



JAVA™



eclipse

# Les IDEs

## **Integrated Development Environment**

**NetBeans (Oracle)**

**Eclipse (IBM)**

**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 puissants qui améliorent la rapidité et la qualité du code produit
- Aucun développeur professionnel ne travaille sans

# Avantages d'un IDE (1)

- **Ergonomie :**

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

# Avantages d'un IDE (2)

- **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 (3)

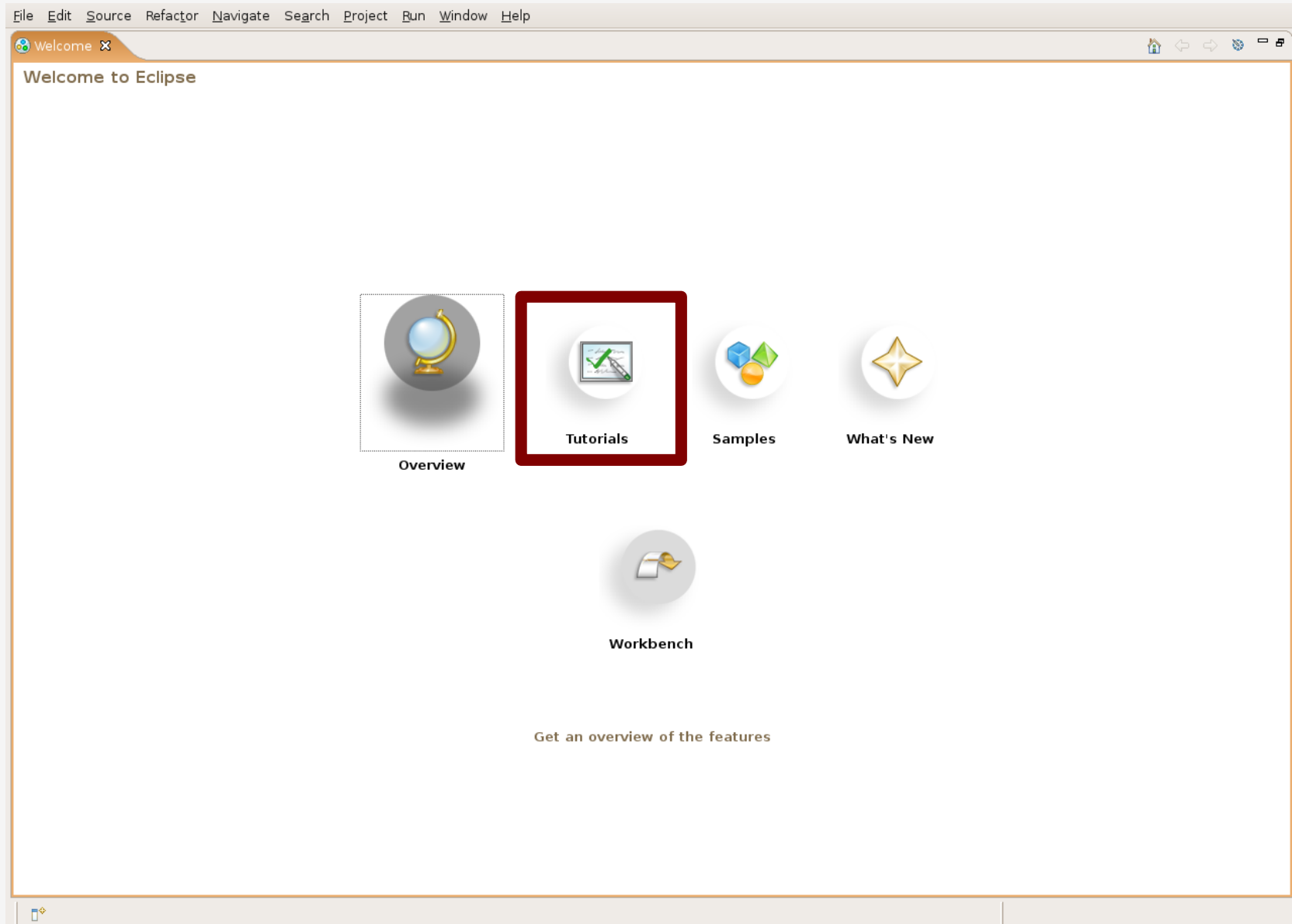
- **Fonctionnalités avancées**

- Édition d'interfaces graphiques façon WYSIWYG
- Gestion de projet (du simple Todo à Redmine)
- Outils de Debug puissants
- Outils d'analyse des performances
- Intégration d'outils tierces:
  - Ant, Maven, Graddle, Ivy, etc.
  - Intégration de gestionnaires de version (Git, svn)
  - Serveurs Web intégrés

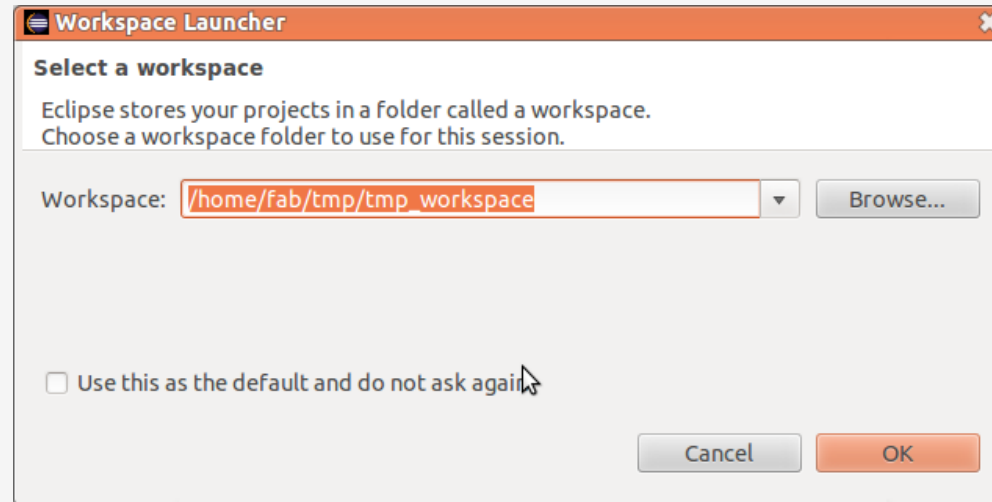
# I Prise en main



# eclipse - démarrage



# Démarrage : sélection d'un workspace



- Un **workspace** contient des projets : des répertoires portant le nom du projet et contenant des sources et autres ressources (images, etc.)
- On peut en avoir plusieurs : ils sont **indépendants**

# File → New → Java Project

## Create a Java project

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

Project name:

Contents

Create new project in workspace

Create project from existing source

Directory:

JRE

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

Use a project specific JRE:  ▼

Project layout

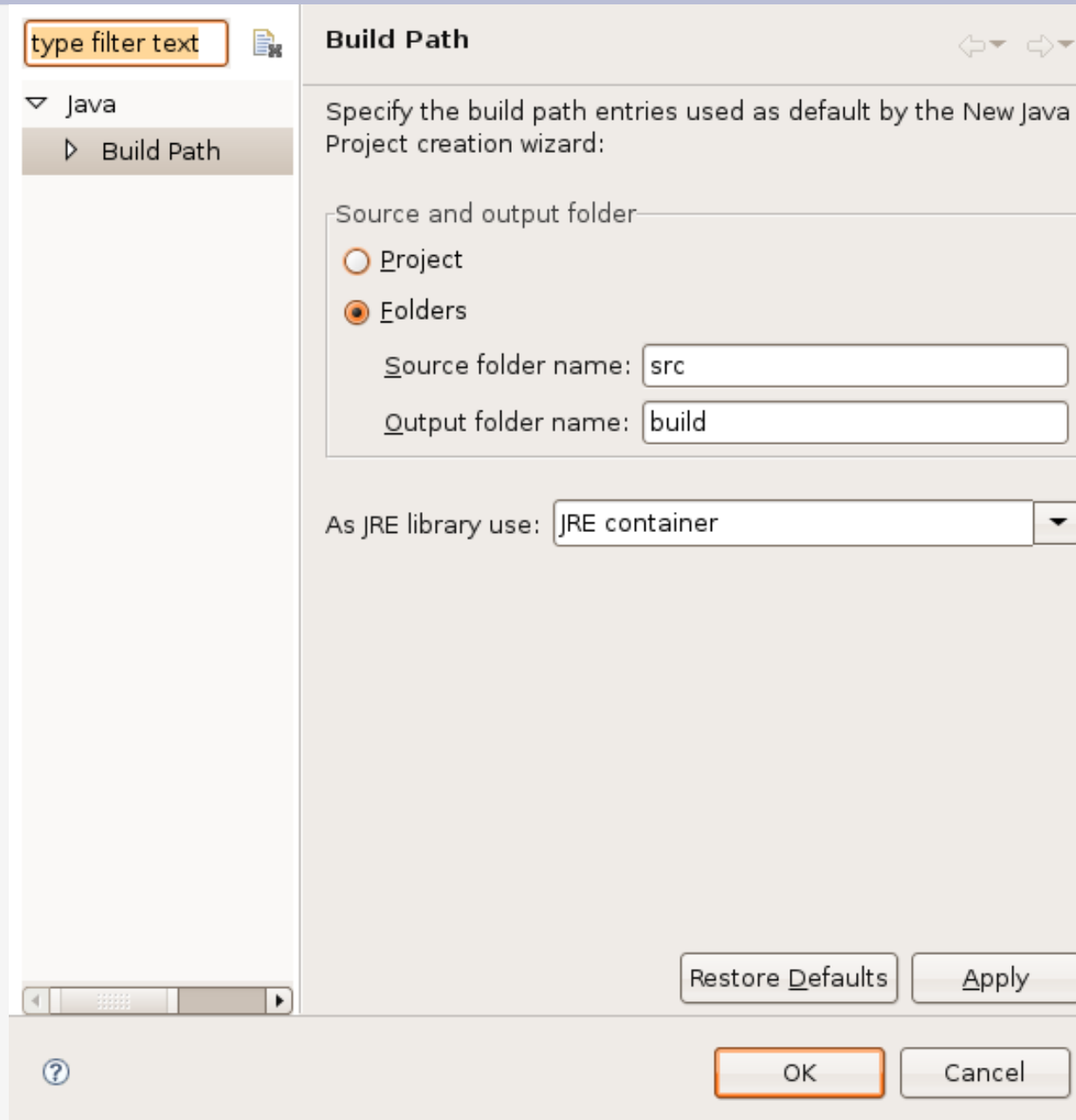
Use project folder as root for sources and class files

Create separate source and output folders [Configure default...](#)

## 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 and a green 'Run' button (a play icon) which 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 lists several steps with checkboxes:
  - 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 and highlighted.
- Console (Bottom):** Shows 'No consoles to display at this time.'
- Bottom Bar:** Displays the current project name 'HelloWorld'.

# 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

# 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





## 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?

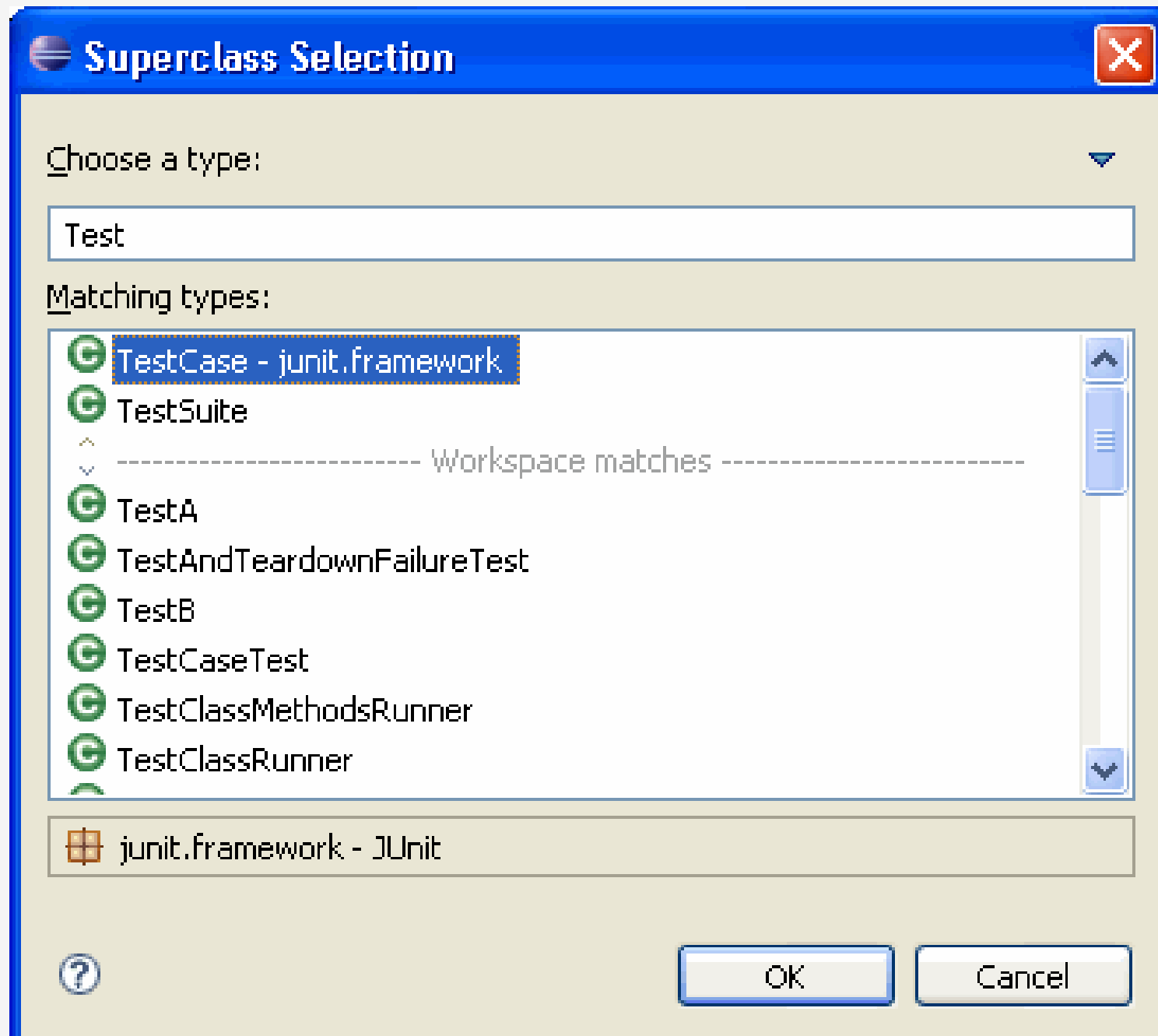
- Generate comments



Finish

Cancel

# 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

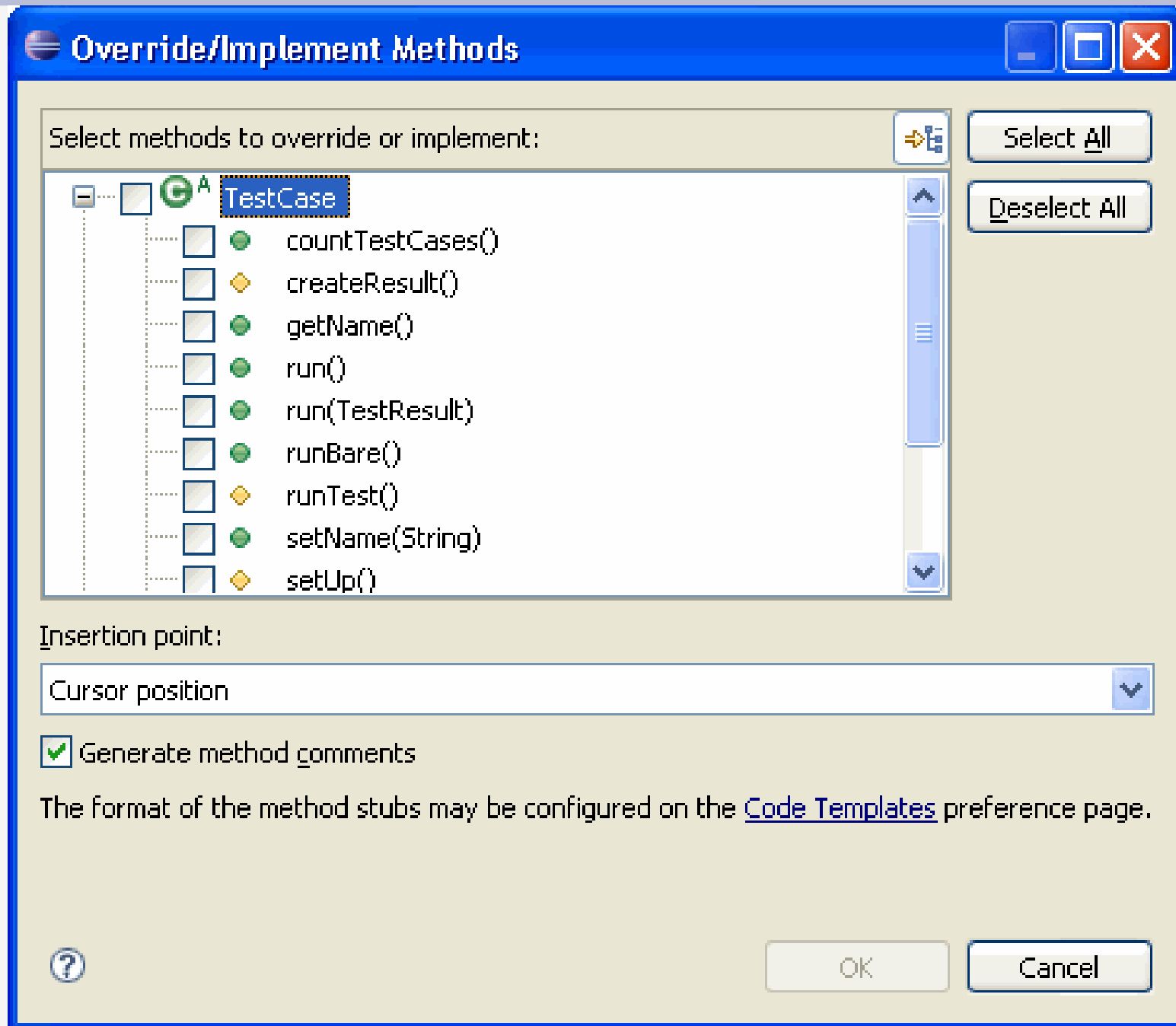
- Source Alt+Shift+S ▶
- Refactor Alt+Shift+T ▶
- Surround With Alt+Shift+Z ▶
- Local History ▶
- References ▶
- Declarations ▶
- Run As ▶
- Debug As ▶
- Team ▶
- Compare With ▶
- Replace With ▶
- Preferences...

- 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

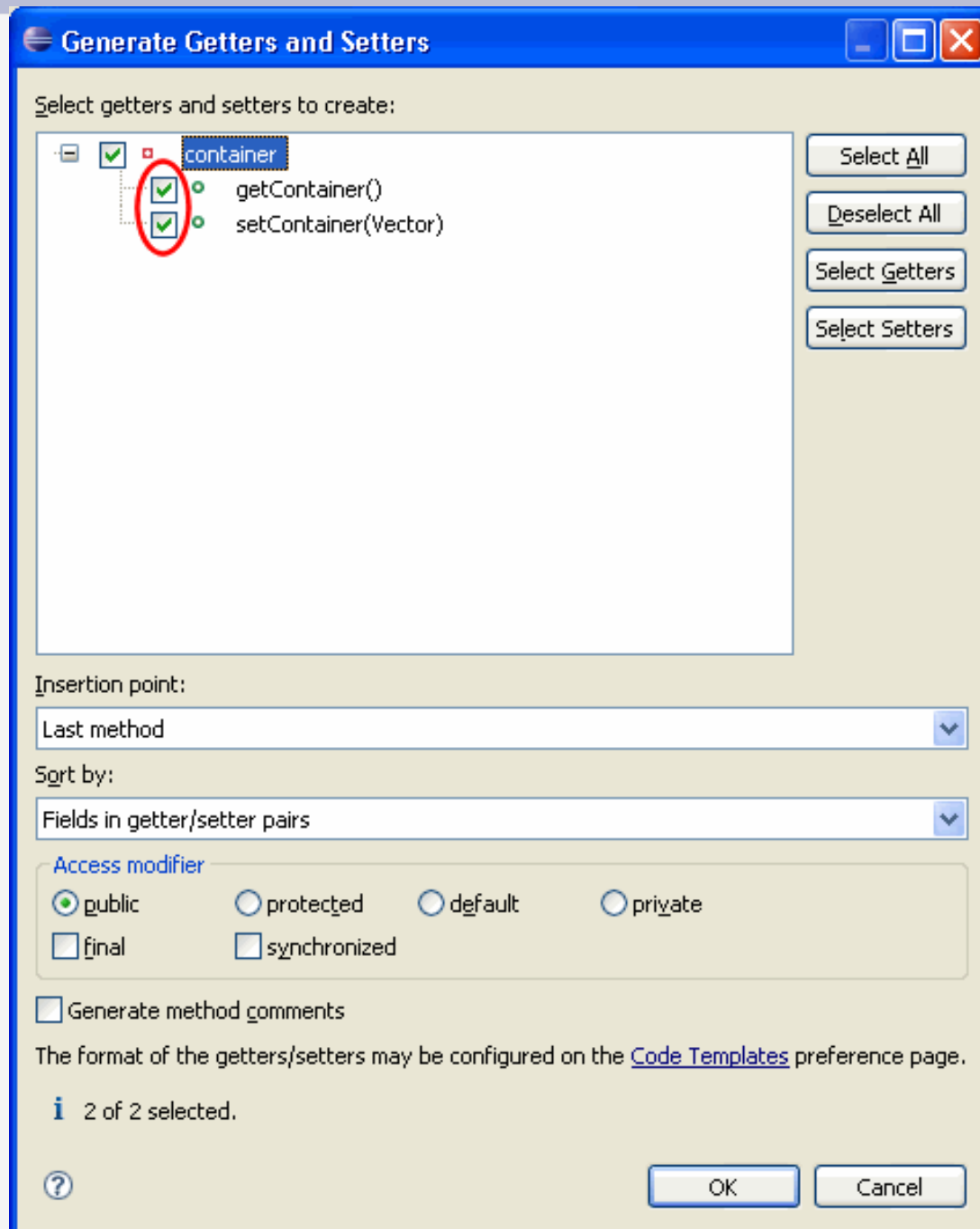
The image shows a screenshot of an IDE with a context menu open over a class named `TestCase`. The menu is divided into two main sections. The left section contains code generation and refactoring options, with `Generate Getters and Setters...` highlighted. The right section contains standard editing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

Menu Item	Shortcut
Open Type Hierarchy	F4
Copy	Ctrl+C
Copy Qualified Name	
Paste	Ctrl+V
Delete	Delete
Source	
Refactor	
References	
Declarations	
Toggle Class Load Breakpoint	
Run As	
Debug As	
Team	
Compare With	
Replace With	
Restore from Local History...	
Properties	Alt+Enter

Left menu items (from top to bottom):

- 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

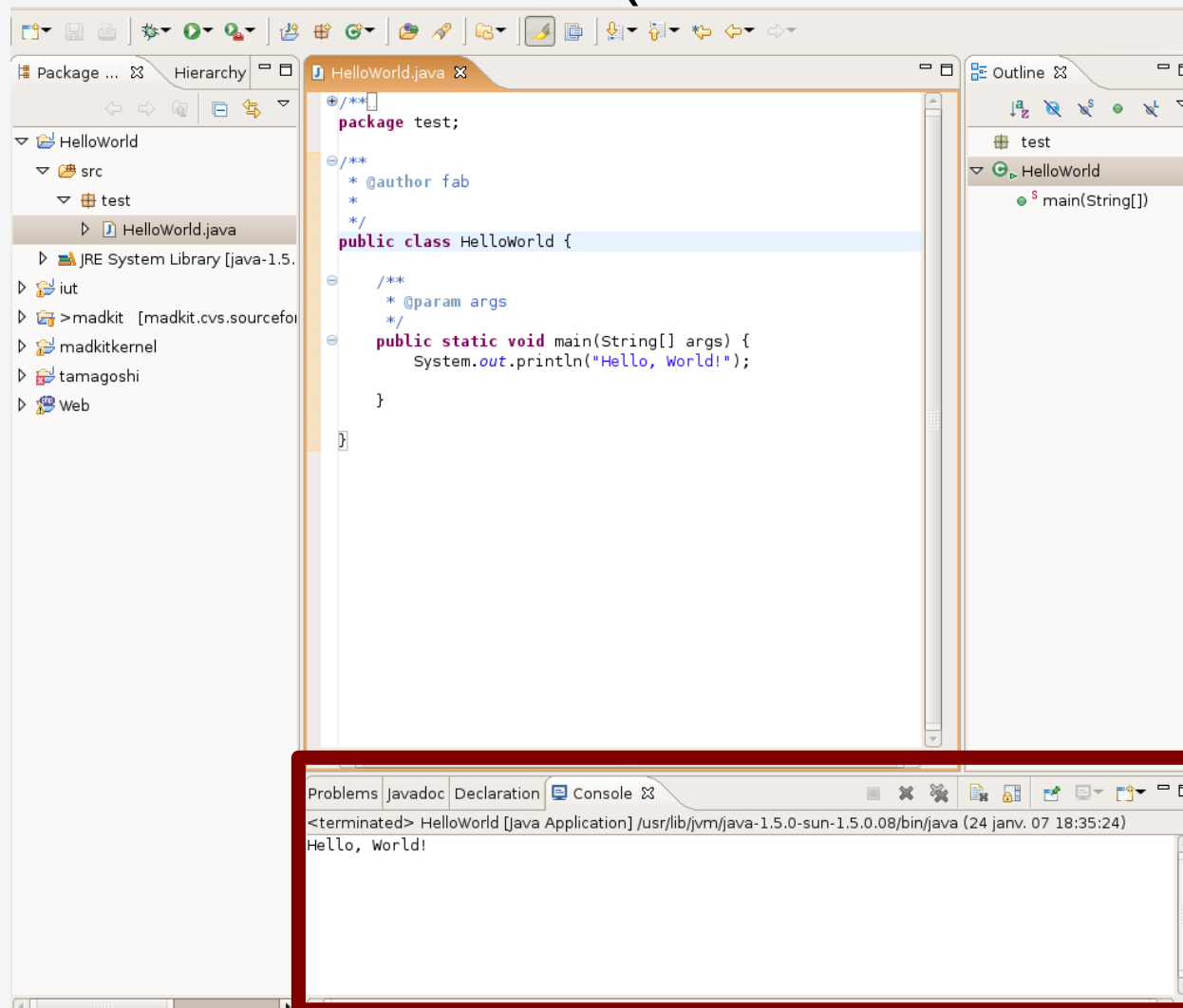
# Générer les accesseurs





# 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]
  - ▶ iut
  - ▶ >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!
```

# II La perspective Java

# Vue globale - perspective java

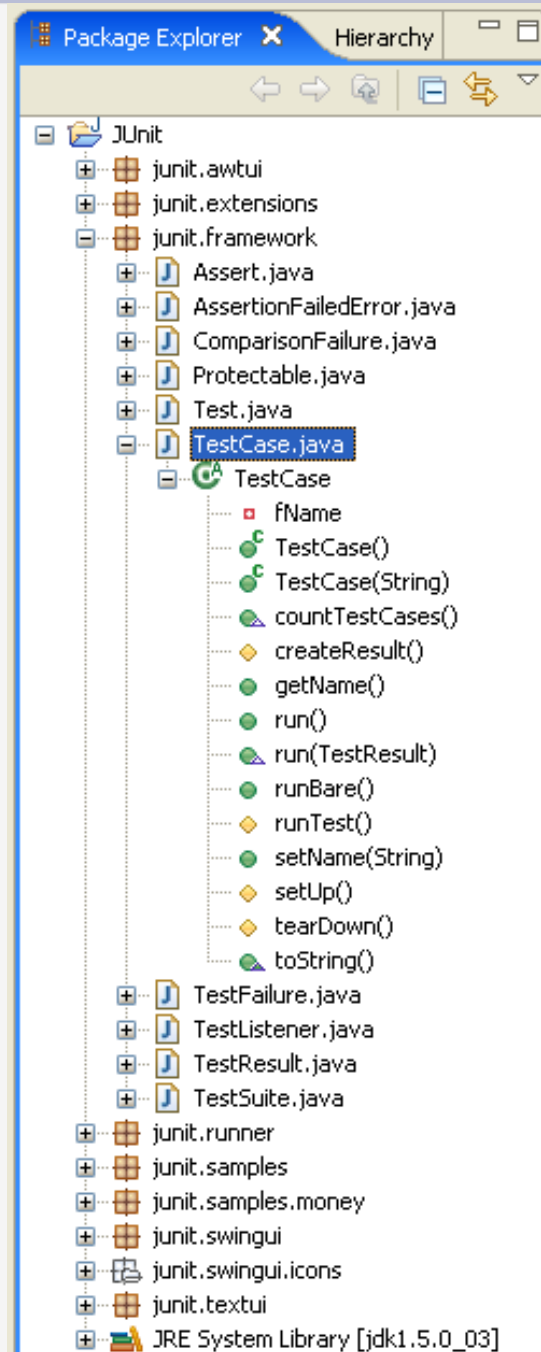
The screenshot displays the Eclipse IDE interface. The main editor window shows the source code of the `AbstractAgent.java` file. The code includes a private method `waitingNextMessage` and two public methods `destroyCommunity` and `destroyGroup`, both with detailed Javadoc comments. The `destroyCommunity` method calls `getKernel().destroyCommunity(this, community);`. The `destroyGroup` method has a similar structure. The right-hand side of the IDE shows the Outline view, which lists the public methods of the class, such as `getKernel()`, `getKernelAddress()`, `getLogger()`, `getMadkitConfig()`, `getMadkitKernel()`, `getMadkitProperty(String)`, `getName()`, `getNetworkID()`, `getOrganizationSnapShot()`, `getReplyTo(Message)`, `getServerInfo()`, `getSimpleNetworkID()`, `getState()`, `handleException(Influence)`, `handleInterruptedException`, `hasGUI()`, `hashCode()`, `isAlive()`, `isCommunity(String)`, `isFinestLogOn()`, `isGroup(String, String)`, and `isKernelOnline()`.

