRemoveAll() {  
    fFull.removeAllElements();  
    fEmpty.removeAllElements();  
    assertTrue(fFull.isEmpty());  
    assertTrue(fEmpty.isEmpty());  
}  
  
public void testRemoveElement() {  
    fFull.removeElement(new Integer(3));  
    assertTrue(!fFull.contains(new Integer(3)) );  
}  
  
public void testSizeIsThree()  
}
```

Syntax error on token ")", { expected after this token

Press 'F2' for focus.

Les erreurs en temps réel

```
*VectorTest.java x
public void testElementAt() {
    Integer i= (Integer)fFull.elementAt(0);
    assertTrue(i.intValue() == 1);

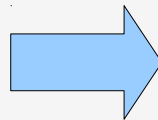
    try {
        fFull.elementAt(fFull.size());
    } catch (ArrayIndexOutOfBoundsException e) {
        return;
    }
    fail("Should raise an ArrayIndexOutofBoundsExcept:");
}

public void testRemoveAll() {
    fFull.removeAllElements();
    fEmpty.removeAllElements();
    assertTrue(fFull.isEmpty());
    assertTrue(fEmpty.isEmpty());
}

public void testRemoveElement() {
    fFull.removeElement(new Integer(3));
    assertTrue(!fFull.contains(new Integer(3)) );
}

public void testSizeIsThree() {
}
}
```

Syntax error on token ")", { expected after this token
Press 'F2' for focus.



Package Explorer Hierarchy

- JUnit
 - + junit.awtui
 - + junit.extensions
 - + junit.framework
 - + junit.runner
 - junit.samples
 - + AllTests.java
 - + SimpleTest.java
 - + VectorTest.java
 - + junit.samples.money

Utiliser la complétion

```
public void testSizeIsThree() {  
    assertTrue(fFull.size() == 3);  
    Vector v = new Vector();  
    for (int i = 0; i < 3; i++) {  
        v.addElement(new Object());  
    }  
}
```

assert

- assert
- assertEquals(boolean expected, boolean actual) void - As
- assertEquals(byte expected, byte actual) void - Assert
- assertEquals(char expected, char actual) void - Assert
- assertEquals(int expected, int actual) void - Assert
- assertEquals(long expected, long actual) void - Assert
- assertEquals(Object expected, Object actual) void - Asse
- assertEquals(short expected, short actual) void - Assert

Press 'Ctrl+Space' to show Template Proposals

Problems Javadoc Declaration

3 errors, 1 warning, 0 infos (Filter matched 4 of 22 items)

La complétion affiche la javadoc

```
}
```

```
assertt
```

- `assertTrue(boolean condition) void - Assert`
- `assertTrue(String message, boolean condition) void - Assert`

Press 'Ctrl+Space' to show Template Proposals

Asserts that a condition is true. If it isn't it throws an `AssertionFailedError`.

Identifier les problèmes de code

```
*TestCase.java x
package junit.framework

import java.lang.reflect.*;
```

Package Explorer

- JUnit
 - junit.awtui
 - junit.extensions
 - junit.framework
 - Assert.java
 - AssertionFailedError.java
 - ComparisonFailure.java
 - Protectable.java
 - Test.java
 - TestCase.java
 - TestFailure.java
 - TestListener.java
 - TestResult.java
 - TestSuite.java
 - junit.runner
 - junit.samples
 - junit.samples.money
 - junit.swingui
 - junit.swingui.icons
 - junit.textui
 - JRE System Library [jdk1.5.0_03]

```
TestCase.java x
package junit.framework

import java.lang.reflect.*;

/**
 * A test case defines the fixture to run multip
 * 1) implement a subclass of TestCase<br>
 * 2) define instance variables that store the s
 * 3) initialize the fixture state by overriding
 * 4) clean-up after a test by overriding <code>
 * Each test runs in its own fixture so there
 * can be no side effects among test runs.
 * Here is an example:
 * <pre>
 * public class MathTest extends TestCase {
 *     protected double fValue1;
 *     protected double fValue2;
```

Outline

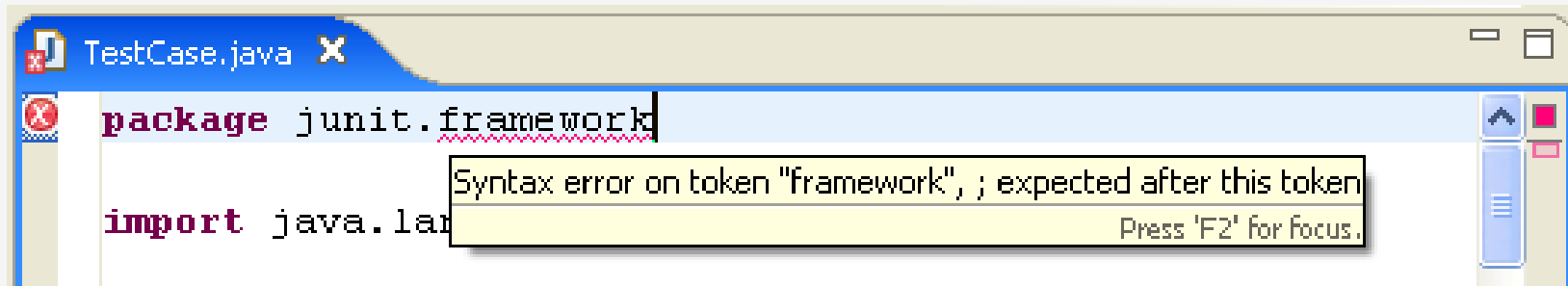
- junit.framework
 - import declarations
 - TestCase
 - TestCase()
 - TestCase(String)
 - countTestCases()
 - createResult()
 - run()
 - run(TestResult)
 - runBare()
 - runTest()
 - setUp()
 - tearDown()
 - toString()
 - getName()
 - setName(String)

Search Problems x Javadoc Declaration

1 error, 0 warnings, 0 infos

Description	Resource	In Folder	Location
Syntax error on token "framework", ...	TestCase....	JUnit/junit/framework	line 1

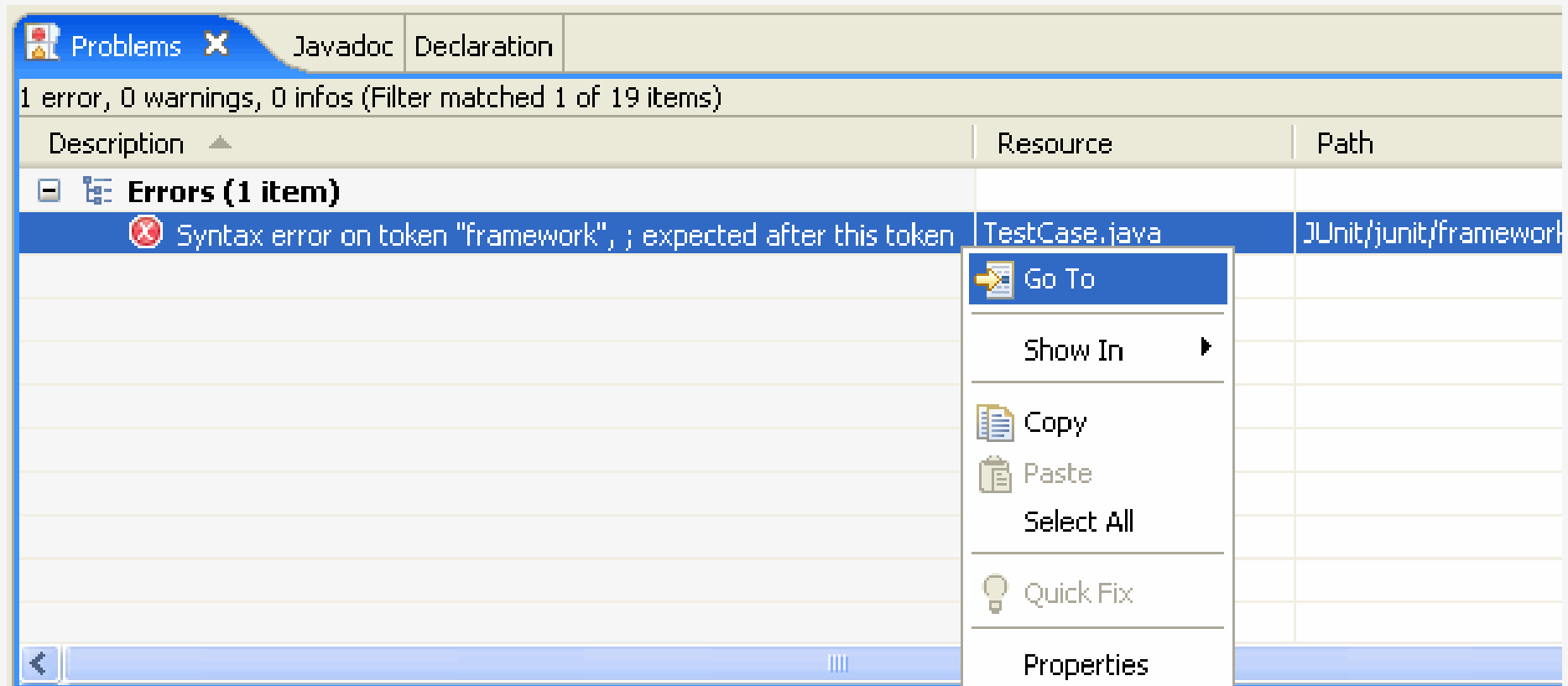
Identifier les problèmes de code



```
package junit.framework  
  
import java.la
```


Syntax error on token "framework", ; expected after this token
Press 'F2' for focus.

ET



Problems Javadoc Declaration

1 error, 0 warnings, 0 infos (Filter matched 1 of 19 items)

Description	Resource	Path
Errors (1 item)		
 Syntax error on token "framework", ; expected after this token	TestCase.java	JUnit/junit/framework

- Go To
- Show In
- Copy
- Paste
- Select All
- Quick Fix
- Properties

Résoudre les problèmes de code

```
/**
 * Gets the name of a TestCase
 * @return returns a String
 */
public String getName() {
    return fTestName;
}
```



Clique gauche
sur la croix

The screenshot shows an IDE interface. On the left, a code editor displays the following code:

```
/**
 * Gets the name of a TestCase
 * @return returns a String
 */
public String getName() {
    return fTestName;
}

/**
 * Sets the
 * @param n
 */
public void
    fName=
}
```

A red 'x' icon is visible in the gutter next to the `return fTestName;` line. A context menu is open over this line, listing several actions:

- Change to 'getName()'
- Create local variable 'fTestName'
- Create field 'fTestName'
- Change to 'fName'
- Create parameter 'fTestName'
- Create constant 'fTestName'

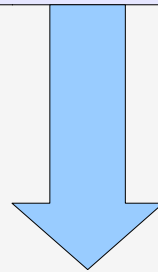
On the right side of the IDE, a class hierarchy view is visible, showing a list of methods with colored icons: `runTest()`, `setUp()`, `tearDown()`, `toString()`, `getName()` (highlighted with a red 'x'), and `setName(St`.

A yellow box on the right contains the following code snippet:

```
...
public String getName() {
    String fTestName;
    return fTestName;
}
...
```

Taper du code : les templates

« sys » suivit de « ctrl+space »



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

Les templates : Window -> preferences-> java-> editor -> content Assist -> Template

Les templates : exemples

```
public void testValues() {  
    Integer[] expected = new Integer[3];  
    for
```

- for - iterate over array
- for - iterate over array with temporary variable
- for - iterate over collection
- foreach - iterate over an array or Iterable
- for
- ForegroundAction - javax.swing.text.StyledEditorKit
- FormAction - javax.swing.text.html.HTMLDocument.HTML
- Format - java.text

Press 'Ctrl+Space' to show Template Proposals

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

Les templates : exemples

TAB

```
public void testValues() {  
    Integer[] expected = new Integer[3];  
    for (int i = 0; i < expected.length; i++) {  
        |  
    }  
}
```

TAB

```
public void testValues() {  
    Integer[] expected = new Integer[3];  
    for (int e = 0; e < expected.length; e++) {  
        |  
    }  
}
```

```
public void testValues() {  
    Integer[] expected = new Integer[3];  
    for (int e = 0; e < expected.length; e++) {  
        |  
    }  
}
```

Les templates : exemples

```
public void testValue() {  
    Integer[] expected = new Integer[3];  
    for (int e= 0; e < expected.length; e++) {  
        expected[e] = new Integer(e+1);  
    }  
    Integer[] actual = to
```

```
(type[]) collection.toArray(new type[col
```

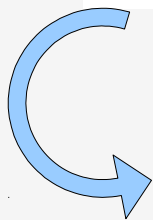
- toString() String - TestCase
- toArray - convert collection to array
- ToggleButtonBorder - com.sun.java.swing.plaf.motif.Motif
- ToggleButtonBorder - javax.swing.plaf.basic.BasicBorders
- ToggleButtonBorder - javax.swing.plaf.metal.MetalBorder
- ToggleButtonModel - javax.swing.JToggleButton
- ToHTMLSAXHandler - org.apache.xml.serializer
- ToHTMLStream - org.apache.xml.serializer

Press 'Ctrl+Space' to show Template Proposals

Les templates : exemples

```
public void testValues() {  
    Integer[] expected = new Integer[3];  
    for (int e = 0; e < expected.length; e++) {  
        expected[e] = new Integer(e + 1);  
    }  
    Integer[] actual = (type[]) collection.toArray(new type[collection.size()])  
}
```

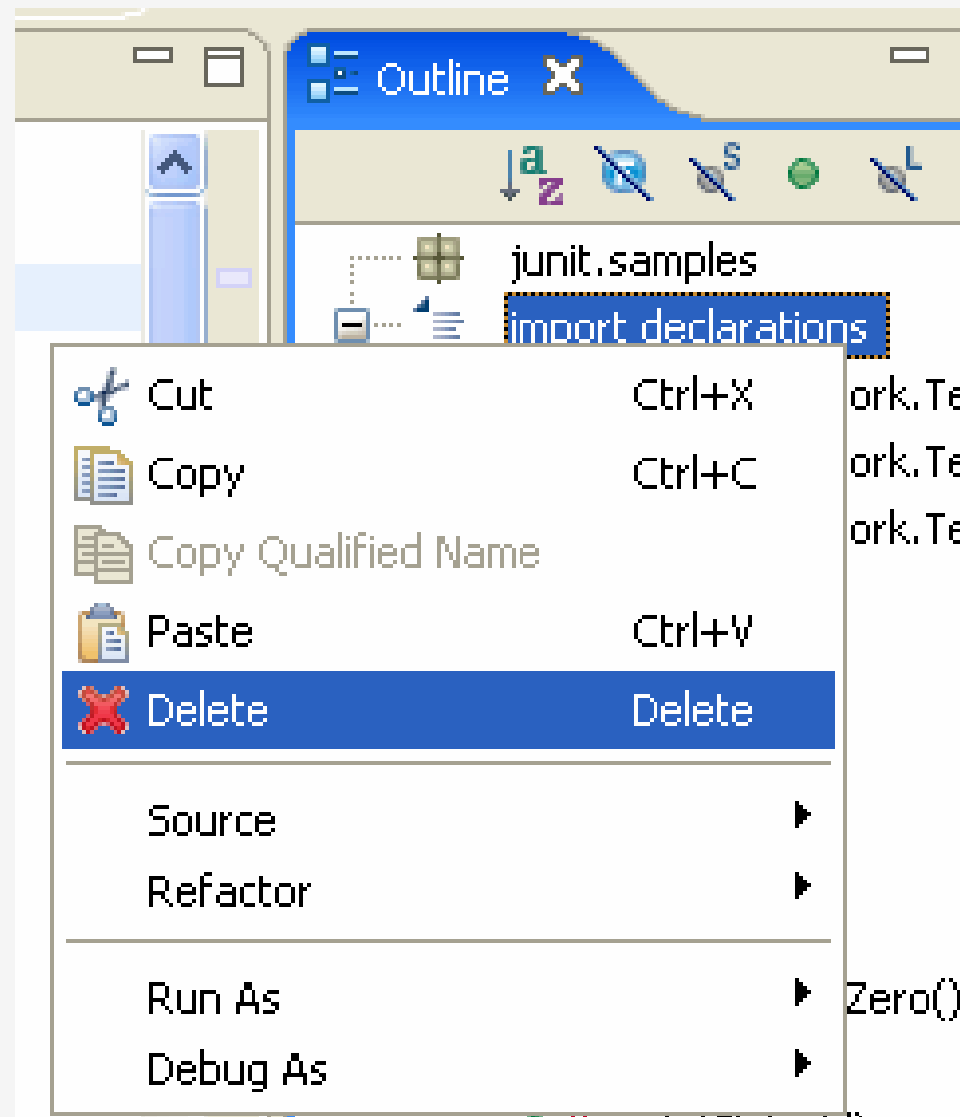
TAB



```
public void testValues() {  
    Integer[] expected = new Integer[3];  
    for (int e = 0; e < expected.length; e++) {  
        expected[e] = new Integer(e + 1);  
    }  
    Integer[] actual = (Integer[]) fFull.toArray(new Integer[fFull.size()])  
}
```

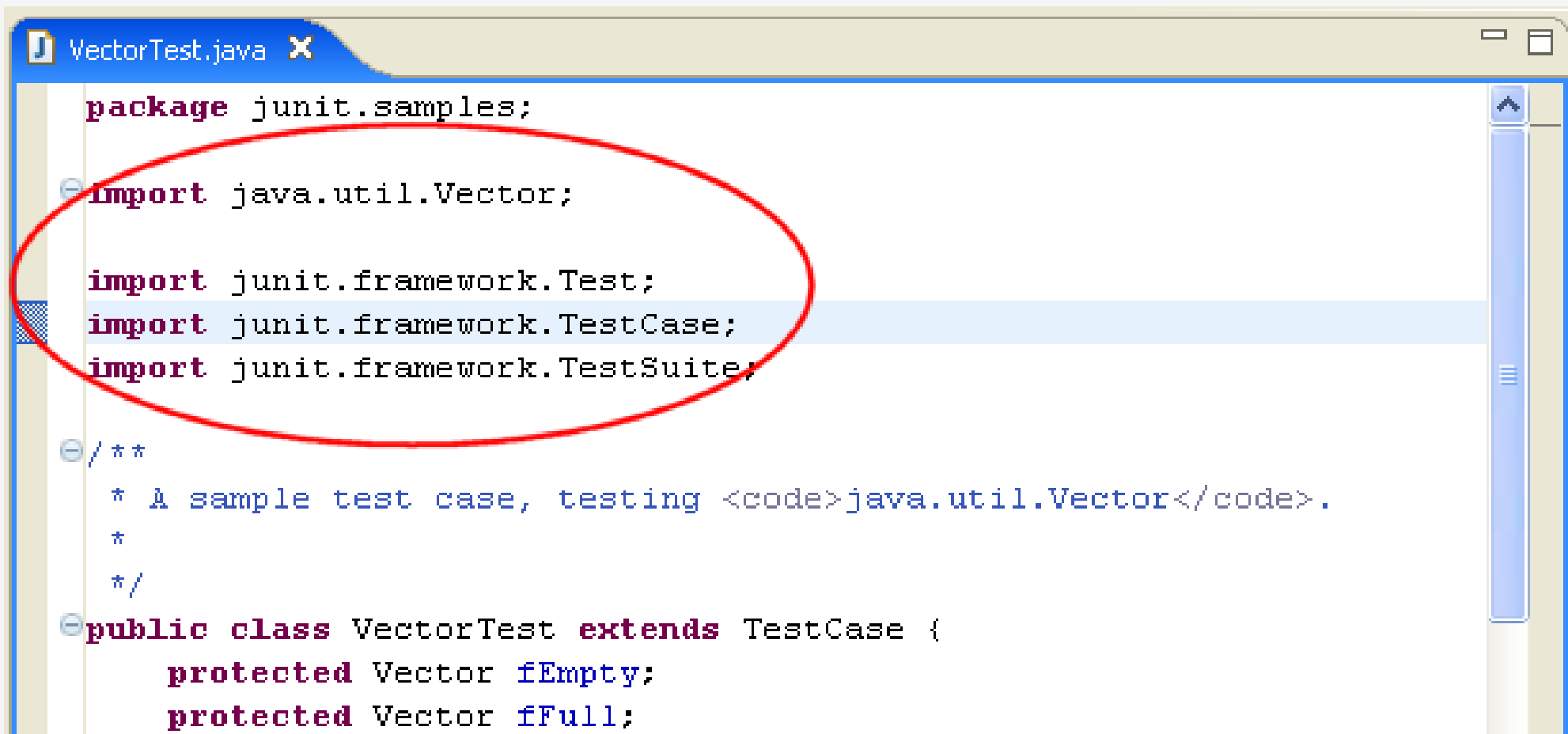
Insertion automatique des imports

- Suppression des imports pour l'exemple :



Insertion automatique des imports

- Bouton droit -> source -> organize imports
- ou « Shift + ctrl + o »



```
VectorTest.java x
package junit.samples;

import java.util.Vector;

import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

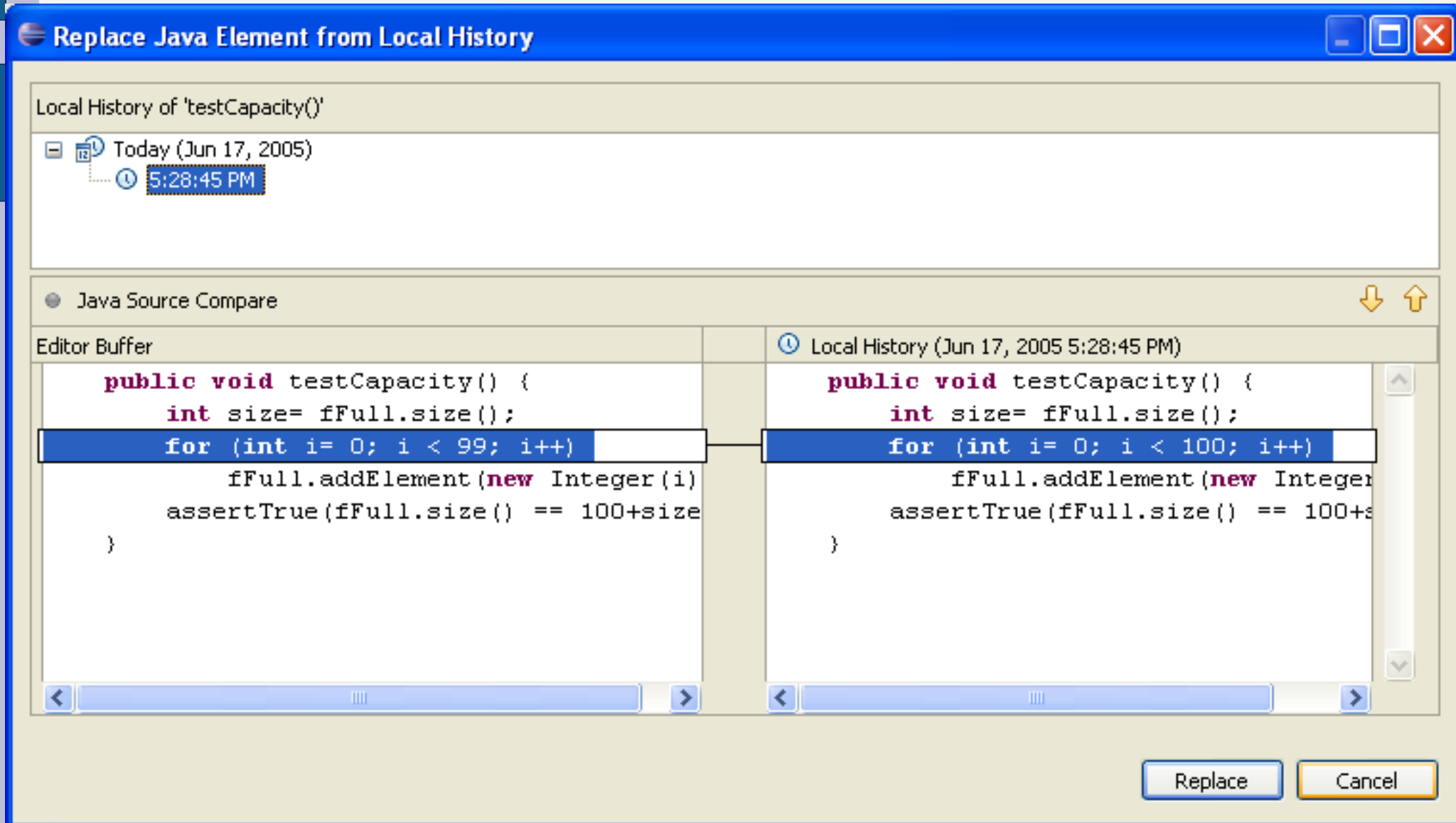
/**
 * A sample test case, testing <code>java.util.Vector</code>.
 *
 */
public class VectorTest extends TestCase {
    protected Vector fEmpty;
    protected Vector fFull;
```


Utiliser l'historique

- Eclipse gère l'historique des modifications par éléments : (méthodes, classes) à chaque sauvegarde
- Il est possible de comparer la version actuelle avec une précédente
- Il est possible de récupérer une version antérieure
- Bouton droit sur un élément (outline) -> Menus :
 - Compare With -> (comparaison)
 - Replace With -> (récupération des modifications)
 - Restore From Local History (récupérations d'éléments effacés)

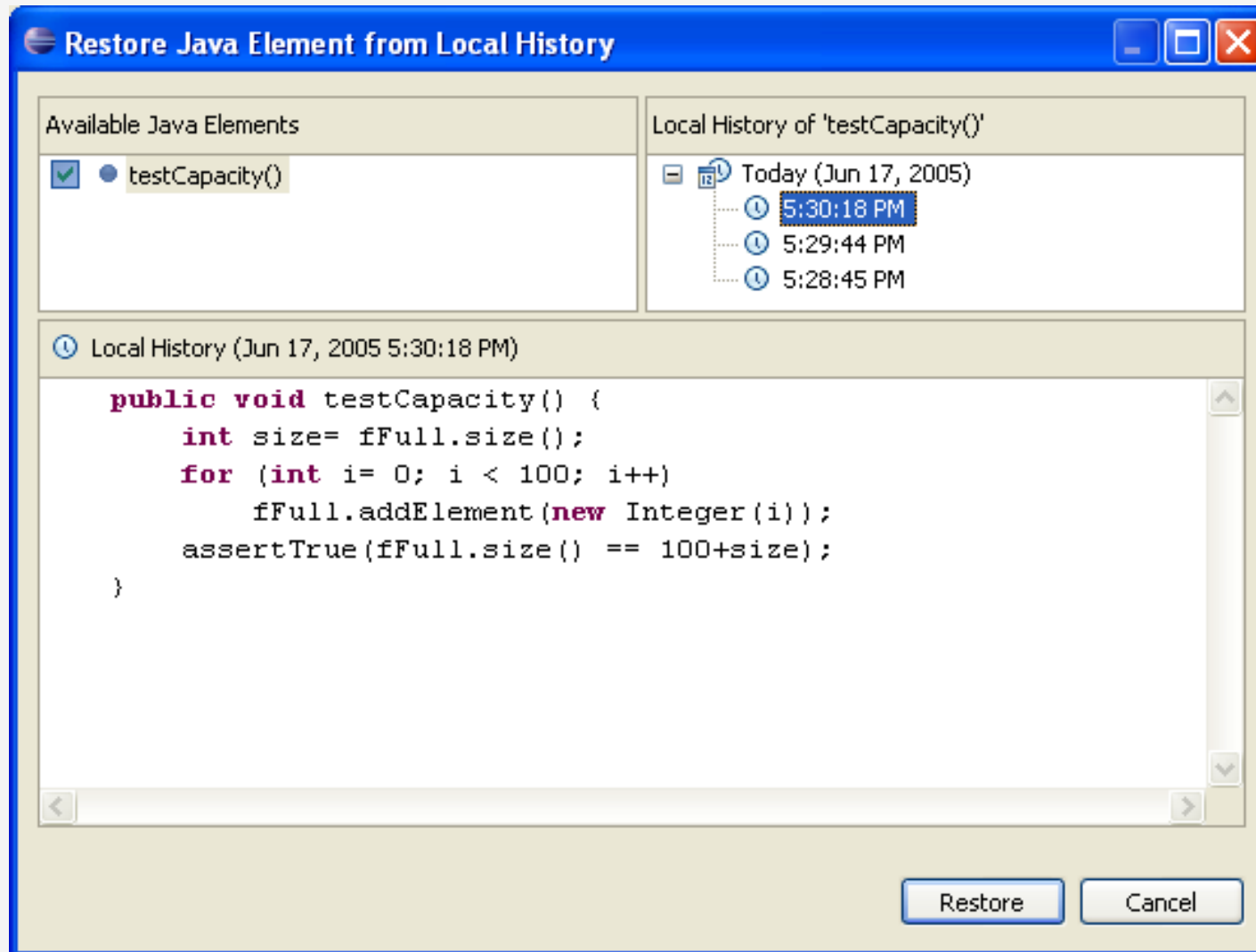
« Undo » sur un élément

- Replace With :



« Undo » pour récupérer des éléments effacés

- Restore from local history :



Extraction d'une méthode

- Souvent, on souhaite définir une nouvelle méthode à partir d'un morceau de code existant :
 - lisibilité
 - factorisation
 - etc.
- Menu refactor -> extract method (Alt+shift+M)
- Après sélection du code source concerné

Extract Method ✖

Method name:

Access modifier: public protected default private

Parameters:

Type	Name
Class	theClass

Edit...

Up

Down

- Add thrown runtime exceptions to method signature
- Generate method comment
- Replace duplicate code fragments

Method signature preview:

```
private void collectedInheritedTests(final Class  
theClass)
```

preview

Extract Method

Changes to be performed

- TestSuite.java - JUnit/junit/framework
 - TestSuite
 - TestSuite(Class)
 - substitute statement(s) with call to collectedInheritedTests
 - add new method collectedInheritedTests

TestSuite.java

Original Source	Refactored Source
<pre> } 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]); superClass= superClass.getSuperclass(); } if (fTests.size() == 0)</pre>	<pre> } if (!Modifier.isPublic(modifier)) addTest(warning("Class " + name + " is not public")); return; } collectedInheritedTests(); if (fTests.size() == 0) addTest(warning("No tests found for " + name)); }</pre>

Preview > OK Cancel

-> outline est mis à jour

The image shows an IDE window with two tabs: `VectorTest.java` and `TestSuite.java`. The `TestSuite.java` tab is active, displaying the following code:

```
private void collectedInheritedTests(final Class theClass) {  
    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, theClass);  
        }  
        superClass= superClass.getSuperclass();  
    }  
}
```

Below the code, there is a Javadoc comment:

```
/**  
 * Constructs an empty TestSuite.  
 */
```

On the right side, the Outline view is open, showing the class hierarchy and methods. The `TestSuite` class is expanded, and the `collectedInheritedTests(Class)` method is highlighted in blue. The Outline view also shows other methods like `createTest(Class, String)`, `exceptionToString(Throwable)`, `getTestConstructor(Class)`, `warning(String)`, `TestSuite()`, `TestSuite(Class)`, `TestSuite(Class, String)`, `TestSuite(String)`, `addTest(Test)`, `addTestMethod(Method, Vector)`, `addTestSuite(Class)`, `countTestCases()`, and `netName()`.

Options de création d'une classe

- Une grande partie du code peut être écrite à la création de la classe :
 - superclass (extends)
 - package
 - modificateurs (final abstract)
 - interfaces implémentées (implements)
 - fonction main
 - constructeur et méthodes abstraites héritées
 - commentaires



Java Class

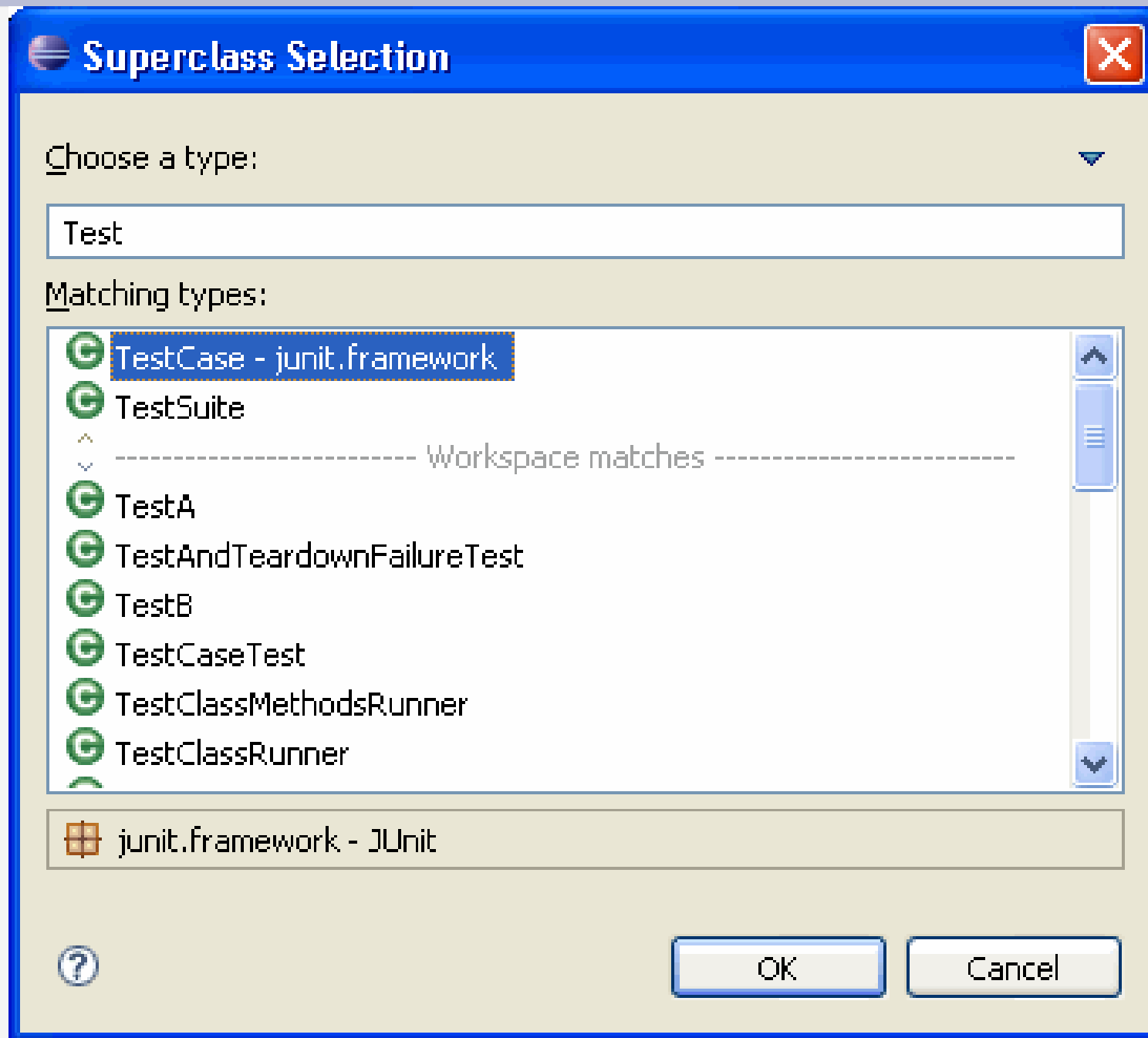
Create a new Java class.

Source folder: Package: Enclosing type: Name: Modifiers: public default private protected abstract final staticSuperclass: Interfaces:

Which method stubs would you like to create?

 public static void main(String[] args) Constructors from superclass Inherited abstract methodsDo you want to add comments as configured in the [properties](#) of the current project? Generate comments

Utiliser la complétion



New Java Class



Java Class

Create a new Java class.



Source folder:

Package:

Enclosing type:

Name:

Modifiers: public default private protected

abstract final static

Superclass:

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

Inherited abstract methods

Do you want to add comments as configured in the [properties](#) of the current project?

Generate comments



Surcharger des méthodes

```
import junit.framework.TestCase;
```

```
public class MyTestCase extends TestCase {
```

```
    public MyTestCase()  
        // TODO Auto-generated constructor stub  
    }
```

```
    public MyTestCase(  
        super(name);  
        // TODO Auto-generated constructor stub  
    }
```

- Undo Typing Ctrl+Z
- Revert File
- Save
- Open Declaration F3
- Open Type Hierarchy F4
- Open Call Hierarchy Ctrl+Alt+H
- Quick Outline Ctrl+O
- Quick Type Hierarchy Ctrl+T
- Show In Alt+Shift+W ▶
- Cut Ctrl+X
- Copy Ctrl+C
- Paste Ctrl+V

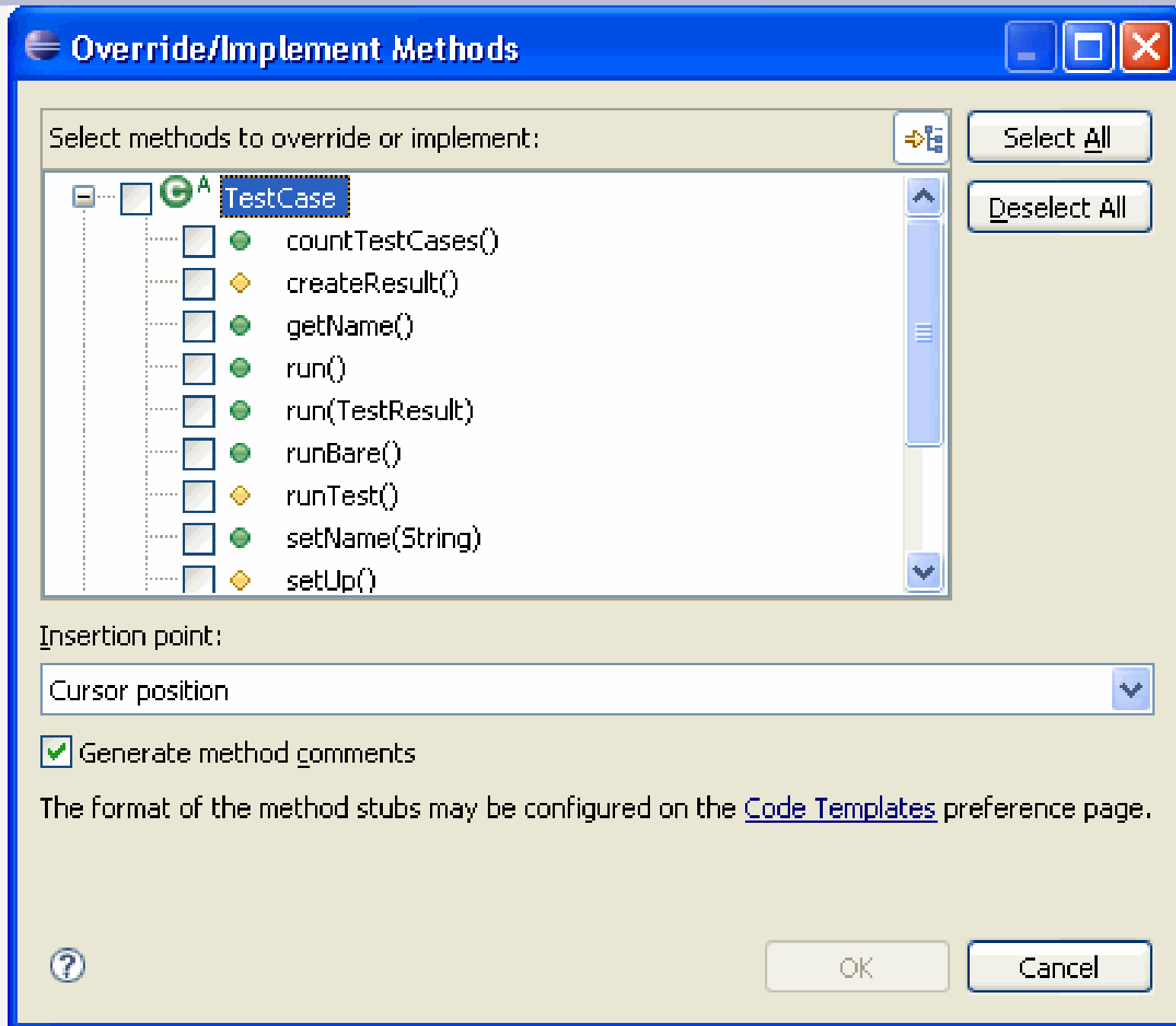
- Source Alt+Shift+S ▶
- Refactor Alt+Shift+T ▶
- Surround With Alt+Shift+Z ▶
- Local History ▶
- References ▶
- Declarations ▶
- Run As ▶
- Debug As ▶
- Team ▶
- Compare With ▶
- Replace With ▶
- Preferences...

- Toggle Comment Ctrl+/
- Add Block Comment Ctrl+Shift+/
- Remove Block Comment Ctrl+Shift+\
- Generate Element Comment Alt+Shift+J
- Correct Indentation Ctrl+I
- Format Ctrl+Shift+F
- Add Import Ctrl+Shift+M
- Organize Imports Ctrl+Shift+O
- Sort Members...
- Clean Up...
- Override/Implement Methods...
- Generate Getters and Setters...
- Generate Delegate Methods...

Warnings (1 item)

- Type safety : A generic array of `Object` is being assigned to an array of `String`.

Surcharger des méthodes



ajouter un attribut

```
protected void setUp() throws Exception {  
    container = new Vector();  
}
```

...
import java.util.Vector;
import junit.framework.TestCase;
...

- Import 'Vector' (java.util)
- Create class 'Vector'
- Change to 'VectorTest' (junit.samples)

```
protected void setUp() throws Exception {  
    container = new Vector();  
}
```

...
protected void setUp() throws Exception {
 Vector container = new Vector();
}
...

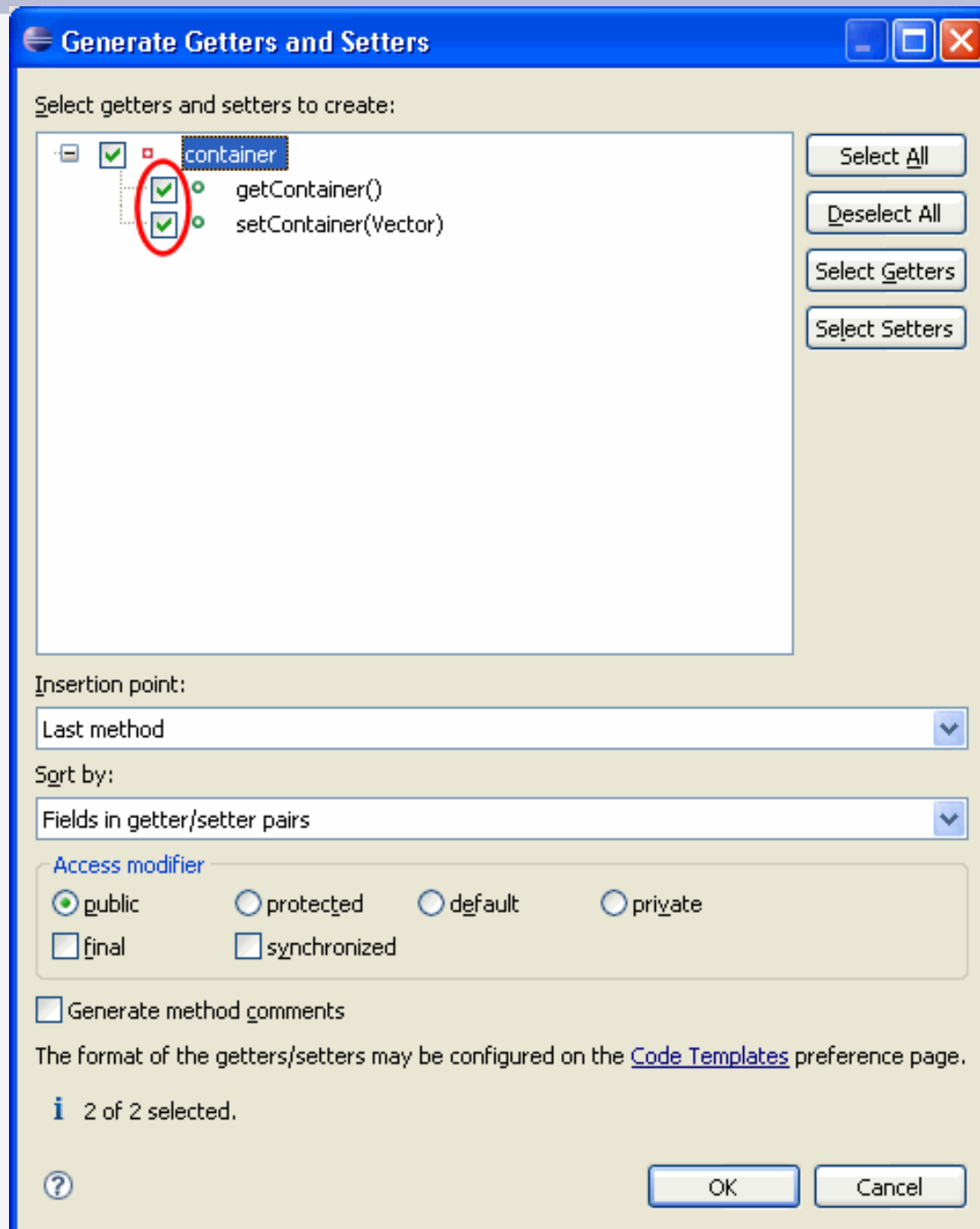
- Create local variable 'container'
- Create field 'container'
- Create parameter 'container'
- Remove assignment
- Rename in file (Ctrl+2, R direct access)

Générer les accesseurs

The image shows a screenshot of an IDE with a context menu open over a class named `TestCase`. The menu is divided into two main sections. The left section contains options for code generation and formatting, with `Generate Getters and Setters...` highlighted. The right section contains options for navigation and development, with `Source` highlighted.

Option	Shortcut
Open Type Hierarchy	F4
Copy	Ctrl+C
Copy Qualified Name	
Paste	Ctrl+V
Delete	Delete
Generate Element Comment	Alt+Shift+J
Format	
Organize Imports	Ctrl+Shift+O
Sort Members...	
Clean Up...	
Override/Implement Methods...	
Generate Getters and Setters...	
Generate Delegate Methods...	
Generate hashCode() and equals()...	
Generate Constructor using Fields...	
Generate Constructors from Superclass...	
Externalize Strings...	
Find Broken Externalized Strings	
Source	
Refactor	
References	
Declarations	
Toggle Class Load Breakpoint	
Run As	
Debug As	
Team	
Compare With	
Replace With	
Restore from Local History...	
Properties	Alt+Enter

Générer les accesseurs



Un peu de « REFACTORING »
Le menu *Refactor*

Renommer des éléments java

- Avoir des noms de variables intelligibles est TRES important !
- Sur un élément (classe, méthode, variable) :
 - bouton droit (ou menu) -> refactor -> rename
 - « Shift + Alt + R »

Renommer des éléments java

Rename Type

New name:

Update references

Update similarly named variables and methods [Configure...](#)

Update textual occurrences in comments and strings (forces preview)

Update fully qualified names in non-Java text files (forces preview)

File name patterns:

The patterns are separated by commas (* = any string, ? = any character)

< Back **Next >** Finish Cancel

Rename Type

Changes to be performed

- ✓ MoneyTest.java - JUnit/junit/samples/money
- ✓ ExceptionTestCase.java - JUnit/junit/extensions
- ✓ TestRunner.java - JUnit/junit/swingui
- ✓ VectorTest.java - JUnit/junit/samples
- ✓ TestRunner.java - JUnit/junit/awtui
- ✓ TestResult.java - JUnit/junit/framework
- ✓ MyTestCase.java - JUnit/junit/framework

MoneyTest.java

Original Source

```
public class MoneyTest extends Test
{
    private Money f12CHF;
    private Money f14CHF;
    private Money f7USD;
    private Money f21USD;

    private IMoney fMB1;
    private IMoney fMB2;

    public static void main(String
        junit.textui.TestRunner.run
    }
}
```

Refactored Source