```
Java - MaDKit/src/madkit/kernel/AbstractAgent.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access Java Debug Git Repository Exploring
XMLFailureTestA XMLSuccessTestA XmlFeaturesTest AbstractAgent.j ConfigFileTest test.prop 20
MaDKit > src > madkit.kernel > AbstractAgent > launchNode(Node) : ReturnCode
*/
private Message waitingNextMessage(final long timeout, final TimeUnit unit) {
    try {
        return messageBox.poll(timeout, unit);
    } catch (InterruptedException e) {
        handleInterruptedException();
        return null;
    }
}

/**
 * Wipes out an entire community at once. Mostly useful when doing simulated
 * systems. This greatly optimizes the time required to make all the agents
 * leave a community.
 *
 * @since MaDKit 5.0.0.9
 * @param community
 *         the community to destroy
 */
public void destroyCommunity(String community) {
    getKernel().destroyCommunity(this, community);
}

/**
 * Wipes out an entire group at once. Mostly useful when doing simulated
 * systems. This greatly optimizes the time required to make all the agents
 * leave a group.
 *
 * @since MaDKit 5.0.0.10
 * @param community
 */
Outline
getKernel() : MadkitKernel
getKernelAddress() : Kerne
getLogger() : AgentLogger
getMadkitConfig() : Madkit
getMadkitKernel() : Madkit
getMadkitProperty(String)
getName() : String
getNetworkID() : String
getOrganizationSnapShot(
getReplyTo(Message) : Mes
getServerInfo() : String
getSimpleNetworkID() : Str
getState() : State
handleException(Influence
handleInterruptedException
hasGUI() : boolean
hashCode() : int
isAlive() : boolean
isCommunity(String) : bool
isFinestLogOn() : boolean
isGroup(String, String) : bo
isKernelOnline() : boolean
isMessageReplyTo(String) : bo
```

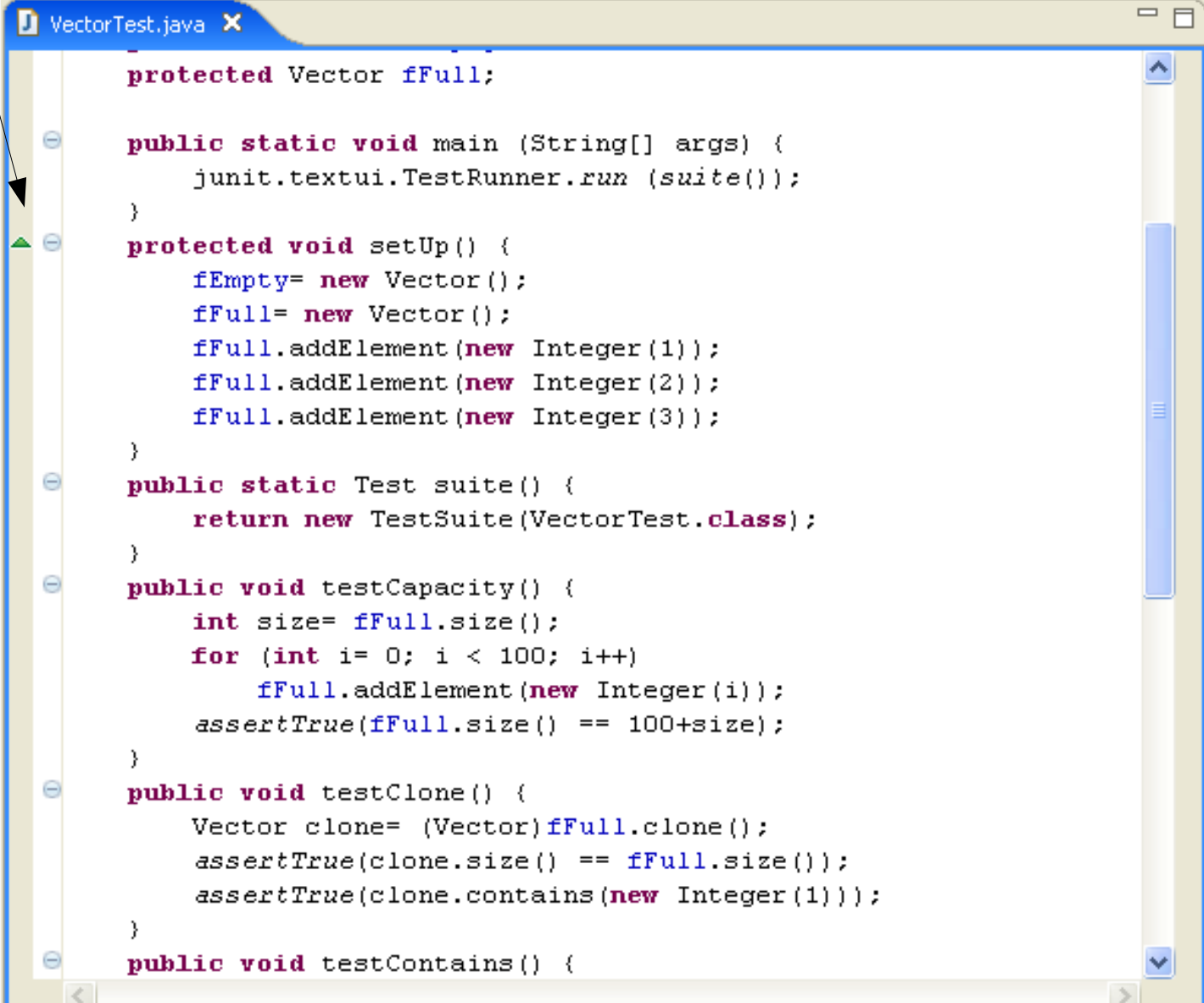
# Le Package Explorer



# La partie édition

Informations :

- erreurs
- surcharge d'une fonction
- etc.



```
VectorTest.java x
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 class structure for `junit.samples`. The structure includes:

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

filtres

Outline window showing the class structure for `junit.samples` after applying filters. The structure includes:

- import declarations
- VectorTest
  - setUp()
  - 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.

**Code Pane:**

```
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 code editor.

**Outline Pane:**

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



# Rajout d'une méthode

The screenshot shows an IDE window titled `*VectorTest.java` with the following 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());
}

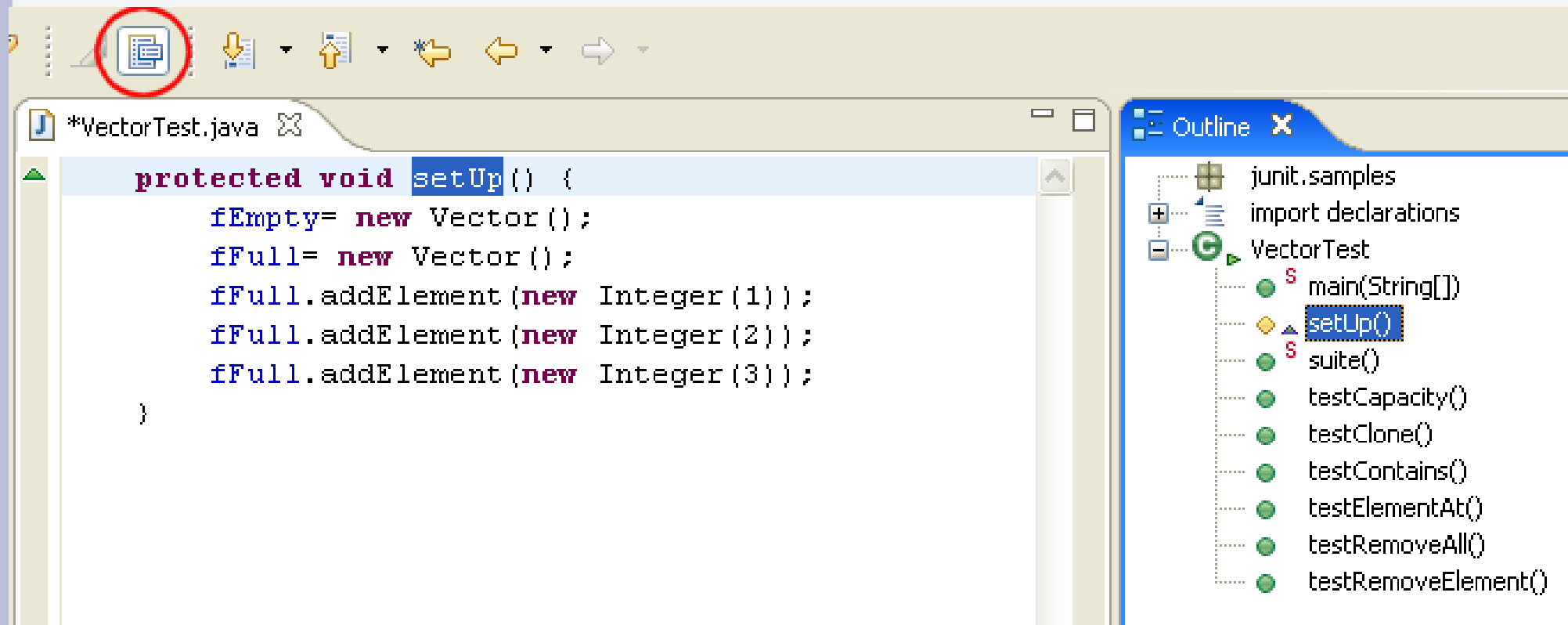
public void testRemoveElement() {
    fFull.removeElement(new Integer(3));
    assertTrue(!fFull.contains(new Integer(3)));
}

public void testSizeIsThree()
```

The `testSizeIsThree()` method is circled in red. In the Outline view on the right, the `testSizeIsThree()` method is also circled in red. The Outline view shows the project structure:

- junit.samples
  - import declarations
  - VectorTest
    - main(String[])
    - suite()
    - setUp()
    - testCapacity()
    - testClone()
    - testContains()
    - testElementAt()
    - testRemoveAll()
    - testRemoveElement()
    - testSizeIsThree()

# Edition par éléments



The screenshot displays an IDE interface with a code editor and an Outline view. The code editor shows the following Java 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 name is highlighted in blue. The Outline view on the right shows the project structure:

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

# III travailler sur le code

# 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

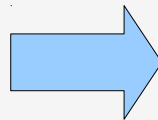
Press 'F2' for focus.

# 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 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.



Package Explorer Hierarchy

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

# 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

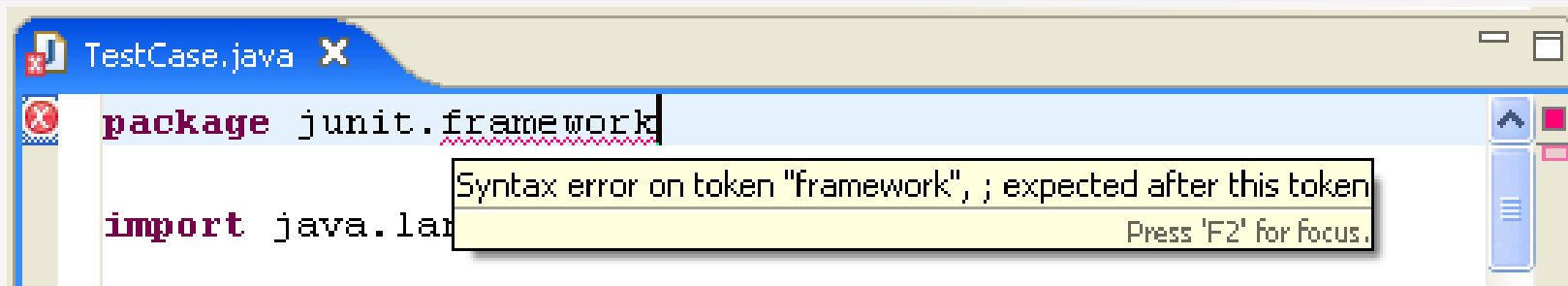
- junit.framework
  - import declarations
  - TestCase
    - TestCase()
    - TestCase(String)
    - countTestCases()
    - createResult()
    - run()
    - run(TestResult)
    - runBare()
    - runTest()
    - setUp()
    - tearDown()
    - toString()
    - getName()
    - setName(String)

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

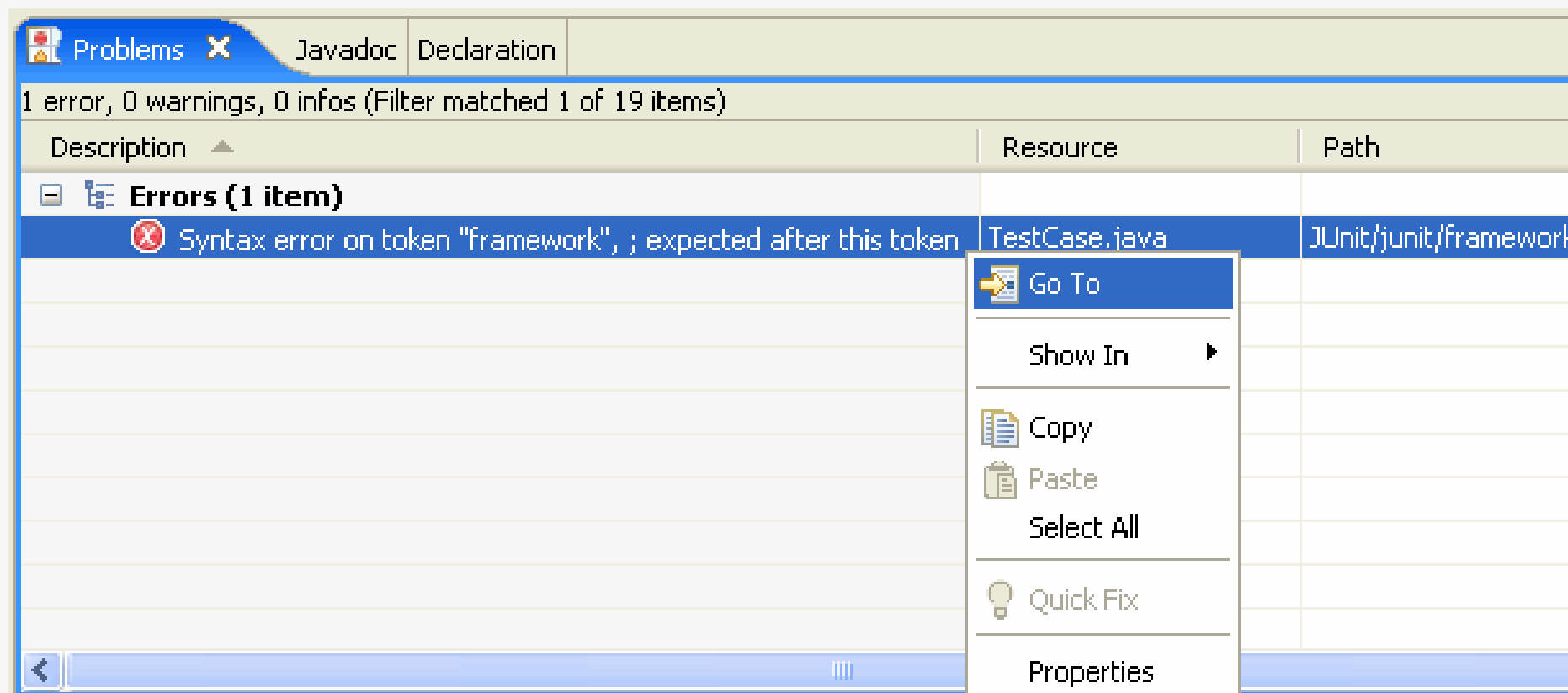
# Identifier les problèmes de code



```
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
<b>Errors (1 item)</b>		
 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 : *quick fix*

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



Clique gauche  
sur la croix

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

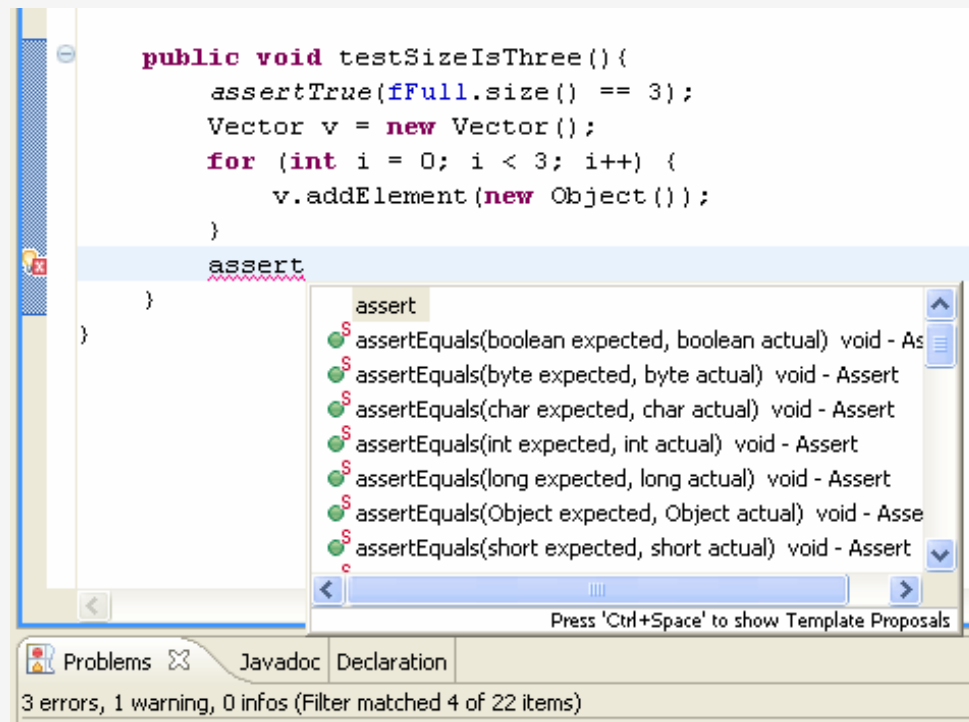
- Change to 'getName()'
- Create local variable 'fTestName'
- Create field 'fTestName'
- Change to 'fName'
- Create parameter 'fTestName'
- Create constant 'fTestName'

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



# IV complétion de code

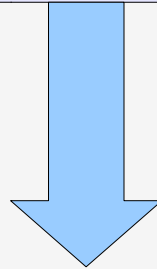
# Complétion « manuelle » : ctrl + espace



- Propose tout ce qu'il est possible de rajouter, en fonction du contexte
  - Templates (code prédéfini)
  - Variables, nom de classes, méthodes possibles

# Code prédéfini : les templates

« syso » suivit de « ctrl+space »



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

- *Tous les templates* : Window → preferences → java → editor → content Assist → Template
- Il est possible de créer ses propres templates !

# Exemple de template : *for*

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

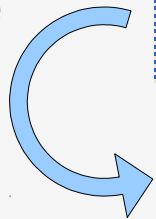
- for - iterate over array
- for - iterate over array with temporary variable
- for - iterate over collection
- foreach - iterate over an array or Iterable
- for
- ForegroundAction - javax.swing.text.StyledEditorKit
- FormAction - javax.swing.text.html.HTMLDocument.HTML
- Format - java.text

Press 'Ctrl+Space' to show Template Proposals

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

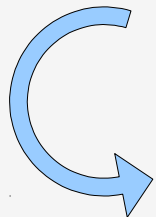
# Compléter un template

touche  
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++) {  
        |  
    }  
}
```

# Prévisualisation du résultat

```
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
```

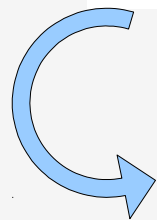
- toString() String - TestCase
- toArray - convert collection to array
- ToggleButtonBorder - com.sun.java.swing.plaf.motif.Motif
- ToggleButtonBorder - javax.swing.plaf.basic.BasicBorders
- ToggleButtonBorder - javax.swing.plaf.metal.MetalBorder
- ToggleButtonModel - javax.swing.JToggleButton
- ToHTMLSAXHandler - 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()])  
}
```

# Javadoc pour les éléments java

```
}
```

```
assertt
```

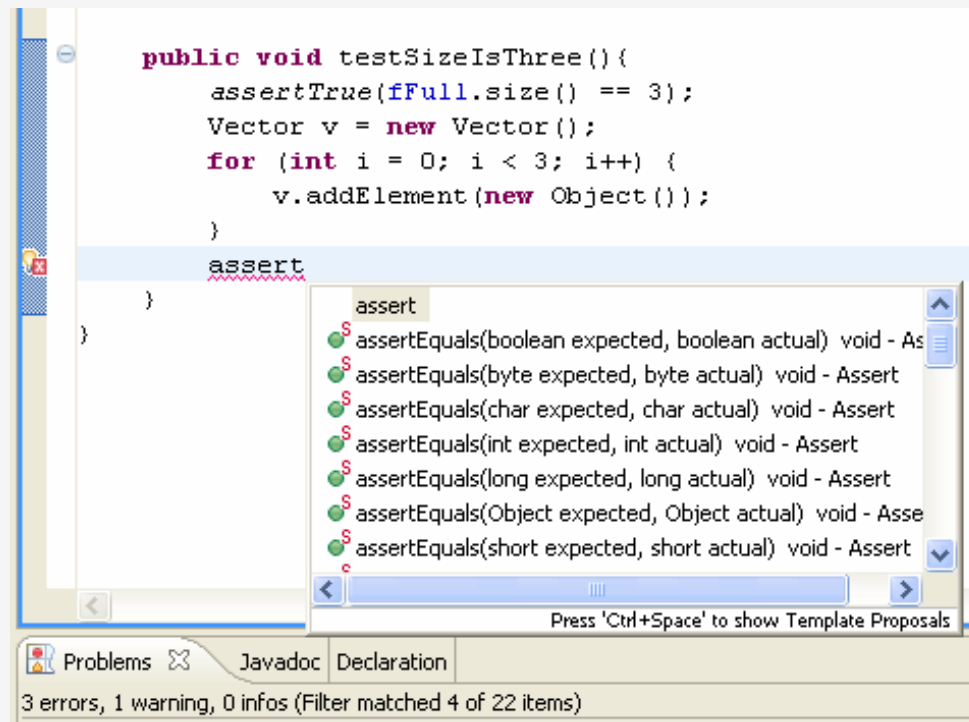
- `assertTrue(boolean condition) void - Assert`
- `assertTrue(String message, boolean condition) void - Assert`

Press 'Ctrl+Space' to show Template Proposals

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



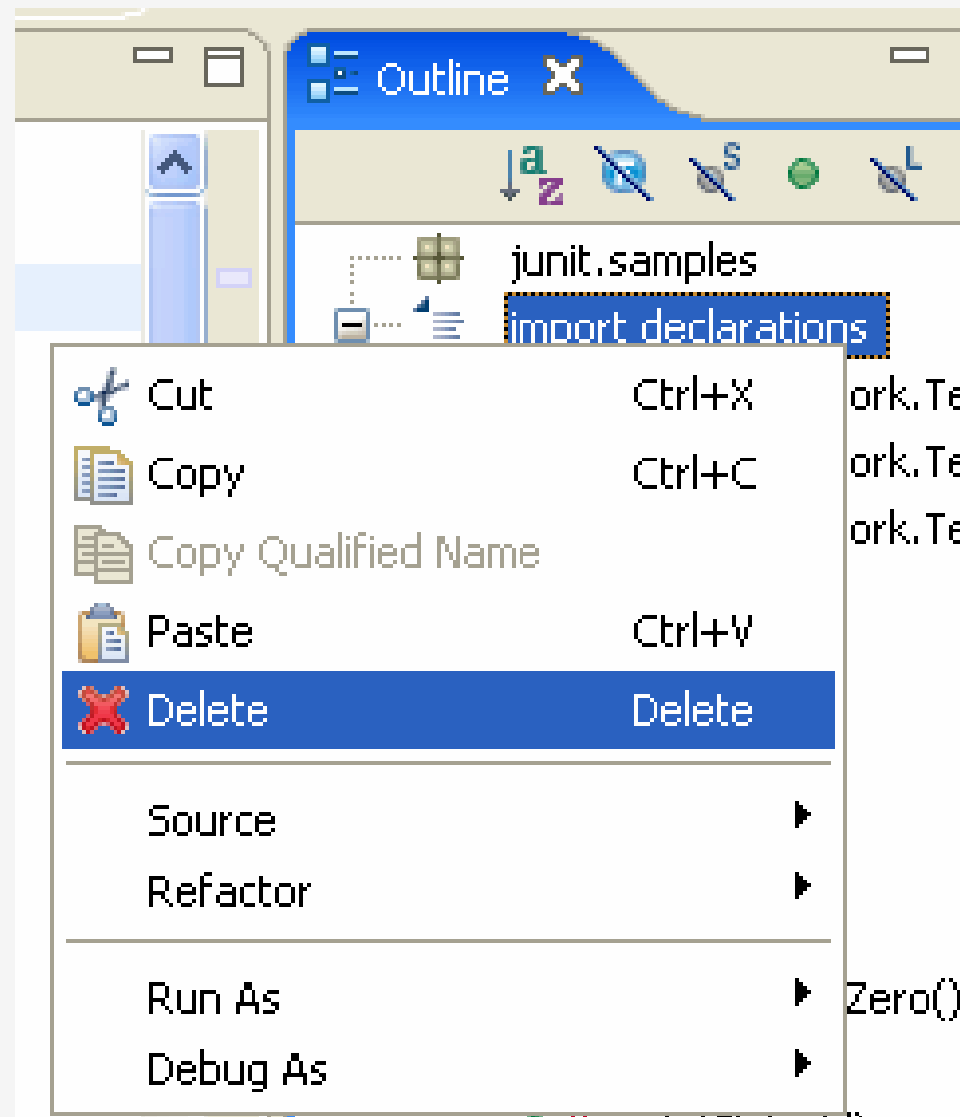
# Complétion «automatique» : après un point (recherche de méthode, attributs)



- Propose tout ce qu'il est possible de rajouter, en fonction du contexte
  - Templates (code prédéfini)
  - Variables, nom de classes, méthodes possibles

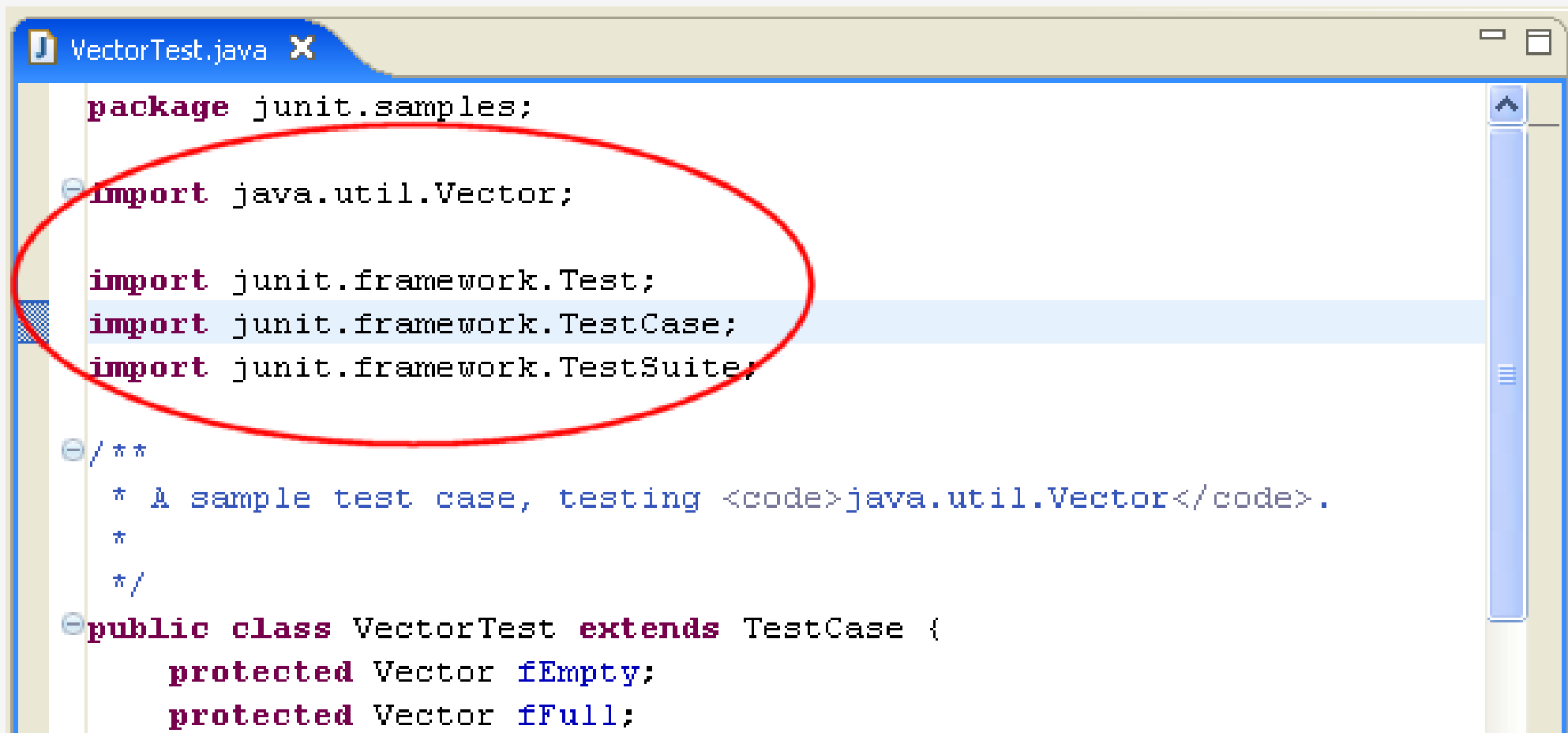
# Insertion automatique des imports

- 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;

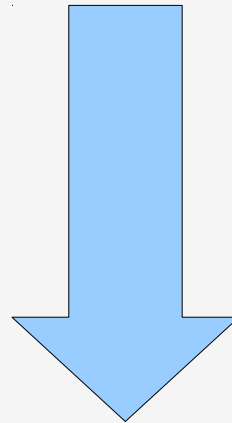
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

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

# V Naviguer dans les sources

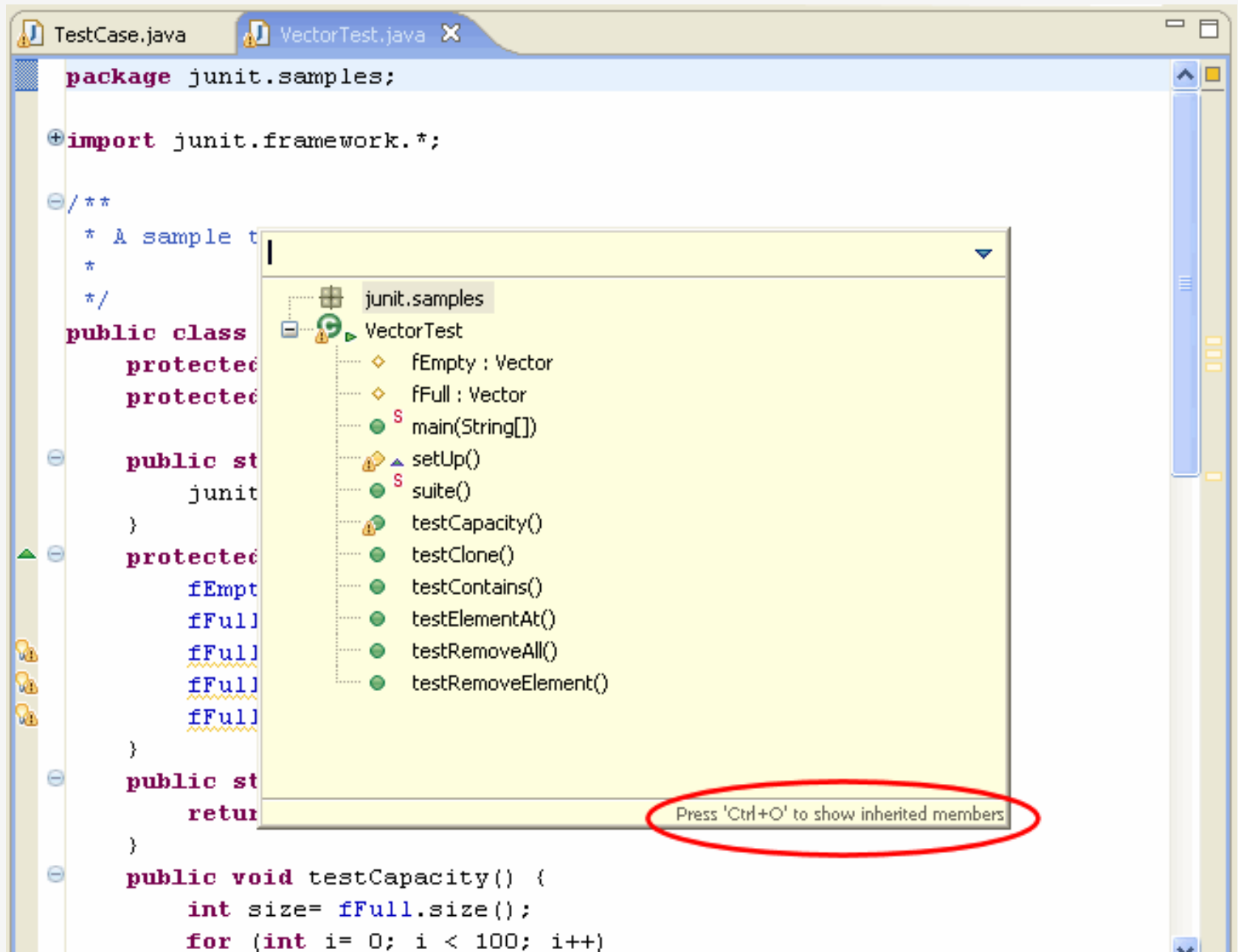
# Naviguer dans les sources

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

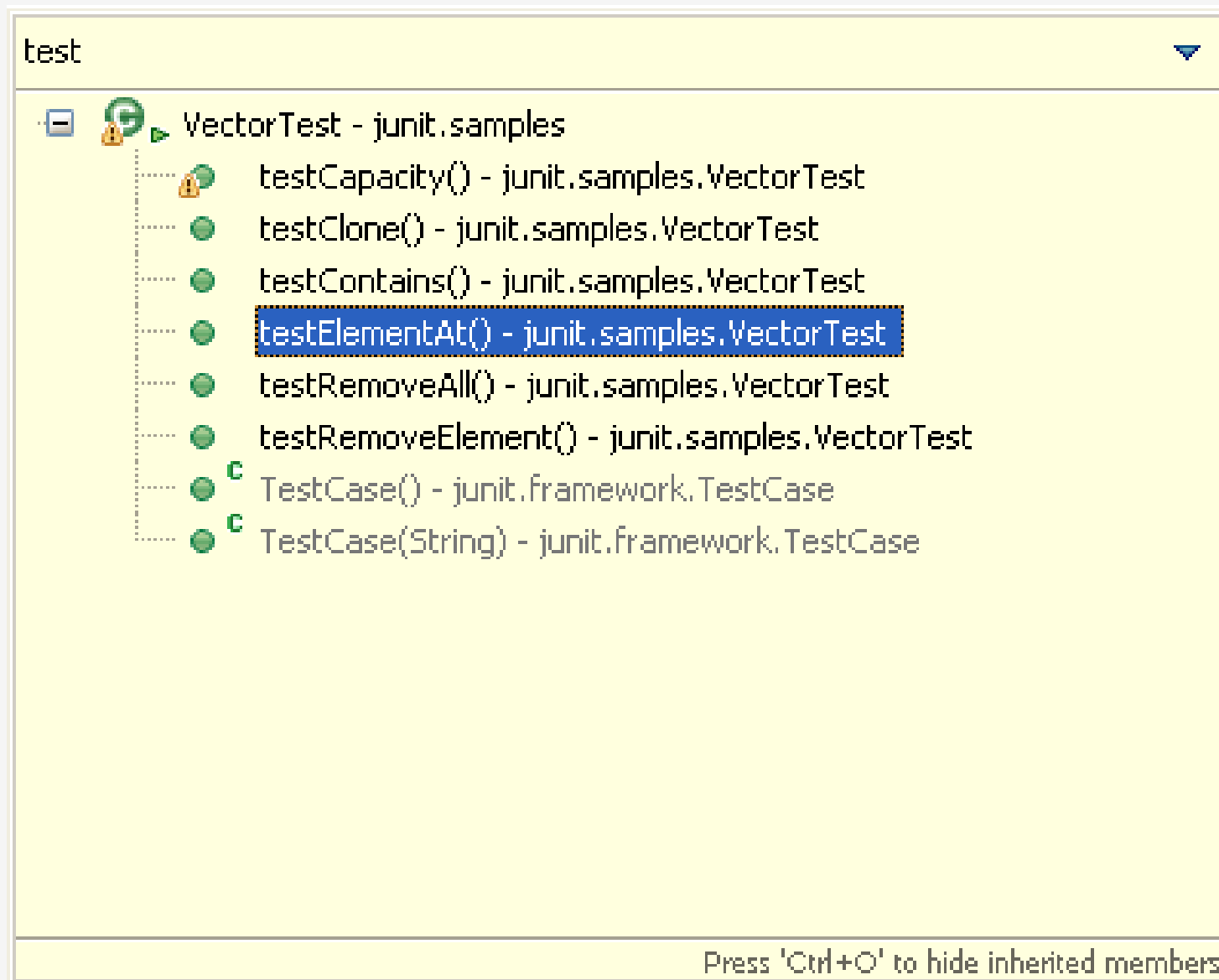


- Savoir naviguer efficacement dans les sources est **fondamental**

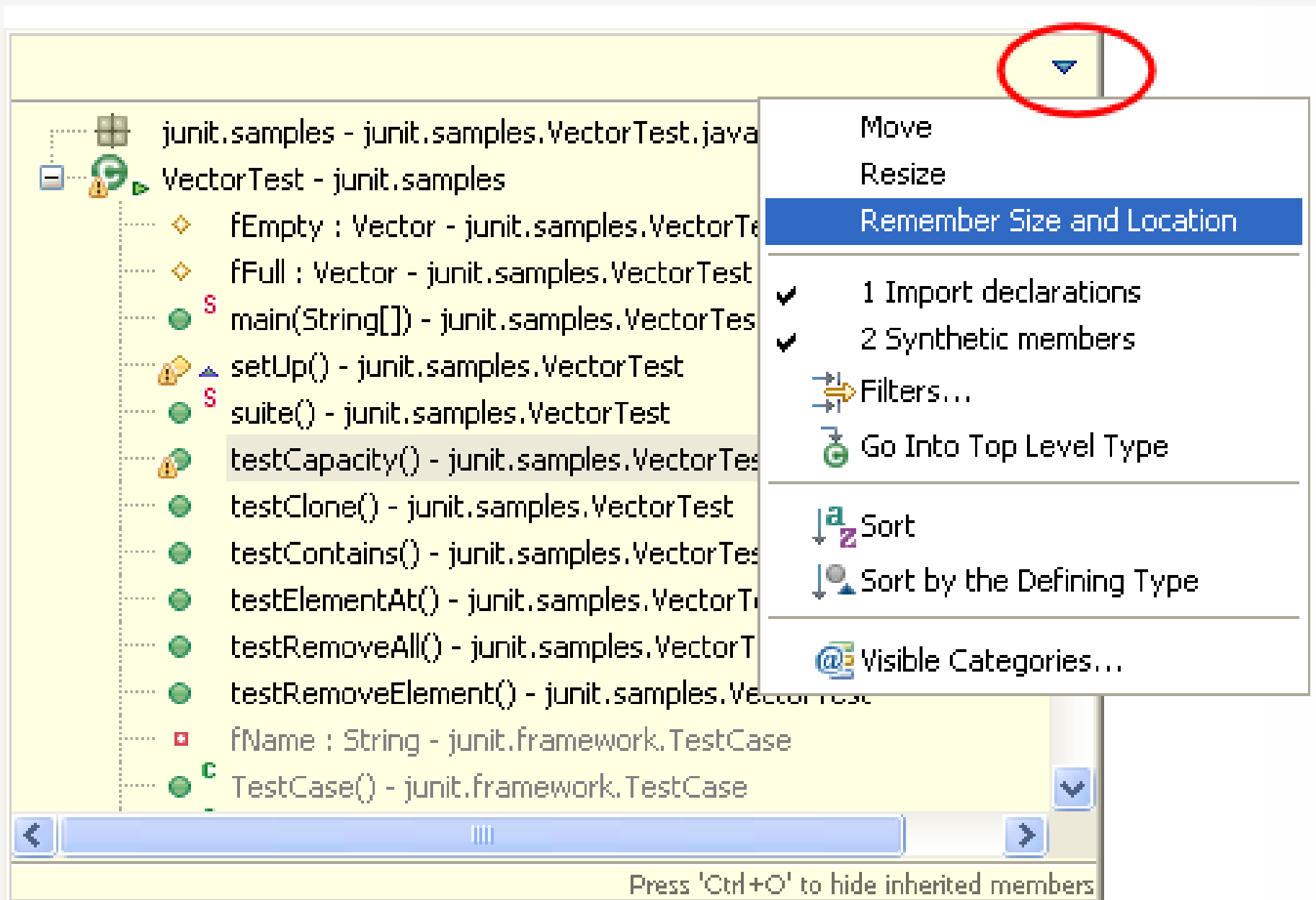
# Dans une classe → Quick outline ctrl+o



# Quick outline : recherche par frappe

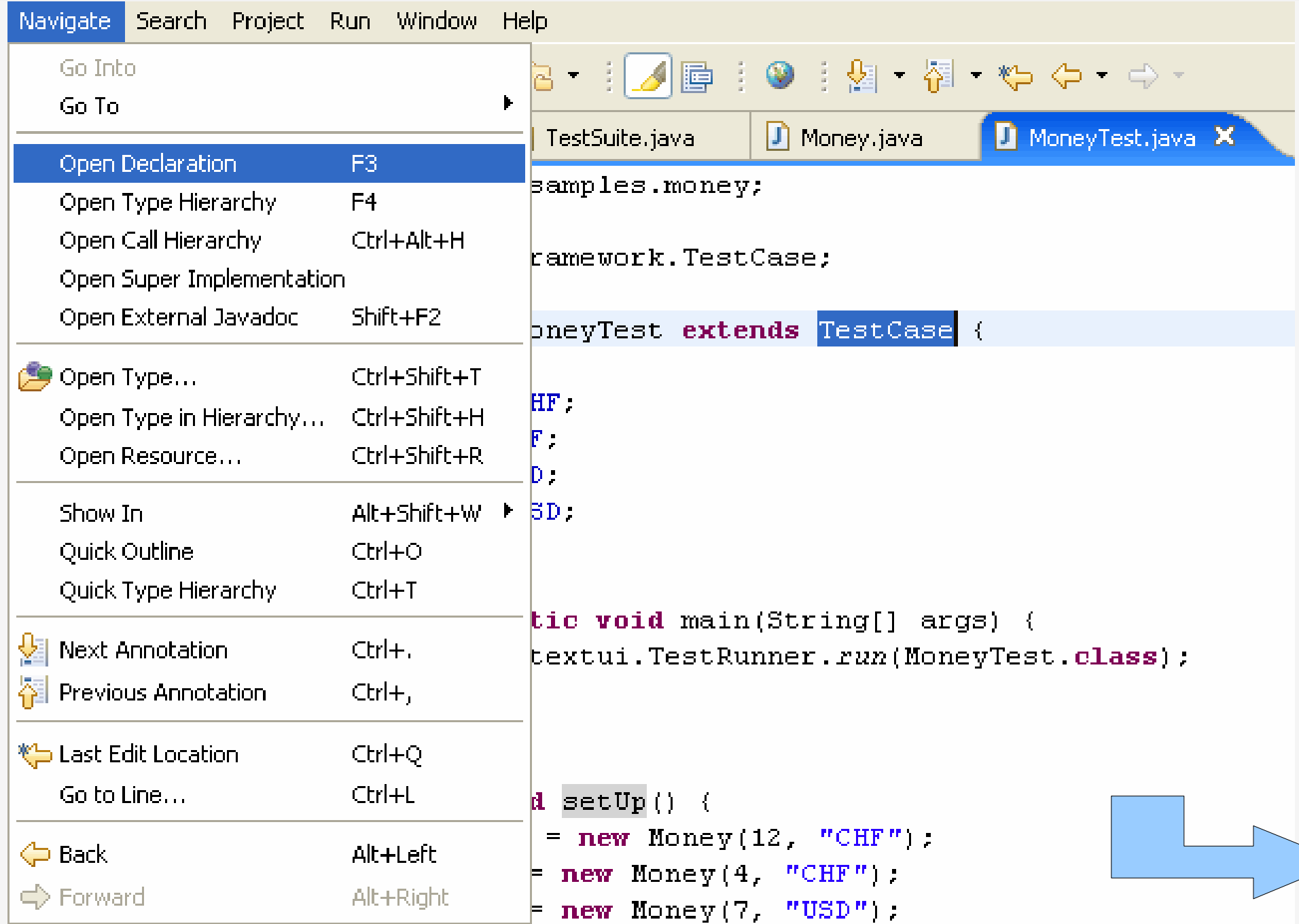


# Quick outline : menu préférences





# Ouvrir la définition d'une classe, d'une méthode ou d'un attribut : F3 ou Ctrl + clique



The screenshot shows an IDE interface with the 'Navigate' menu open. The menu items are as follows:

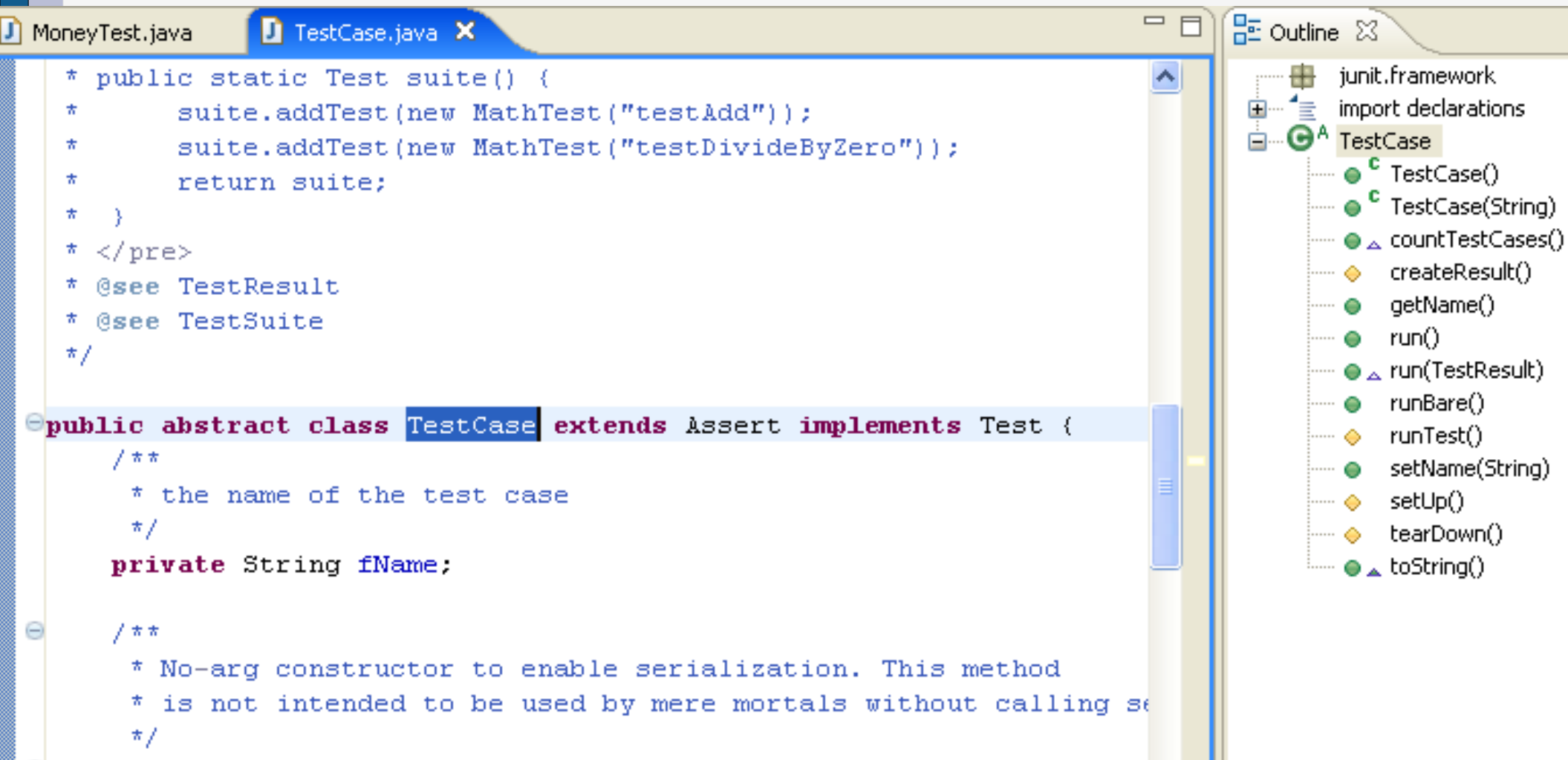
Action	Shortcut
Go Into	
Go To	
<b>Open Declaration</b>	<b>F3</b>
Open Type Hierarchy	F4
Open Call Hierarchy	Ctrl+Alt+H
Open Super Implementation	
Open External Javadoc	Shift+F2
<hr/>	
Open Type...	Ctrl+Shift+T
Open Type in Hierarchy...	Ctrl+Shift+H
Open Resource...	Ctrl+Shift+R
<hr/>	
Show In	Alt+Shift+W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

The code editor shows the following code:

```
samples.money;  
framework.TestCase;  
MoneyTest extends TestCase {  
    HF;  
    F;  
    D;  
    SD;  
    tic void main(String[] args) {  
        textui.TestRunner.run(MoneyTest.class);  
    }  
    d setUp() {  
        = new Money(12, "CHF");  
        = new Money(4, "CHF");  
        = new Money(7, "USD");  
    }  
}
```

A large blue arrow points to the right, indicating the direction of the 'Forward' navigation action.

# Ouvrir la définition d'une classe



The screenshot shows an IDE with two tabs: MoneyTest.java and TestCase.java. The TestCase.java tab is active, displaying 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;

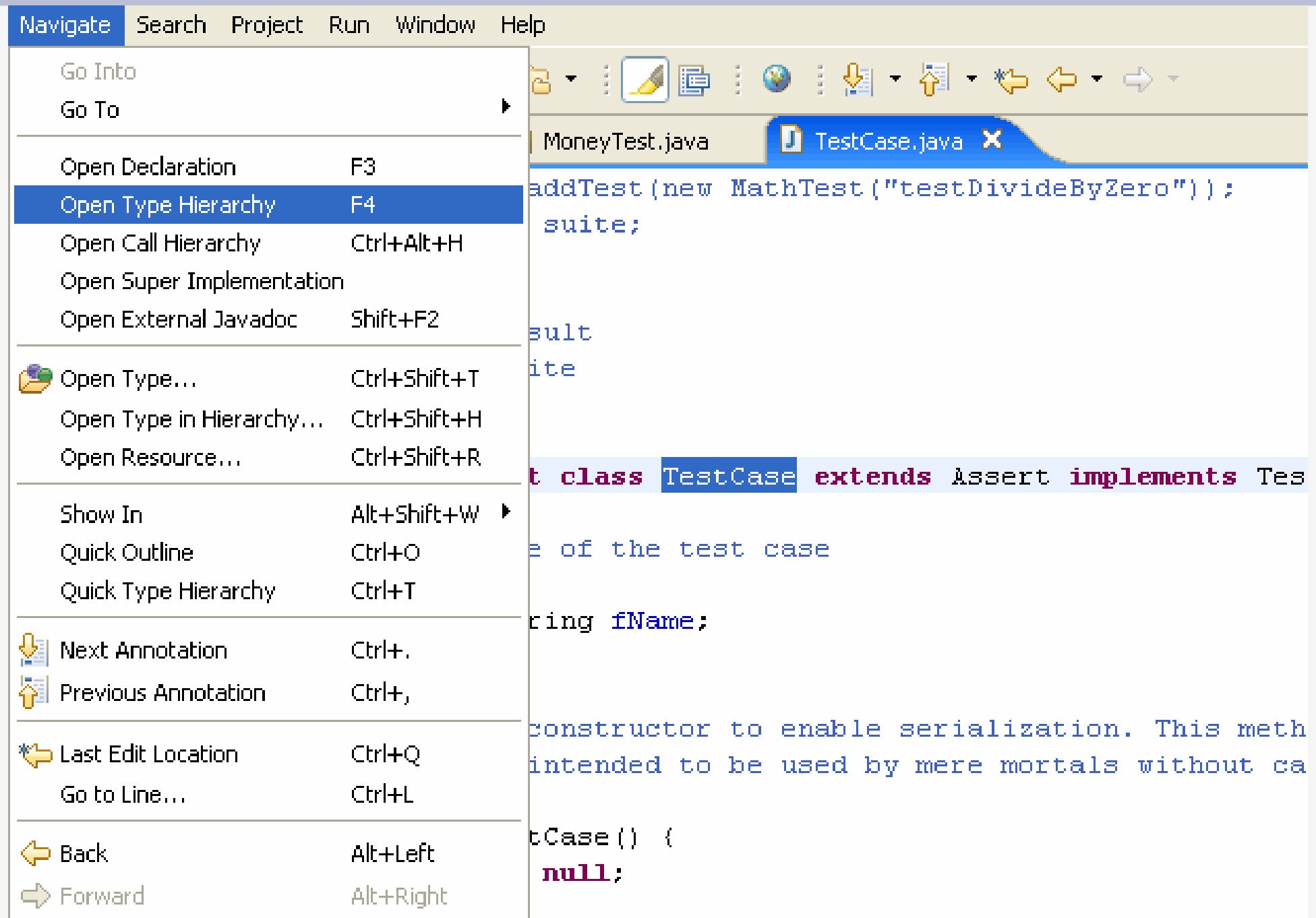
    /**
     * No-arg constructor to enable serialization. This method
     * is not intended to be used by mere mortals without calling se
     */
```

The Outline pane 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 + clique »

# Voir la hiérarchie d'un type



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

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

result
ite

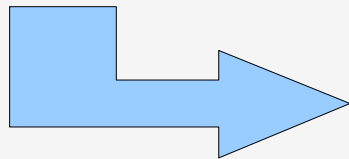
t class TestCase extends Assert implements Tes

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 package `junit.framework` is expanded, showing several Java files. `TestCase.java` is selected and highlighted with a blue dashed border.
- Hierarchy View:** Shows the class hierarchy for `TestCase`. The text `public abstract class TestCase` is visible, with `TestCase` 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 `public abstract class TestCase` line is highlighted in blue.