```
public class MoneyTest extends Test
{
    private Money f12CHF;
    private Money f14CHF;
    private Money f7USD;
    private Money f21USD;

    private IMoney fMB1;
    private IMoney fMB2;

    public static void main(String
        junit.textui.TestRunner.run
    }
}
```

Preview >

OK

Cancel

« undo groupé »

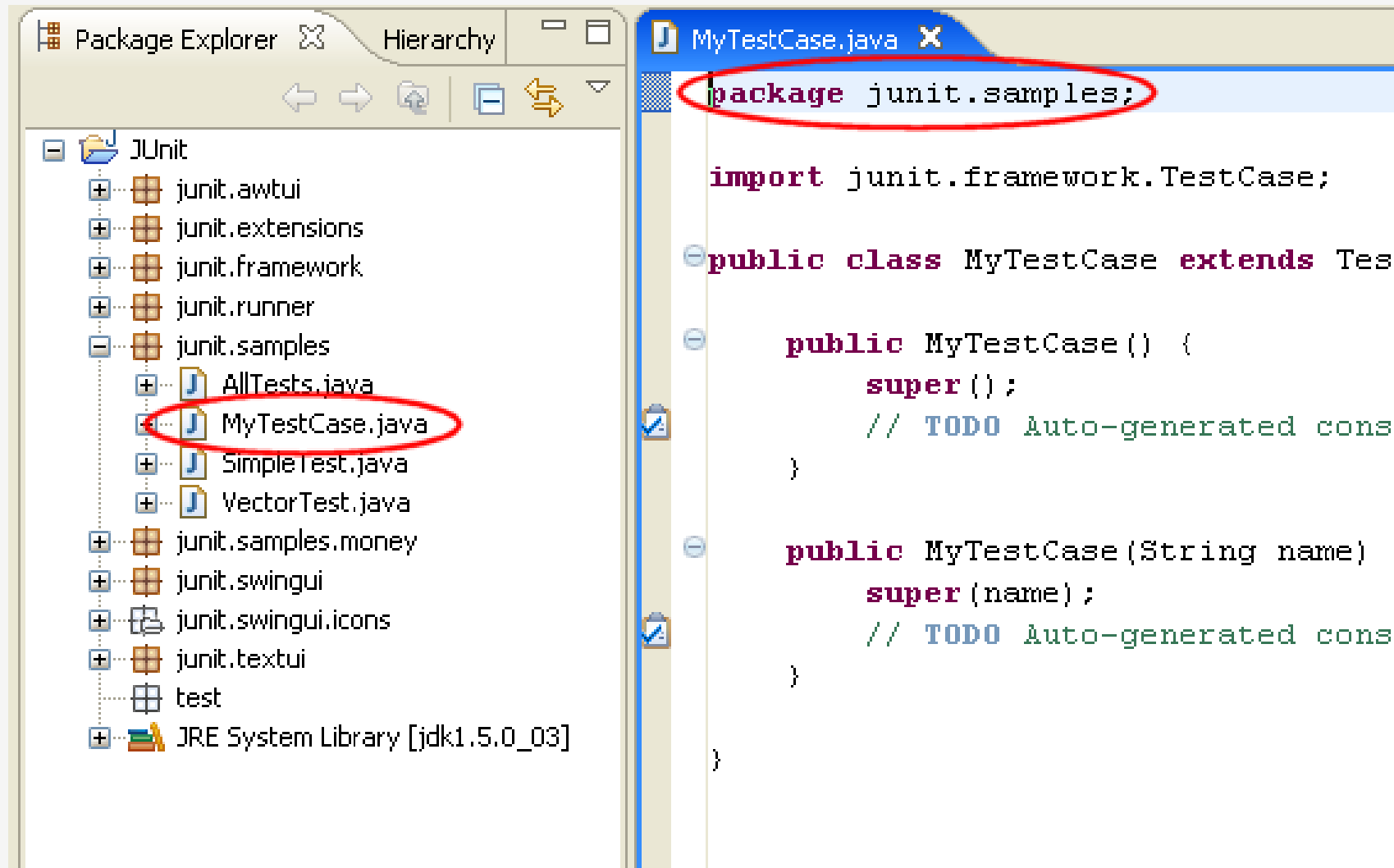
The image shows a screenshot of an IDE's menu bar and a dropdown menu. The menu bar includes 'Edit', 'Source', 'Refactor', 'Navigate', 'Search', 'Project', 'Run', and 'Window'. The 'Edit' menu is open, and the item 'Undo Rename Compilation Unit' is highlighted with a red rectangular box. The menu items and their keyboard shortcuts are as follows:

MenuItem	Shortcut
Undo Rename Compilation Unit	Ctrl+Z
Redo Typing	Ctrl+Y
<hr/>	
Cut	Ctrl+X
Copy	Ctrl+C
Copy Qualified Name	
Paste	Ctrl+V
<hr/>	
Delete	Delete
Select All	Ctrl+A
Expand Selection To	
<hr/>	
Find/Replace...	Ctrl+F
Find Next	Ctrl+K

On the right side of the screenshot, a portion of a code editor is visible, showing a file named '2.java' and some code snippets including 'pre>', 'ee T', and 'c ab'.

Changer le package d'une classe

- « Drag and drop »



The screenshot shows an IDE interface with two main panes. On the left is the Package Explorer, displaying a tree view of the project structure. The 'JUnit' package is expanded, and 'MyTestCase.java' is highlighted with a red circle. On the right is the editor window for 'MyTestCase.java'. The first line of code, 'package junit.samples;', is also circled in red. The code in the editor is as follows:

```
package junit.samples;

import junit.framework.TestCase;

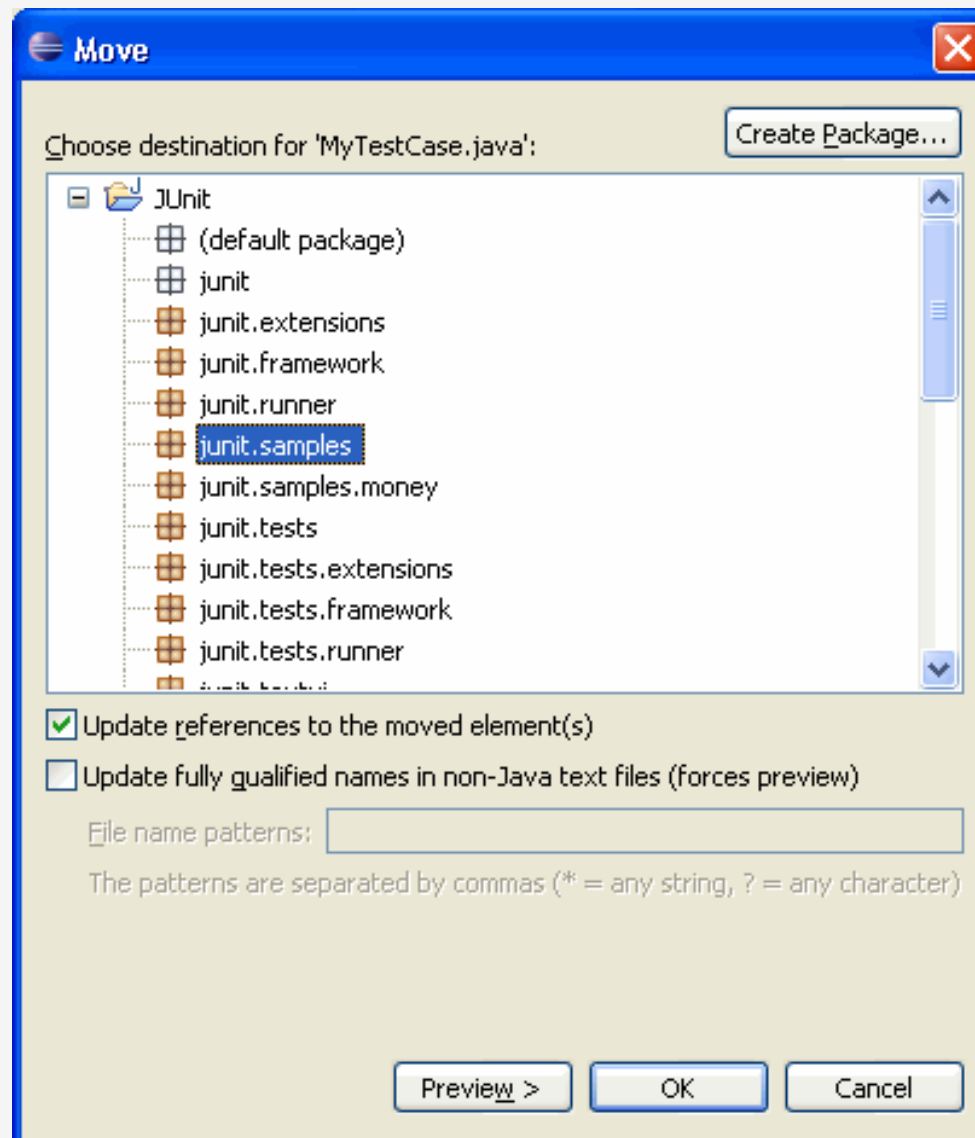
public class MyTestCase extends TestCase {

    public MyTestCase() {
        super();
        // TODO Auto-generated constructor stub
    }

    public MyTestCase(String name) {
        super(name);
        // TODO Auto-generated constructor stub
    }
}
```

Changer le package d'une classe

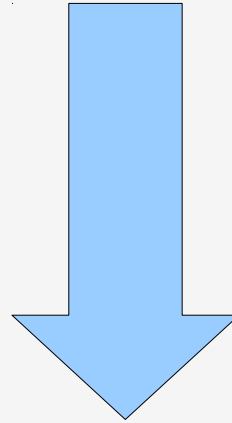
- Par le menu : refactor -> move (« shift+alt+v »)



Naviguer dans les sources

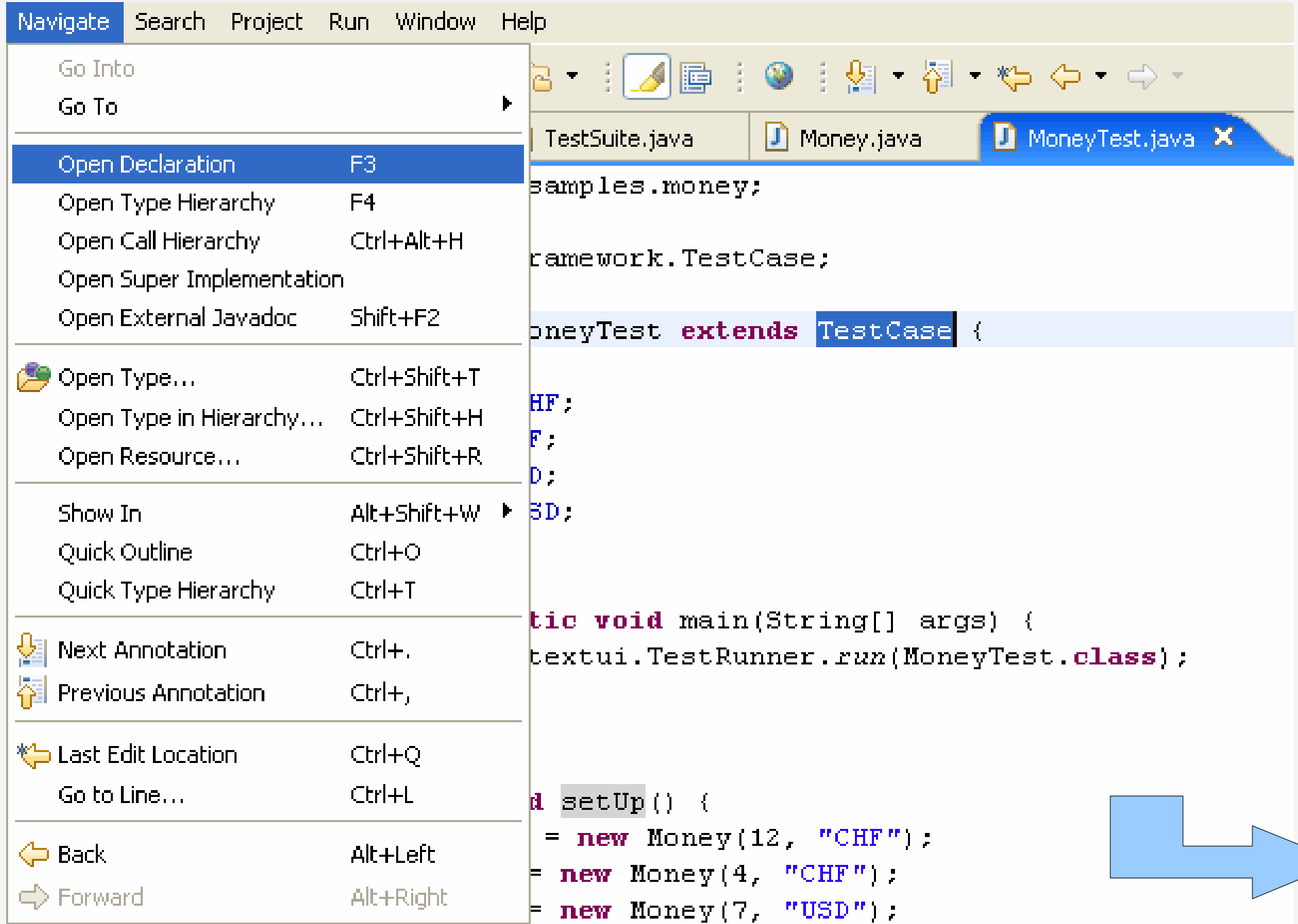
Naviguer dans les sources

- Les « vrais » programmes contiennent généralement un grand nombre de fichiers sources



- Savoir naviguer efficacement dans les sources est **fondamental**

Ouvrir la définition d'une classe



The screenshot shows an IDE window with the 'Navigate' menu open. The menu items are:

- Go Into
- Go To
- Open Declaration (F3)
- Open Type Hierarchy (F4)
- Open Call Hierarchy (Ctrl+Alt+H)
- Open Super Implementation
- Open External Javadoc (Shift+F2)
- Open Type... (Ctrl+Shift+T)
- Open Type in Hierarchy... (Ctrl+Shift+H)
- Open Resource... (Ctrl+Shift+R)
- Show In (Alt+Shift+W)
- Quick Outline (Ctrl+O)
- Quick Type Hierarchy (Ctrl+T)
- Next Annotation (Ctrl+.)
- Previous Annotation (Ctrl+,)
- Last Edit Location (Ctrl+Q)
- Go to Line... (Ctrl+L)
- Back (Alt+Left)
- Forward (Alt+Right)

The background code in 'MoneyTest.java' is:

```
samples.money;

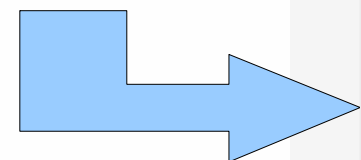
framework.TestCase;

MoneyTest extends TestCase {

    HF;
    F;
    D;
    SD;

    public void main(String[] args) {
        textui.TestRunner.run(MoneyTest.class);
    }

    public void setUp() {
        m = new Money(12, "CHF");
        m2 = new Money(4, "CHF");
        m3 = new Money(7, "USD");
    }
}
```



Ouvrir la définition d'une classe

The screenshot shows an IDE with two windows: MoneyTest.java and TestCase.java. The TestCase.java window is active and shows the following code:

```
* public static Test suite() {
*     suite.addTest(new MathTest("testAdd"));
*     suite.addTest(new MathTest("testDivideByZero"));
*     return suite;
* }
* </pre>
* @see TestResult
* @see TestSuite
*/

public abstract class TestCase extends Assert implements Test {
    /**
     * the name of the test case
     */
    private String fName;

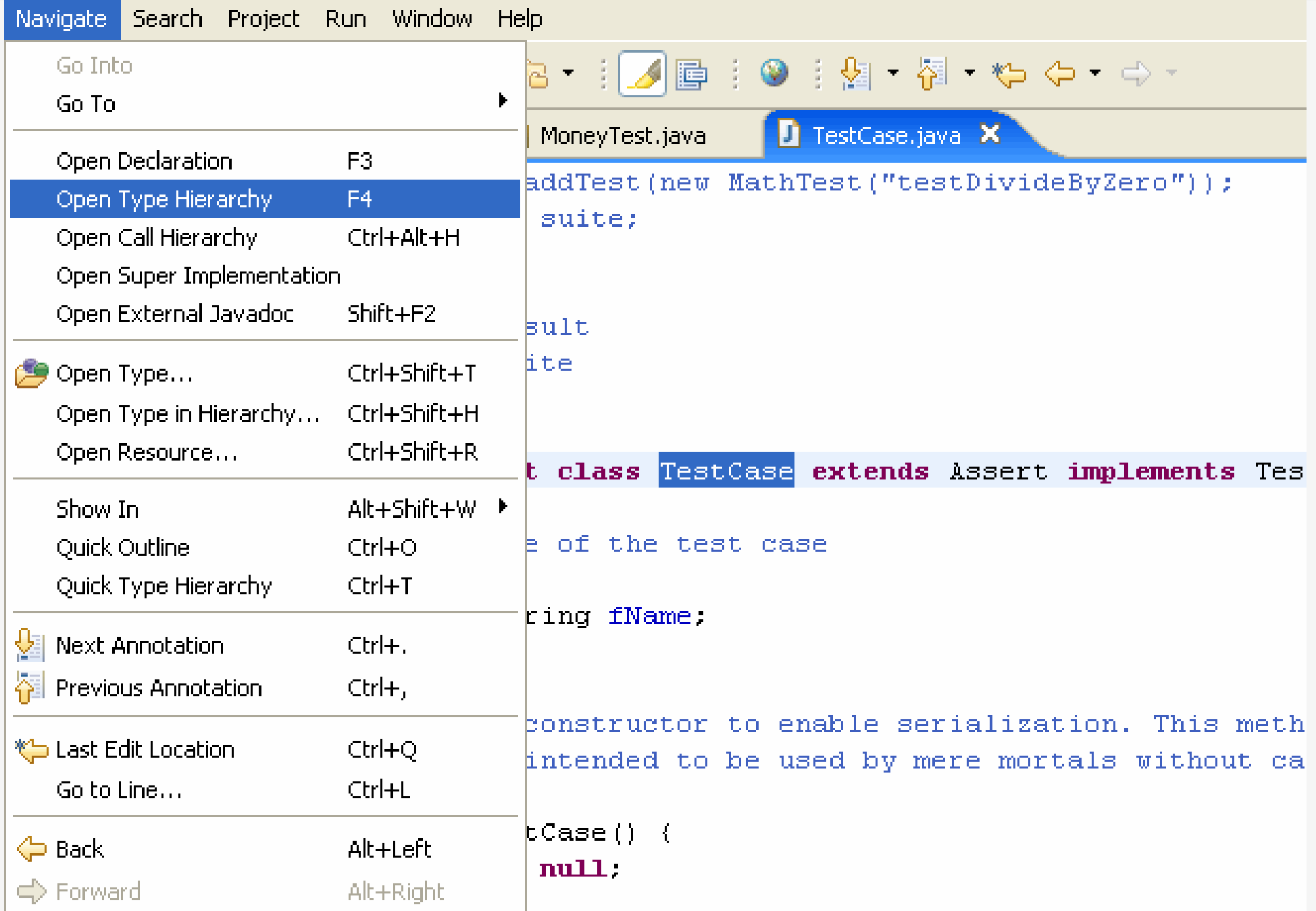
    /**
     * No-arg constructor to enable serialization. This method
     * is not intended to be used by mere mortals without calling se
     */
```

The Outline view on the right shows the following structure:

- junit.framework
- import declarations
- TestCase
 - TestCase()
 - TestCase(String)
 - countTestCases()
 - createResult()
 - getName()
 - run()
 - run(TestResult)
 - runBare()
 - runTest()
 - setName(String)
 - setUp()
 - tearDown()
 - toString()

Même résultat avec « ctrl + click »

Voir la hiérarchie des types



The image shows a screenshot of an IDE interface. The 'Navigate' menu is open, and the 'Open Type Hierarchy' option is highlighted. The background shows a Java code editor with the following code:

```
addTest (new MathTest ("testDivideByZero"));
suite;

result
ite

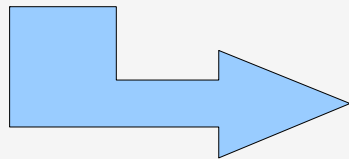
t class TestCase extends Assert implements Tes

e of the test case

ring fName;

constructor to enable serialization. This meth
intended to be used by mere mortals without ca

tCase () {
    null;
```



Package Explorer Hierarchy

TestCase

- Object
 - Assert
 - TestCase
 - ExceptionTestCase
 - MoneyTest
 - MyTestCase
 - SimpleTest
 - VectorTest
 - new TestCase() {...}

TestCase

- fName
- TestCase()
- TestCase(String)
- countTestCases()
- createResult()
- getName()
- run()
- run(TestResult)
- runBare()
- runTest()
- setName(String)
- setUp()
- tearDown()
- toString()

Naviguer avec la vue hiérarchique

The screenshot shows an IDE interface with three main components:

- Package Explorer:** Displays a tree view of the project structure under the package `JUnit`. The file `TestCase.java` is selected and highlighted with a blue border.
- Hierarchy View:** Shows the class hierarchy for `TestCase`. The class `TestCase` is highlighted in blue.
- Code Editor:** Displays the source code for `TestCase.java`. The code includes a `public static Test suite()` method and a `public abstract class TestCase` declaration. The `TestCase` class name is highlighted in blue.

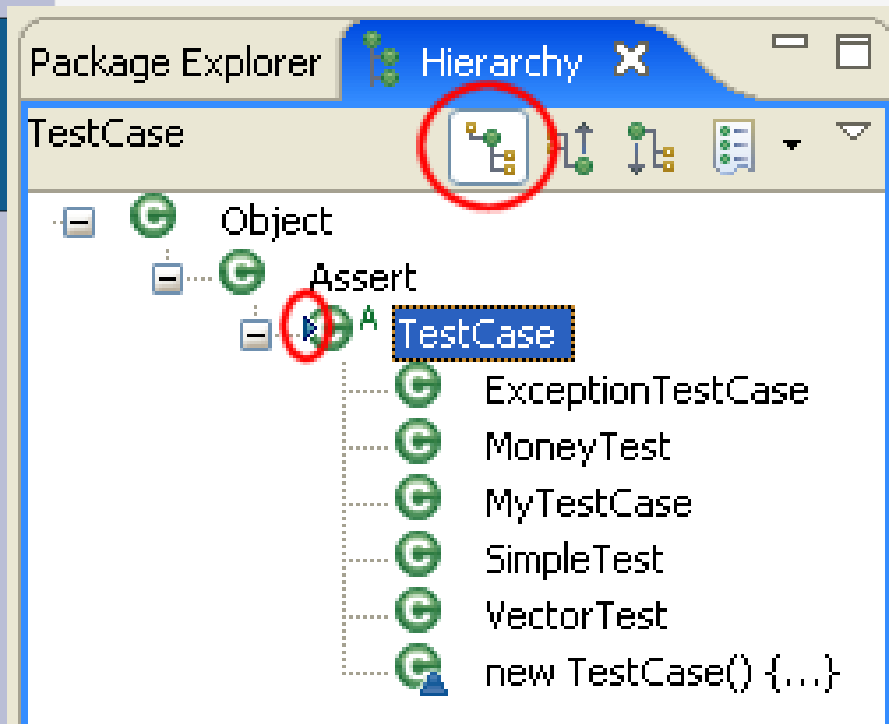
A context menu is open over the `TestCase.java` file in the Package Explorer, listing the following actions:

- New
- Open (F3)
- Open With
- Open Type Hierarchy (F4) - This option is highlighted in blue.
- Copy (Ctrl+Insert)

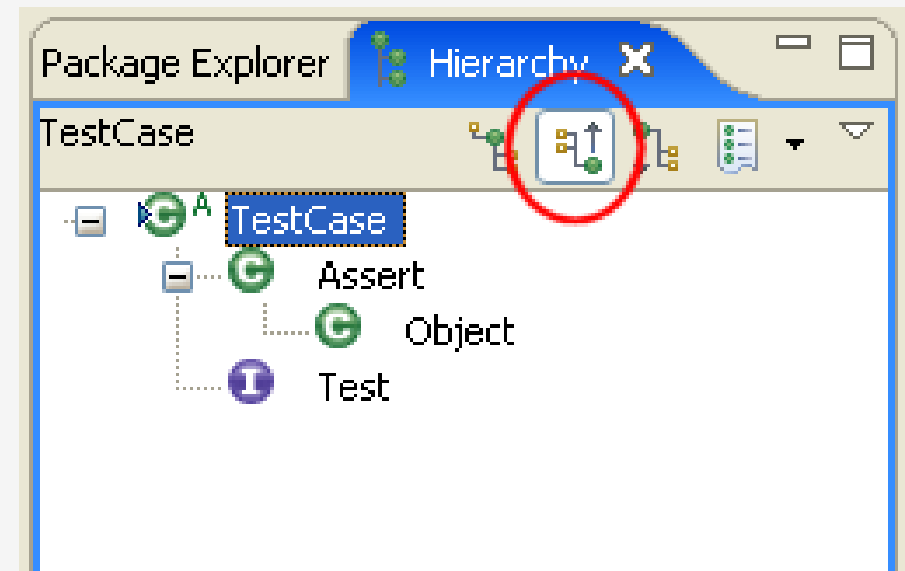
```
* public static Test suite() {  
*     suite.addTest(new MathTe  
*     suite.addTest(new MathTe  
*     return suite;  
* }  
* </pre>  
* @see TestResult  
* @see TestSuite  
*/  
  
public abstract class TestCase  
  
name of the test case  
  
String fName;  
  