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

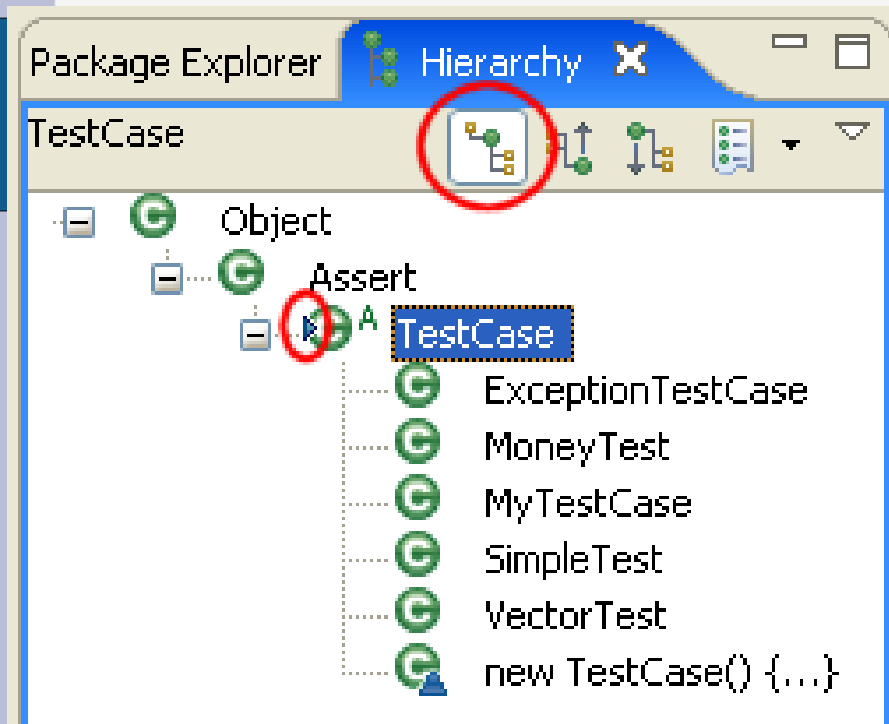
- New
- Open (F3)
- Open With
- Open Type Hierarchy (F4)
- Copy (Ctrl+Insert)

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

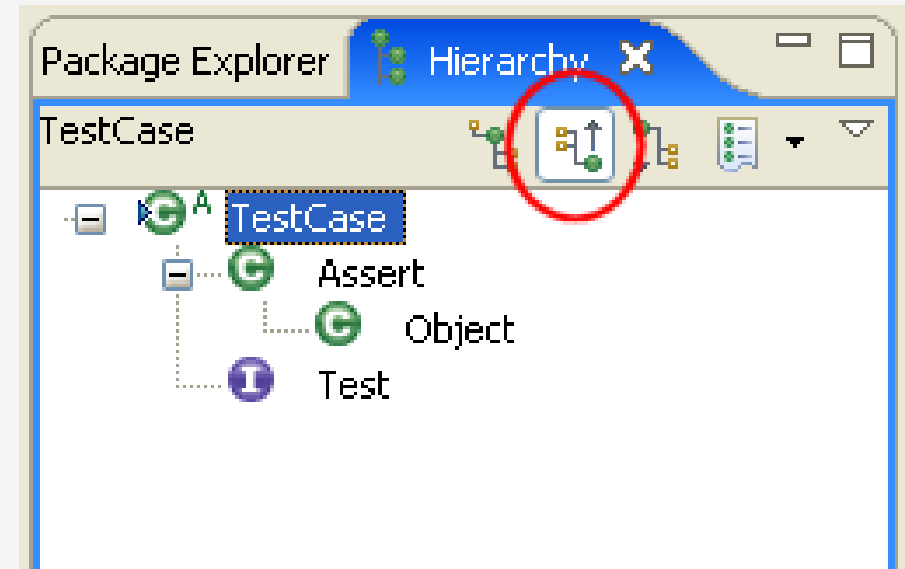
public abstract class TestCase
name of the test case
String fName;
```

# Naviguer avec la vue hiérarchique

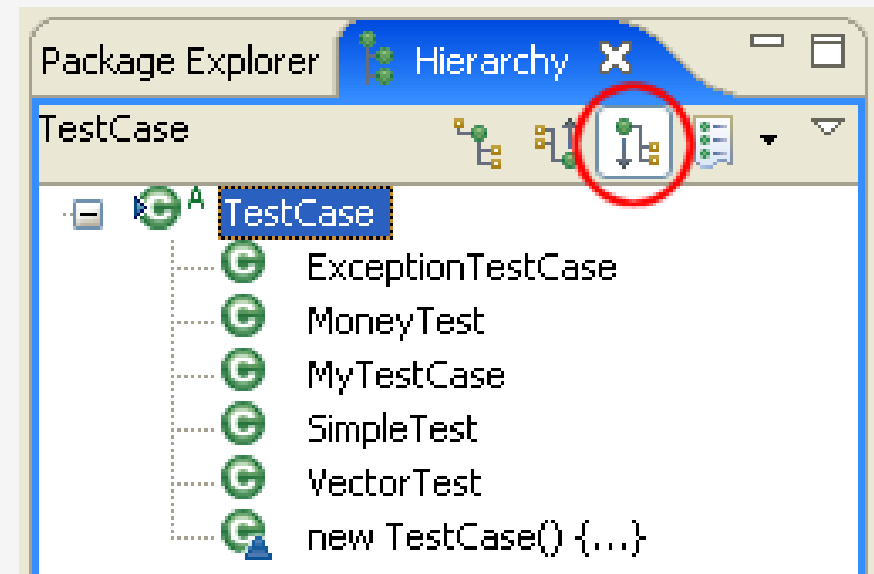
Vue globale



Vue des super types



Vue des sous types



- Quelles sous classes redéfinissent une méthode ? :

The screenshot shows an IDE window with the Package Explorer on the left and the Editor on the right. The Package Explorer displays a hierarchy for the `TestCase` class, showing its subclasses: `ExceptionTestCase` and `new TestCase() {...}`. Both subclasses have a red arrow icon next to their `runTest()` method, indicating they override it. The Editor shows the source code for the `TestCase` class, with the `runTest()` method highlighted in blue and circled in red. The toolbar above the editor has a red circle around the 'Go to Method' icon.

```
Package Explorer:
- TestCase
  - runTest()
  - ExceptionTestCase
    - runTest()
  - new TestCase() {...}
    - runTest()

Editor:
public class TestCase {
    fName
    TestCase()
    TestCase(String)
    countTestCases()
    createResult()
    getName()
    run()
    run(TestResult)
    runBare()
    runTest()
    setName(String)
    setUp()
    tearDown()
    toString()
}
```

- Où cette méthode est-elle définie ? :

The screenshot shows the Package Explorer and Hierarchy views of an IDE. The Hierarchy view at the top shows a tree structure where the `countTestCases()` method is associated with the `Test` class. The Implementation view at the bottom shows the list of methods for the `TestCase` class, with `countTestCases()` highlighted in blue. Two red circles highlight the navigation icons in the Hierarchy view: one for navigating to the implementation of the selected method, and another for navigating to the implementation of the selected class.

```
graph TD
    TestCase[TestCase] -.-> countTestCases_Hierarchy[countTestCases()]
    countTestCases_Hierarchy -.-> Test[Test]
    Test -.-> countTestCases_Implementation[countTestCases()]
```

Implementation of `TestCase`:

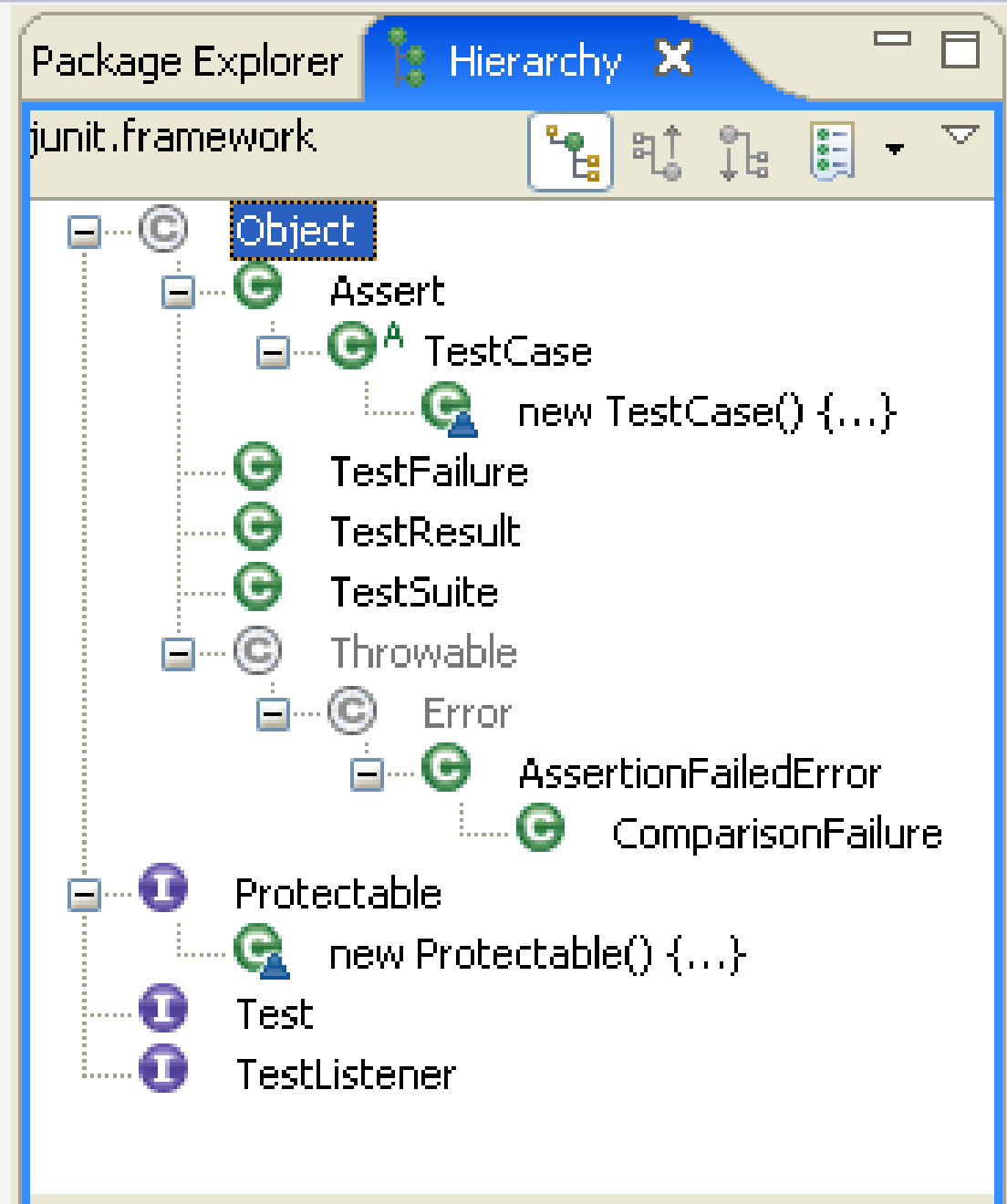
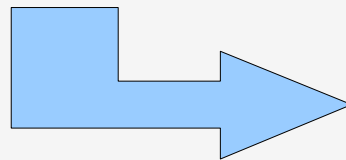
- fName
- TestCase()
- TestCase(String)
- countTestCases()**
- createResult()
- getName()
- run()
- run(ActionResult)
- runBare()
- runTest()
- setName(String)
- setUp()
- tearDown()
- toString()



# Changer d'élément référence dans la vue

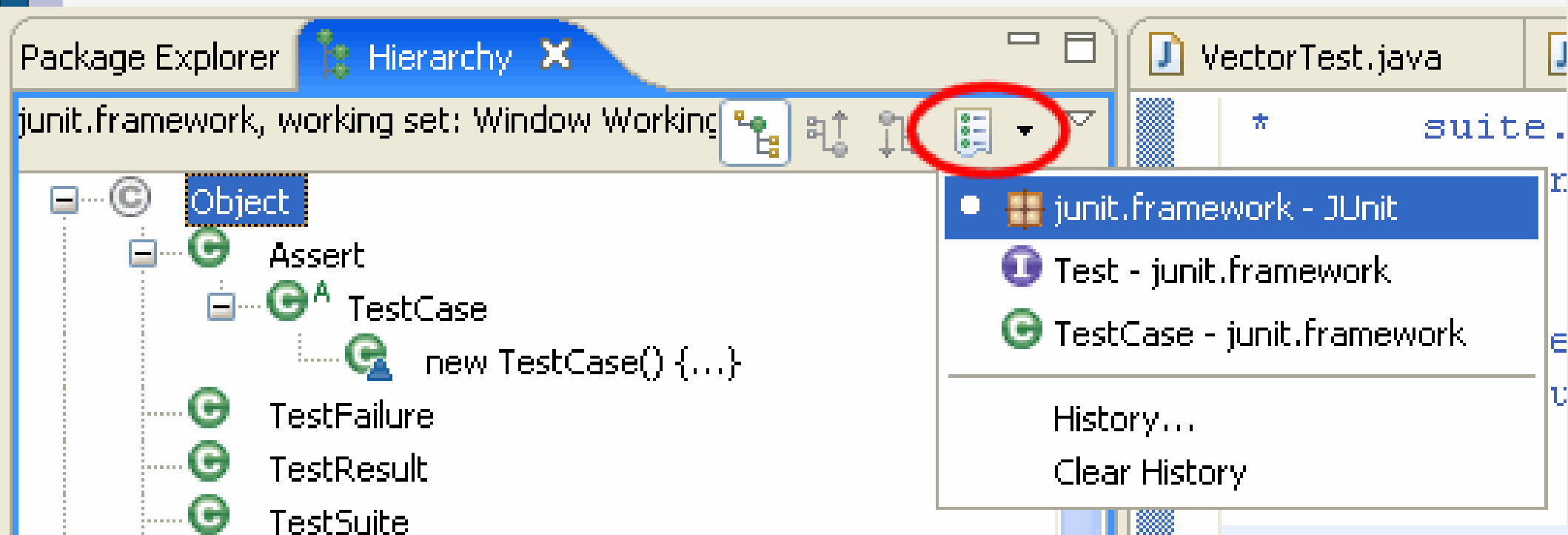
- Bouton droit sur un élément de la hiérarchie -> « focus on ... »
- Pour voir toutes les classes d'un package avec la vue hiérarchique:
  - dans le package explorer, sur un package -> « open type hierarchy »

# Naviguer avec la vue hiérarchique



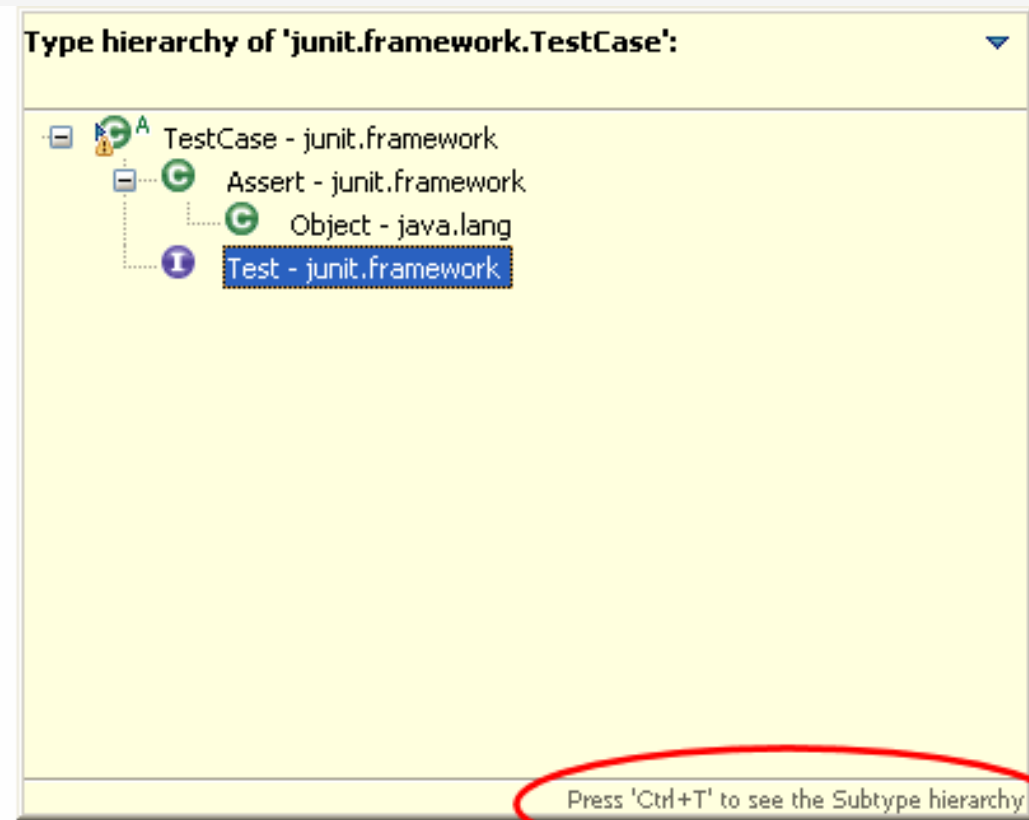
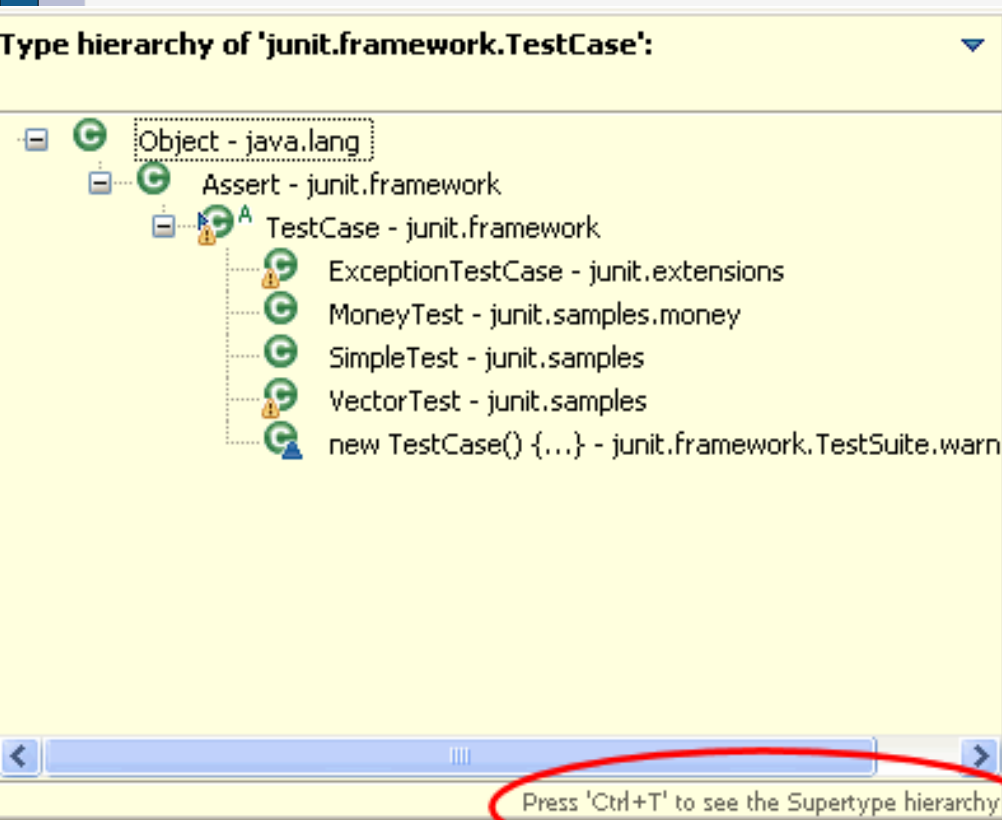
# Naviguer avec la vue hiérarchique

- 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 :

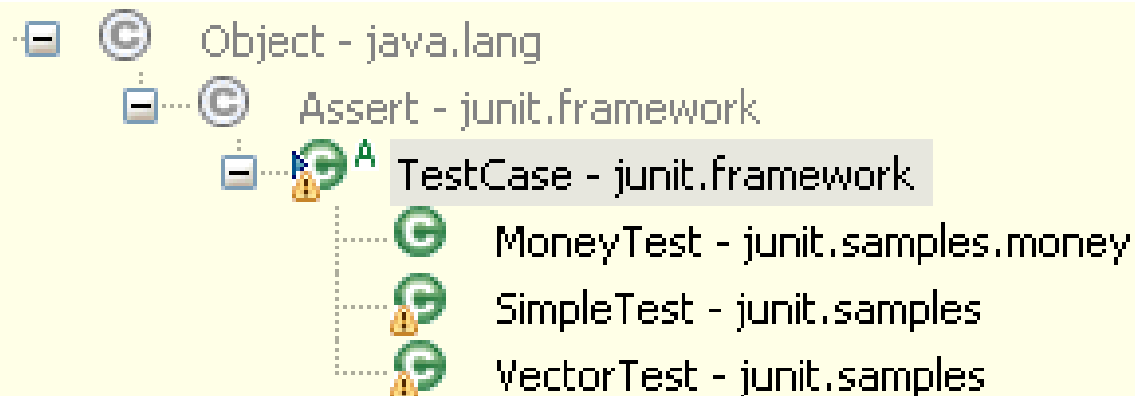


# « Quick type hierarchy »

- « 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()'



# « Java Browsing » perspective

- Window → Open Perspective → Java Browsing :

Java Browsing - TestCase.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Projects: JUnit

Packages: junit.samples, junit.samples.money, junit.tests, junit.tests.extensions, junit.tests.framework, junit.tests.runner, junit.textui, org.junit, org.junit.internal.requests, org.junit.internal.runners

Types: JUnit4TestAdapterCache, JUnit4TestCaseFacade, Protectable, Test, TestCase, TestFailure, TestListener, TestResult, TestSuite

Members: import declarations, fName : String, TestCase(), TestCase(String), countTestCases(), createResult(), run(), run(TestResult)

```
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.
 */
```

junit.framework.TestCase.run() : TestResult - JUnit

# 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 displays the Eclipse IDE interface with the following components:

- Package Explorer (Left):** Shows the project structure for 'tamagoshi', including sub-packages like 'tamagoshi.graphic' and 'tamagoshi.jeu', and source files such as 'TamaFrame.java', 'TamaPanel.java', 'TamaGame.java', etc.
- Editor (Center):** Displays the source code of 'Tamagoshi.java'. The code includes:

```
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;
}
```
- Outline (Right):** Shows the class hierarchy for 'Tamagoshi', listing 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 (Bottom):** Shows the output of a Java application, including a cycle indicator and dialogue lines:

```
<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 !"
```



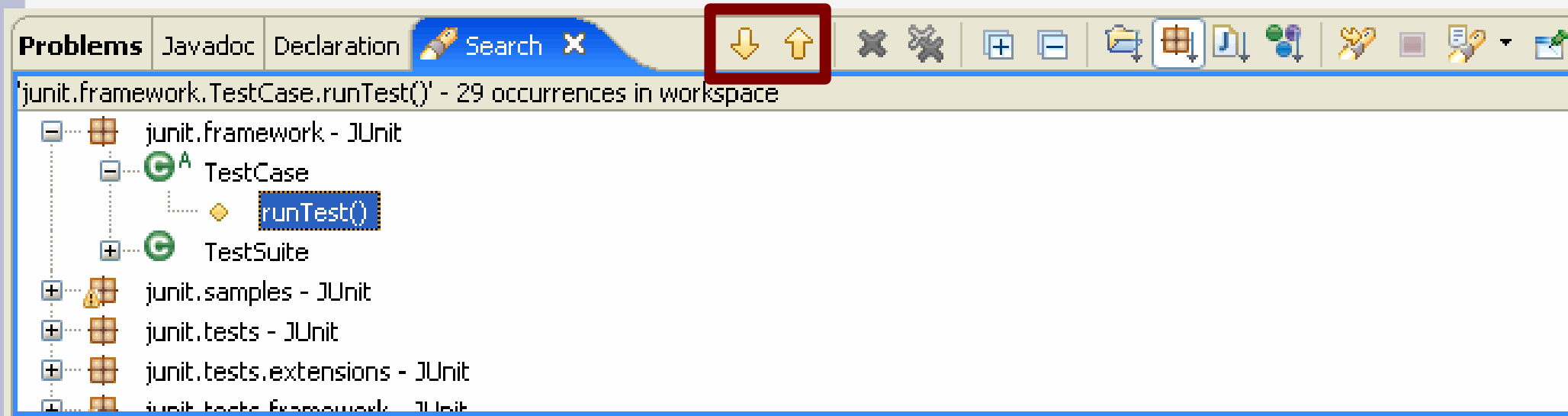
# Rechercher par nom de méthode

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: [Empty text box] [Choose...]

Buttons at the bottom: [Customize...], [Search], [Cancel].

# Rechercher par nom de méthode



```

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);
}

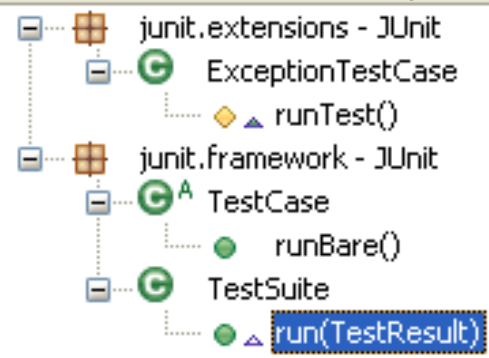
/**
 * Returns the number of tests in this suite

```

- 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



# Recherche depuis l'éditeur

The screenshot shows an IDE window with a Java file named 'Assert.java'. The code is as follows:

```
    }  
    /**  
     * 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,  
        if (expected == null && actual == null)  
            return;  
        if (expected != null && expected.equals(actual))  
            return;  
        failNotEquals(message, expected, actual);  
    }  
    /**  
     * Asserts that two objects are equal.  
     * an AssertionError is thrown with the given message.  
     */  
    static public void assertEquals(Object expected, Object actual)
```

The 'fail()' method is selected, and a context menu is open. The 'References' option is highlighted, which has opened a sub-menu showing the following options:

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

In the background, the 'Outline' window shows the project structure with 'Assert 1.22 (ASCII-k)' expanded, listing methods: Assert(), assertTrue(String), assertTrue(boolean), assertFalse(String), assertFalse(boolean), fail(String), and fail().

```

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

```

```

public abstract class TestCase 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
- fNa
- Tes
- Tes
- col
- cre
- run
- 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:

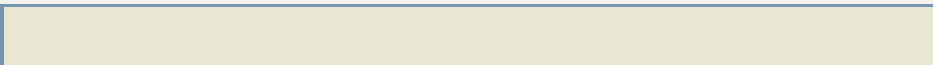
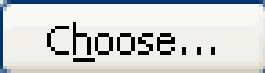
\*.java  




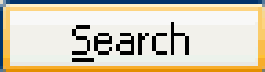

Patterns are separated by a comma (\* = any string, ? = any character)

Consider derived resources

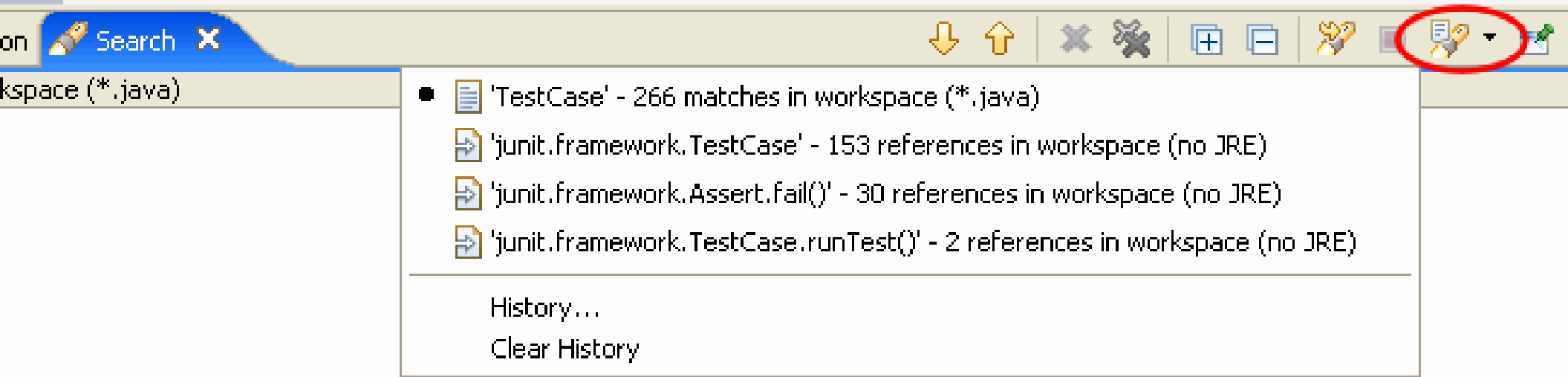
Scope

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 is a toolbar with various icons, including a search icon circled in red. The main content area displays a list of search results for the query "workspace (\*.java)".

workspace (\*.java)

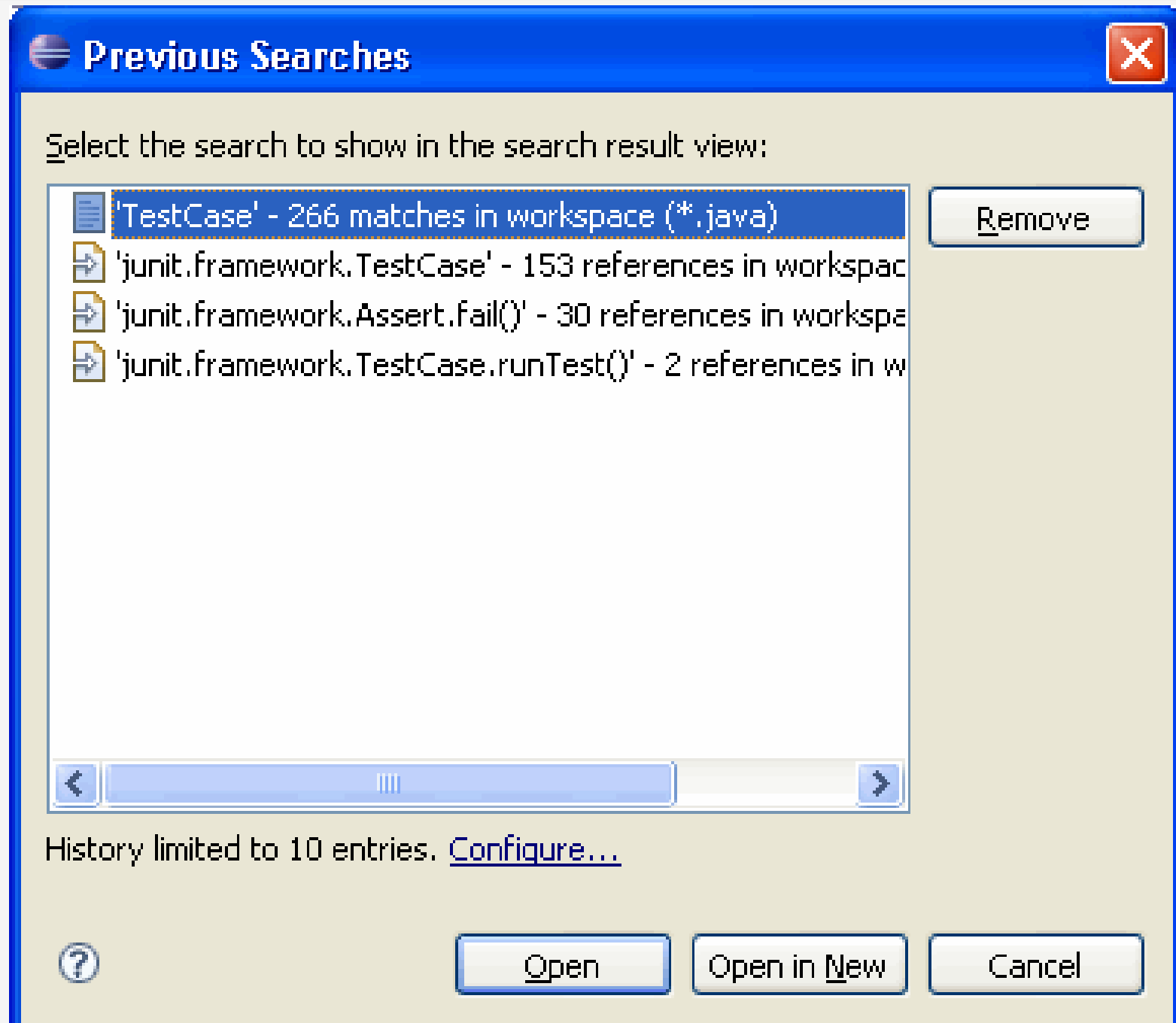
- 'TestCase' - 266 matches in workspace (\*.java)
- 'junit.framework.TestCase' - 153 references in workspace (no JRE)
- 'junit.framework.Assert.fail()' - 30 references in workspace (no JRE)
- 'junit.framework.TestCase.runTest()' - 2 references in workspace (no JRE)

---

History...