xx constructor to ex
```

Naviguer avec la vue hiérarchique

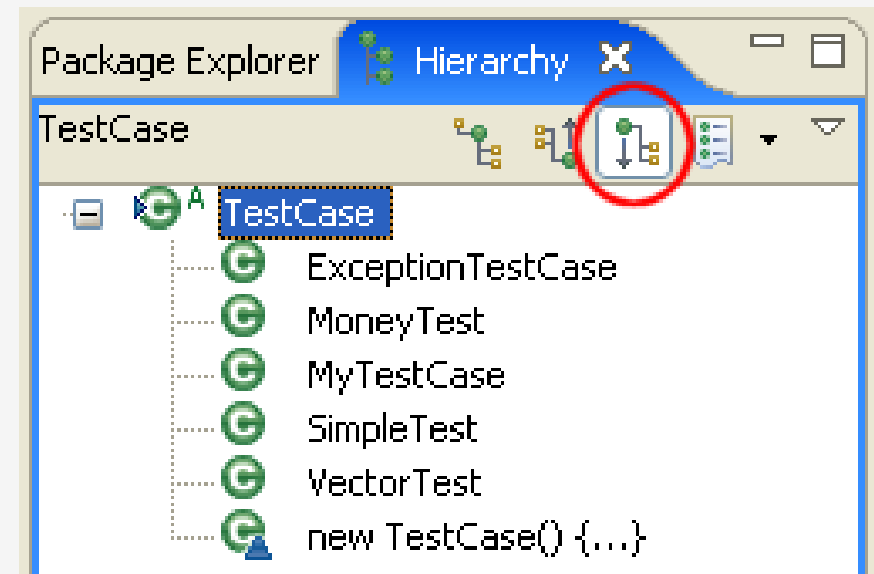
Vue globale



Vue des super types



Vue des sous types



- Quelles sous classes redéfinissent une méthode ? :

MCours.com

The screenshot shows an IDE window with two panes. The top pane, titled 'Package Explorer', displays a class hierarchy for 'TestCase'. It shows 'TestCase' at the top, with 'ExceptionTestCase' and 'new TestCase() {...}' as subclasses. Each class has a 'runTest()' method listed next to it. The bottom pane shows the 'TestCase' class with a list of its methods: 'fName', 'TestCase()', 'TestCase(String)', 'countTestCases()', 'createResult()', 'getName()', 'run()', 'run(TestResult)', 'runBare()', 'runTest()', 'setName(String)', 'setUp()', 'tearDown()', and 'toString()'. The 'runTest()' method is highlighted with a red circle. In the toolbar above the bottom pane, the 'Go to Method' icon (a blue square with a white arrow pointing to a list) is also circled in red.

- Où cette méthode est-elle définie ? :

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left shows the 'TestCase' package. The Hierarchy view on the right shows the class hierarchy for 'TestCase'. The 'countTestCases()' method is highlighted in blue in the Hierarchy view. The toolbar of the Hierarchy view has a red circle around the 'Go to Definition' icon (a blue square with a white arrow pointing right).

Package Explorer Hierarchy

TestCase

- TestCase
 - countTestCases()
 - Test
 - countTestCases()

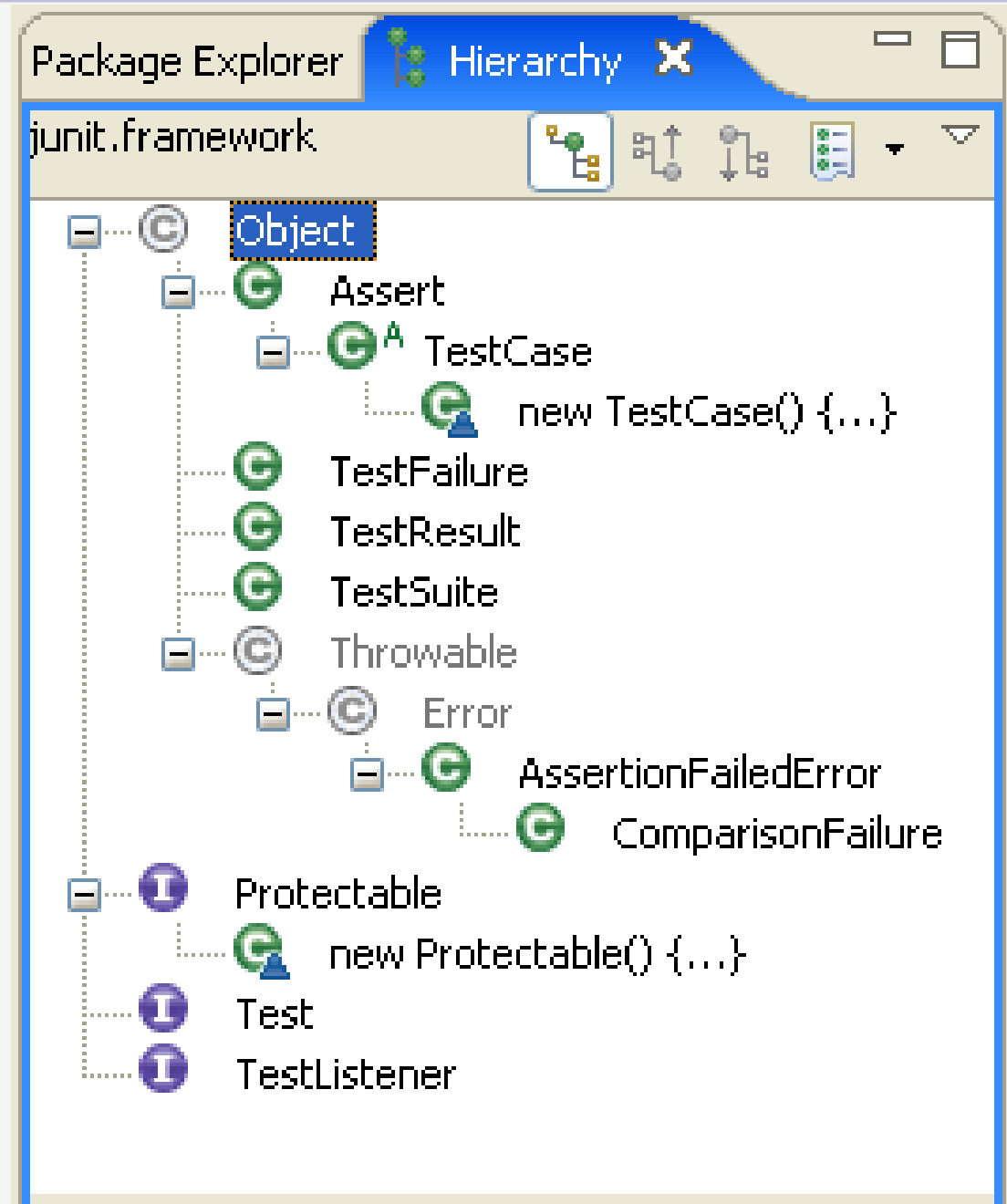
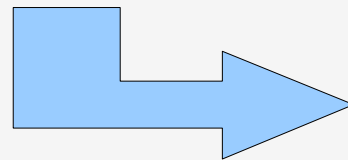
TestCase

- fName
- TestCase()
- TestCase(String)
- countTestCases()
- createResult()
- getName()
- run()
- run(TestResult)
- runBare()
- runTest()
- setName(String)
- setUp()
- tearDown()
- toString()

Changer d'élément référence dans la vue

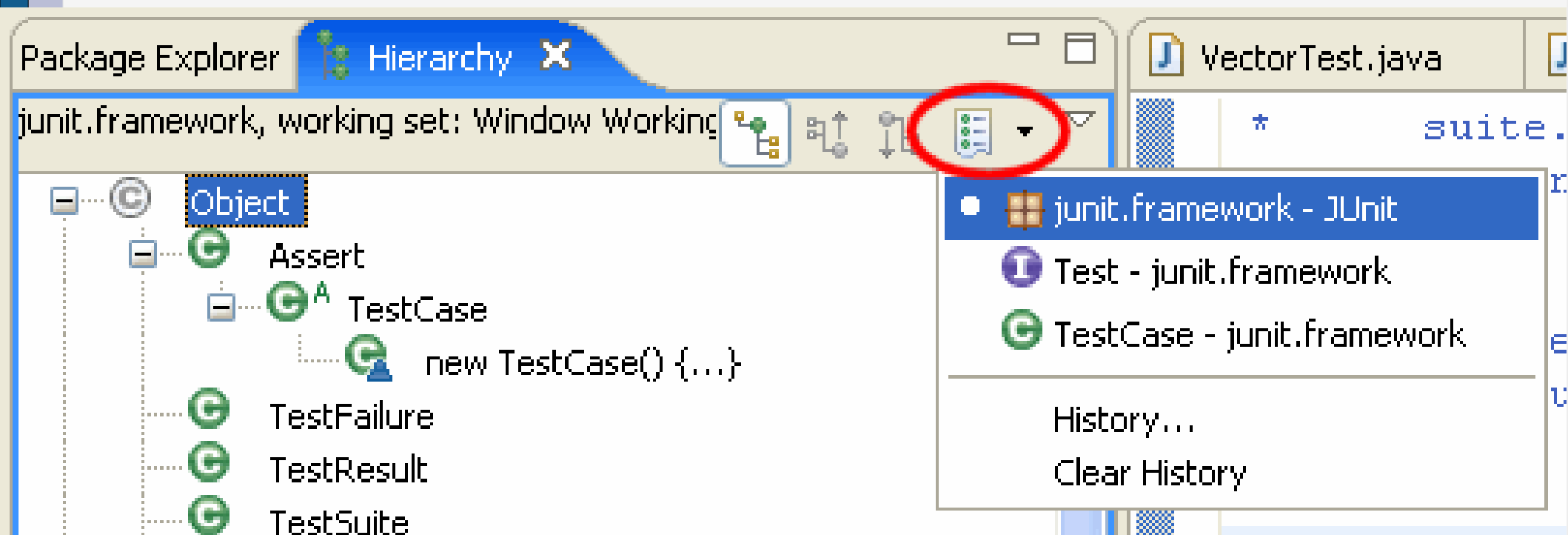
- Bouton droit sur un élément de la hiérarchie -> « focus on ... »
- Pour voir toutes les classes d'un packages avec la vue hiérarchique:
 - dans le package explorer, sur un package -> « open type hierarchy »

Naviguer avec la vue hiérarchique



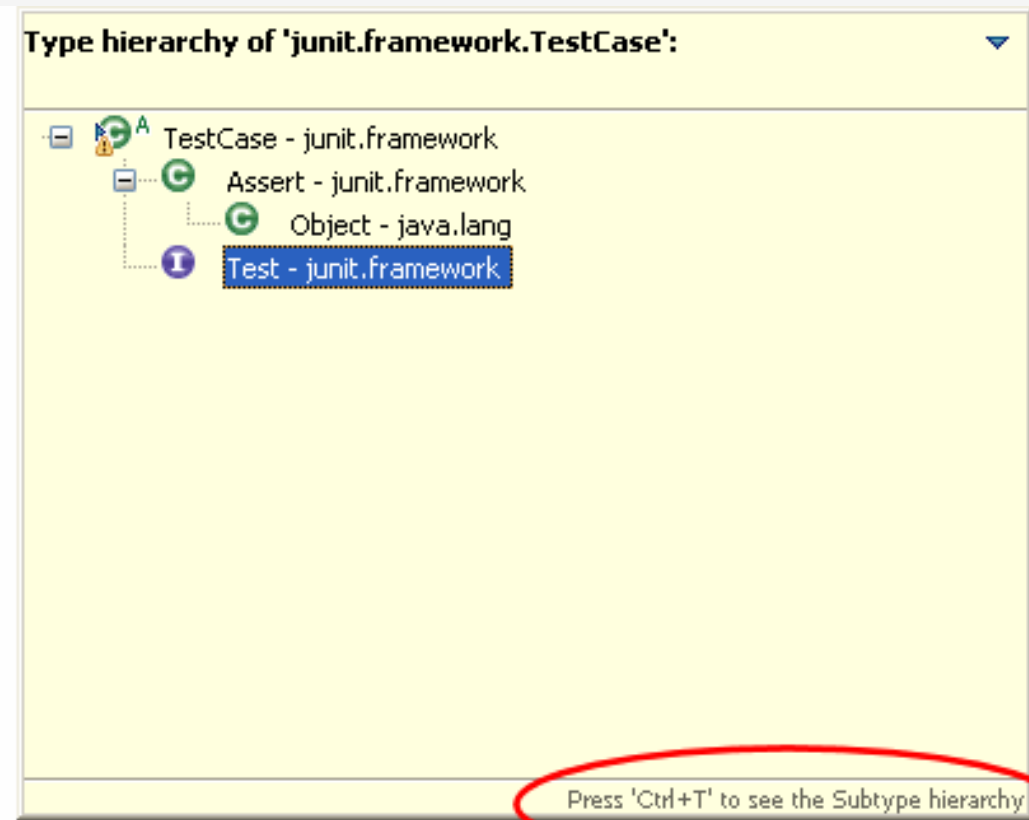
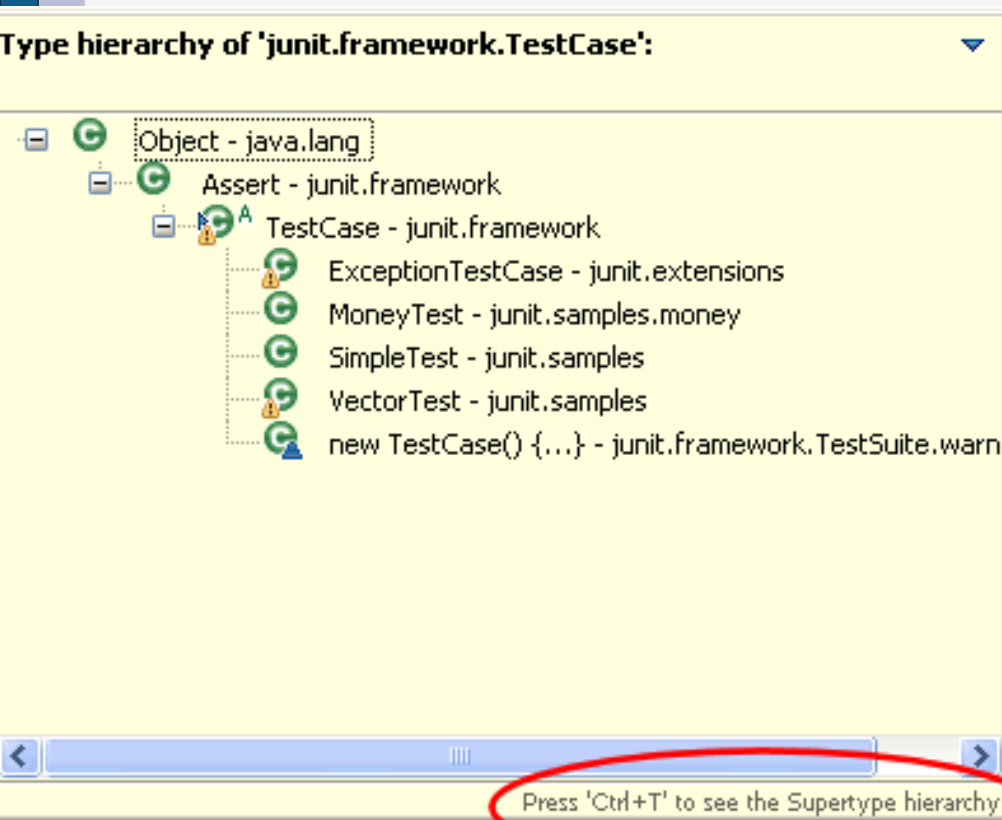
Naviguer avec la vue hiérarchique

- Retour aux éléments ouverts précédemment



« Quick type hierarchy »

- « ctrl + T » dans l'éditeur, sur le nom de la classe ou à un « endroit neutre » du fichier :

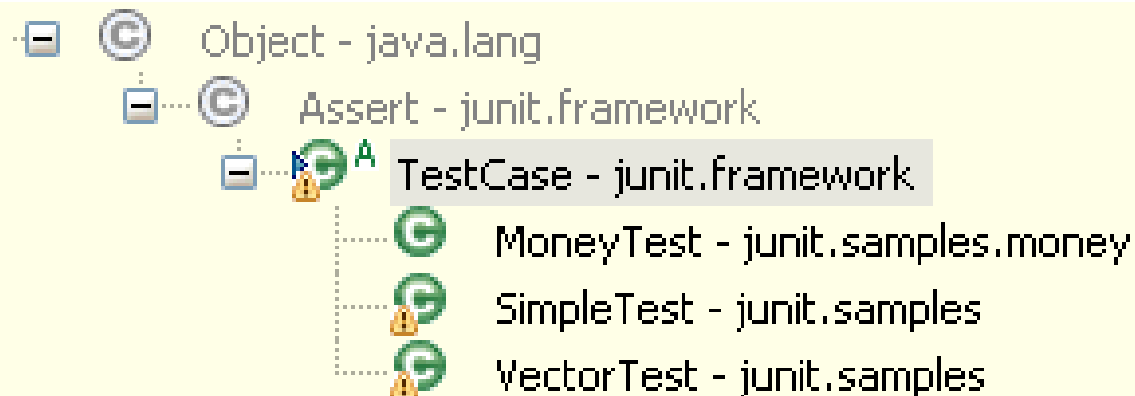


« Quick type hierarchy »

- « ctrl + T » dans l'éditeur, sur une méthode :

```
/**  
 * Runs the b  
 * @exception  
 */  
public void r  
    setUp();  
    try {  
        runTe  
    }  
    finally {  
        teard  
    }
```

Types implementing or defining 'TestCase.setUp()' ▼



Rechercher des éléments java

- Il existe plusieurs type de recherche sous Eclipse menu *search* ou « ctrl + H » :
 - Par nom de fichier
 - Par texte contenu
 - Par éléments java dans les sources (plus rapide)

Recherche des éléments java

The screenshot shows the Eclipse IDE interface with the following components:

- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Includes icons for file operations and search. A red box highlights the search icon.
- Package Explorer:** Shows the project structure for 'tamagoshi', including 'src' and 'tamagoshi.jeu' packages.
- Editor:** Displays the source code of 'Tamagoshi.java'. A search for 'Exo 6' has been performed, highlighting the following code block:

```
public boolean consommeFun(){ //Exo 6
    fun--;
    if(fun<=0){
        parler("snif : je fais une dépression, ciao!", true);
        parler("", false);
        return false;
    }
    return true;
}
```
- Outline:** Lists the class members of 'Tamagoshi', including attributes (name, generateur, age, maxEnergy, maxFun, fun, energy, lifeTime, monPanel) and methods (Tamagoshi(String), main(String[]), parle(), parler(String), parler(String, boolean), mange(), vieillit(), consommeEnergie(), consommeFun(), getAge(), getName()).
- Console:** Shows the output of the application, including a cycle count and character dialogue:

```
<terminated> TamaGameGraphic [Java Application] /usr/lib/jvm/java-1.5.0-sun-1.5.0.08/bin/java (25 janv. 07 11:13:46)
-----Cycle n01-----
Neo : "je m'ennuie à mourrir !"
Jacques : "Tout va bien !"
Zizou : "Tout va bien !"
```

Rechercher par nom de méthode

The screenshot shows the Eclipse Search dialog box with the following configuration:

- Search String:** runTest
- Case sensitive:**
- Search For:**
 - Type
 - Method
 - Package
 - Constructor
 - Field
- Limit To:**
 - Declarations
 - References
 - Implementors
 - All occurrences
 - Read accesses
 - Write accesses
- Search the JRE system libraries:**
- Scope:**
 - Workspace
 - Selected resources
 - Enclosing projects
 - Working set:

Buttons at the bottom:

Rechercher par nom de méthode

The screenshot shows an IDE's search results window. The title bar includes tabs for 'Problems', 'Javadoc', 'Declaration', and 'Search'. The search results are for the method `junit.framework.TestCase.runTest()`, showing 29 occurrences in the workspace. The results are organized into a tree view with the following structure:

- junit.framework - JUnit
 - TestCase
 - runTest()
 - TestSuite
- junit.samples - JUnit
- junit.tests - JUnit
- junit.tests.extensions - JUnit
- junit.tests.framework - JUnit

The `runTest()` method name is highlighted with a blue dashed border. A red box highlights the navigation arrows (down and up) in the toolbar above the results list.

```

public void run(TestResult result) {
    for (Enumeration e= tests(); e.hasMoreElements(); )
        if (result.shouldStop() )
            break;
    Test test= (Test)e.nextElement();
    runTest(test, result);
}

public void runTest(Test test, TestResult result) {
    test.run(result);
}

/**
 * Returns the test at the given index
 */
public Test testAt(int index) {
    return (Test)fTests.elementAt(index);
}

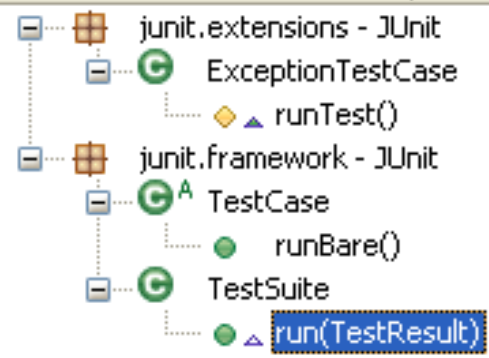
/**
 * Returns the number of tests in this suite

```

- TestSuite(String)
- addTest(Test)
- addTestMethod(Method, Vector)
- addTestSuite(Class)
- collectedInheritedTests(Class)
- countTestCases()
- getName()
- isPublicTestMethod(Method)
- isTestMethod(Method)
- run(TestResult)
- runTest(Test, TestResult)
- setName(String)
- testAt(int)
- testCount()
- tests()
- toString()

Problems Javadoc Declaration Search

'runTest' - 3 references in workspace



Recherches depuis l'éditeur

The screenshot shows an IDE with a code editor on the left and a search results window on the right. The code editor displays the following Java code:

```
Assert.java
}
/**
 * Fails a test with the given message.
 */
static public void fail(String message) {
    throw new AssertionError(message);
}
/**
 * Fails a test with no message.
 */
static public void fail() {
    fail(null);
}
/**
 * Asserts that two objects are equal. If they are not
 * an AssertionError is thrown with the given message.
 */
static public void assertEquals(String message, Object expected, Object actual) {
    if (expected == null && actual == null)
        return;
    if (expected != null && expected.equals(actual))
        return;
    failNotEquals(message, expected, actual);
}
/**
 * Asserts that two objects are equal. If they are not
 * an AssertionError is thrown with the given message.
 */
static public void assertEquals(Object expected, Object actual)
```

The search results window, titled 'References', shows the following results:

Method	Shortcut
Open Type Hierarchy	F4
Open Call Hierarchy	Ctrl+Alt+H
Cut	Ctrl+X
Copy	Ctrl+C
Copy Qualified Name	
Paste	Ctrl+V
Delete	Delete
Source	
Refactor	
References	
Declarations	
Toggle Method Breakpoint	
Run As	
Debug As	
Compare With	

The search results window also shows a list of references to the 'fail()' method, including:

- Assert 1.22 (ASCII-k)
- Assert()
- assertTrue(String)
- assertTrue(boolean)
- assertFalse(String)
- assertFalse(boolean)
- fail(String)
- fail()

```

*      suite.addTest(new MathTest("testAdd"));
*      suite.addTest(new MathTest("testDivideByZero"));
*      return suite;
*    }
* </pre>
* @see TestResult
* @see TestSuite
*/

```

public abstract class TestCase extends Assert implements Test {

```

/**
 * the name of the test
 */
private String fName;

/**
 * No-arg constructor that
 * is not intended to be
 */
public TestCase() {
    fName = null;
}