Clear History

# Historique des recherches





# VI Le menu *Source*

# Le menu *Source* : commentaires

Name	Function	Keyboard Shortcut
Toggle Comment	Comment or uncomment all lines containing the current selection.	Ctrl + /
Add Block Comment	Adds a block comment around all lines containing the current selection.	Ctrl + Shift + /
Remove Block Comment	Removes a block comment from all lines containing the current selection.	Ctrl + Shift + \
Generate Element Comment	Adds a comment to the selected element. See the <a href="#">Code templates preference page</a> to specify the format of the generated comments. Available on types, fields, constructors, and methods.	Alt + Shift + J

# Le menu *Source* : *formatage*

Shift Right	Increments the level of indentation of the currently select lines. Only activated when the selection covers multiple lines or a single whole line.	
Shift Left	Decrements the level of indentation of the currently select lines. Only activated when the selection covers multiple lines or a single whole line.	
Correct Indentation	Corrects the indentation of the lines denoted by the current text selection.	Ctrl + I
Format	Uses the code formatter to format the current text selection. The formatting options are configured on the <a href="#">Code Formatter preference page</a> .	Ctrl + Shift + F
Format Element	Uses the code formatter to format the Java element comprising the current text selection. The Format Element action works on method and type level. The formatting options are configured on the <a href="#">Code Formatter preference page</a> .	
Add Import	Creates an import declaration for a type reference currently selected. If the type reference is qualified, the qualification will be removed if possible. If the referenced type name can not be mapped uniquely to a type of the current project you will be prompted to specify the correct type. Add Import tries to follow the import order as specified in the <a href="#">Organize Import preference page</a>	Ctrl + Shift + M
Organize Imports	Organizes the import declarations in the compilation unit currently open or selected. Unnecessary import declarations are removed, and required import declarations are ordered as specified in the <a href="#">Organize Imports preference page</a> . Organize imports can be executed on incomplete source and will prompt you when a referenced type name can not be mapped uniquely to a type in the current project. You can also organize multiple compilation units by invoking the action on a package or selecting a set of compilation units.	Ctrl + Shift + O
Sort Members	Sorts the members of a type according to the sorting order specified in the <a href="#">Member Sort Order preference page</a>	
Clean Up	Performs various changes in order to clean up your code according to the settings specified in the <a href="#">Clean Up preference page</a>	

# Source → génération de code

Override/Implement Methods	Opens the <a href="#">Override Method dialog</a> that allows you to override or implement a method in the current type. Available on types or on a text selection inside a type.	
Generate Getter and Setter	Opens the <a href="#">Generate Getters and Setters dialog</a> that allows you to create Getters and Setters for fields in the current type. Available on fields and types or on a text selection inside a type.	
Generate Delegate Methods	Opens the Generate Delegate Methods dialog that allows you to create method delegates for fields in the current type. Available on fields and types with fields.	
Generate hashCode() and equals()	Opens the Generate HashCode and Equals dialog that allows you to start and control the generation of hashCode and equals methods in the current type.	
Generate toString()	Opens the <a href="#">Generate toString() dialog</a> that allows you to start and control the generation of a toString() method in the current type.	
Generate Constructor using Fields	Adds constructors which initialize fields for the currently selected types. Available on types, fields or on a text selection inside a type.	
Add Constructor from Superclass	Adds constructors as defined in the super class for the currently selected types. Available on types or on a text selection inside a type.	
Surround With	Surround the selected statements with a code template. Create your own templates on the <a href="#">Template preference page</a> . Further, you can use <i>Expand Selection to</i> from the <a href="#">Edit</a> menu to get a valid selection range.	Alt + Shift + Z
Externalize Strings	Opens the Externalize strings wizard. This wizard allows you to replace all strings in the code by statements accessing a property file.	
Find Broken Externalized Strings	Searches for broken externalized strings in a selected property file, package, project or set of projects.	

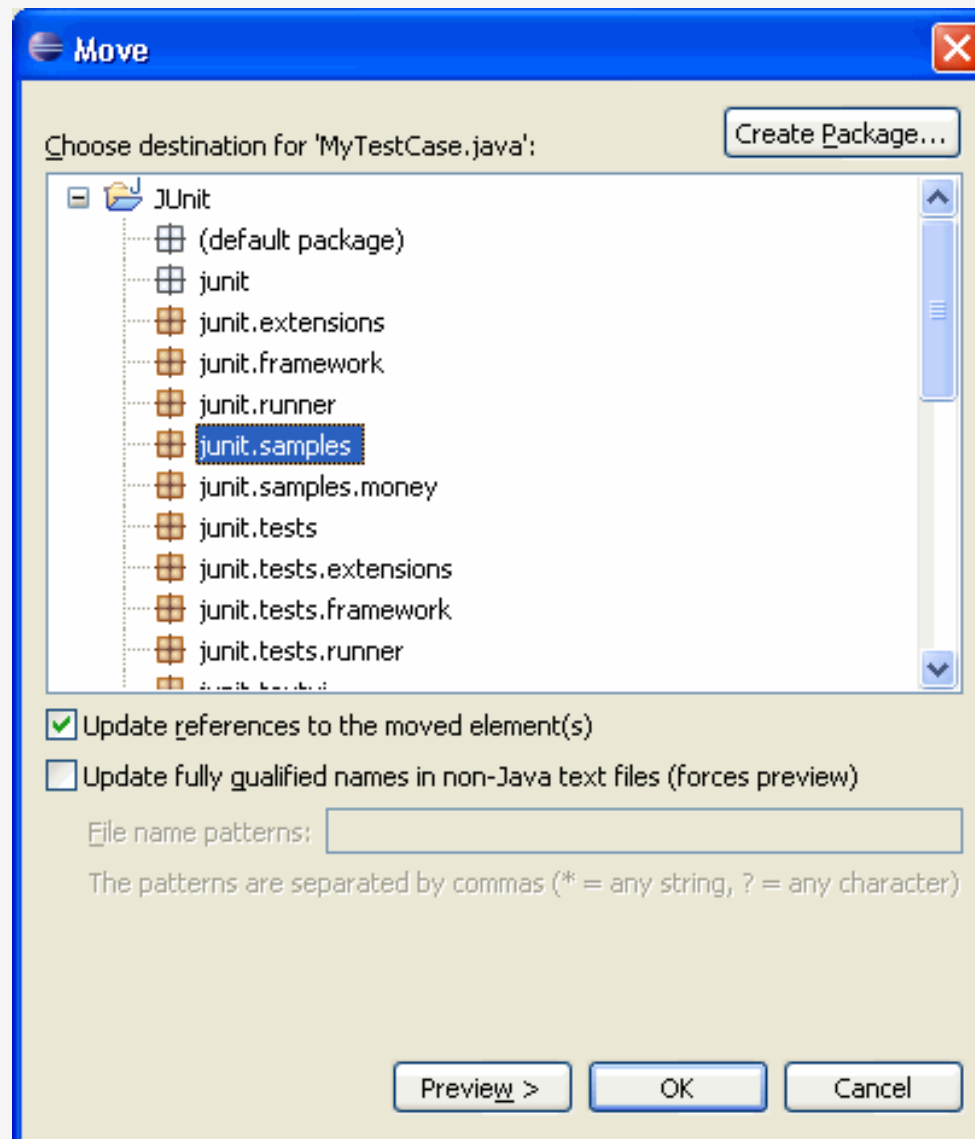
# VII Le menu *Refactor*

# 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 »

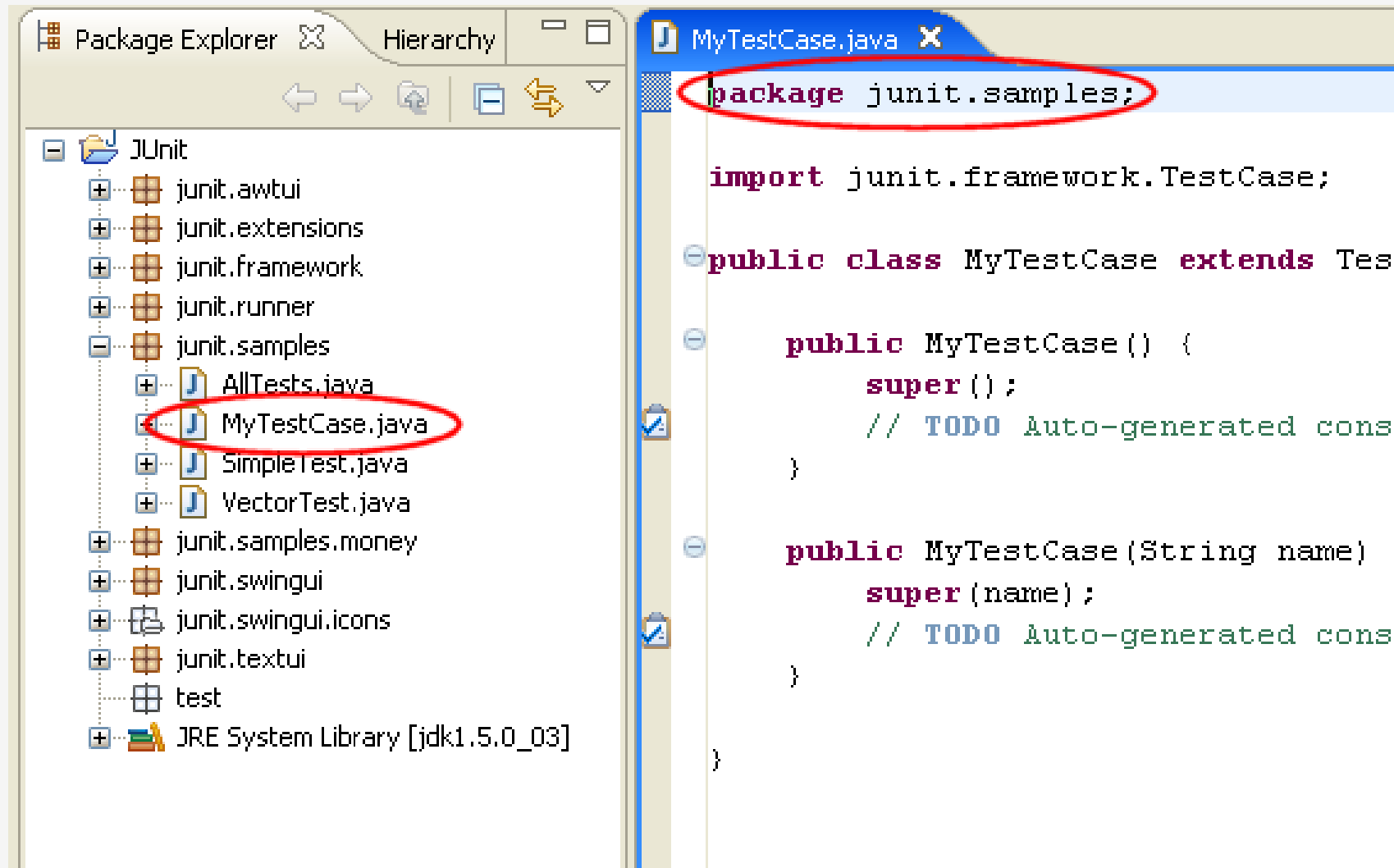
# Changer le package d'une classe

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



# Changer le package d'une classe

- « Drag and drop »



The screenshot shows an IDE interface with two main panels. 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', showing the source code. The first line of the code, 'package junit.samples;', is also circled in red. The code includes an import statement for 'junit.framework.TestCase', a class declaration 'public class MyTestCase extends TestCase', and two constructor methods: 'public MyTestCase()' and 'public MyTestCase(String name)'. Both constructors call 'super()' or 'super(name)' and have a comment '// TODO Auto-generated constructor stub'.

```
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
    }
}
```



# 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

# 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

```
Class superClass= theClass;
Vector names= new Vector();
```

```
wh
```

- 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 ▶
- Add to Snippets...
- Run As ▶
- Debug As ▶
- Profile As ▶
- Team ▶
- Compare With ▶
- Replace With ▶
- Preferences...

```
From(superClass) {
    class.getDeclaredMethods();
    .length; i++) {
        i], names, theClass);
    }
}
Superclass();

found in "+theClass.getName());
```

- Move... Alt+Shift+V
- Change Method Signature... Alt+Shift+C
- Extract Method... Alt+Shift+M**
- Extract Interface...
- Use Supertype Where Possible...

Path	Location

```

}
if
}
/**
 * Cons
 */
public
set
}
/**
 * Adds
 */
public
fTe
```

```
ms
warnings, 0
ion
```

## 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 >

OK

Cancel

# 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 &lt; 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 comment:

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

The `Outline` view on the right shows the class structure for `TestSuite`. The methods listed are:

- `createTest(Class, String)`
- `exceptionToString(Throwable)`
- `getTestConstructor(Class)`
- `warning(String)`
- `TestSuite()`
- `TestSuite(Class)`
- `TestSuite(Class, String)`
- `TestSuite(String)`
- `addTest(Test)`
- `addTestMethod(Method, Vector)`
- `addTestSuite(Class)`
- `collectedInheritedTests(Class)` (highlighted)
- `countTestCases()`
- `netName()`

# « 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:

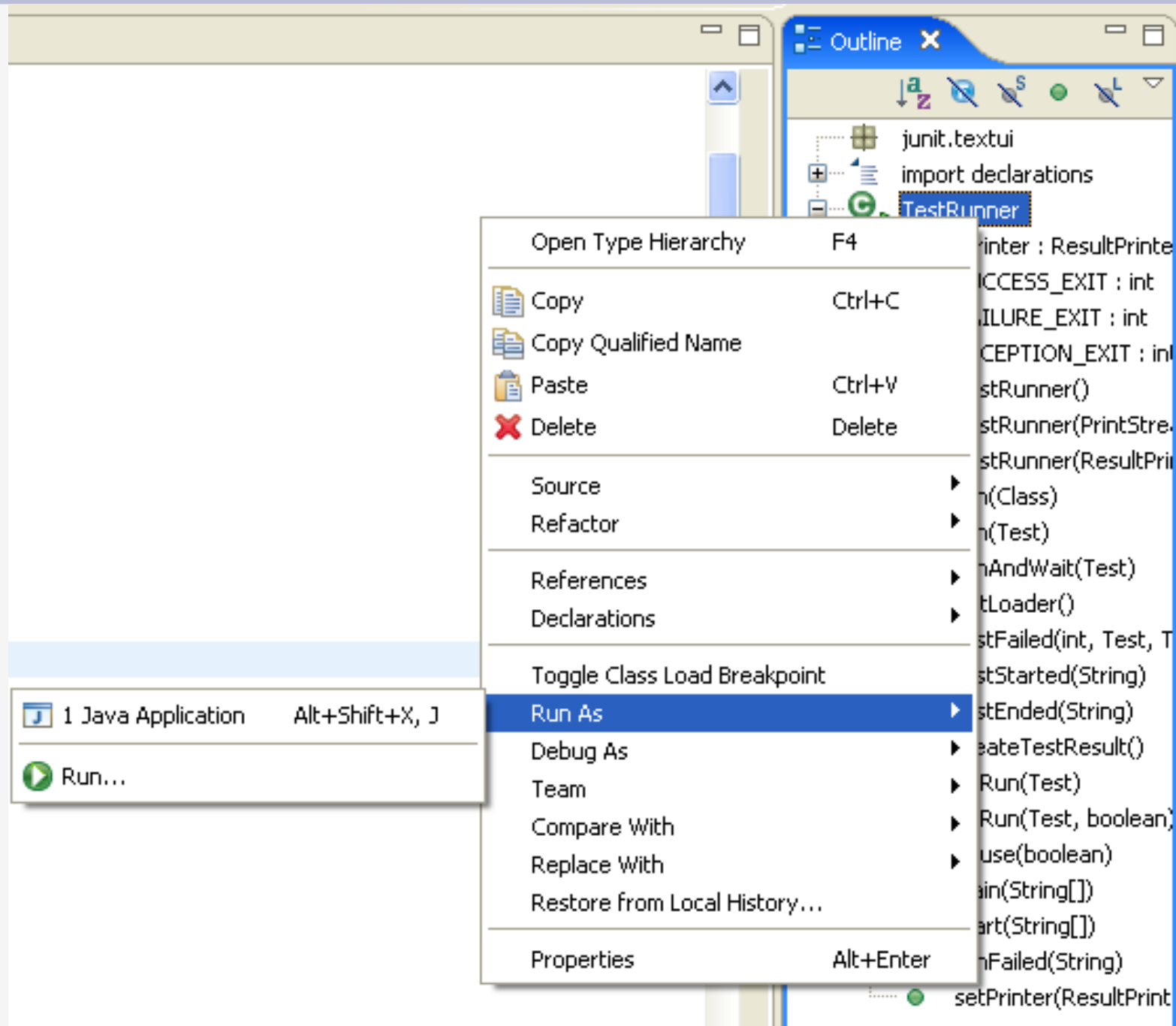
MenuItem	Shortcut
Undo Rename Compilation Unit	Ctrl+Z
Redo Typing	Ctrl+Y
<hr/>	
Cut	Ctrl+X
Copy	Ctrl+C
Copy Qualified Name	
Paste	Ctrl+V
<hr/>	
Delete	Delete
Select All	Ctrl+A
Expand Selection To	
<hr/>	
Find/Replace...	Ctrl+F
Find Next	Ctrl+K

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', 'ee T', 'c ab', and '\*\*'.

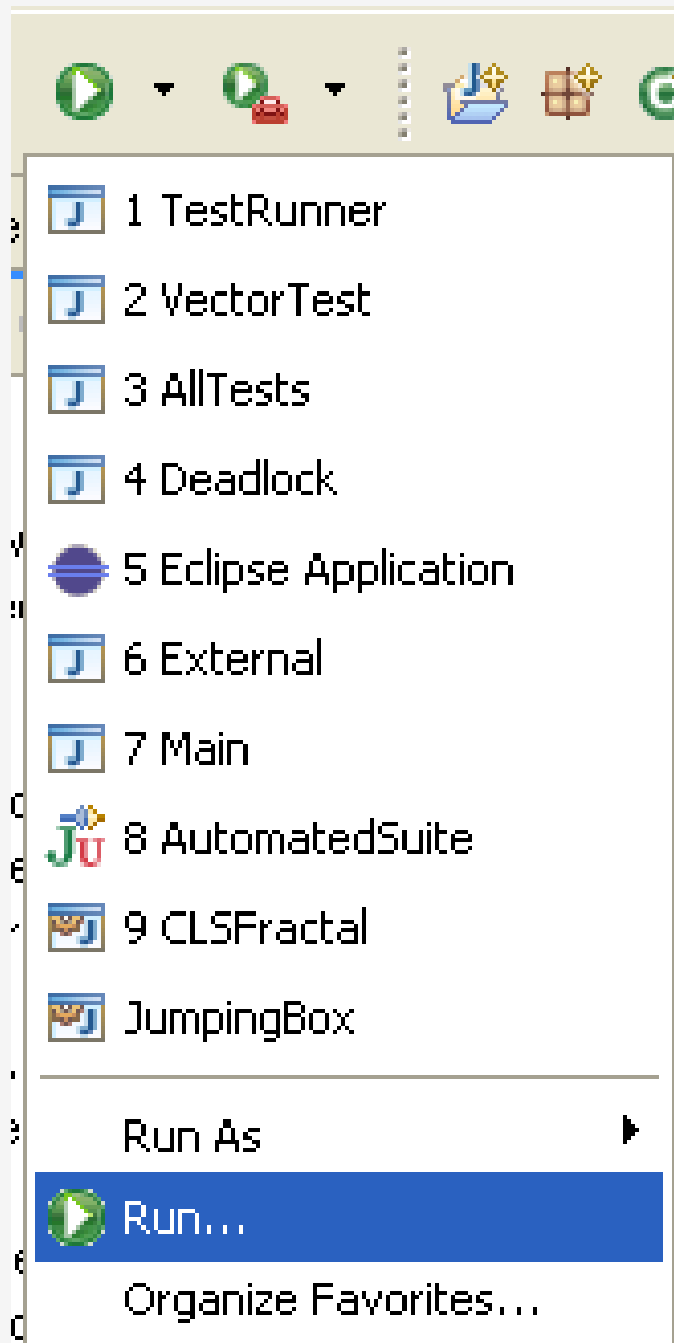


## VIII Configurer les « runs »

# « Run As »



# Spécifier des arguments



# Spécifier des arguments

The screenshot shows the 'Run' dialog box in the Eclipse IDE. The title bar reads 'Run' and the subtitle is 'Create, manage, and run configurations'. Below the subtitle, it says 'Run a Java application' and there is a green play button icon. The dialog is divided into two main sections. On the left is a tree view of configuration types, with 'TestRunner' selected under 'Java Application'. On the right, the configuration details are shown for 'TestRunner'. The 'Name' field contains 'TestRunner'. Below the name are several tabs: 'Main' (selected), 'Arguments', 'JRE', 'Classpath', 'Source', 'Environment', and 'Common'. The 'Project' field contains 'JUnit' with a 'Browse...' button. The 'Main class' field contains 'junit.textui.TestRunner' with 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', and 'Stop in main', all of which are currently unchecked.

Run

Create, manage, and run configurations

Run a Java application

Name: TestRunner

Main Arguments JRE Classpath Source Environment Common

Project: JUnit Browse...

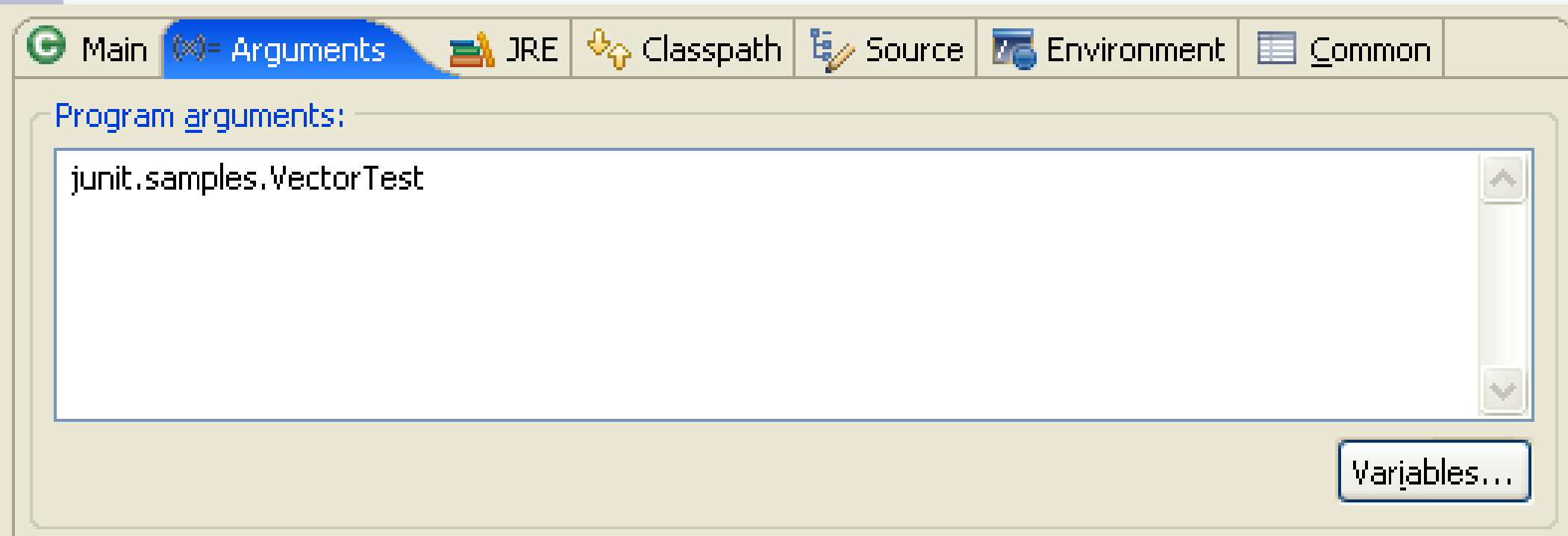
Main class: junit.textui.TestRunner Search...

Include libraries when searching for a main class

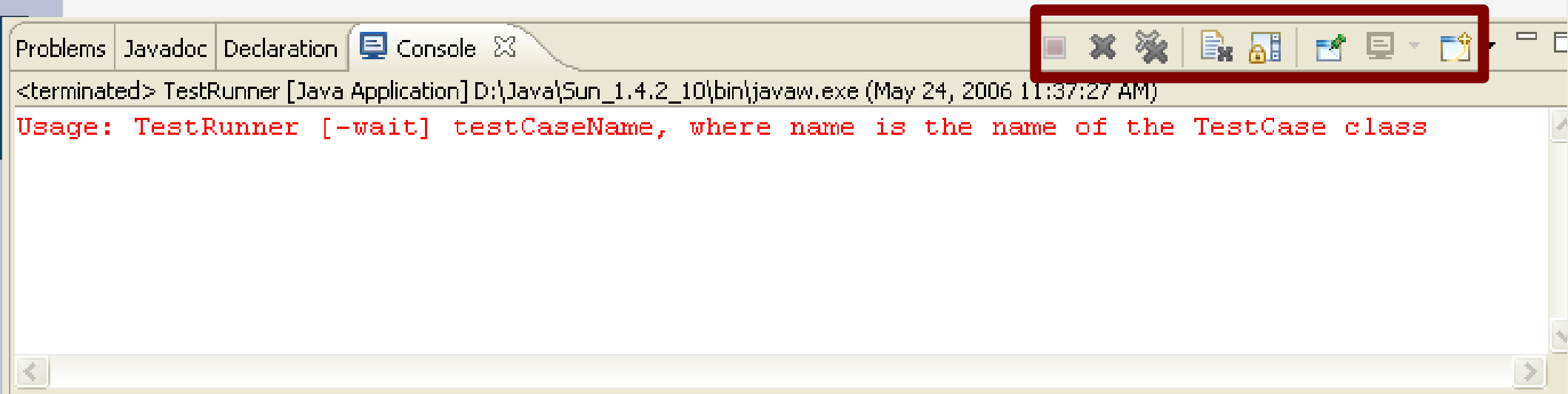
Include inherited mains when searching for a main class

Stop in main

# Spécifier des arguments



# A propos de la console



# IX Le debugger et la perspective *Debug*

# 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 (mauvaise) solution : créer, dans une classe du programme, un booléen *debug* et une méthode *debug(String s)* statiques :

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

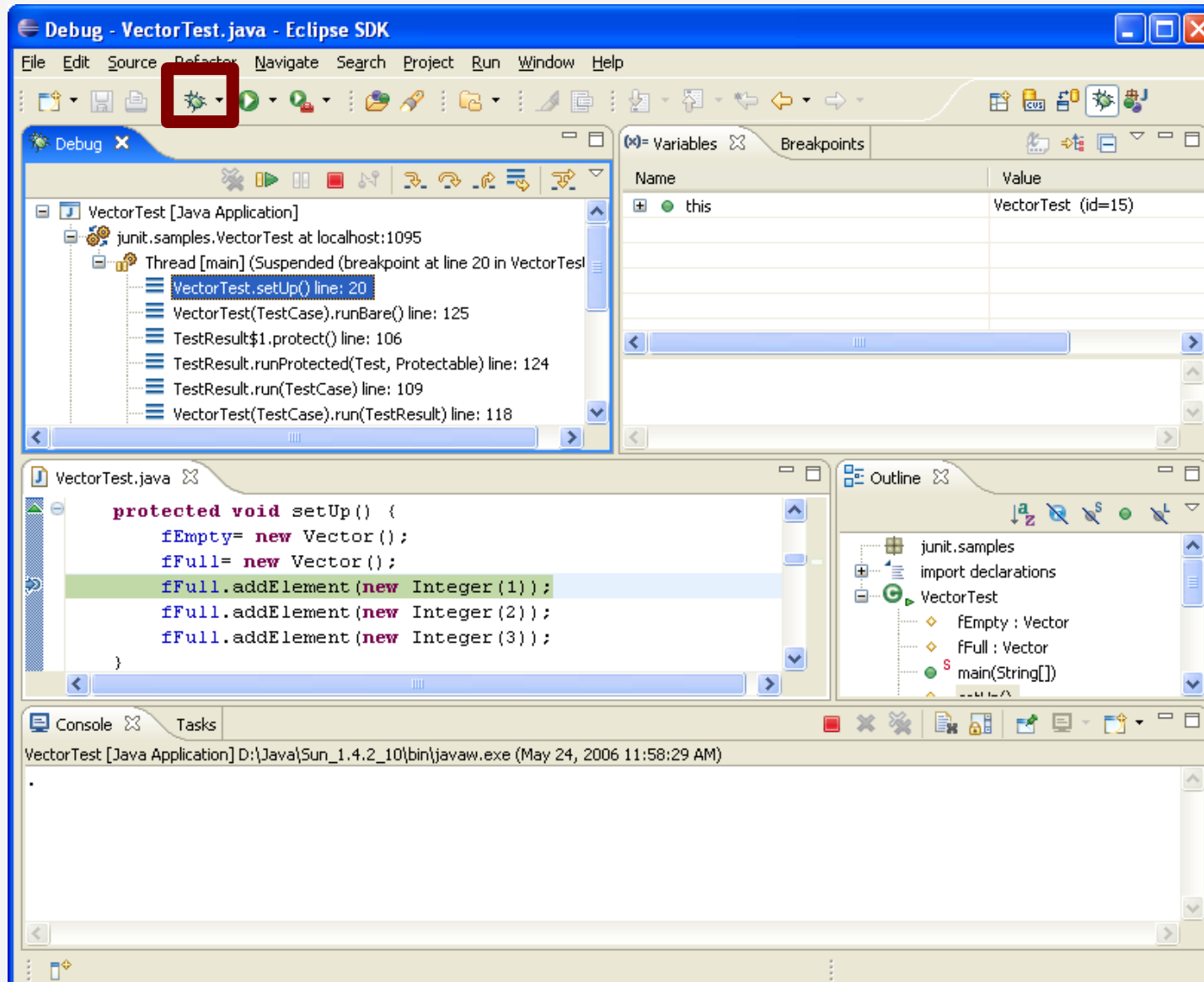
- Intérêt :
  - permet d'activer ou de désactiver le mode debug
- Problèmes :
  - On ne peut pas sélectionner les messages ! (cf. `java.util.logging` pour une vraie solution de traçage)
  - Trop d'information tue l'information.
- Solution : un debugger n'ajoute rien dans les sources

# 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.

# La perspective *Debug*

- « 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. The Outline view on the right shows the class structure:

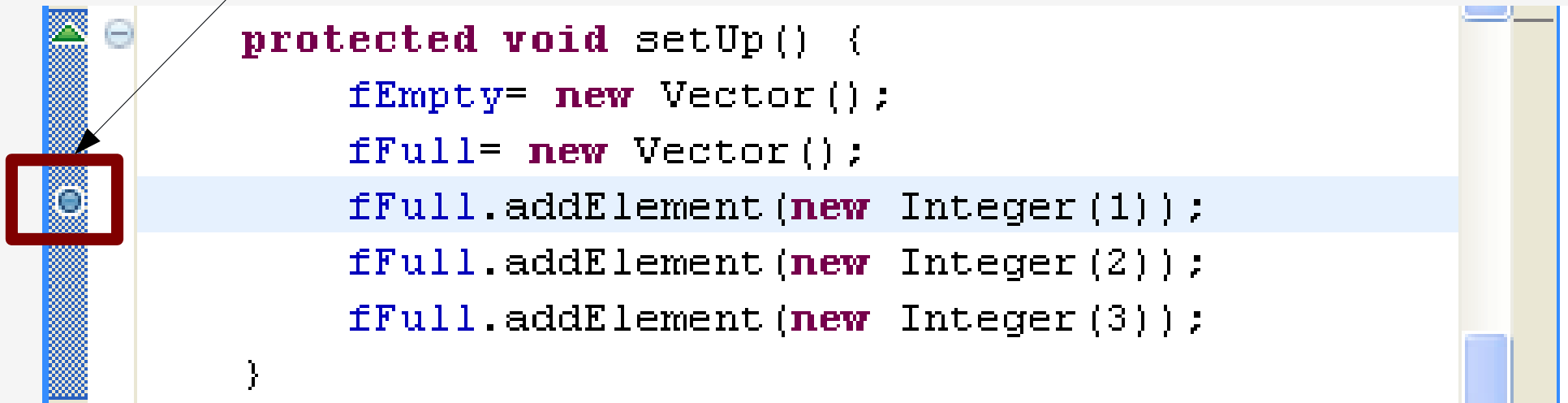
- test
  - import declarations
  - Test
    - myList : ArrayList
    - Test()
    - fillList()
    - main(String[])

The bottom toolbar includes: Problems, Javadoc, Declaration, PHP Browser, Console, and Search. The console shows the command: `<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 : il faut des points d'arrêt, des *breakpoints*

# Insérer un point d'arrêt

Bouton droit :  
« toggle breakpoint »



The screenshot shows an IDE window with a code editor. On the left side, there is a vertical toolbar with a blue background. A red square highlights a breakpoint icon (a circle with a dot) in the toolbar. An arrow points from this icon to a blue box containing the text 'Bouton droit : « toggle breakpoint »'. In the code editor, the following code is displayed:

```
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 line `fFull.addElement(new Integer(1));` is highlighted in light blue. A vertical scrollbar is visible on the right side of the code editor.

# 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)

```

Test.java
    myList = new ArrayList<Integer>();
}

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

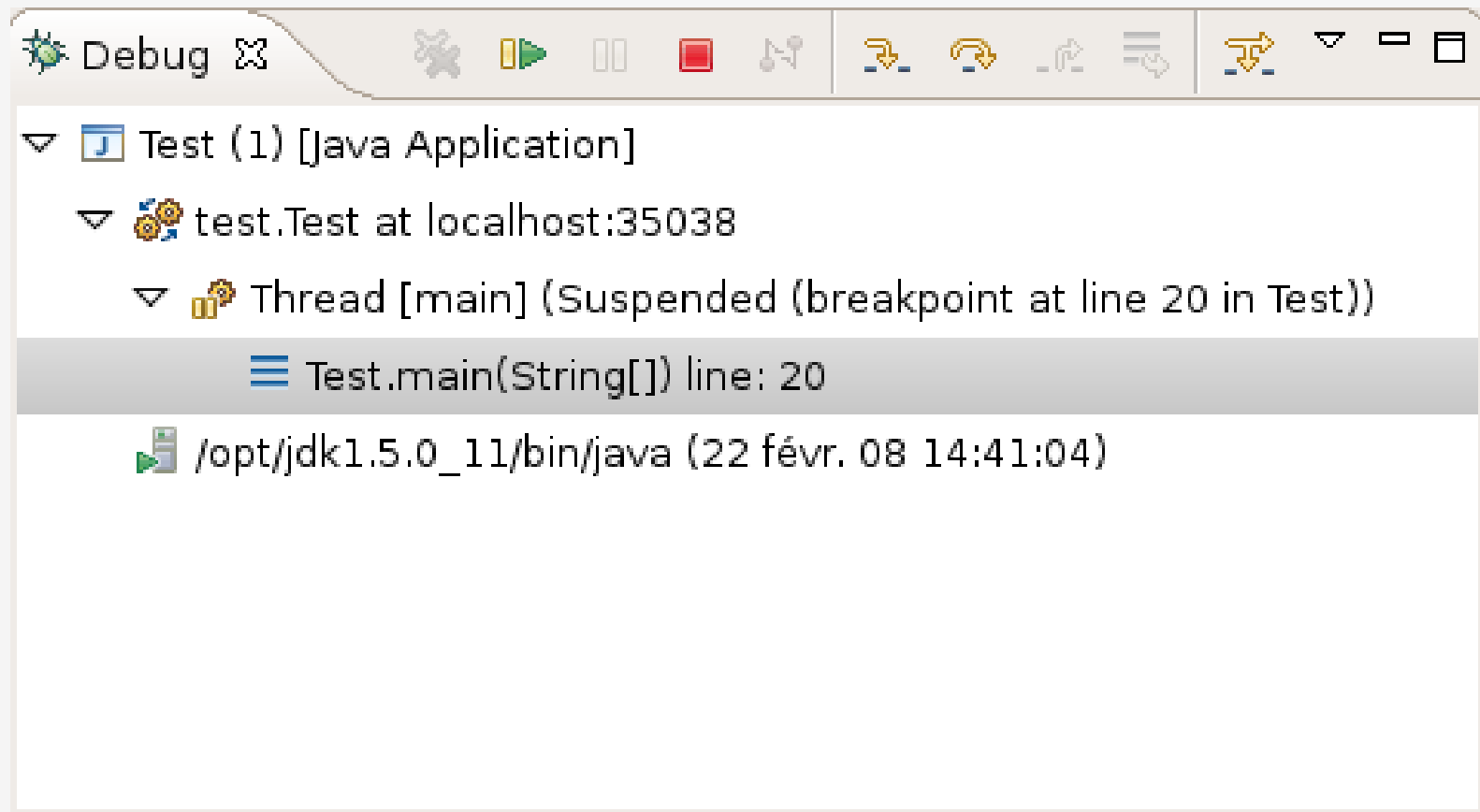
public static void main(String[] args) {
    Test name = new Test();
    name.fillList();
}
    
```

Outline

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

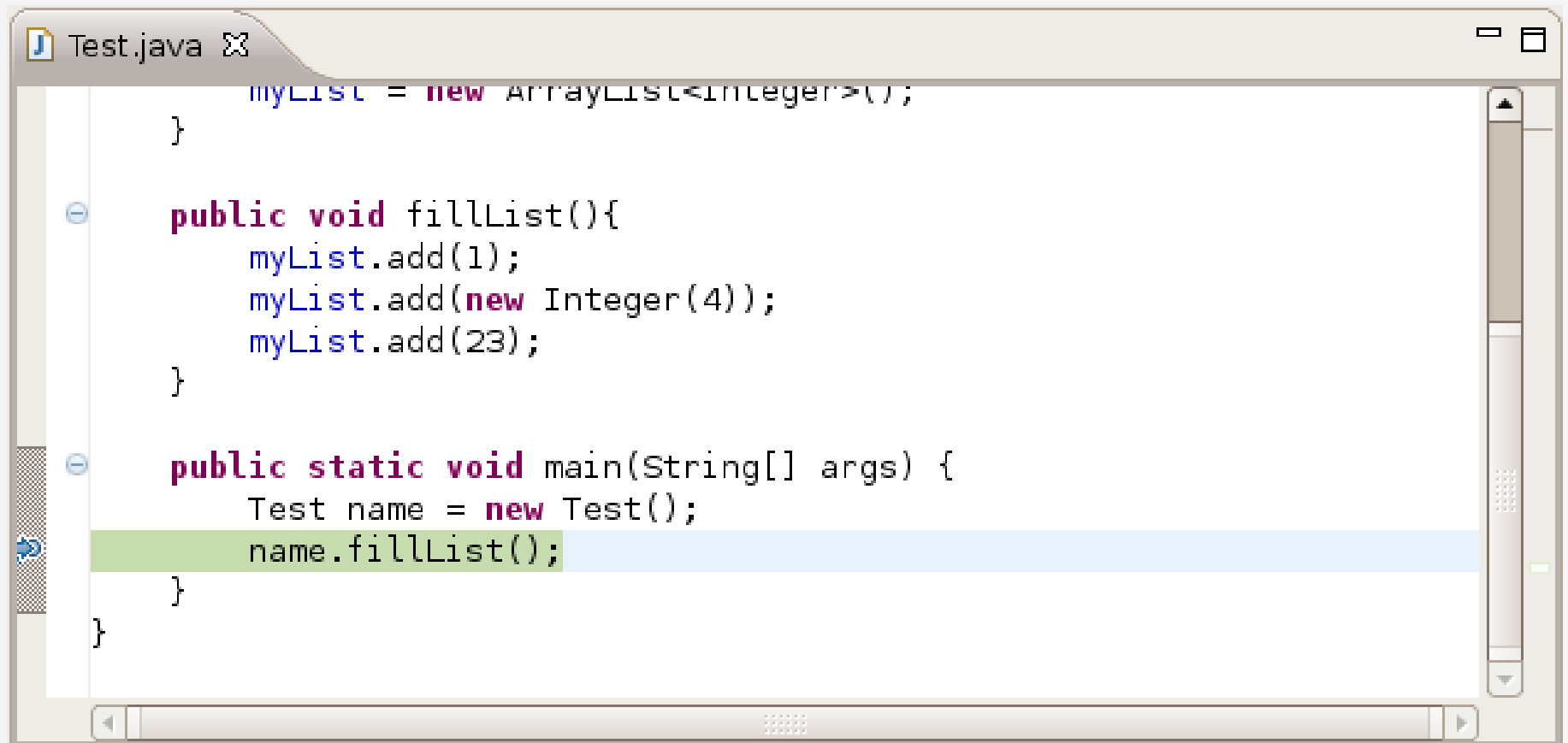
Test (1) [Java Application] /opt/jdk1.5.0\_11/bin/java (22 févr. 08 14:41:04)

# État de l'exécution



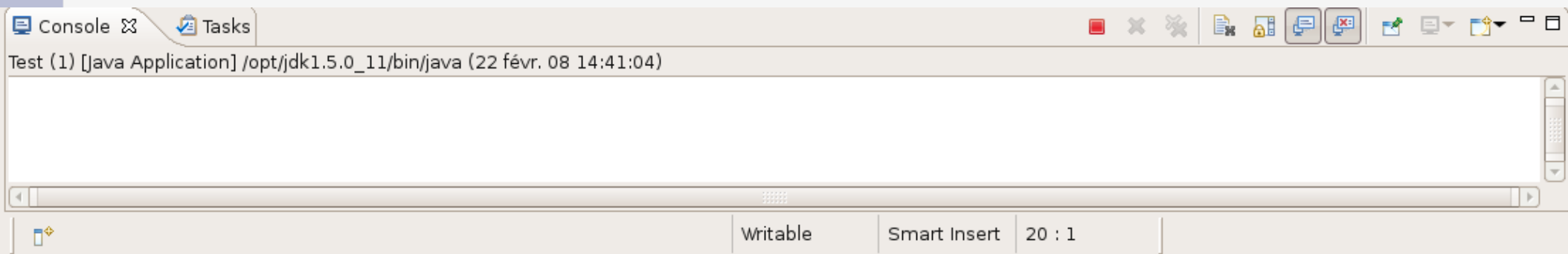


# Ligne de code correspondante

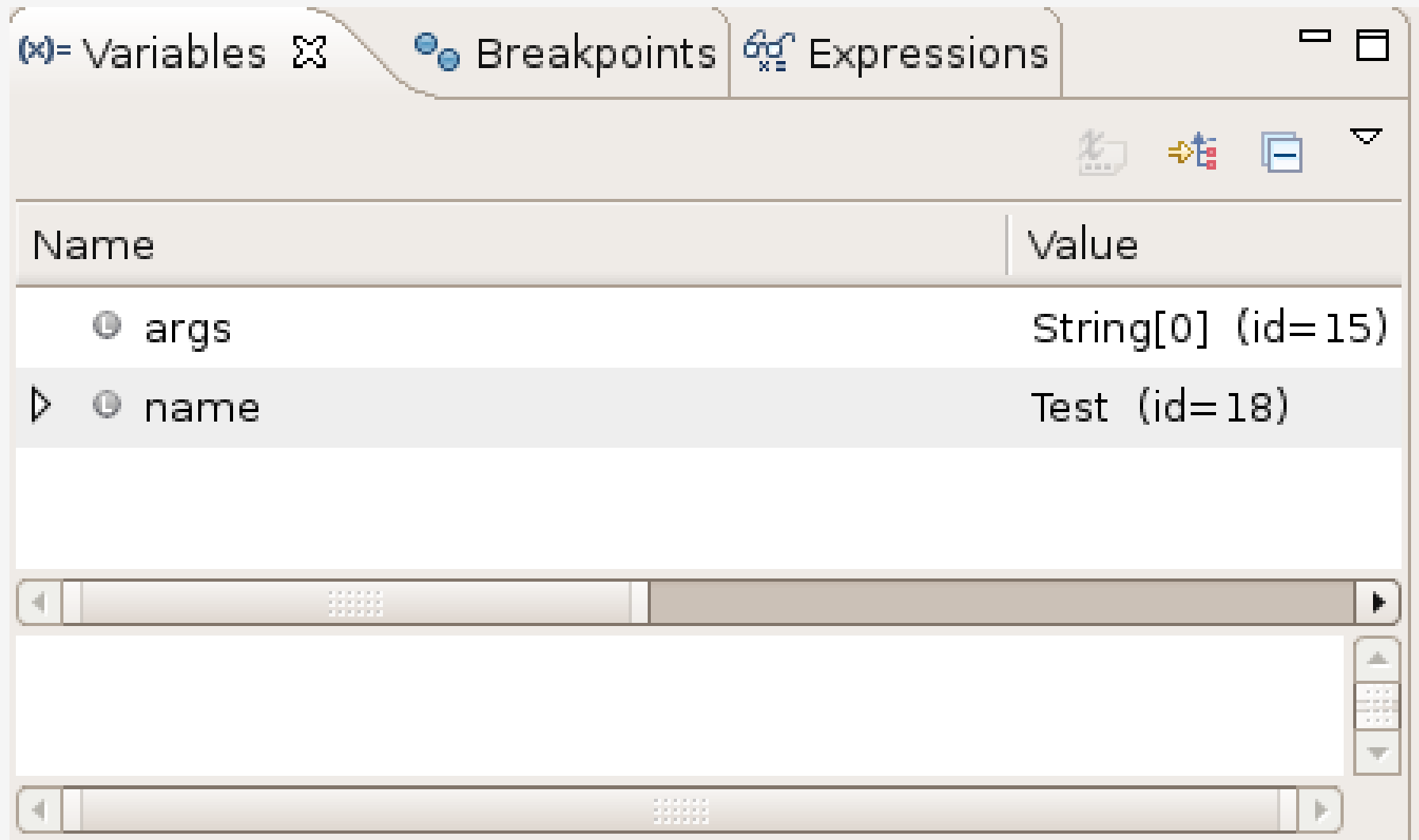


```
Test.java ✕  
    myList = new ArrayList<Integer>();  
}  
  
public void fillList(){  
    myList.add(1);  
    myList.add(new Integer(4));  
    myList.add(23);  
}  
  
public static void main(String[] args) {  
    Test name = new Test();  
    name.fillList();  
}  
}
```

# 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 listed with a value of 'String[0] (id=15)'. The 'name' variable is listed with a value of 'Test (id=18)'. The 'name' row is highlighted. Below the table are two horizontal scroll bars and a vertical scroll bar on the right side.

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

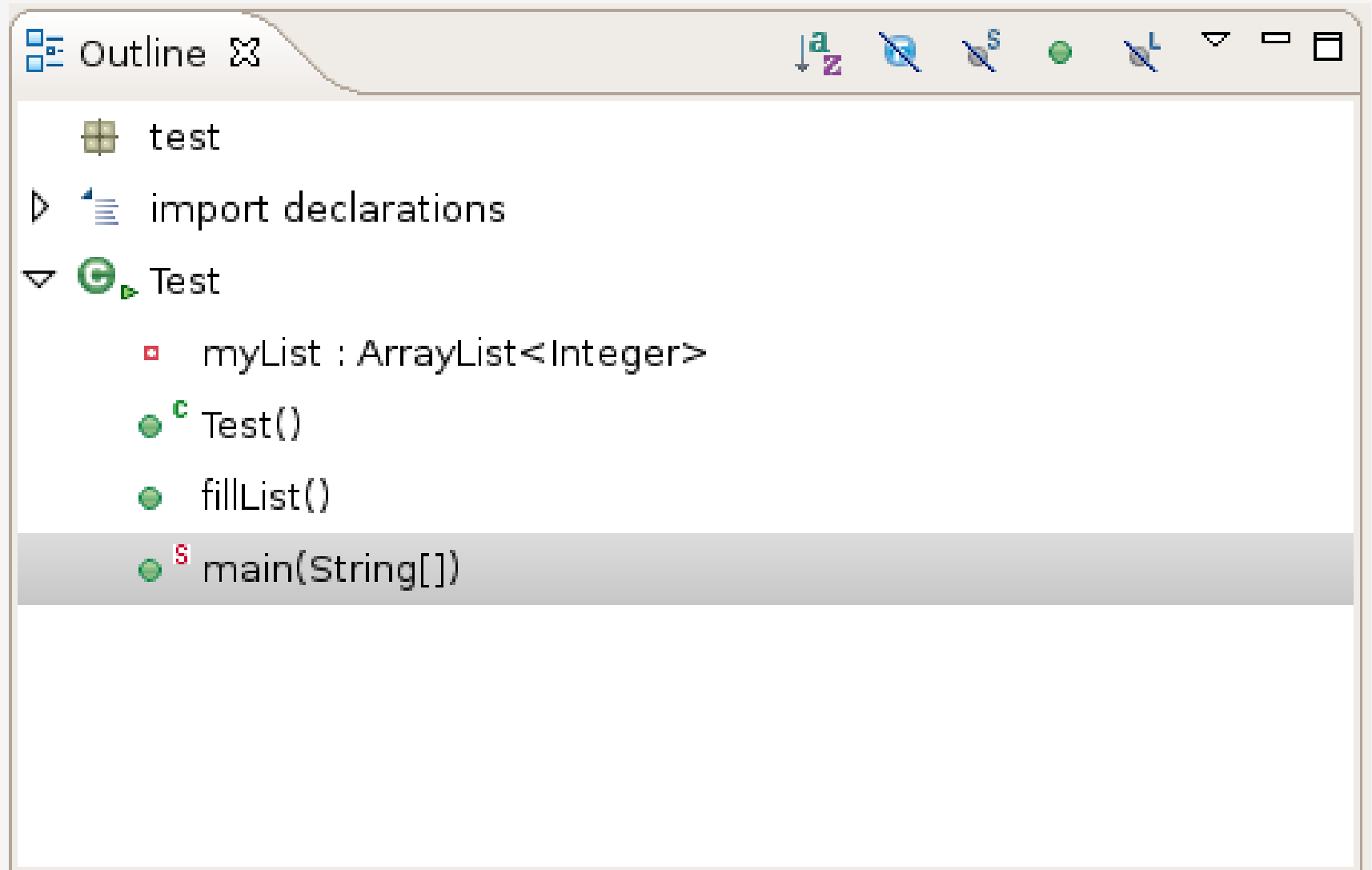
# L'état des variables

The screenshot shows the 'Variables' window in an IDE. The window has tabs for 'Variables', 'Breakpoints', and 'Expressions'. The 'Variables' tab is active and displays a table of variables:

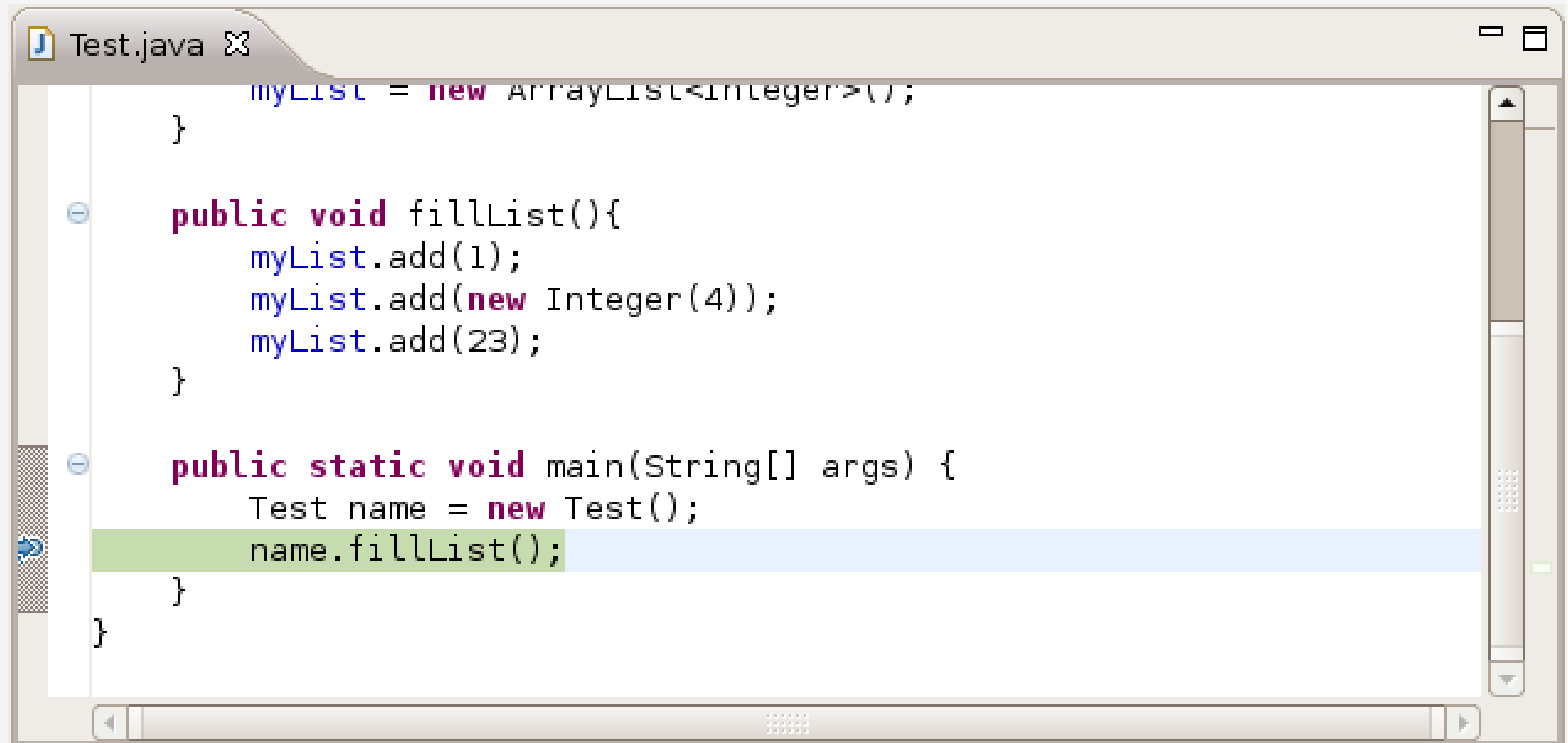
Name	Value
args	String[0] (id=15)
name	Test (id=18)
myList	ArrayList<E> (id=19)
elementData	Object[10] (id=29)
modCount	0
size	0

Below the table is a scrollable area containing an empty array representation: []

# La méthode courante (outline)

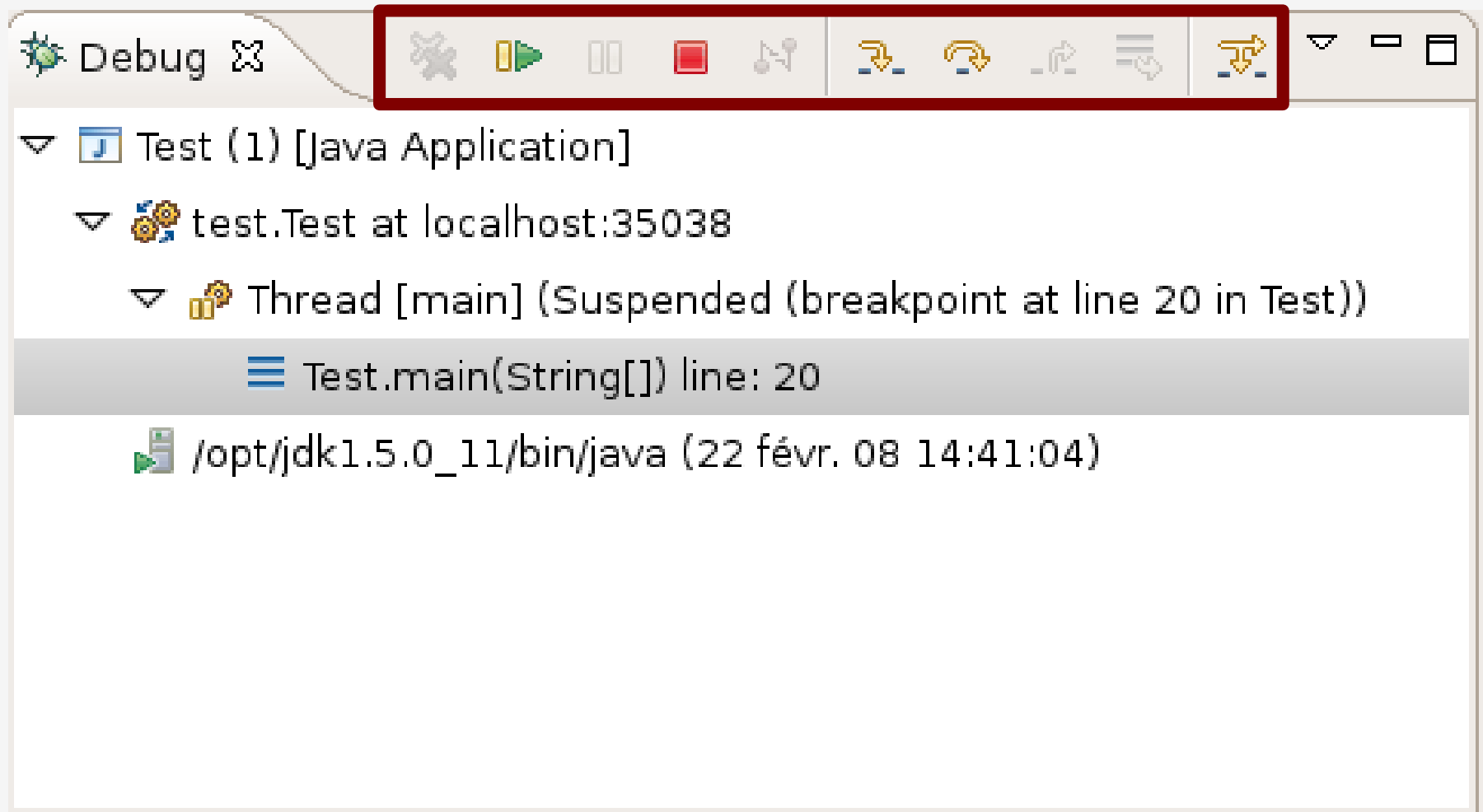


# Sur-lignage au point d'arrêt