/**
 * Constructs a test case
 */
public TestCase(String
    fName = name;
}

/**
 * Counts the number of

```

- Undo Ctrl+Z
- Revert File
- Save

- Open Declaration F3
- Open Type Hierarchy F4
- Open Call Hierarchy Ctrl+Alt+H
- Quick Outline Ctrl+O
- Quick Type Hierarchy Ctrl+T
- Show In Alt+Shift+W

- Cut Ctrl+X
- Copy Ctrl+C
- Paste Ctrl+V

- Source Alt+Shift+S
- Refactor Alt+Shift+T
- Surround With Alt+Shift+Z
- Local History

- References
- Declarations
- Run As
- Debug As
- Team
- Compare With
- Replace With
- Preferences...



- Workspace Ctrl+Shift+G
- Project
- Hierarchy
- Working Set...




- junit.frame
- import d
- >TestCa
- fNe
- Test
- Test
- col
- cre
- run
- run
- run
- run
- set
- tea
- toS
- get
- set

'runTest' - 3 references in workspace (no JRE)


- >junit.framework - JUnit
- >TestCase 1.19 (ASCII-kkv)
 - runBare()
- >TestSuite 1.16 (ASCII-kkv)

Recherche de fichier

 **Search** 


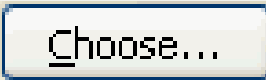
 File Search  Java Search  Plug-in Search

Containing text:

TestCase  Case sensitive

(* = any string, ? = any character, \ = escape for literals: * ? \) Regular expression

File name patterns:


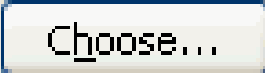
*.java  






Patterns are separated by a comma (* = any string, ? = any character)

Consider derived resources

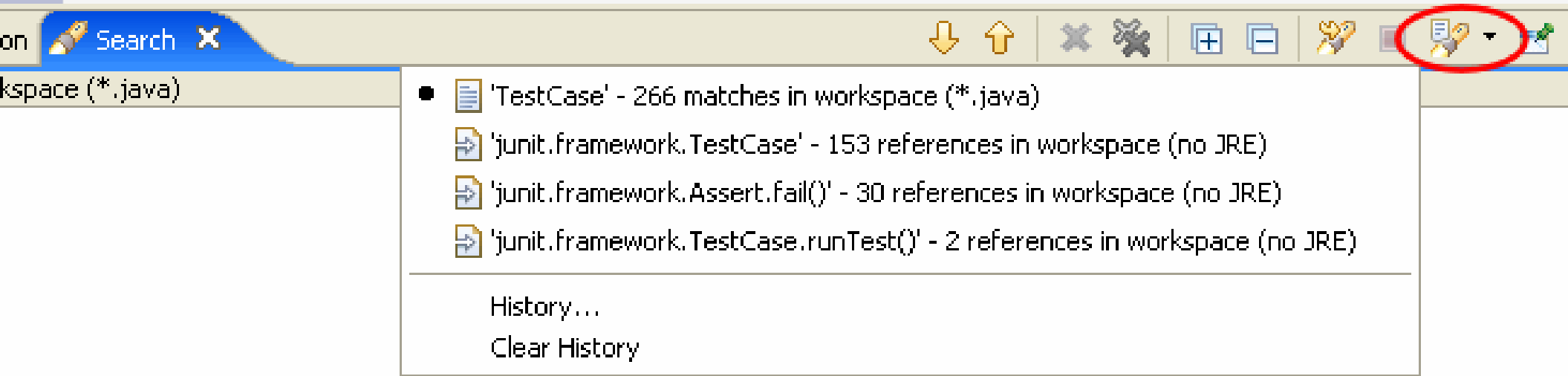
Scope

Workspace Selected resources Enclosing projects

Working set:  

Historique des recherches

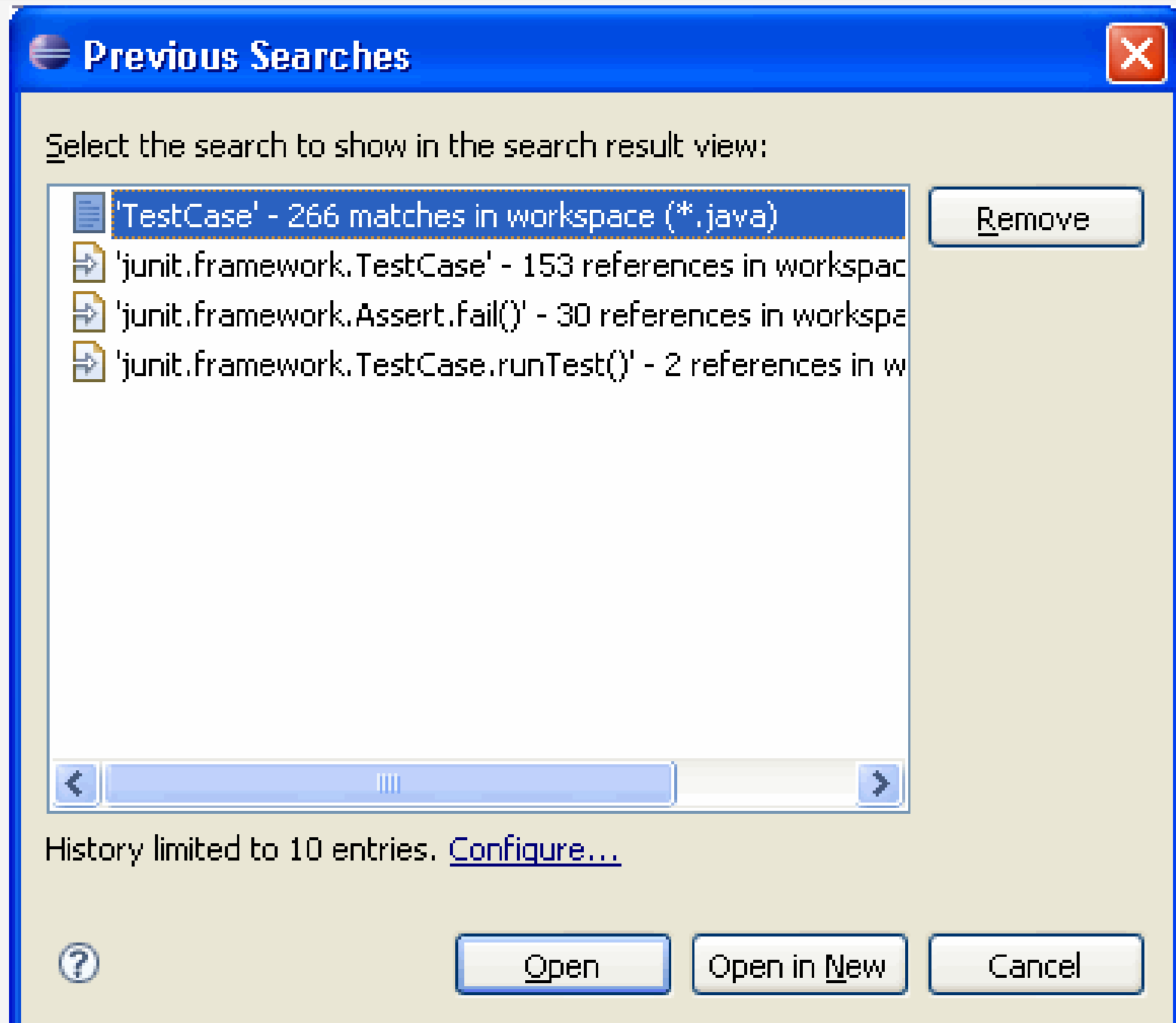


The screenshot shows an IDE search results window. The title bar includes a search icon, the text "Search", and a close button. Below the title bar, there are several icons: a downward arrow, an upward arrow, a close button, a refresh button, a plus button, a minus button, a search icon, and a search icon with a dropdown arrow. The search results are displayed in a list format, with a search icon next to each item. The results are as follows:

- 'TestCase' - 266 matches in workspace (*.java)
- 'junit.framework.TestCase' - 153 references in workspace (no JRE)
- 'junit.framework.Assert.fail()' - 30 references in workspace (no JRE)
- 'junit.framework.TestCase.runTest()' - 2 references in workspace (no JRE)

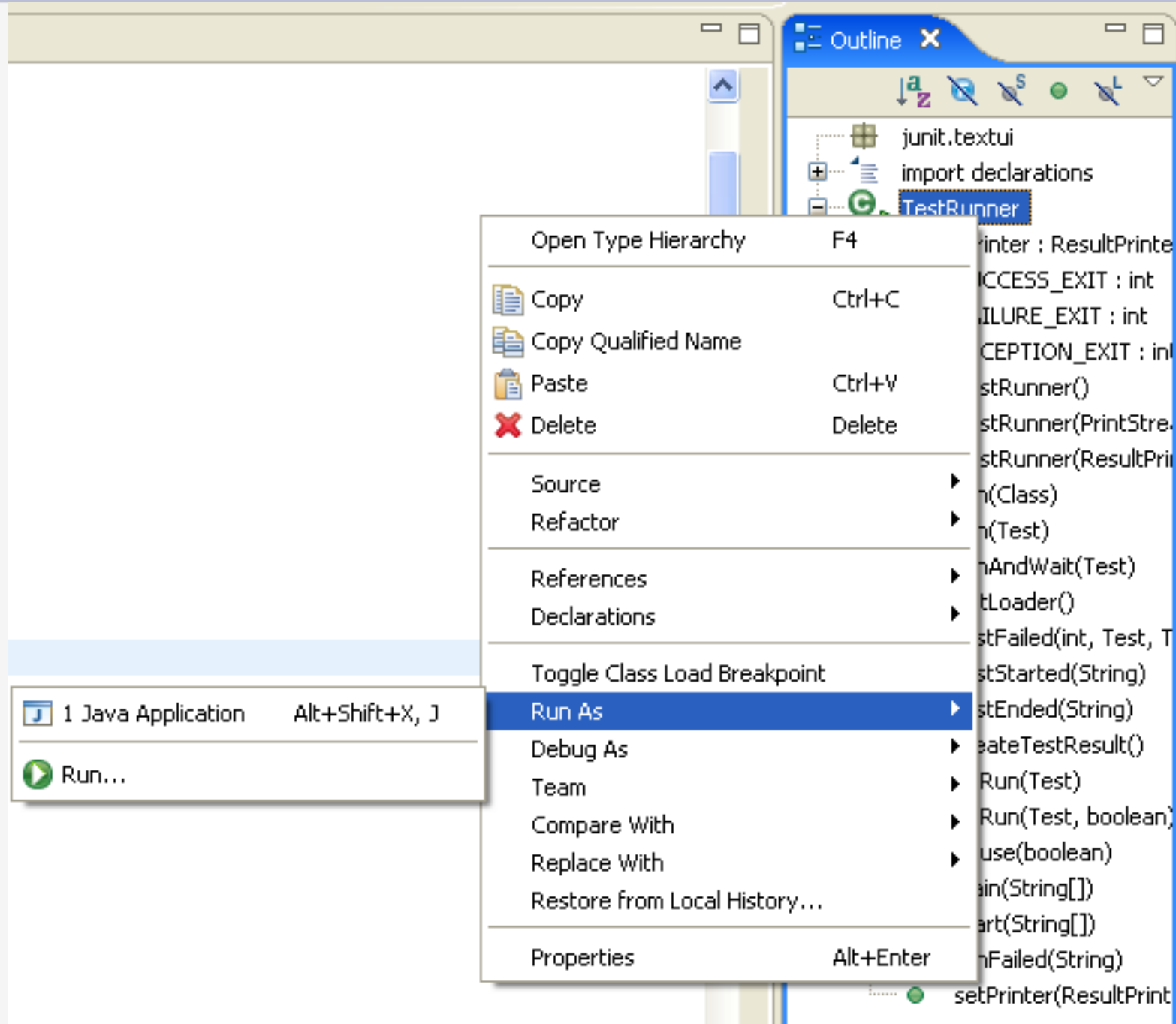
Below the list, there are two options: "History..." and "Clear History".

Historique des recherches

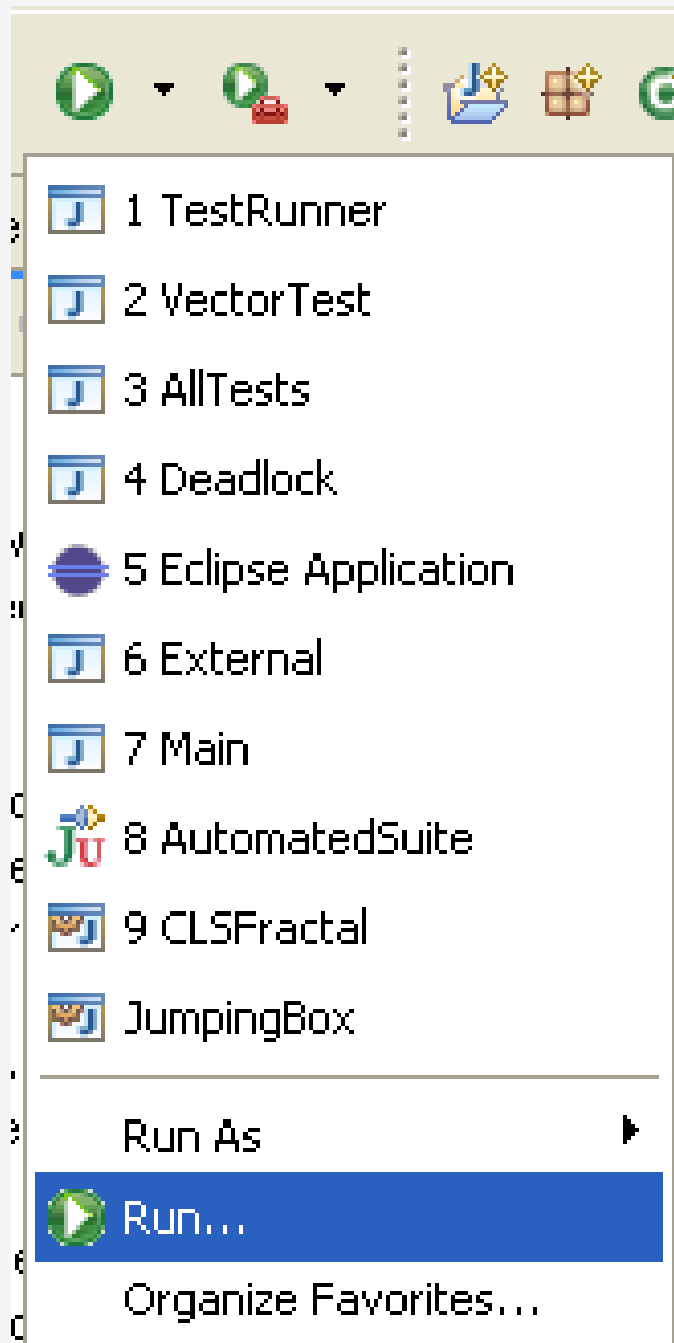


Configurer les « runs »

« Run As »



Spécifier des arguments



Spécifier des arguments

The screenshot shows the Eclipse IDE's 'Run' dialog box. The title bar reads 'Run' and the subtitle is 'Create, manage, and run configurations'. Below the subtitle, it says 'Run a Java application' next to a green play button icon.

The dialog is divided into two main sections. On the left is a tree view of configuration types, with a search box containing 'type filter text'. The tree includes categories like 'Eclipse Application', 'Equinox OSGi Framework', 'Java Applet', 'Java Application', 'JUnit', and 'SWT Application'. Under 'Java Application', 'TestRunner' is selected.

The right section is for configuration details. At the top, the 'Name' field contains 'TestRunner'. Below this is a tabbed interface with tabs for 'Main', 'Arguments', 'JRE', 'Classpath', 'Source', 'Environment', and 'Common'. The 'Main' tab is active.

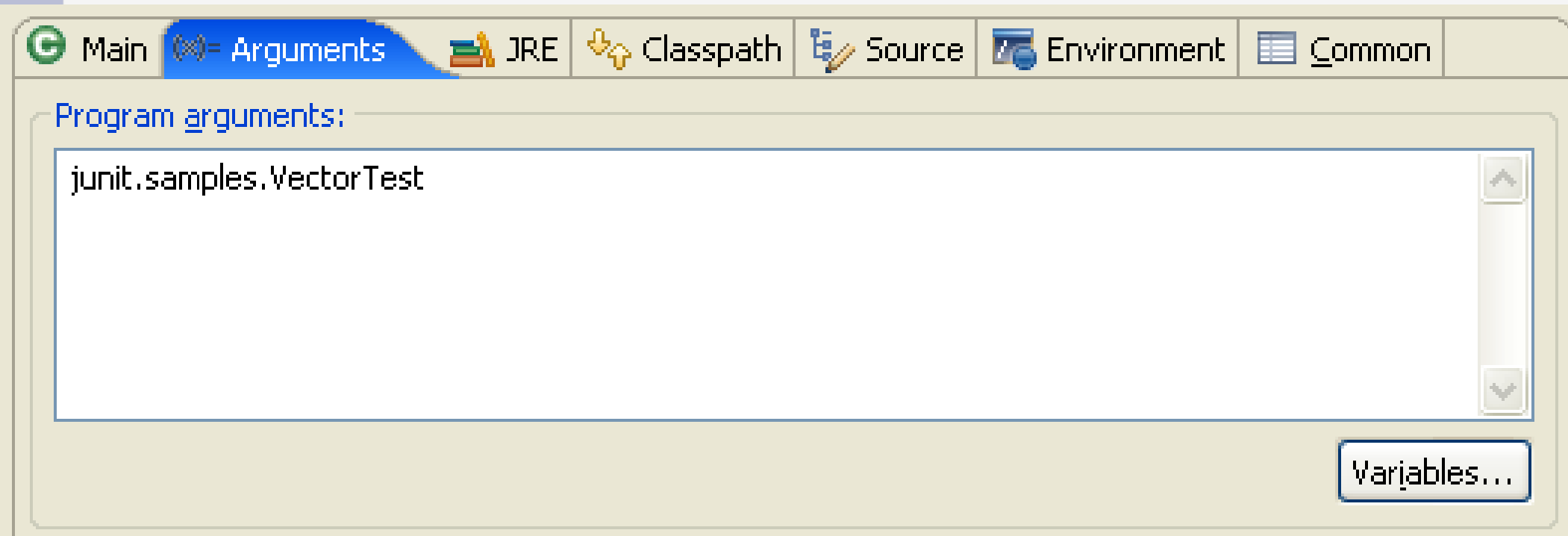
Under the 'Main' tab, there are two main fields:

- 'Project': A text box containing 'JUnit' and a 'Browse...' button.
- 'Main class': A text box containing 'junit.textui.TestRunner' and a 'Search...' button.

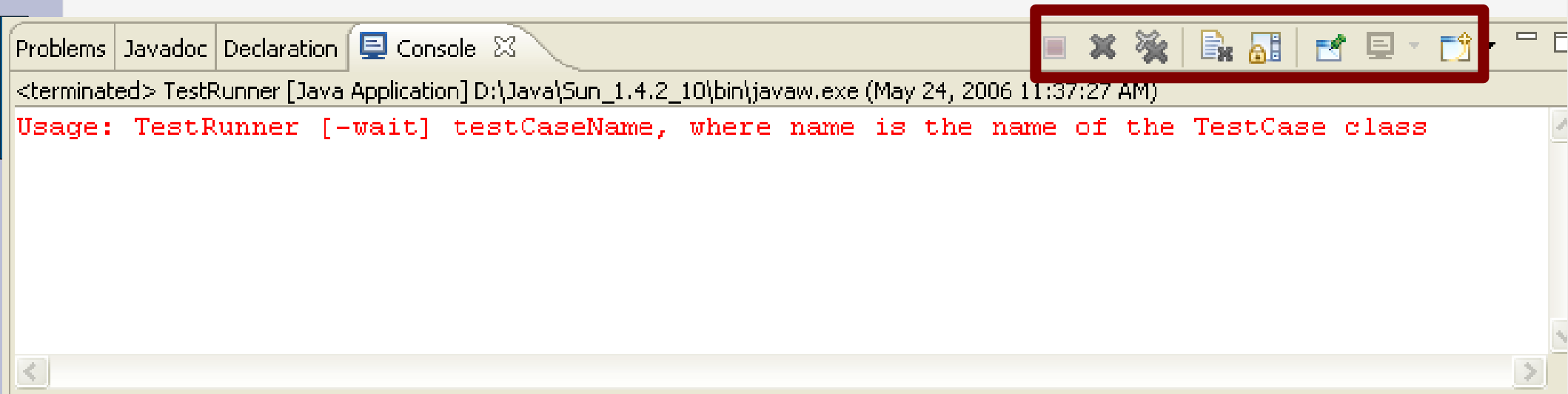
Below these fields are three checkboxes:

- Include libraries when searching for a main class
- Include inherited mains when searching for a main class
- Stop in main

Spécifier des arguments



A propos de la console



Debugger un programme

Debugger « à la main »

- Première solution :
 - Sysout ... etc . (d'où l'intérêt de redéfinir toString)
- Intérêts :
 - être sûr de la valeur d'un objet,
 - suivre l'évolution du programme, etc.
- Problèmes :
 - Il faut parfois beaucoup de sysout pour s'en sortir ...
 - Il faut tout enlever une fois debugger.
 - On ne pas remettre/enlever tous les sysout en une fois

Debugger « à la main »

- Une solution : créer, dans une classe du programme, une variable booléenne *debug* et une méthode *debug(String s)* statiques :

```
public static debug(String s){  
    if(debug) sysout(s);  
}  
  
...  
ClasseProg.debug(message);
```

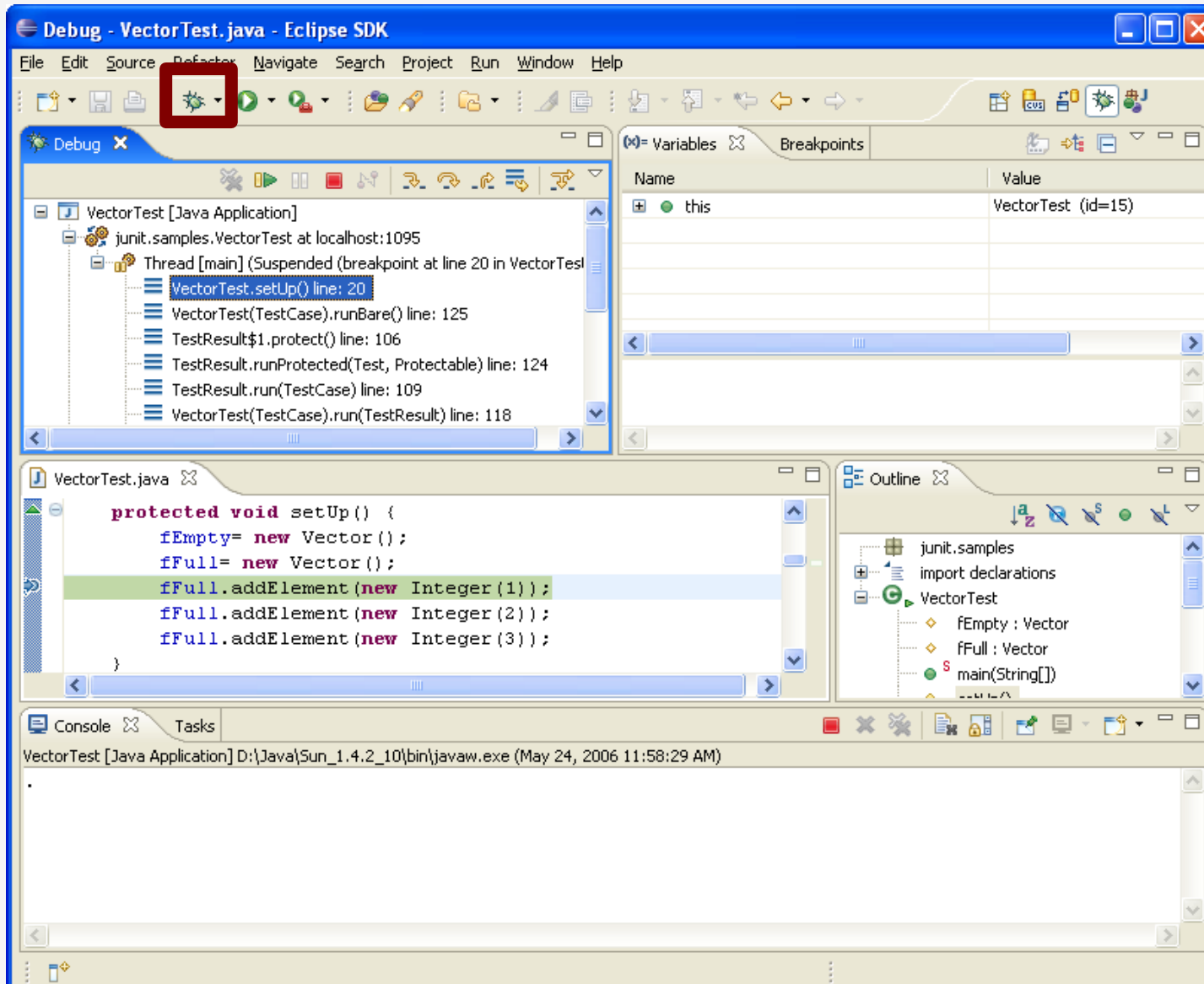
- Intérêt :
 - permet d'activer ou de désactiver le mode debug
- Problèmes :
 - On ne peut pas sélectionner les messages !
 - Trop d'information tue l'information.

Le debugger d'eclipse

- La majorité des IDEs possède un debbuger
- Caractéristiques communes :
 - fonctionne sans programmer de code supplémentaire
 - permet de poser des points d'arrêt dans le programme
 - permet de faire fonctionner le programme en pas à pas
 - permet de modifier la valeur des variables en cours d'exécution
 - etc.

Le debugger d'eclipse

- « Alt+Shift+D » -> X (java application)



Le debugger dans le détail

The screenshot shows an IDE window with the following code in `Test.java`:

```
package test;
import java.util.ArrayList;

public class Test {

    private ArrayList<Integer> myList;

    public Test() {
        myList = new ArrayList<Integer>();
    }

    public void fillList(){
        myList.add(1);
        myList.add(new Integer(4));
        myList.add(23);
    }

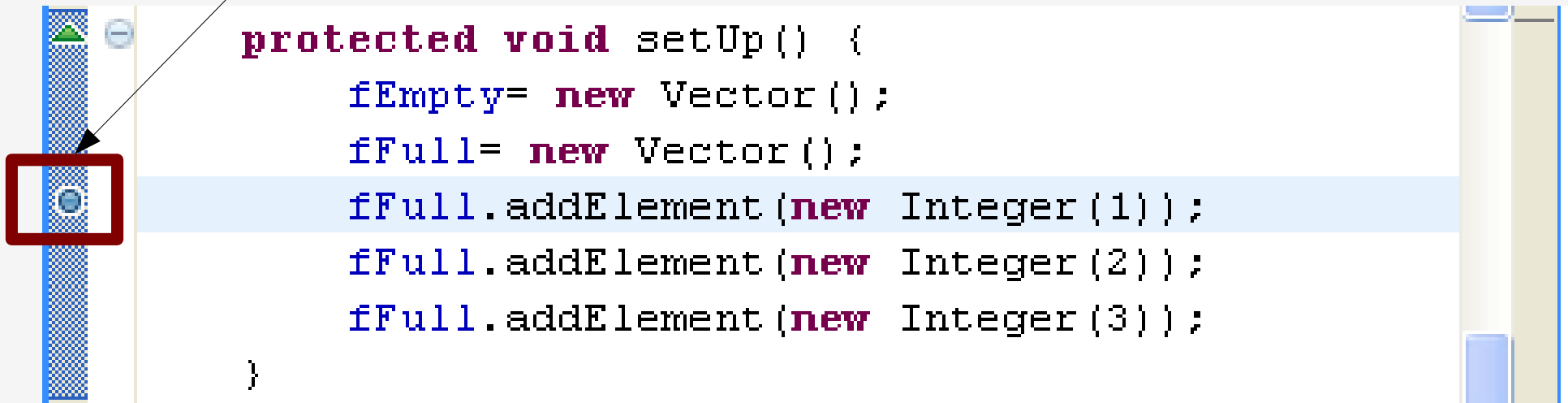
    public static void main(String[] args) {
        Test name = new Test();
        name.fillList();
    }
}
```

The `main` method is highlighted in blue, and a debugger breakpoint (red 'S' icon) is set on the line `name.fillList();`. The Outline view on the right shows the class structure with `main(String[])` selected. The bottom toolbar includes buttons for Problems, Javadoc, Declaration, PHP Browser, Console, and Search. The Console window at the bottom displays the message: `<terminated> Test (1) [Java Application] /opt/jdk1.5.0_11/bin/java (22 févr. 08 14:32:07)`.

« Alt+Shift+D » -> X (java application) → rien

Insérer un point d'arrêt

Bouton droit :
« toggle breakpoint »

A screenshot of an IDE's editor window. On the left, a vertical toolbar contains a breakpoint icon (a blue circle with a white dot) which is highlighted by a red square. An arrow points from this icon to a blue callout box containing the text 'Bouton droit : « toggle breakpoint »'. The main editor area shows a code snippet in a monospaced font. The third line of the code is highlighted in light blue. The code is as follows:

```
protected void setUp() {  
    fEmpty= new Vector();  
    fFull= new Vector();  
    fFull.addElement(new Integer(1));  
    fFull.addElement(new Integer(2));  
    fFull.addElement(new Integer(3));  
}
```

Le debugger dans le détail

```
Test.java ⌵  
  
package test;  
import java.util.ArrayList;  
  
public class Test {  
  
    private ArrayList<Integer> myList;  
  
    public Test() {  
        myList = new ArrayList<Integer>();  
    }  
  
    public void fillList(){  
        myList.add(1);  
        myList.add(new Integer(4));  
        myList.add(23);  
    }  
  
    public static void main(String[] args) {  
        Test name = new Test();  
        name.fillList();  
    }  
}
```



Debug

- Test (1) [Java Application]
 - test.Test at localhost:35038
 - Thread [main] (Suspended (breakpoint at line 20 in Test))
 - Test.main(String[]) line: 20

/opt/jdk1.5.0_11/bin/java (22 févr. 08 14:41:04)

Variables Breakpoints Expressions

Name	Value
args	String[0] (id=15)
name	Test (id=18)

```

myList = new ArrayList<Integer>();
}

public void fillList(){
    myList.add(1);
    myList.add(new Integer(4));
    myList.add(23);
}

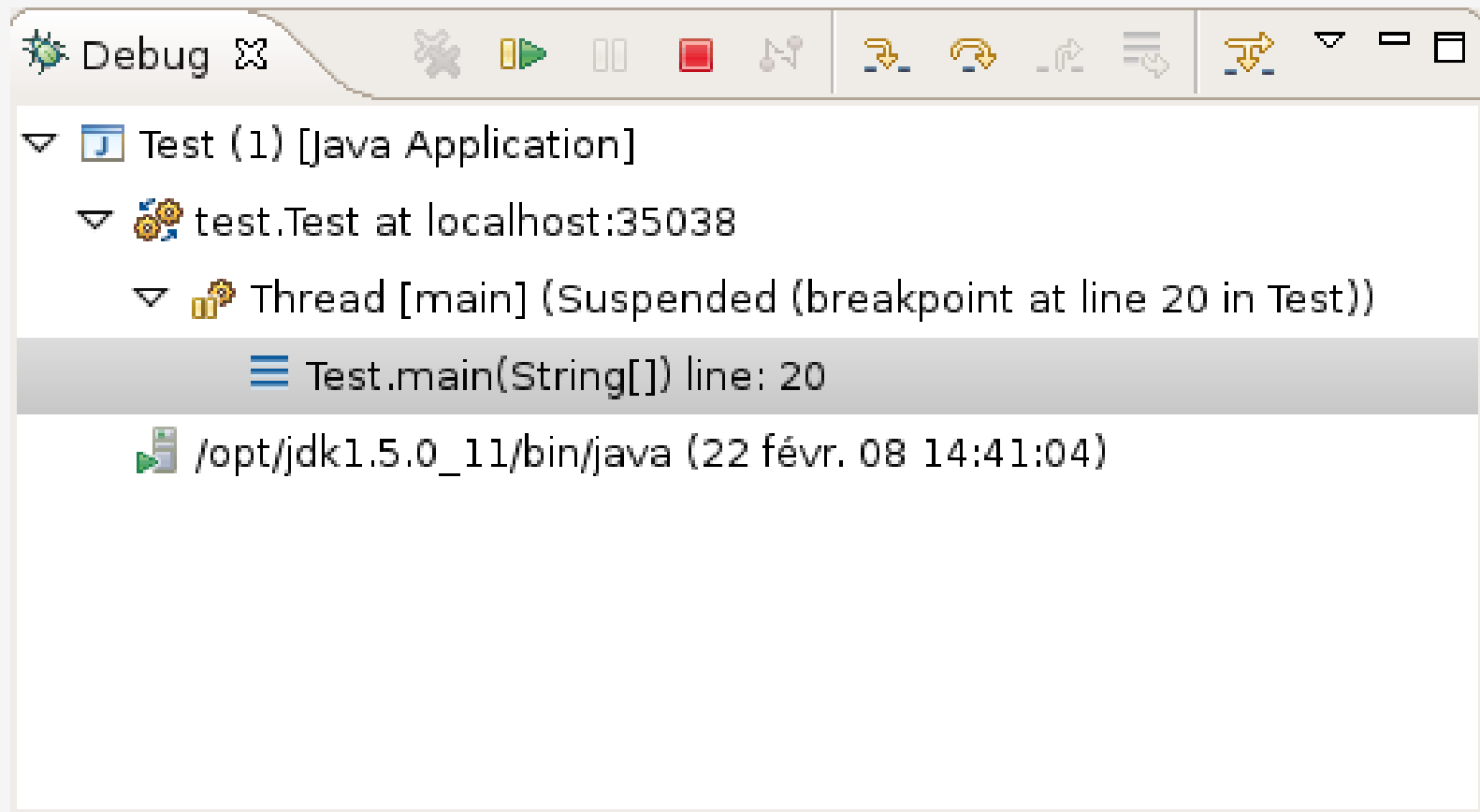
public static void main(String[] args) {
    Test name = new Test();
    name.fillList();
}
    
```

Outline

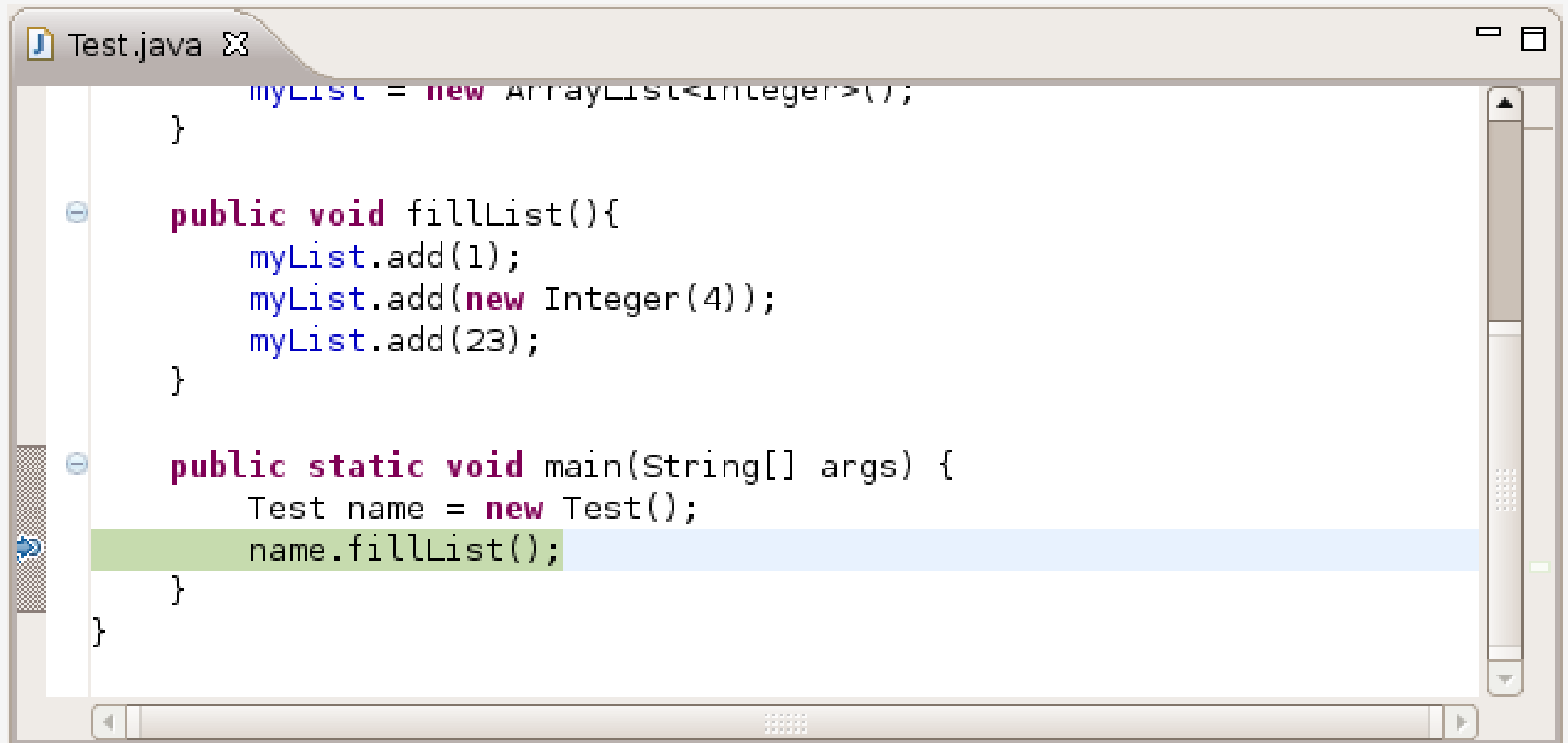
- test
 - import declarations
 - Test
 - myList : ArrayList<Integer>
 - Test()
 - fillList()
 - main(String[])

Test (1) [Java Application] /opt/jdk1.5.0_11/bin/java (22 févr. 08 14:41:04)

État de l'exécution

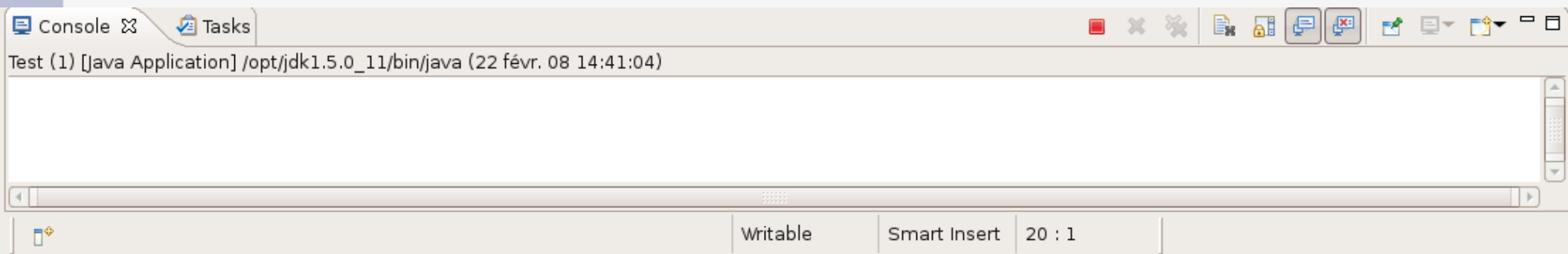


Ligne de code correspondante



```
Test.java ✕  
    myList = new ArrayList<Integer>();  
}  
  
public void fillList(){  
    myList.add(1);  
    myList.add(new Integer(4));  
    myList.add(23);  
}  
  
public static void main(String[] args) {  
    Test name = new Test();  
    name.fillList();  
}  
}
```

La console habituelle



L'état des variables

The screenshot shows a variable inspection window with three tabs: 'Variables', 'Breakpoints', and 'Expressions'. The 'Variables' tab is active. The window contains a table with two columns: 'Name' and 'Value'. The 'args' variable is shown as 'String[0] (id=15)' and the 'name' variable is shown as 'Test (id=18)'. The 'name' variable is currently selected. Below the table are two horizontal scroll bars and a vertical scroll bar.

Name	Value
args	String[0] (id=15)
name	Test (id=18)

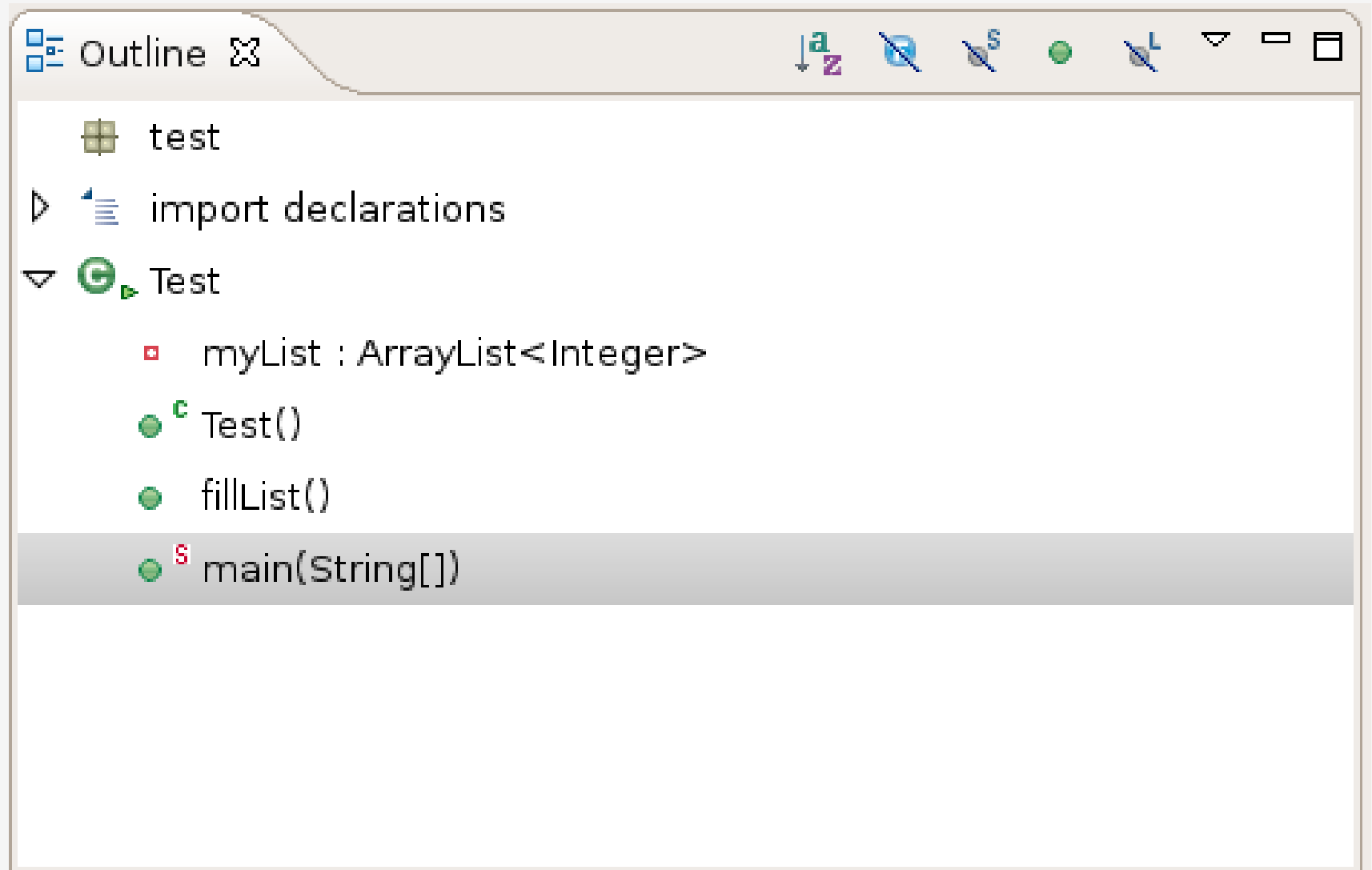
L'état des variables

The screenshot shows the 'Variables' window in an IDE. The window has tabs for 'Variables', 'Breakpoints', and 'Expressions'. The 'Variables' tab is active, displaying a table of variables and their values. Below the table is a scrollable area showing an empty array '[]'.

Name	Value
args	String[0] (id=15)
name	Test (id=18)
myList	ArrayList<E> (id=19)
elementData	Object[10] (id=29)
modCount	0
size	0

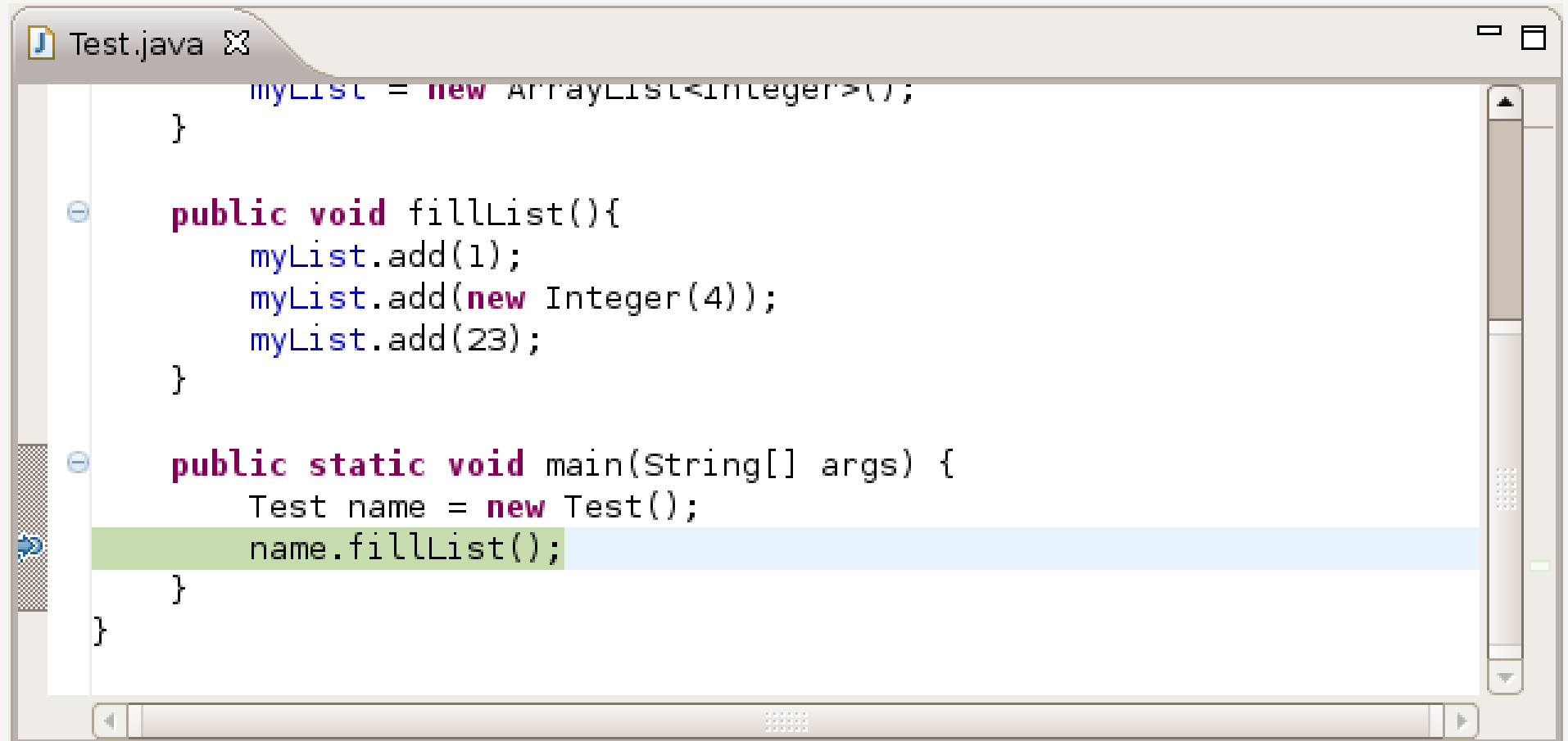
Below the table, a scrollable area displays the value of the selected variable, 'myList', which is an empty array: []

La méthode courante (outline)



Le debugger dans le détail

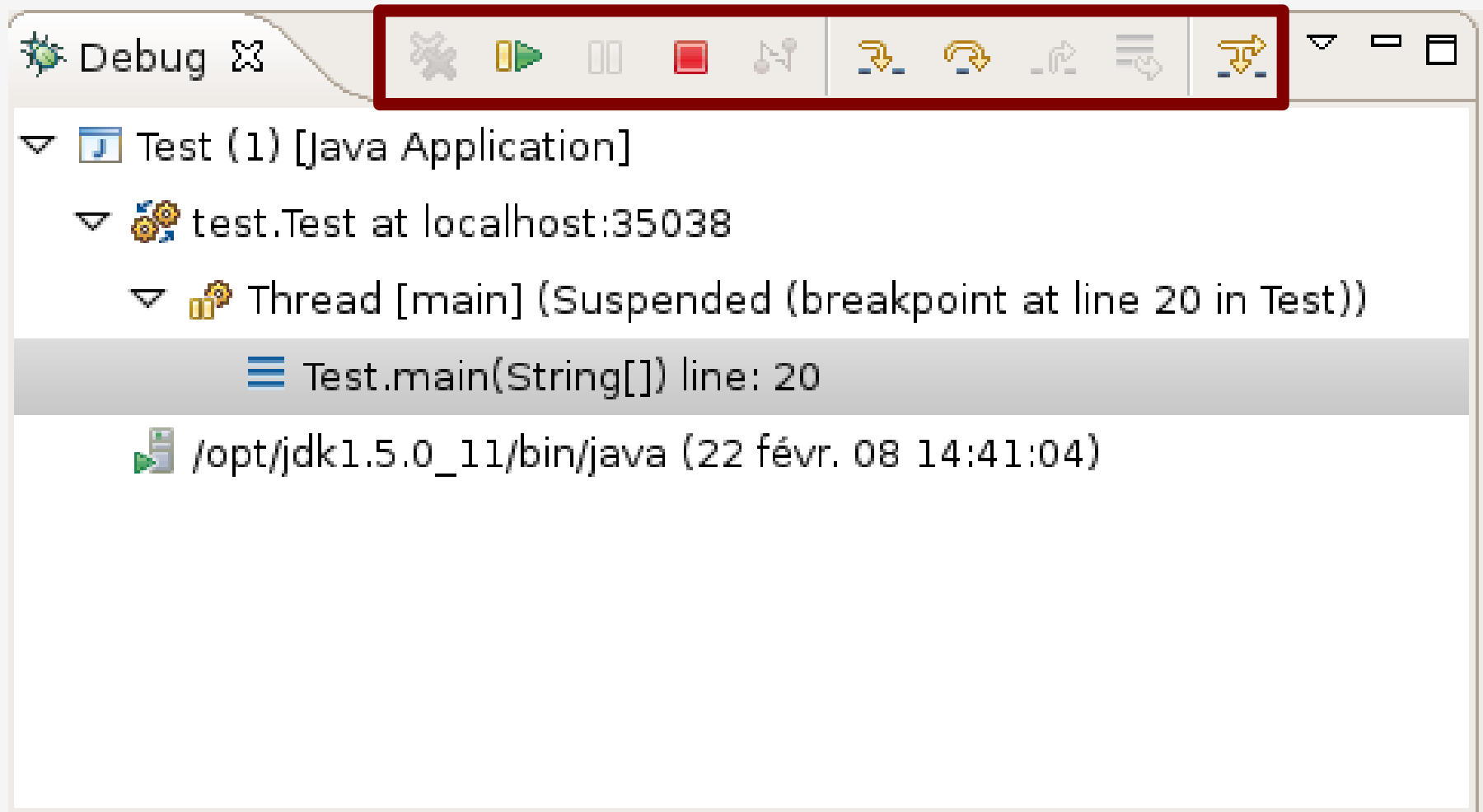
MCours.com



```
Test.java ✖  
    myList = new ArrayList<Integer>();  
  }  
  
  public void fillList(){  
    myList.add(1);  
    myList.add(new Integer(4));  
    myList.add(23);  
  }  
  
  public static void main(String[] args) {  
    Test name = new Test();  
    name.fillList();  
  }  
}
```

The screenshot shows a code editor window titled "Test.java" with a Java logo icon and a close button. The code contains a class with a constructor, a `fillList()` method, and a `main` method. A blue arrow icon on the left margin indicates a debugger breakpoint is set on the line `name.fillList();`. The line is highlighted with a light blue background. The code is color-coded: keywords like `public`, `void`, `static`, and `new` are in purple; identifiers like `myList`, `Integer`, and `String` are in blue; and literals like `1`, `4`, `23`, and `args` are in black. The editor has a scrollbar on the right and a status bar at the bottom.

Exécution contrôlée



Exécution contrôlée : Step over (F6)

```
Test.java ☒  
    myList = new ArrayList<Integer>();  
    }  
    public void fillList(){  
        myList.add(1);  
        myList.add(new Integer(4));  
        myList.add(23);  
    }  
    public static void main(String[] args) {  
        Test name = new Test();  
        name.fillList();  
    }  
}
```

F6 (step over)