```
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();  
}  
}
```

# Exécution contrôlée



# Exécution contrôlée

Java Execution Control Commands

Command	Name	Description	Availability
	<a href="#"><u>Resume</u></a>	Resumes a suspended thread.	Context menu, Run menu and view action
	<a href="#"><u>Step Into</u></a>	Steps into the highlighted statement.	Context menu, Run menu and view action
	<a href="#"><u>Step Over</u></a>	<p>Steps over the highlighted statement. Execution will continue at the next line either 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.</p> <p>The cursor jumps to the declaration of the method and selects this line.</p>	Context menu, Run menu and view action
	<a href="#"><u>Step Return</u></a>	Steps out of the current method. This option stops execution after exiting the current method.	Context menu, Run menu and view action
	<a href="#"><u>Suspend</u></a>	Suspends the selected thread of a target so that you can browse or modify code, inspect data, step, and so on.	Context menu, Run menu and view action
	<a href="#"><u>Terminate</u></a>	Terminates the selected debug target.	Context menu, Run menu and view action
	<a href="#"><u>Terminate &amp; Relaunch</u></a>	Terminates the selected debug target and relaunches it.	Context menu
	<a href="#"><u>Terminate &amp; Remove</u></a>	Terminates the selected debug target and removes it from the view.	Context menu
	<a href="#"><u>Terminate/Disconnect All</u></a>	Terminates all active launches in the view.	Context menu



# 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 Breakpoints Expressions

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)

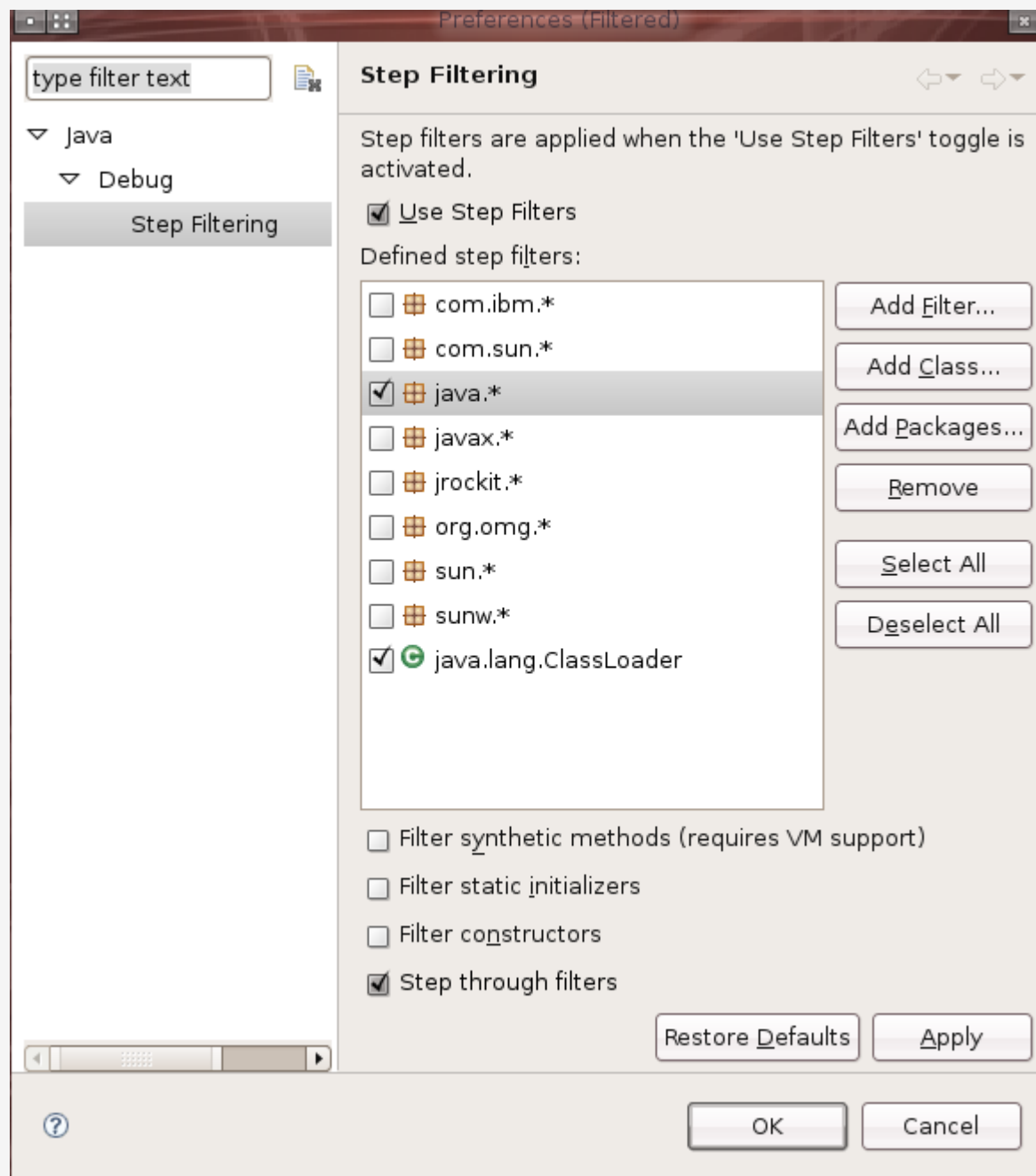
Name	Value
group	ThreadGroup (id=10)
inheritableThreadLocals	null
inheritedAccessControlCon	AccessControlContext (id=)
name	char[4] (id=41)
priority	5
single_step	false

```
/**
 * 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 debug session. The top toolbar includes standard debugging icons like 'Step into' (F5). The 'Debug' console on the left shows the execution path: 'Test (1) [Java Application]' -> 'test.Test at localhost:50108' -> 'Thread [main] (Suspended)' -> 'Test.fillList() line: 14' (highlighted) -> 'Test.main(String[]) line: 20'. The 'Variables' window on the right shows the current state of variables: 'this' is 'Test (id=18)' and 'myList' is 'ArrayList<E> (id=19)'. Below the variables, the array content '[1]' is visible. The 'Test.java' editor at the bottom left shows the source code with the line `myList.add(new Integer(4));` highlighted. The 'Outline' window on the bottom right shows the class structure with 'fillList()' selected.

```
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();
}
```

# 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: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:

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

Below the variable list, the value of myList is shown as [1, 4].

The bottom-left pane shows the source code of 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();  
}  
}
```





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<Integer>();
fFull.addElement(1);
fFull.addElement(1);
fFull.addElement(1);
}
public static Test suite() {
    return new TestRunner();
}
public void testCapacity() {
    int size= fFull.size();
    for (int i= 0; i < size; i++)
        fFull.addElement(i);
    assertTrue(fFull.size() == size);
}
public void testClone() {
    Vector clone= (Vector)fFull.clone();
    assertTrue(clone.containsAll(fFull));
    assertTrue(clone.containsAll(fFull));
}
public void testContains() {
    assertTrue(fFull.contains(1));
    assertTrue(!fFull.contains(2));
}
public void testElementAt() {
    Integer i= (Integer)fFull.elementAt(0);
    assertTrue(i.intValue() == 1);
}

```

	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

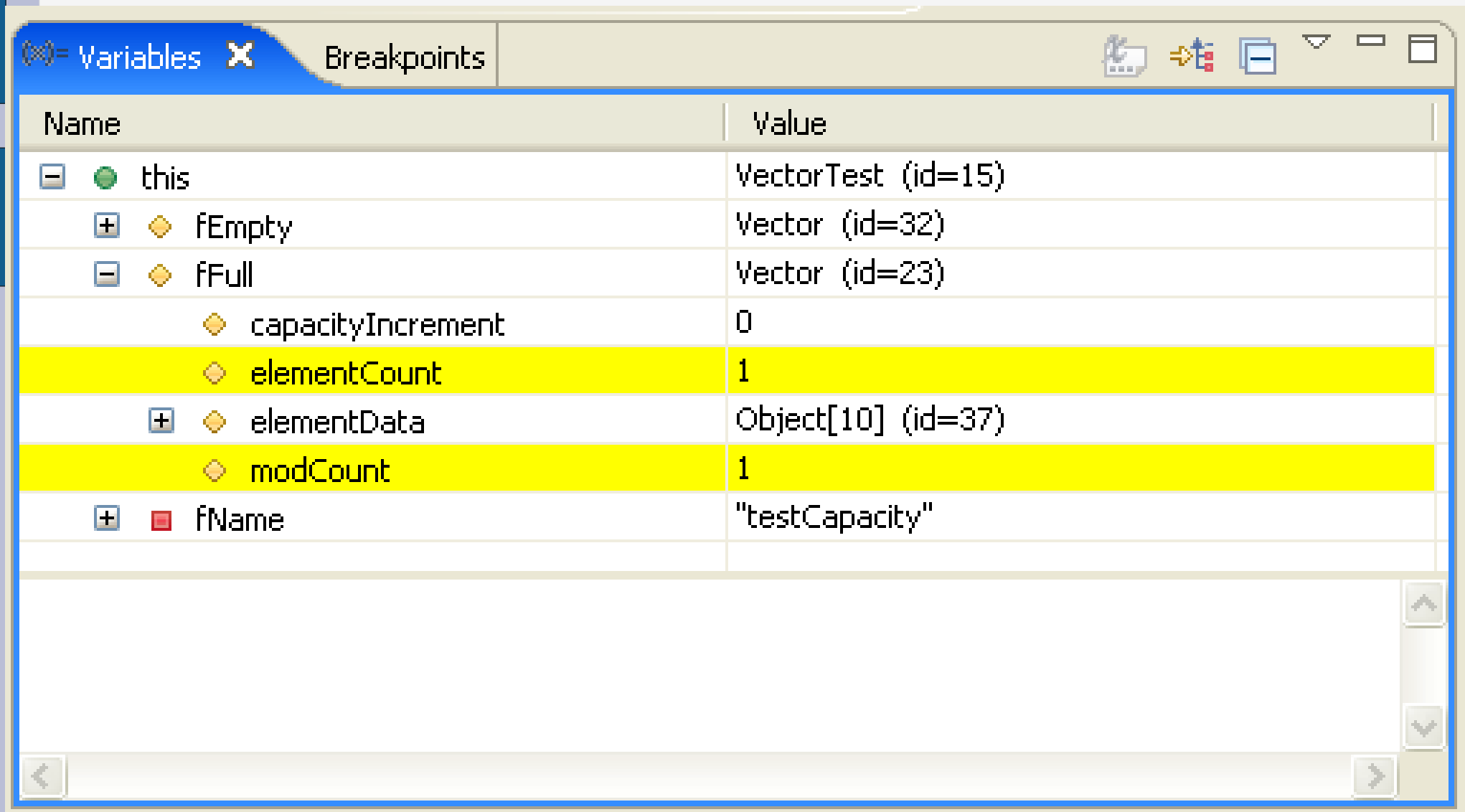
The screenshot shows an IDE with the following components:

- 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 showing the state of the selected variable:
  - 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



# Modifier une variable

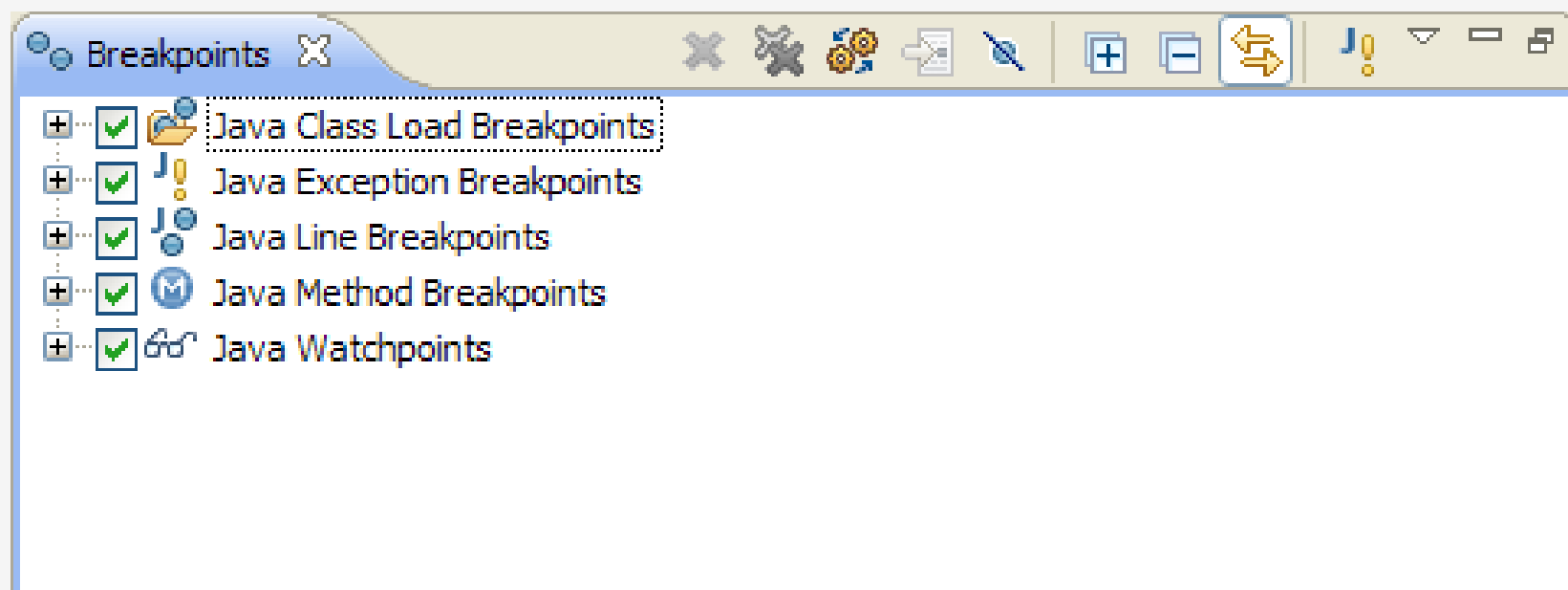
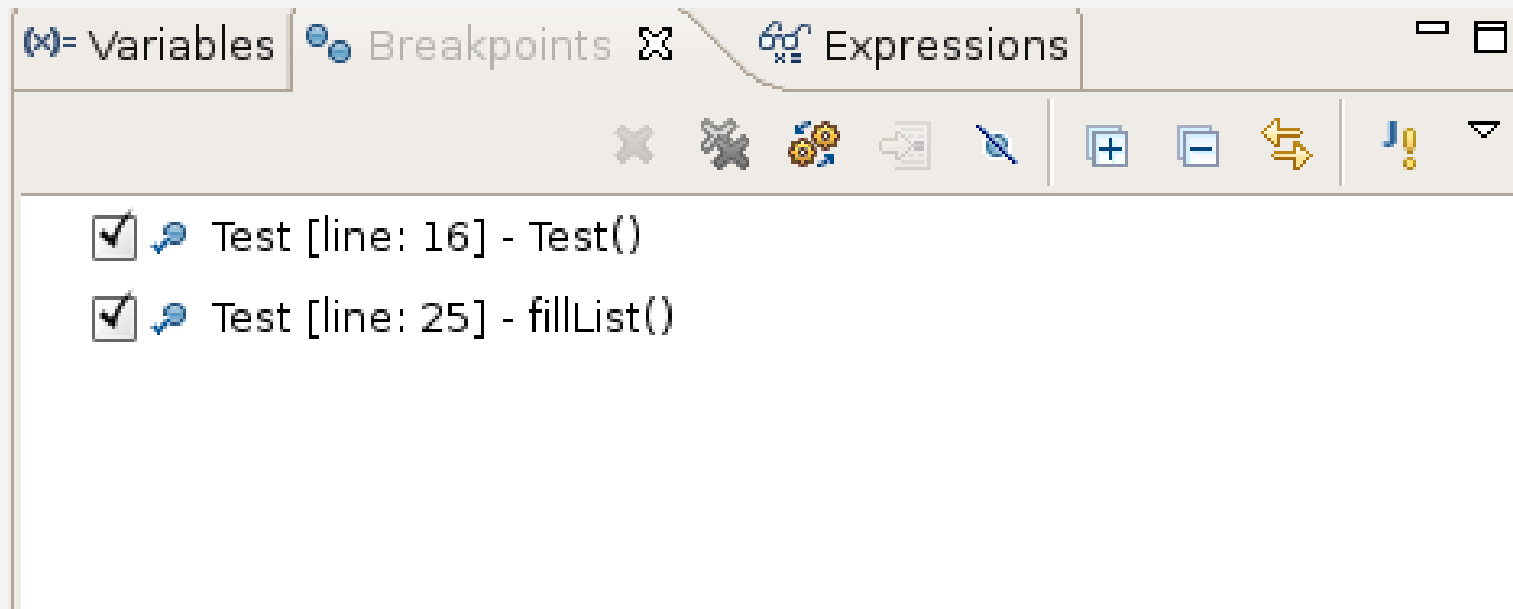


The screenshot shows the 'Variables' window in an IDE. The window title is 'Variables' and it has a 'Breakpoints' tab. The main area is a table with two columns: 'Name' and 'Value'. The table lists the following variables and their values:

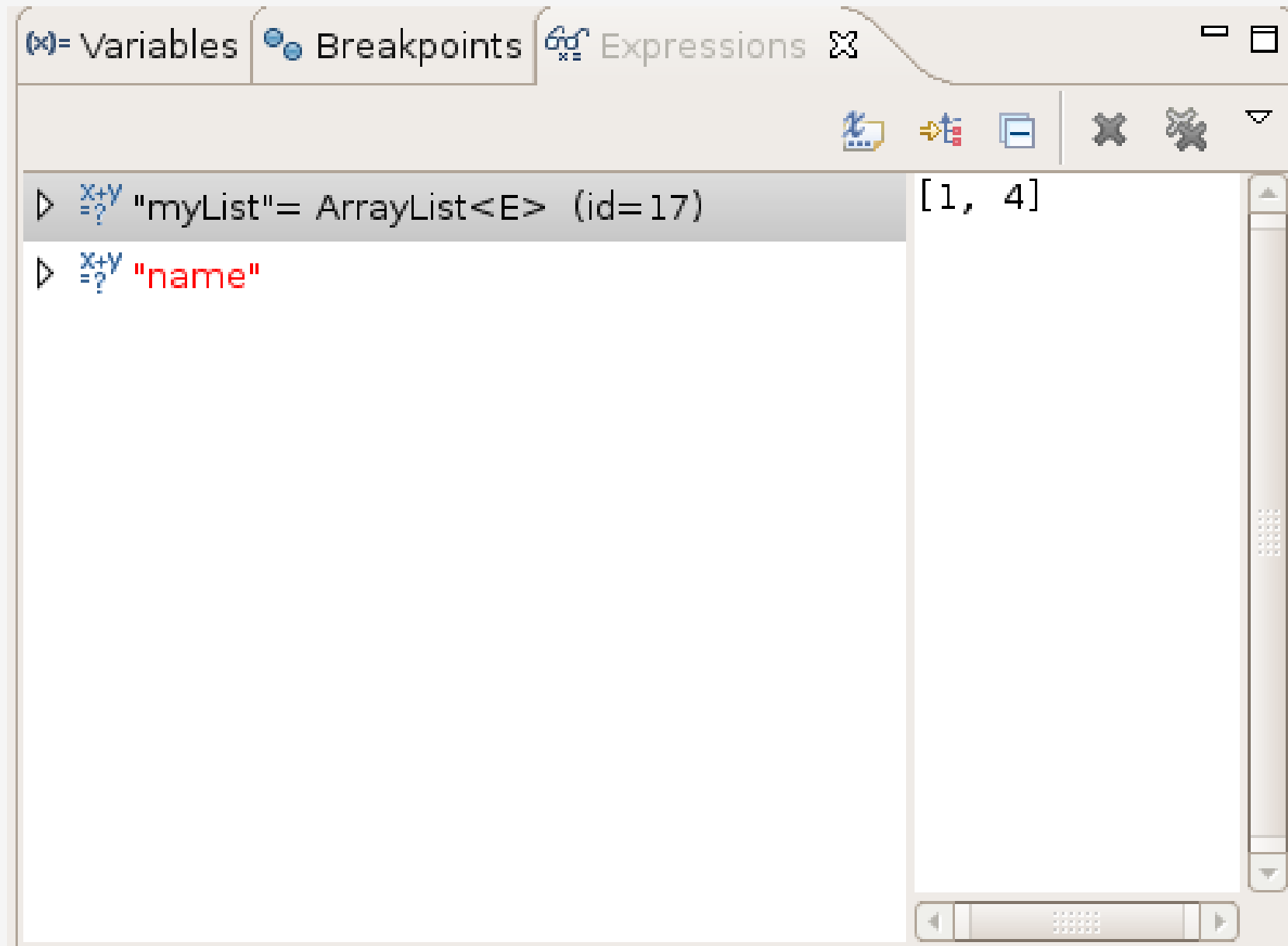
Name	Value
this	VectorTest (id=15)
fEmpty	Vector (id=32)
fFull	Vector (id=23)
capacityIncrement	0
elementCount	1
elementData	Object[10] (id=37)
modCount	1
fName	"testCapacity"

The 'elementCount' and 'modCount' rows are highlighted in yellow. The 'fName' row has a red square icon next to it. The 'this' row has a green dot icon next to it. The 'fEmpty' and 'fFull' rows have orange diamond icons next to them. The 'capacityIncrement' row has an orange diamond icon next to it. The 'elementData' row has a plus icon next to it. The 'elementCount' and 'modCount' rows have orange diamond icons next to them. The 'fName' row has a plus icon next to it.

# Liste des points d'arrêt

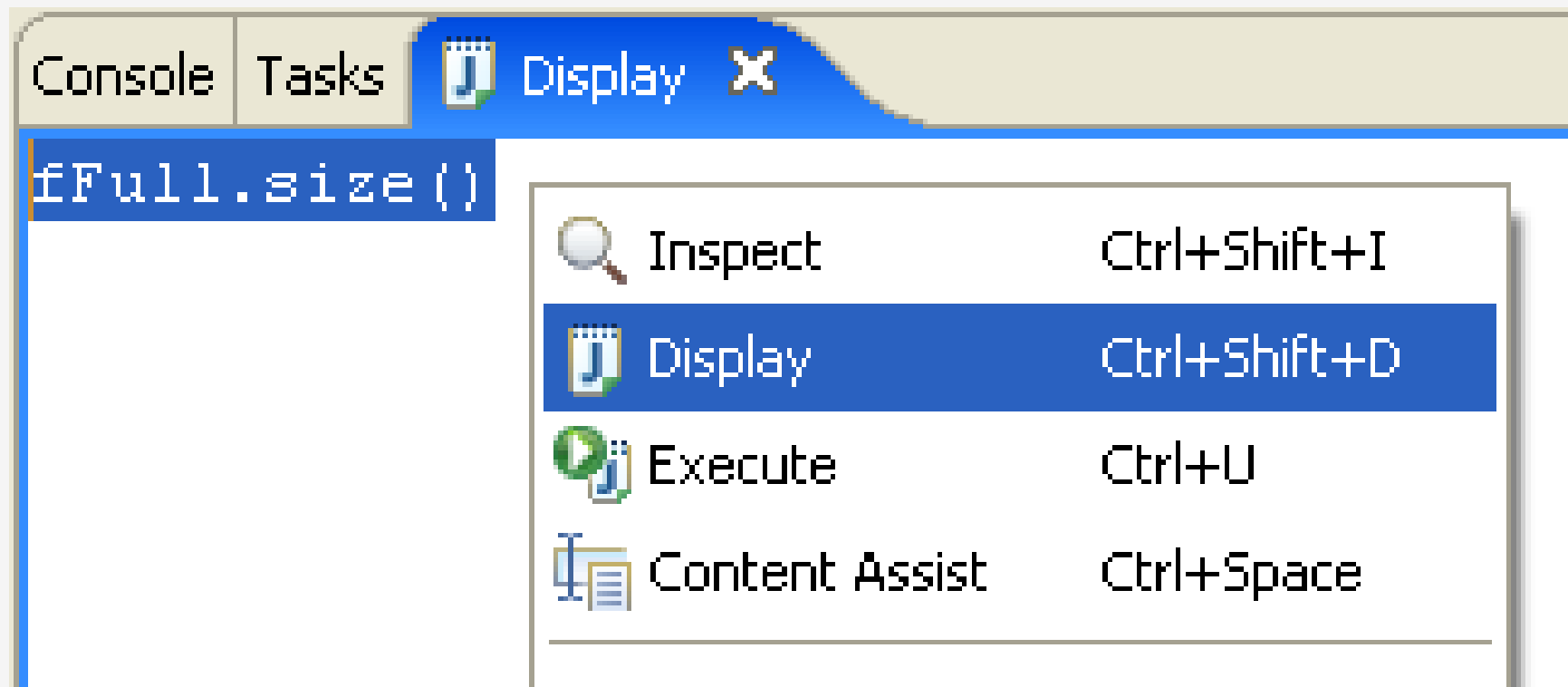


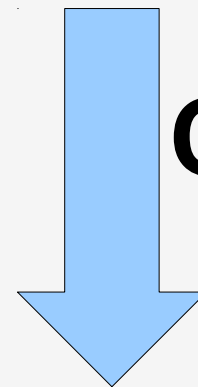
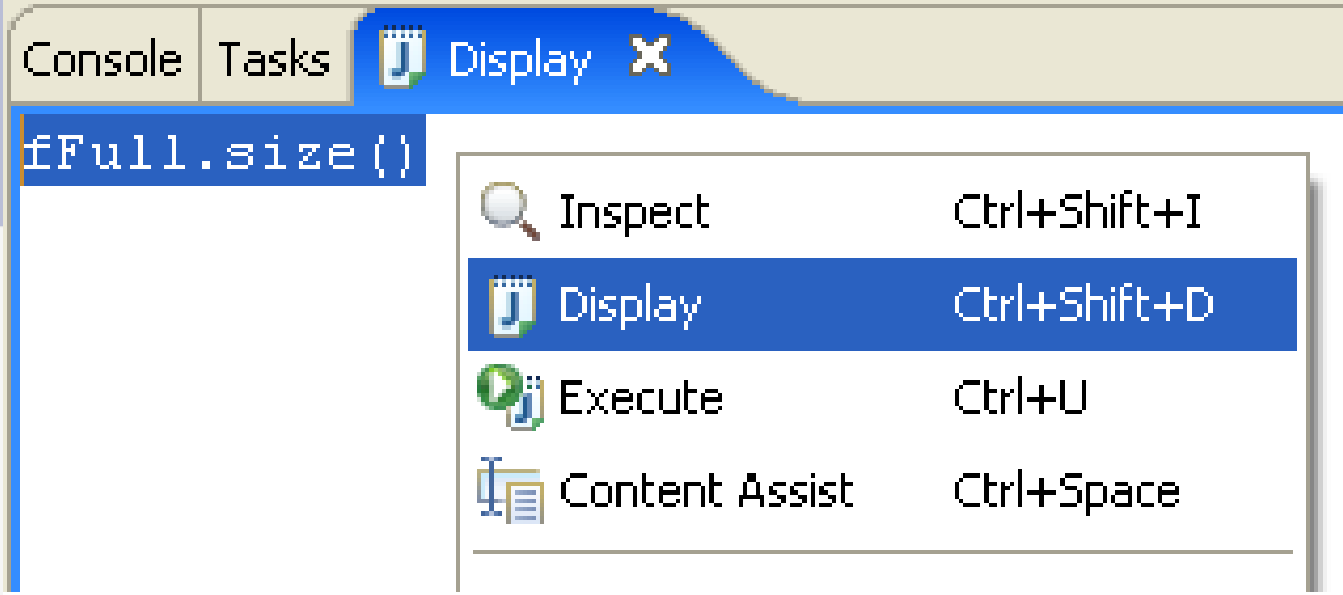
# Surveiller une expression