```
Test.java ☒  
    public void fillList(){  
        myList.add(1);  
        myList.add(new Integer(4));  
        myList.add(23);  
    }  
    public static void main(String[] args) {  
        Test name = new Test();  
        name.fillList();  
    }  
}
```

Variables

Name	Value
args	String[0] (id=15)
name	Test (id=17)
myList	ArrayList<E> (id=19)

[1, 4, 23]

Exécution contrôlée : Step over (F6)

Debug Test (1) [Java Application]

- test.Test at localhost:54871
 - Thread [main] (Suspended)
 - Thread.exit() line: 604 [local variables unavailable]

/opt/jdk1.5.0_11/bin/java (23 févr. 08 15:53:52)

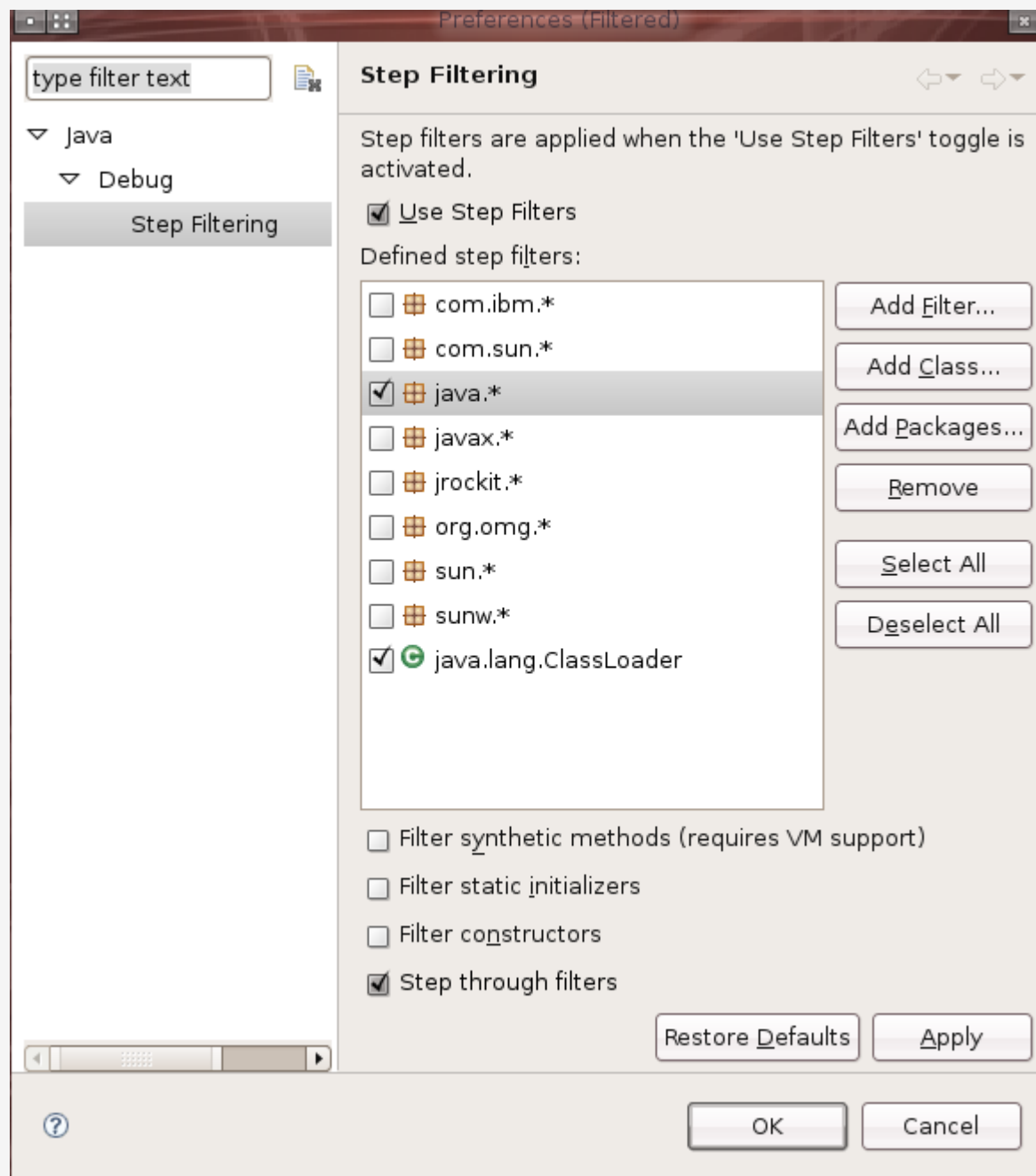
Name	Value
group	ThreadGroup (id=10)
inheritableThreadLocals	null
inheritedAccessControlCon	AccessControlContext (id=)
name	char[4] (id=41)
priority	5
single_step	false

```
/**  
 * This method is called by the system to give  
 * a chance to clean up before it actually exit  
 */  
private void exit() {  
if (group != null) {  
    group.remove(this);  
    group = null;  
}
```

Outline

- Thread(ThreadGroup, String)
- Thread(Runnable, String)
- Thread(ThreadGroup, Runnable, String)
- Thread(ThreadGroup, Runnable, String, long)
- start()
- start0()

Les filtres



Exécution contrôlée : Step into (F5)

The screenshot displays the IDE's debug interface. The top-left pane shows the 'Debug' console with the following structure:

- Test (1) [Java Application]
 - test.Test at localhost:44336
 - Thread [main] (Suspended)
 - Test.fillList() line: 13
 - Test.main(String[]) line: 20

The top-right pane shows the 'Variables' window with the following table:

Name	Value
this	Test (id=17)
myList	ArrayList<E> (id=19)

The bottom-left pane shows the source code for 'Test.java' with the following content:

```
}  
  
public void fillList(){  
    myList.add(1);  
    myList.add(new Integer(4));  
    myList.add(23);  
}  
  
public static void main(String[] args) {  
    Test name = new Test();  
    name.fillList();  
}  
}
```

The bottom-right pane shows the 'Outline' window with the following structure:

- test
 - import declarations
 - Test
 - myList : ArrayList<Integer>
 - Test()
 - fillList()
 - main(String[])

Exécution contrôlée : Step into (F5)

The screenshot displays an IDE interface during a Java application debug session. The top-left pane shows the 'Debug' console with the following structure:

- Test (1) [Java Application]
- test.Test at localhost:50108
 - Thread [main] (Suspended)
 - Test.fillList() line: 14**
 - Test.main(String[]) line: 20
- /opt/jdk1.5.0_11/bin/java (23 févr. 08 16:01:52)

The top-right pane shows the 'Variables' window with the following table:

Name	Value
this	Test (id=18)
myList	ArrayList<E> (id=19)

The bottom-left pane shows the source code of 'Test.java' with the following content:

```
}  
  
public void fillList(){  
    myList.add(1);  
    myList.add(new Integer(4));  
    myList.add(23);  
}  
  
public static void main(String[] args) {  
    Test name = new Test();  
    name.fillList();  
}  
}
```

The bottom-right pane shows the 'Outline' window with the following structure:

- test
- import declarations
- Test
 - myList : ArrayList<Integer>
 - Test()
 - fillList()**
 - main(String[])

Exécution contrôlée : Step into (F5)

The screenshot displays an IDE interface during a debug session. The top-left pane shows the 'Debug' console with the following structure:

- Test (1) [java Application]
- test.Test at localhost:50108
 - Thread [main] (Suspended)
 - Test.fillList() line: 15 (highlighted)
 - Test.main(String[]) line: 20
- /opt/jdk1.5.0_11/bin/java (23 févr. 08 16:01:52)

The top-right pane shows the 'Variables' window with the following table:

Name	Value
this	Test (id=18)
myList	ArrayList<E> (id=19)

The bottom-left pane shows the source code of 'Test.java' with the following content:

```
}  
  
public void fillList(){  
    myList.add(1);  
    myList.add(new Integer(4));  
    myList.add(23);  
}  
  
public static void main(String[] args) {  
    Test name = new Test();  
    name.fillList();  
}  
}
```





The bottom-right pane shows the 'Outline' window with the following structure:

- test
 - import declarations
 - Test
 - myList : ArrayList<Integer>
 - Test()
 - fillList() (highlighted)
 - main(String[])

```

fFull= new Vector<>();
fFull.addElement(1);
fFull.addElement(1);
fFull.addElement(1);
}
public static Test su
return new TestSu
}
public void testCapac
int size= fFull.s
for (int i= 0; i
fFull.addElem
assertTrue(fFull.
}
public void testClone
Vector clone= (Vec
assertTrue(clone.
assertTrue(clone.
}
public void testConta
assertTrue(fFull.
assertTrue(!fEmpt
}
public void testElemen
Integer i= (Intege
assertTrue(i.intVa

```

	Undo	Ctrl+Z
	Revert File	
	Save	
<hr/>		
	Open Declaration	F3
	Open Type Hierarchy	F4
	Open Call Hierarchy	Ctrl+Alt+H
	Quick Outline	
	Quick Type Hierarchy	
	Show In	Alt+Shift+W ▶
<hr/>		
	Cut	Ctrl+X
	Copy	Ctrl+C
	Paste	Ctrl+V
<hr/>		
	Source	Alt+Shift+S ▶
	Refactor	Alt+Shift+T ▶
	Surround With	Alt+Shift+Z ▶
	Local History	▶
<hr/>		
	References	▶
	Declarations	▶
<hr/>		
	Watch	
	Inspect	Ctrl+Shift+I
	Display	Ctrl+Shift+D

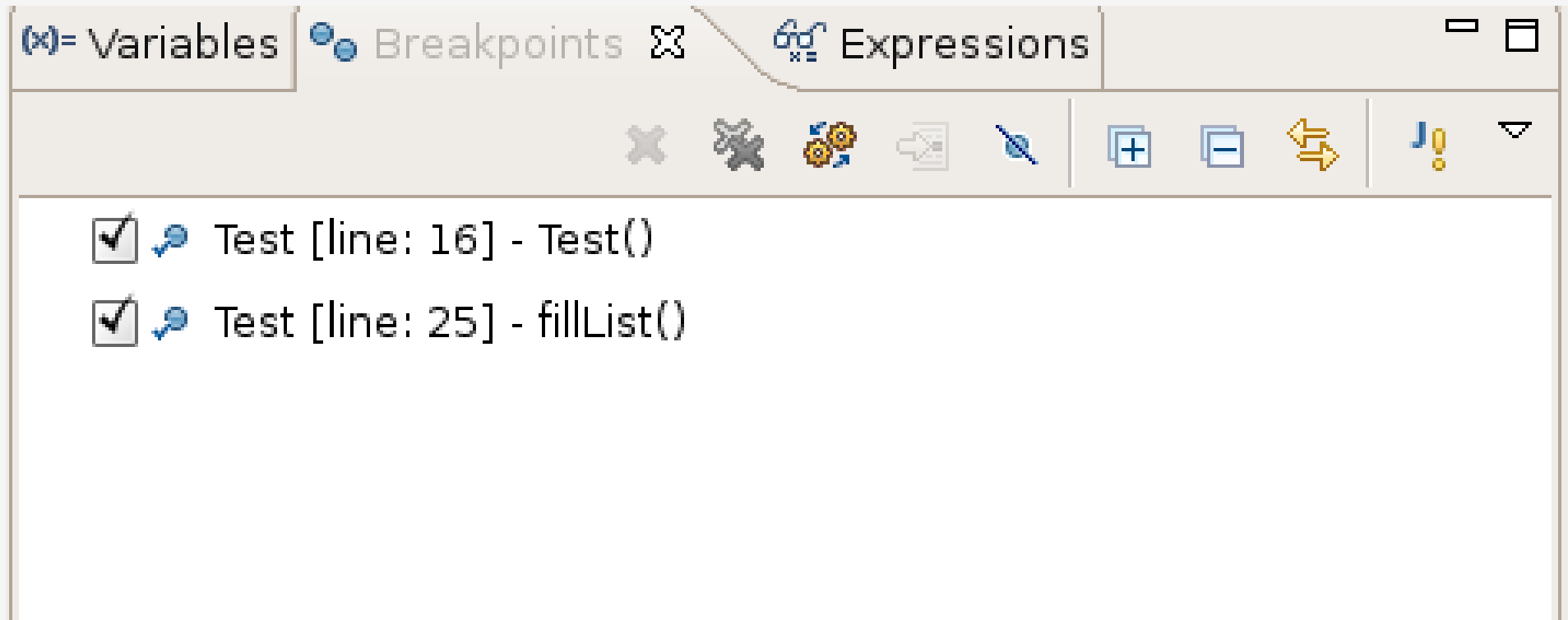
Inspecter une variable

The screenshot shows an IDE with the following components:

- Editor:** Displays the `fillList()` method in `Test.java`. The line `myList.add(23);` is selected.
- Outline:** Shows the project structure with `Test` expanded to show `myList : ArrayList<Integer>`.
- Inspector:** A yellow tooltip window displays the state of the variable `myList`:
 - Variable: `"myList" = ArrayList<E> (id=19)`
 - Property: `elementData = Object[10] (id=35)`
 - Property: `modCount = 2`
 - Property: `size = 2`
- Console:** Shows the output `[1, 4]`.

Press Shift+Ctrl+I to Move to Expressions View

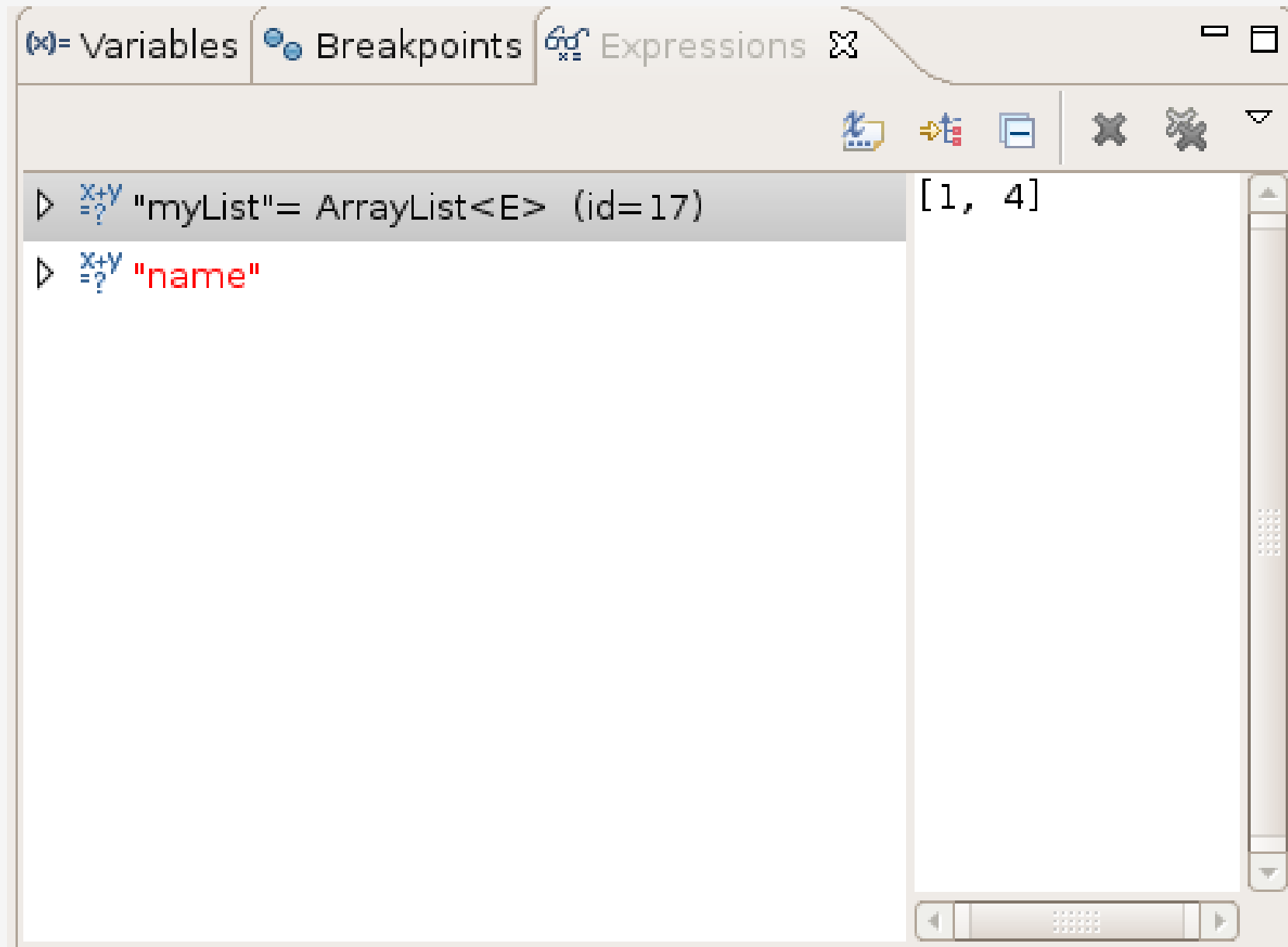
Liste des points d'arrêt



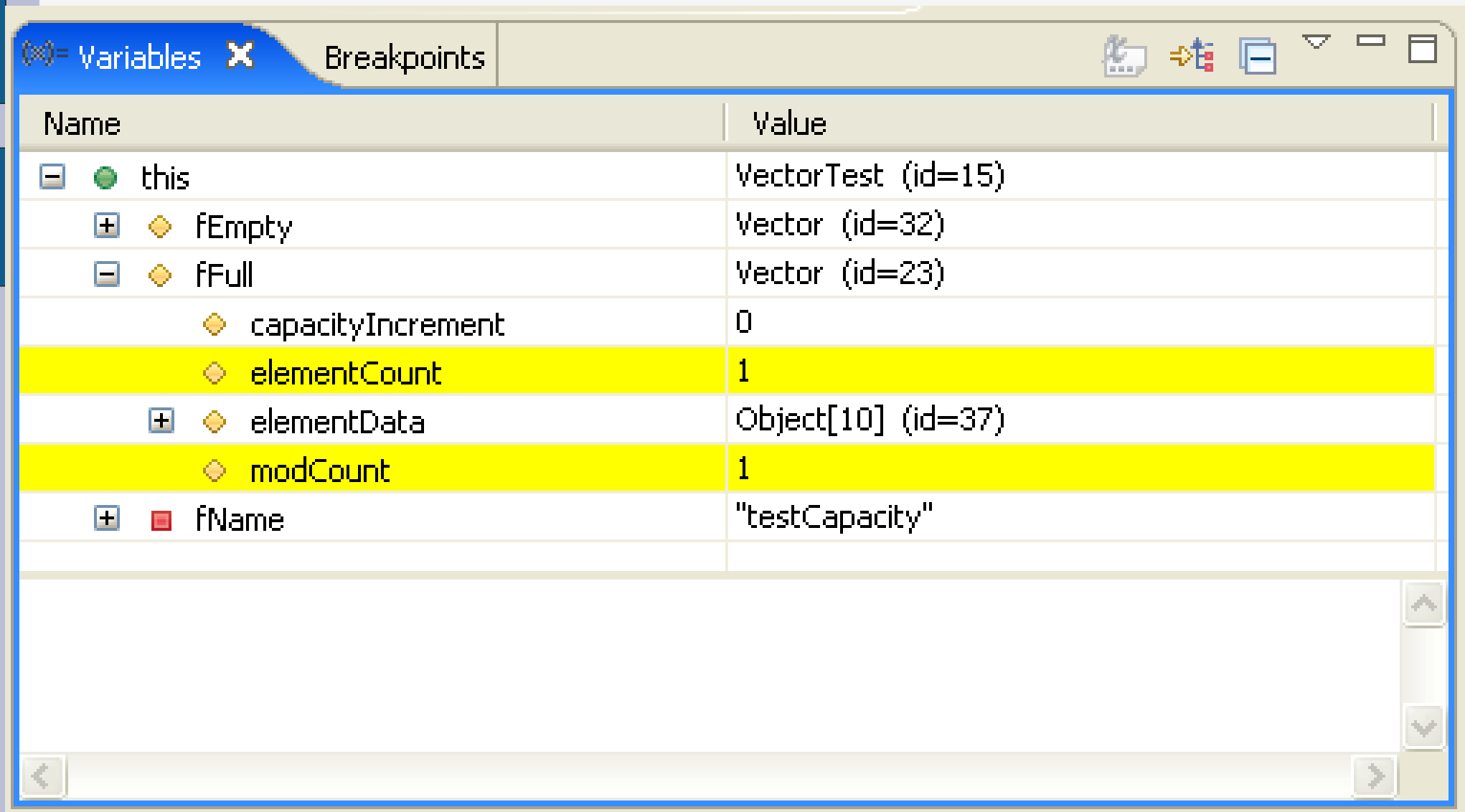
The screenshot shows a debugger window with three tabs: "Variables", "Breakpoints", and "Expressions". The "Breakpoints" tab is active. The toolbar contains icons for deleting, disabling, enabling, and adding breakpoints, as well as zoom and search icons. The main area lists two breakpoints, both of which are checked and active.

Breakpoint Status	Breakpoint Location
<input checked="" type="checkbox"/>	Test [line: 16] - Test()
<input checked="" type="checkbox"/>	Test [line: 25] - fillList()

Surveiller une expression



Modifier une variable



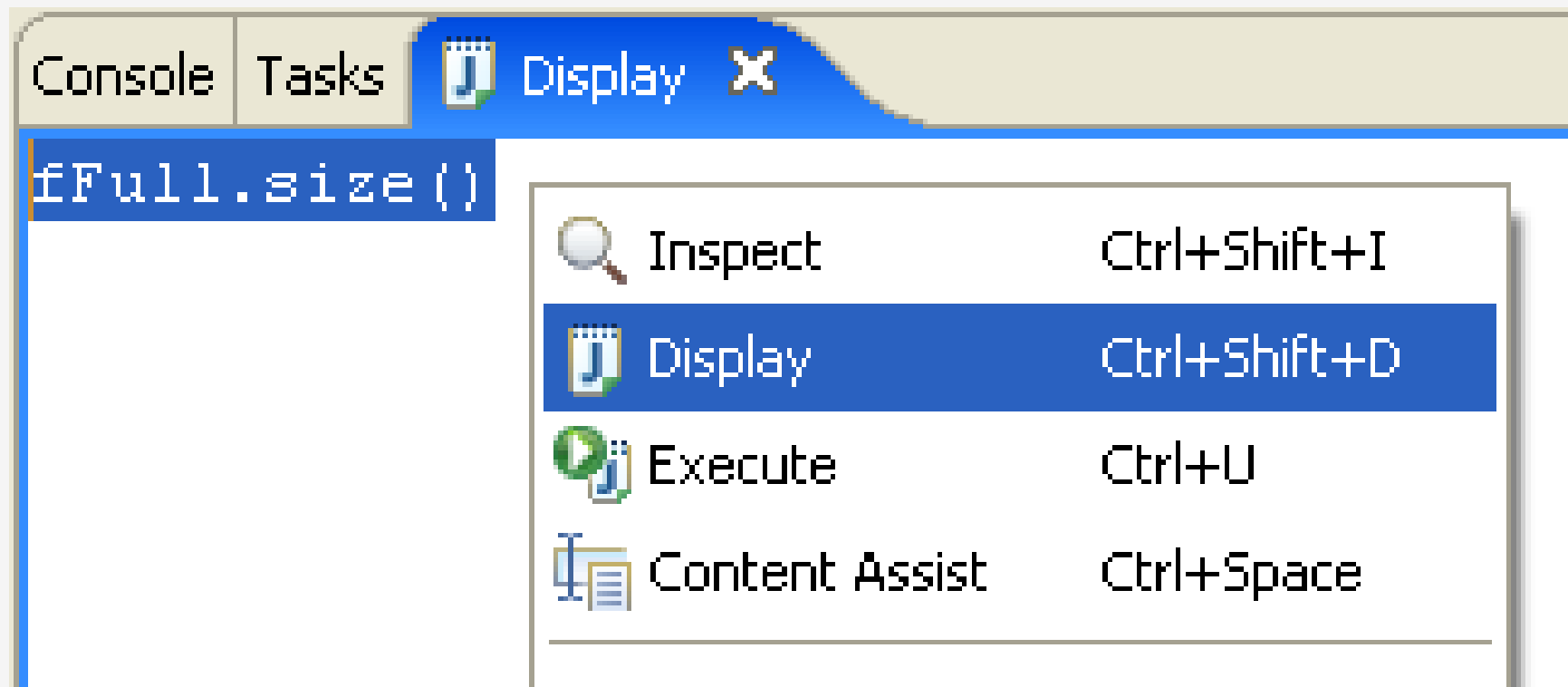
The screenshot shows the 'Variables' window in an IDE. The window title is 'Variables' and it has a 'Breakpoints' tab. The main area is a table with two columns: 'Name' and 'Value'. The table lists the following variables and their values:

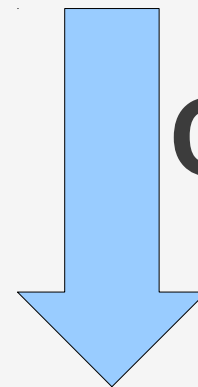
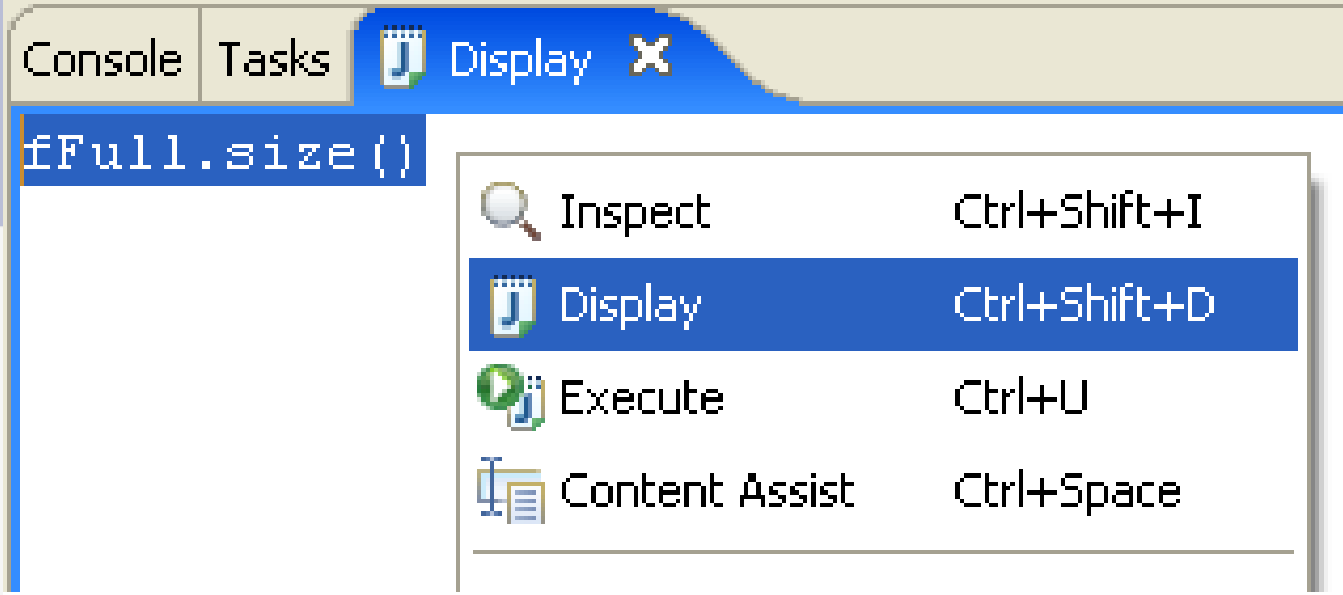
Name	Value
this	VectorTest (id=15)
fEmpty	Vector (id=32)
fFull	Vector (id=23)
capacityIncrement	0
elementCount	1
elementData	Object[10] (id=37)
modCount	1
fName	"testCapacity"

The 'elementCount' and 'modCount' rows are highlighted in yellow. The 'fName' row has a red square icon next to it. The window also features a toolbar with icons for refreshing, stepping through code, and window management, along with standard window controls (minimize, maximize, close) in the top right corner.

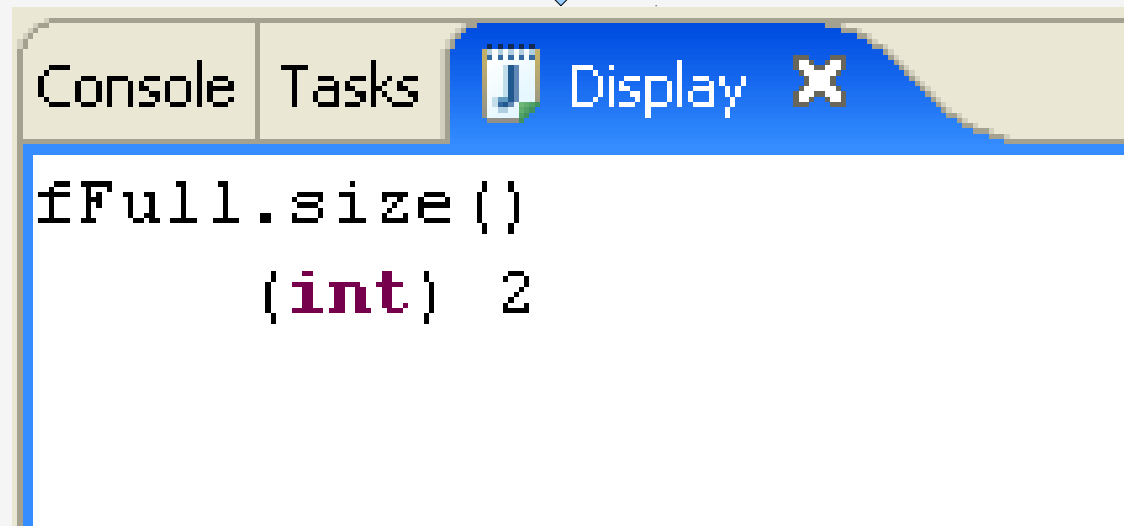
Evaluer des expressions

- Window > Show View > Display :





Ctrl + Shift + D



Inspecter une évaluation

The screenshot shows an IDE's console window with the following content:

```
Console Tasks Display X  
fFull.size()  
    (int) 2  
fFull.toArray()
```

A yellow tooltip is displayed over the `fFull.toArray()` expression, showing the following structure:

- [-] 🔍 "fFull.toArray()"= Object[2] (id=47)
 - [+] 🏔 [0]= Integer (id=43)
 - [+] 🏔 [1]= Integer (id=40)

At the bottom of the console window, there is a message: "Press Ctrl+Shift+I to Move to Expressions View".

Tester des expressions

- New > Other > Java > Java Run/Debug > Scrapbook Page :



Tester des expressions

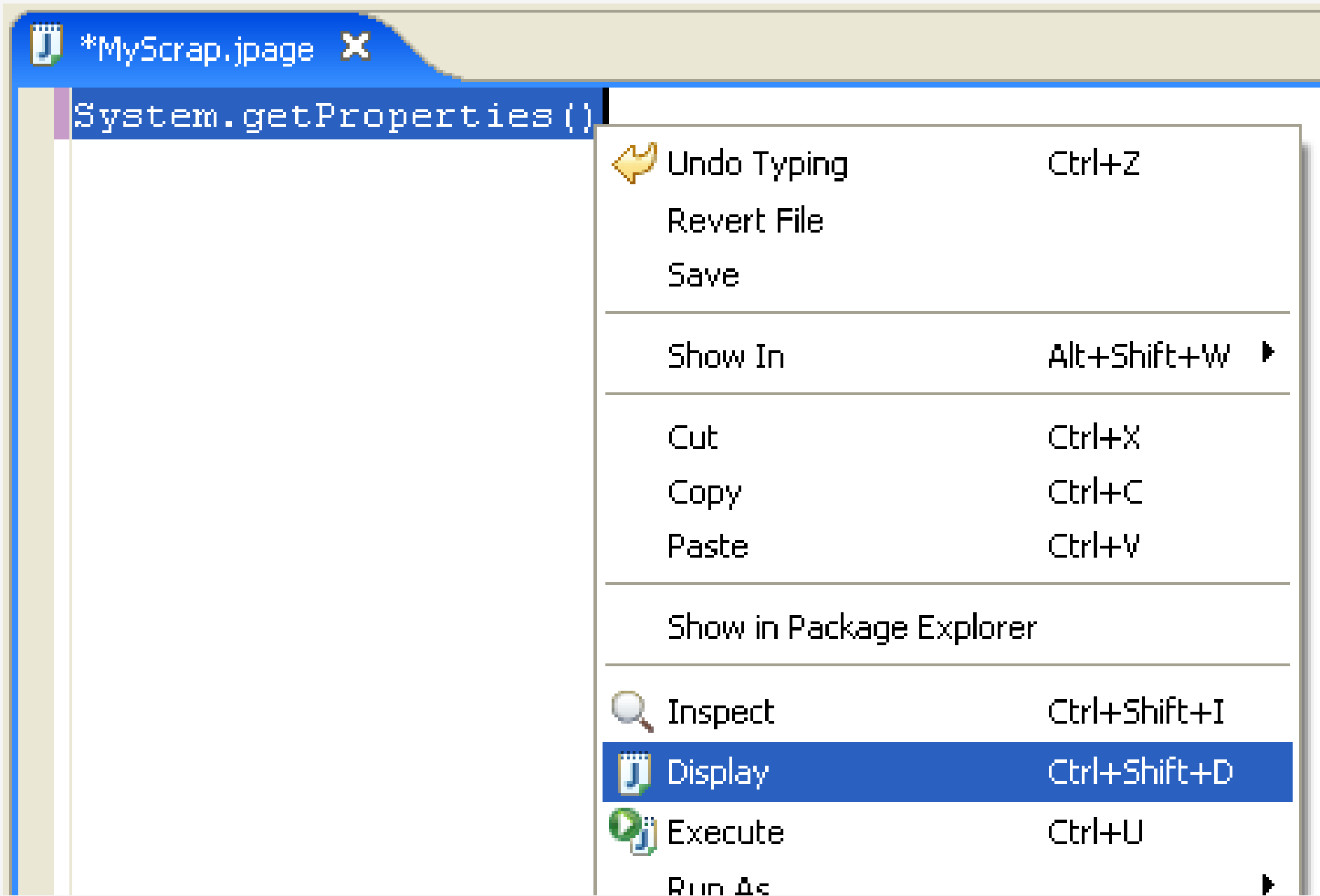
The screenshot shows an IDE window with two tabs: 'VectorTest.java' and '*MyScrap.jpage'. The cursor is positioned at the end of the text 'System.g' in the editor. A popup menu is displayed, listing several methods from the 'System' class. Each method is preceded by a green circle with a red 'S' next to it. The popup menu has a title bar that reads 'JUnit/MyScrap.jpage'. At the bottom of the popup, there is a scroll bar with left and right arrow buttons.

System.g

JUnit/MyScrap.jpage

- gc() void - System
- getenv(String name) String - System
- getProperties() Properties - System
- getProperty(String key) String - System
- getProperty(String key, String def) String - System
- getSecurityManager() SecurityManager - System

Tester des expressions



The screenshot shows an IDE window with a file named `*MyScrap.jpape`. The code `System.getProperties()` is selected, and a context menu is open. The menu items and their keyboard shortcuts are as follows:

MenuItem	Shortcut
Undo Typing	Ctrl+Z
Revert File	
Save	
Show In	Alt+Shift+W ▶
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Show in Package Explorer	
Inspect	Ctrl+Shift+I
Display	Ctrl+Shift+D
Execute	Ctrl+U
Run As	▶

« Java Browsing » perspective

- Window > Open Perspective > Java Browsing :

The screenshot shows the Eclipse IDE in the Java Browsing perspective. The window title is "Java Browsing - TestCase.java - Eclipse SDK". The interface includes a menu bar (File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help), a toolbar, and four main panes:

- Projects:** Shows the JUnit project.
- Packages:** Shows the package hierarchy: junit.samples, junit.samples.money, junit.tests, junit.tests.extensions, junit.tests.framework, junit.tests.runner, junit.textui, org.junit, org.junit.internal.requests, and org.junit.internal.runners.
- Types:** Shows the class hierarchy: JUnit4TestAdapterCache, JUnit4TestCaseFacade, Protectable, Test, TestCase (highlighted), TestFailure, TestListener, TestResult, and TestSuite.
- Members:** Shows the members of the selected class: import declarations, fName : String, TestCase(), TestCase(String), countTestCases(), createResult(), run() (highlighted), and run(TestResult).

The main editor displays the source code for `TestCase.run()`:

```
protected TestResult createResult() {
    return new TestResult();
}
/**
 * A convenience method to run this test, collecting the results with a
 * default TestResult object.
 *
 * @see TestResult
 */
public TestResult run() {
    TestResult result= createResult();
    run(result);
    return result;
}
/**
 * Runs the test case and collects the results in TestResult.
 */
```

The status bar at the bottom shows the current location: `junit.framework.TestCase.run() : TestResult - JUnit`.

Bookmark sur une ligne

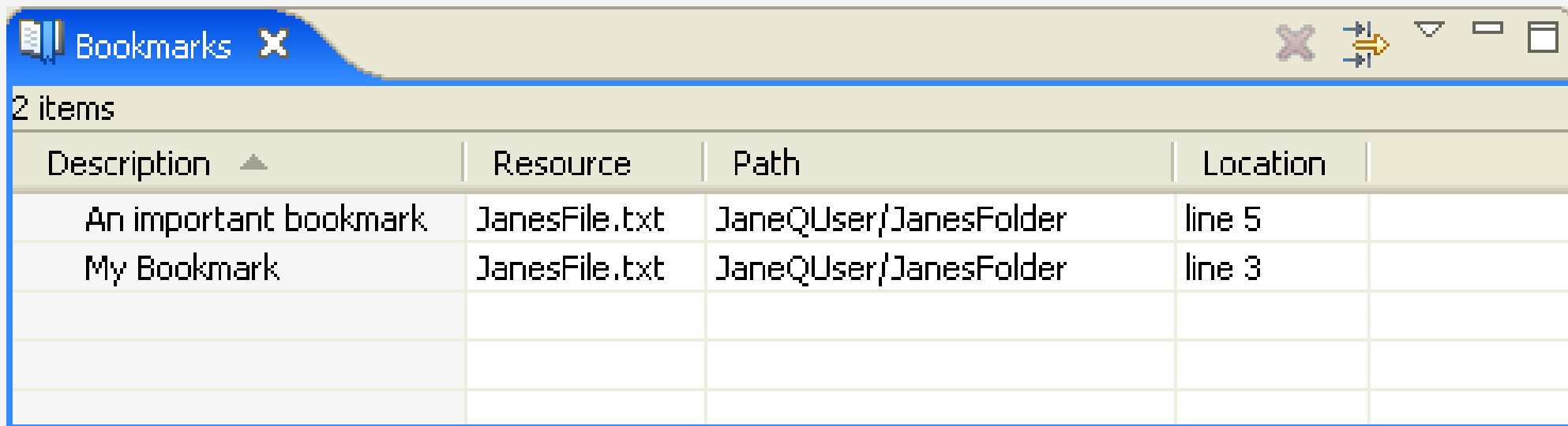
The image illustrates the steps to add a bookmark to a specific line in a text editor. It consists of three sequential screenshots:

- Step 1:** A text editor window with two tabs: 'JanesFile.txt' and 'JanesFile2.txt'. The 'JanesFile.txt' tab is active, showing a text file with five lines. The third line is selected. A context menu is open over the selection, with 'Add Bookmark...' highlighted.
- Step 2:** The same text editor window, but now a small bookmark icon (a blue ribbon) is visible on the left margin of the third line. A red box highlights this icon.
- Step 3:** A 'Bookmarks' panel window is open, displaying a table with one item. The table has columns for Description, Resource, Path, and Location.

Description	Resource	Path	Location
My Bookmark	JanesFile.txt	JaneQUser/JanesFolder	line 3

Bookmark sur un fichier

- Sur un fichier dans l'explorateur :
 - Edit > Add Bookmark

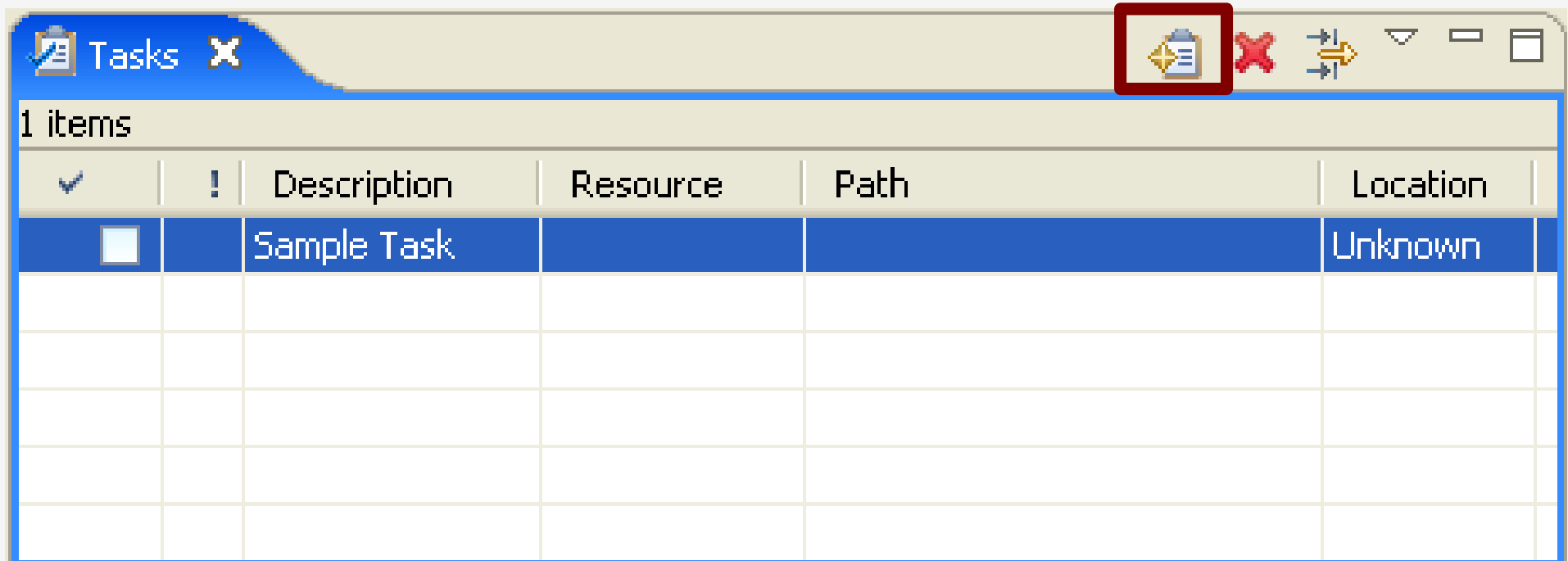


The screenshot shows a Windows 'Bookmarks' window with a table containing two items. The table has four columns: Description, Resource, Path, and Location. The first item is 'An important bookmark' pointing to 'JanesFile.txt' at 'JaneQUser/JanesFolder' on 'line 5'. The second item is 'My Bookmark' pointing to 'JanesFile.txt' at 'JaneQUser/JanesFolder' on 'line 3'.

Description ▲	Resource	Path	Location
An important bookmark	JanesFile.txt	JaneQUser/JanesFolder	line 5
My Bookmark	JanesFile.txt	JaneQUser/JanesFolder	line 3

Création et gestion de tâches

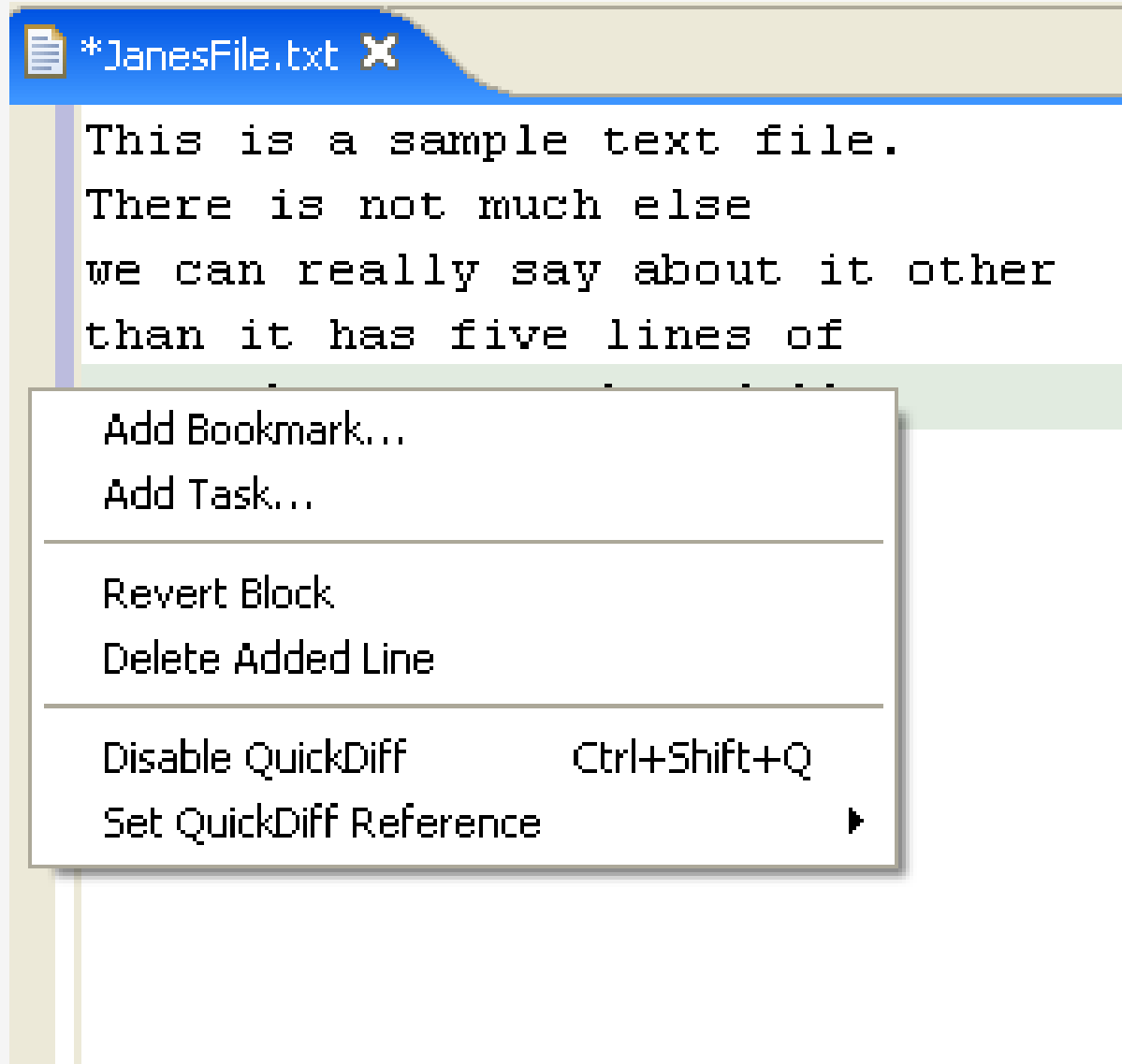
- Création :



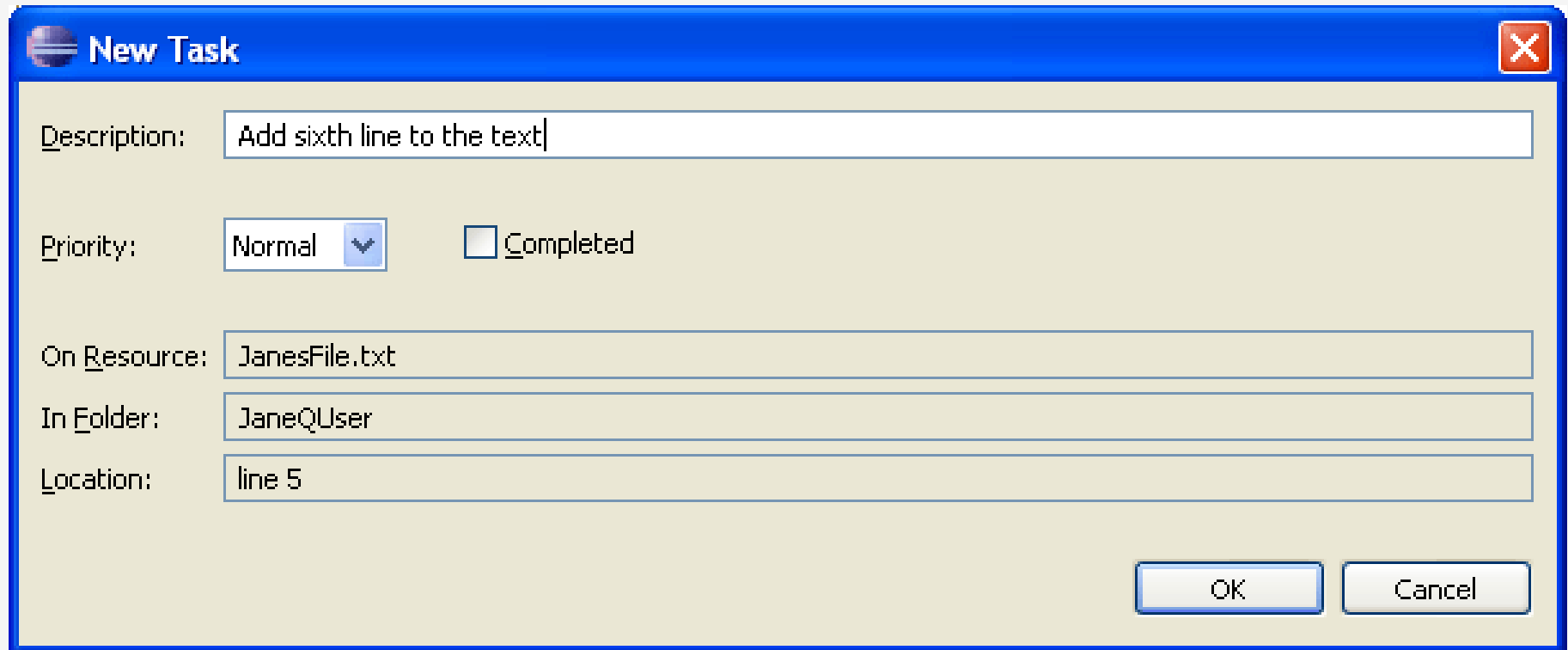
The screenshot shows a software window titled 'Tasks'. The window has a toolbar with several icons. A red box highlights the 'Add' icon, which is a blue document with a plus sign. To the right of the 'Add' icon are a red 'X' (delete), a double-headed arrow (move), a dropdown arrow, a minimize button, and a maximize button. Below the toolbar, the window displays '1 items' and a table with the following columns: a checkbox, an exclamation mark, 'Description', 'Resource', 'Path', and 'Location'. The first row contains a checked checkbox, an exclamation mark, 'Sample Task', an empty 'Resource' field, an empty 'Path' field, and 'Unknown' in the 'Location' column.

<input checked="" type="checkbox"/>	!	Description	Resource	Path	Location
<input checked="" type="checkbox"/>	!	Sample Task			Unknown

Création et gestion de tâches



Création et gestion de tâches



New Task

Description: Add sixth line to the text

Priority: Normal Completed

On Resource: JanesFile.txt

In Folder: JaneQUser

Location: line 5

OK Cancel

Création et gestion de tâches

- Directement dans le code :
 - FIXME (priorité max)
 - *// FIXME cette méthode bug : ...*
 - TODO
 - *// TODO faire en sorte que ...*