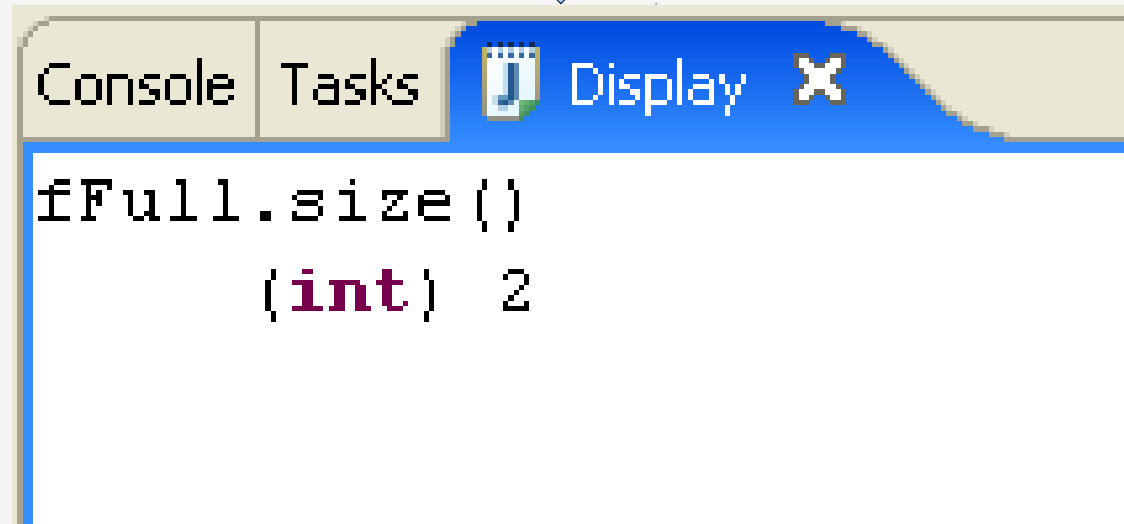
# Evaluer des expressions

- 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 its evaluation result:

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

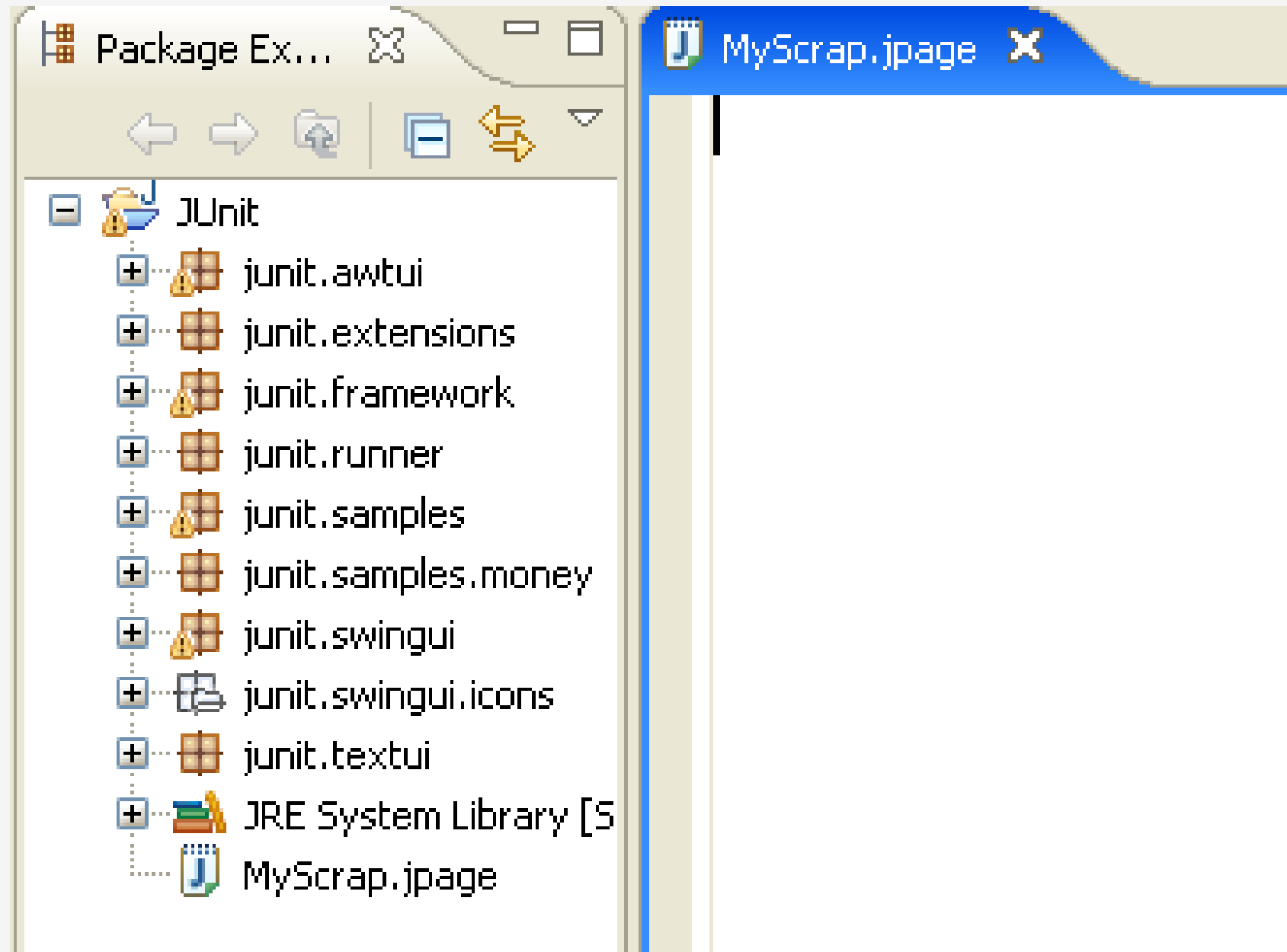
At the bottom of the console window, a message reads: "Press Ctrl+Shift+I to Move to Expressions View".

# Tester des expressions

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



# Tester des expressions





# Tester des expressions

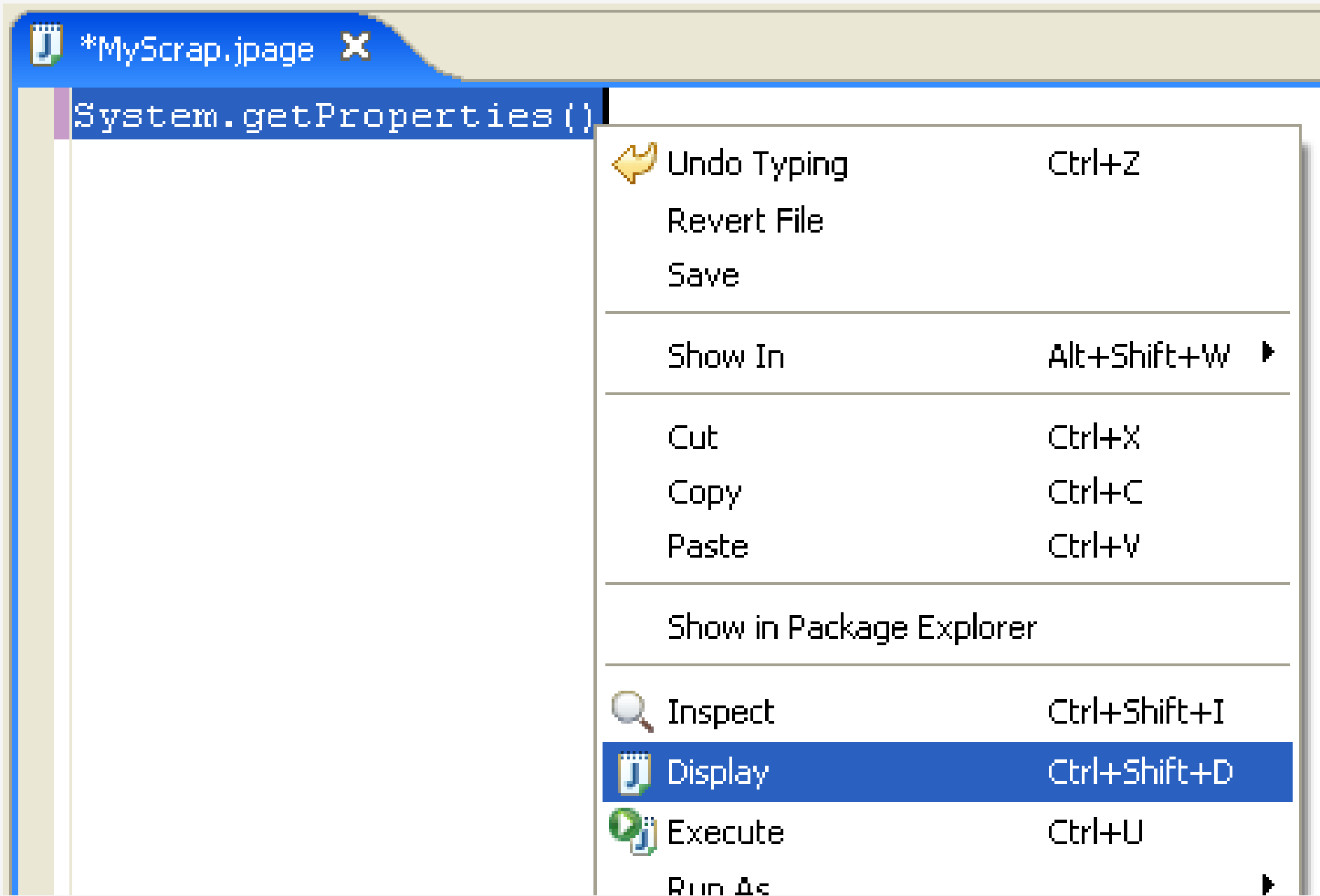
The screenshot shows an IDE window with two tabs: 'VectorTest.java' and '\*MyScrap.jpage'. The main editor area contains the text 'System.g'. A popup menu is displayed, listing several methods from the 'System' class. Each method is preceded by a green circle with a red 'S' next to it. The popup menu has a title bar that reads 'JUnit/MyScrap.jpage'. At the bottom of the popup, there is a scrollbar 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.jpage`. 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
<b>Display</b>	<b>Ctrl+Shift+D</b>
Execute	Ctrl+U
Run As	▶

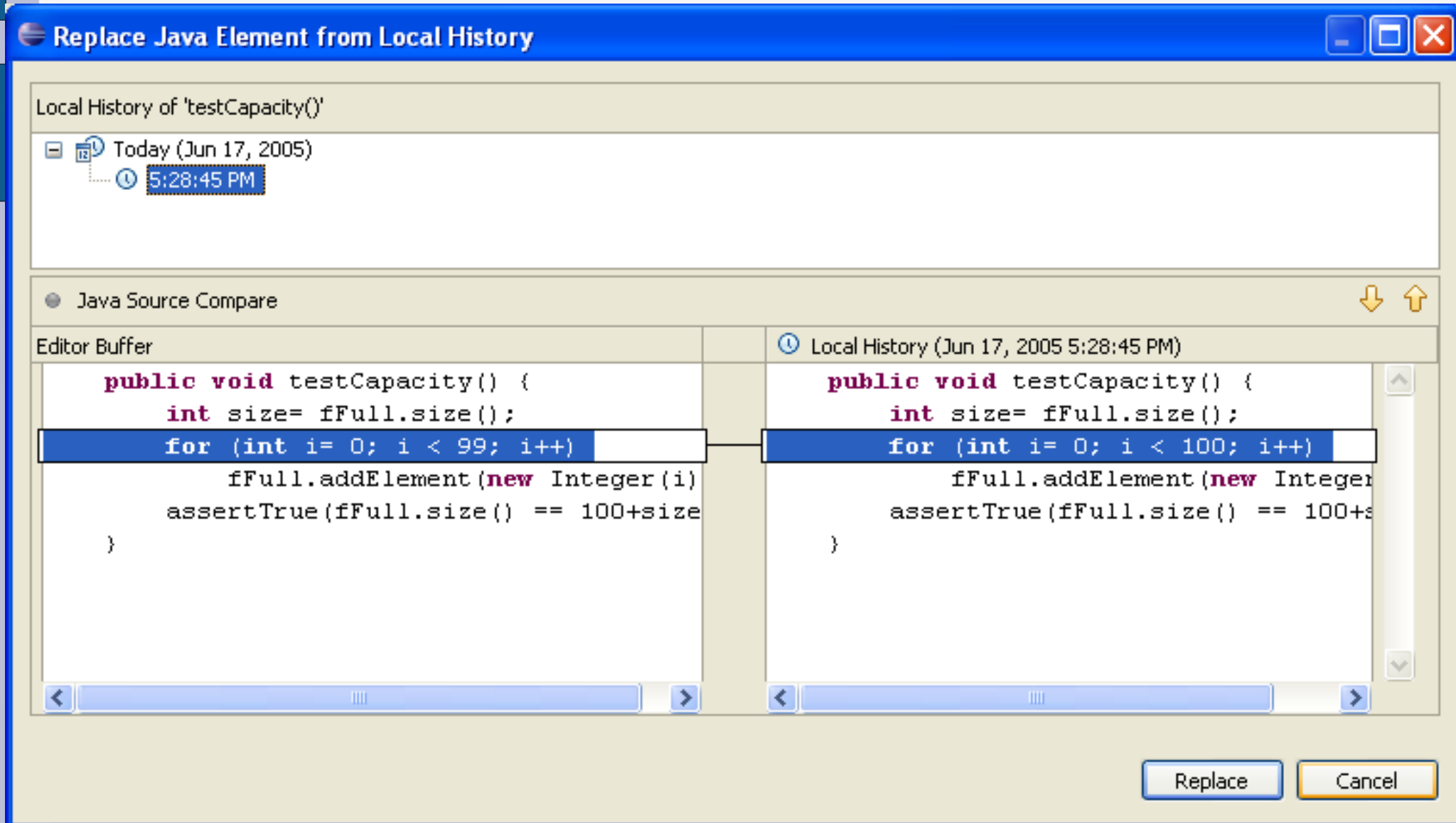
# X Utiliser l'historique

# 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)

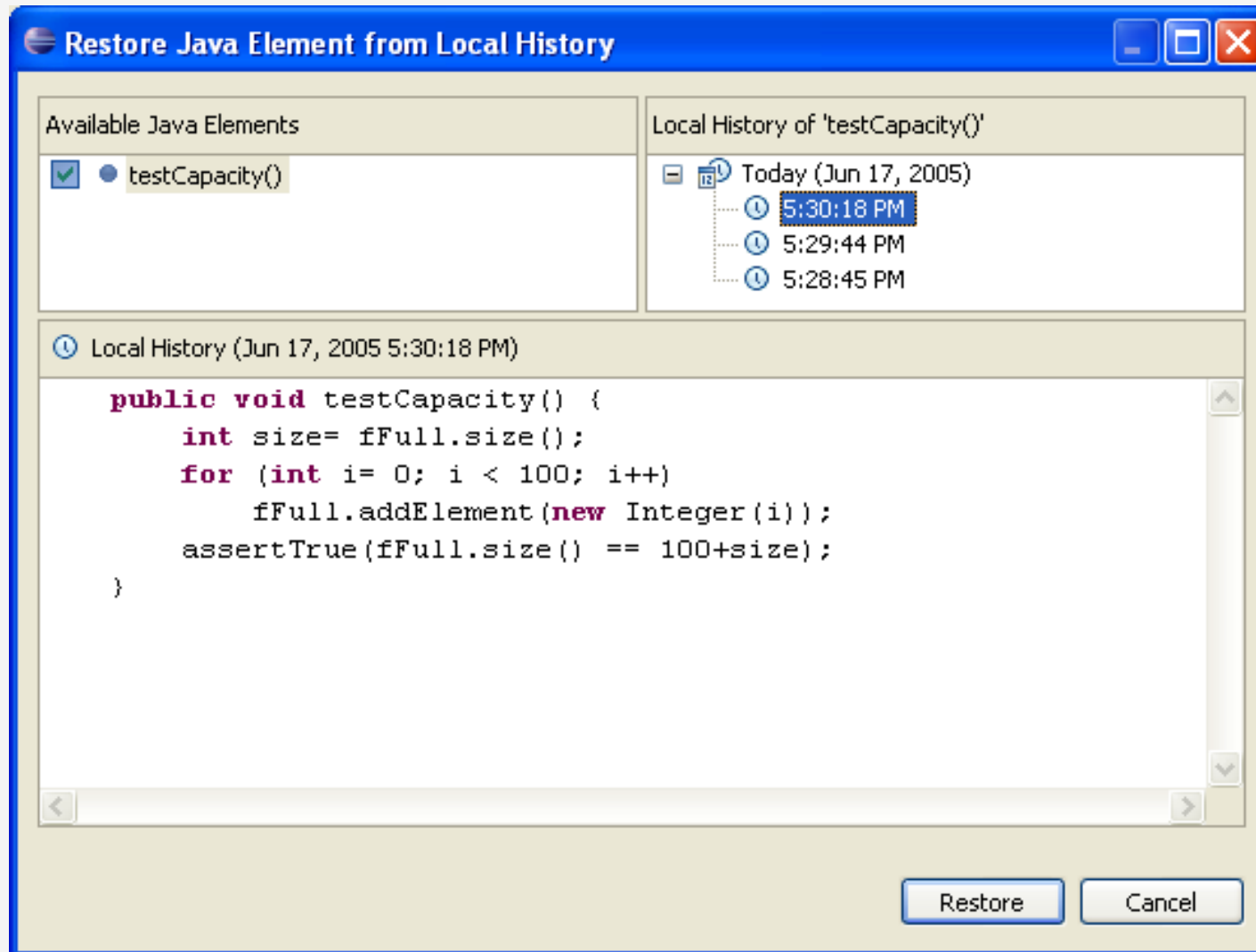
# « Undo » sur un élément

- Replace With :



# « Undo » pour récupérer des éléments effacés

- Restore from local history :

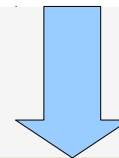
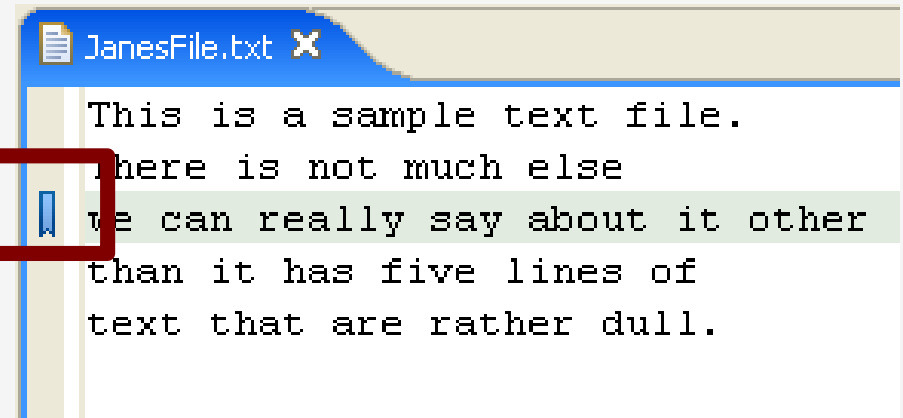
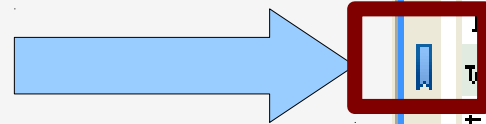
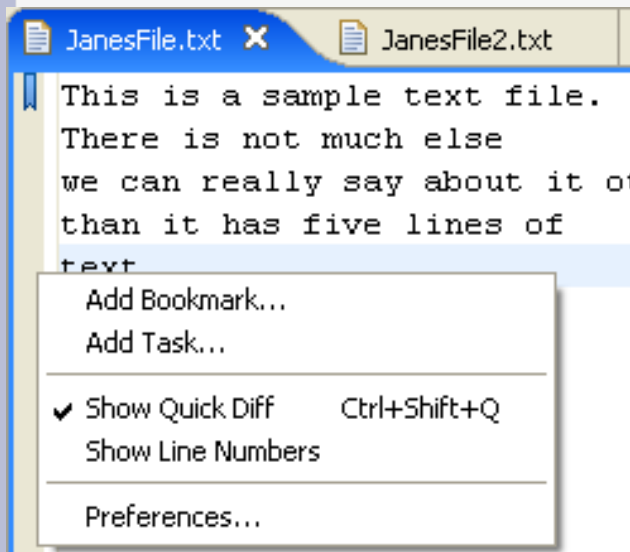


# *XI Bookmarks*





# Bookmark sur une ligne

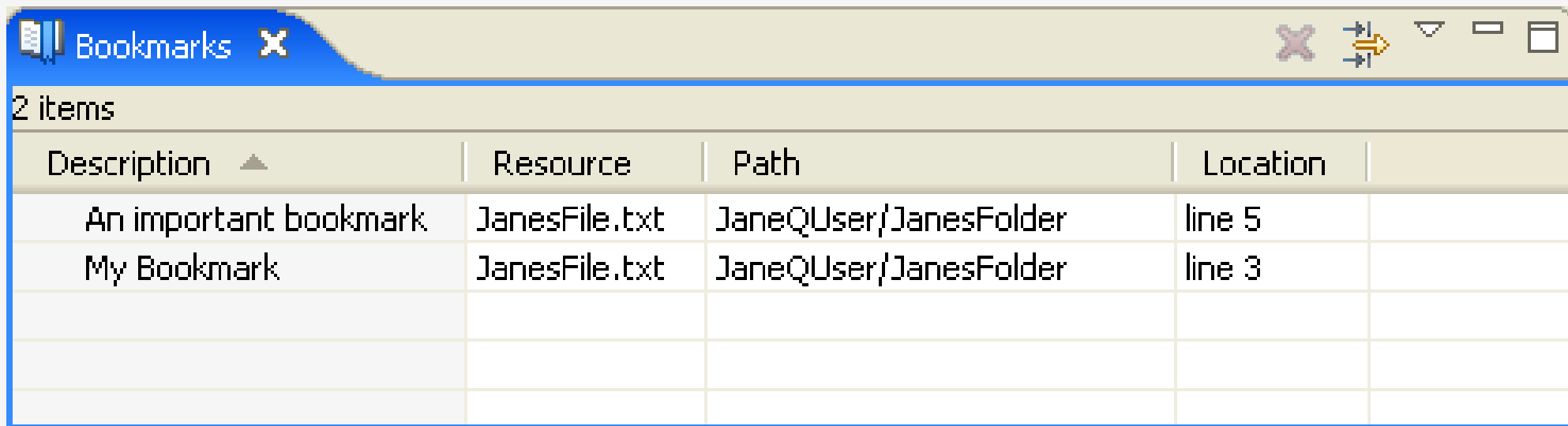


A screenshot of the 'Bookmarks' window. It shows a table with one item. The table has four columns: Description, Resource, Path, and Location.

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



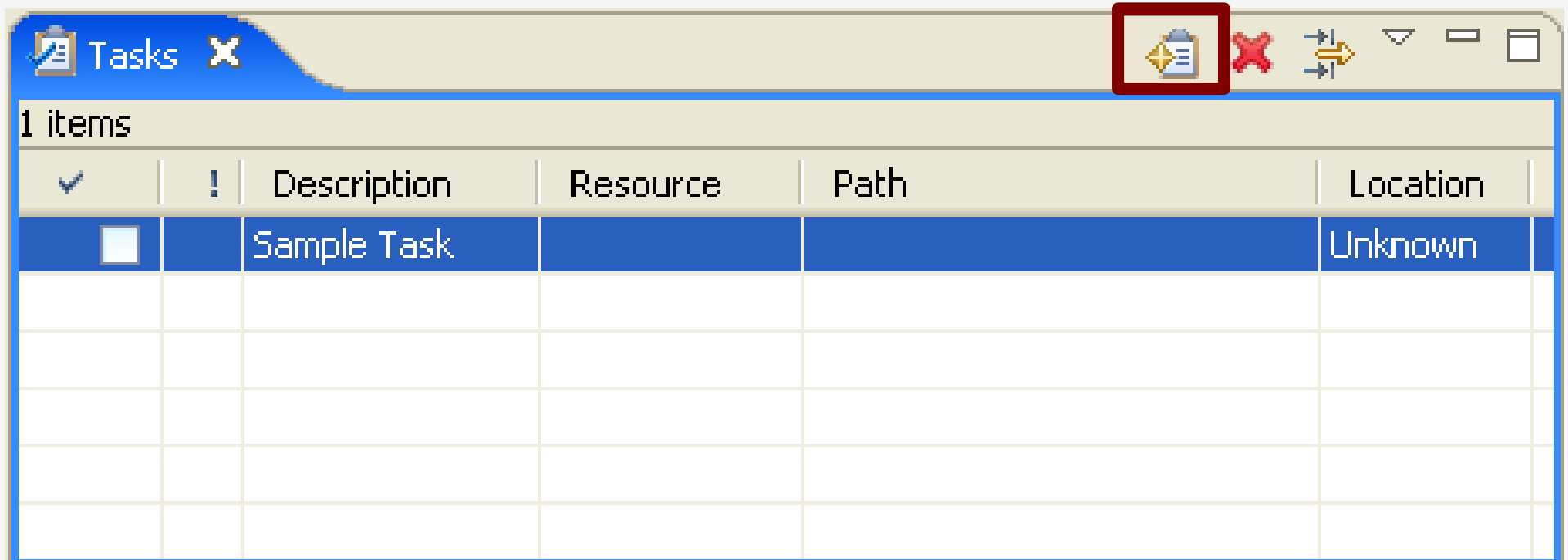
The screenshot shows a window titled "Bookmarks" with a close button (X) and a refresh button (circular arrow). Below the title bar, it says "2 items". The main content is a table with the following data:

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

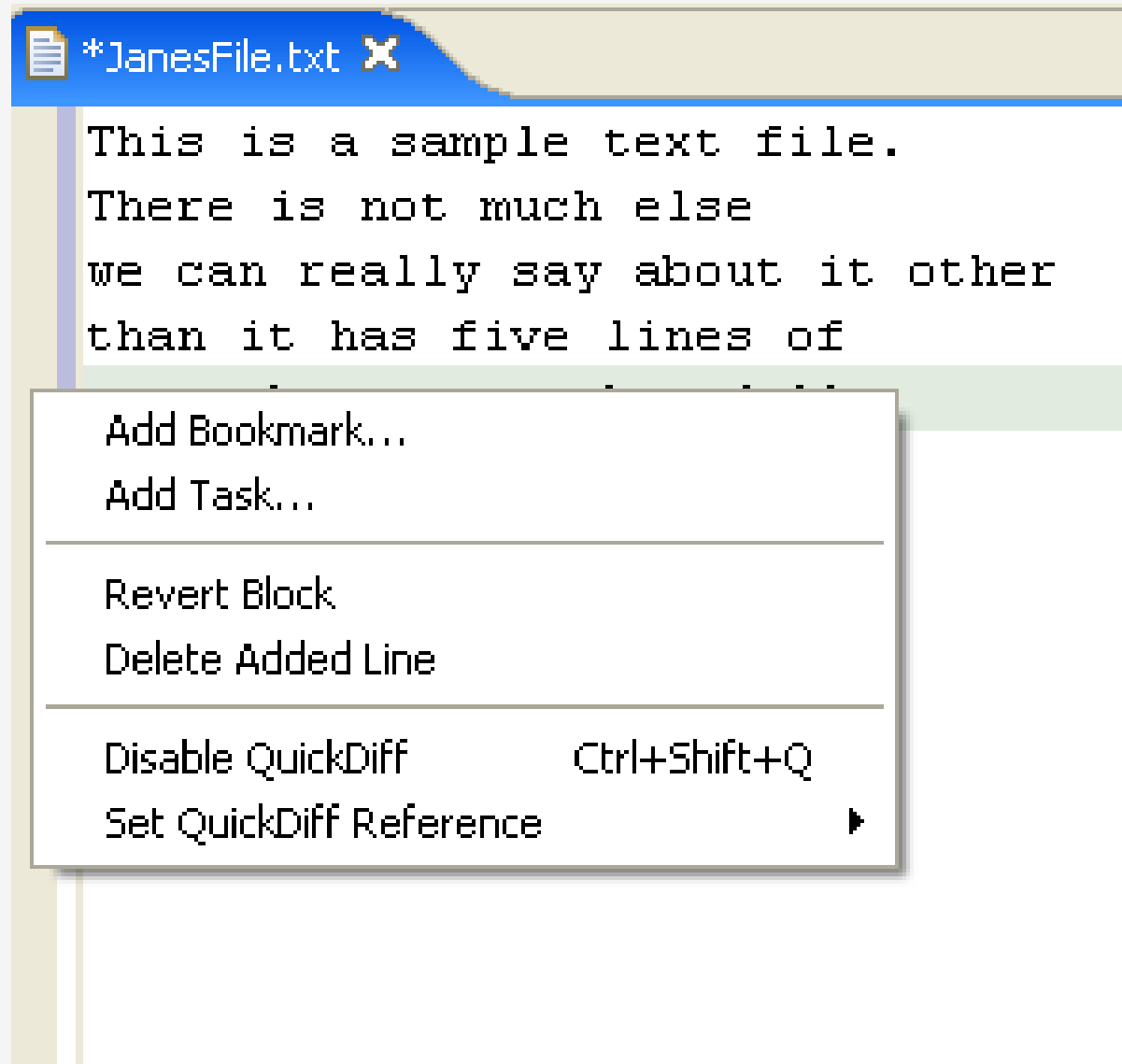
- Création :



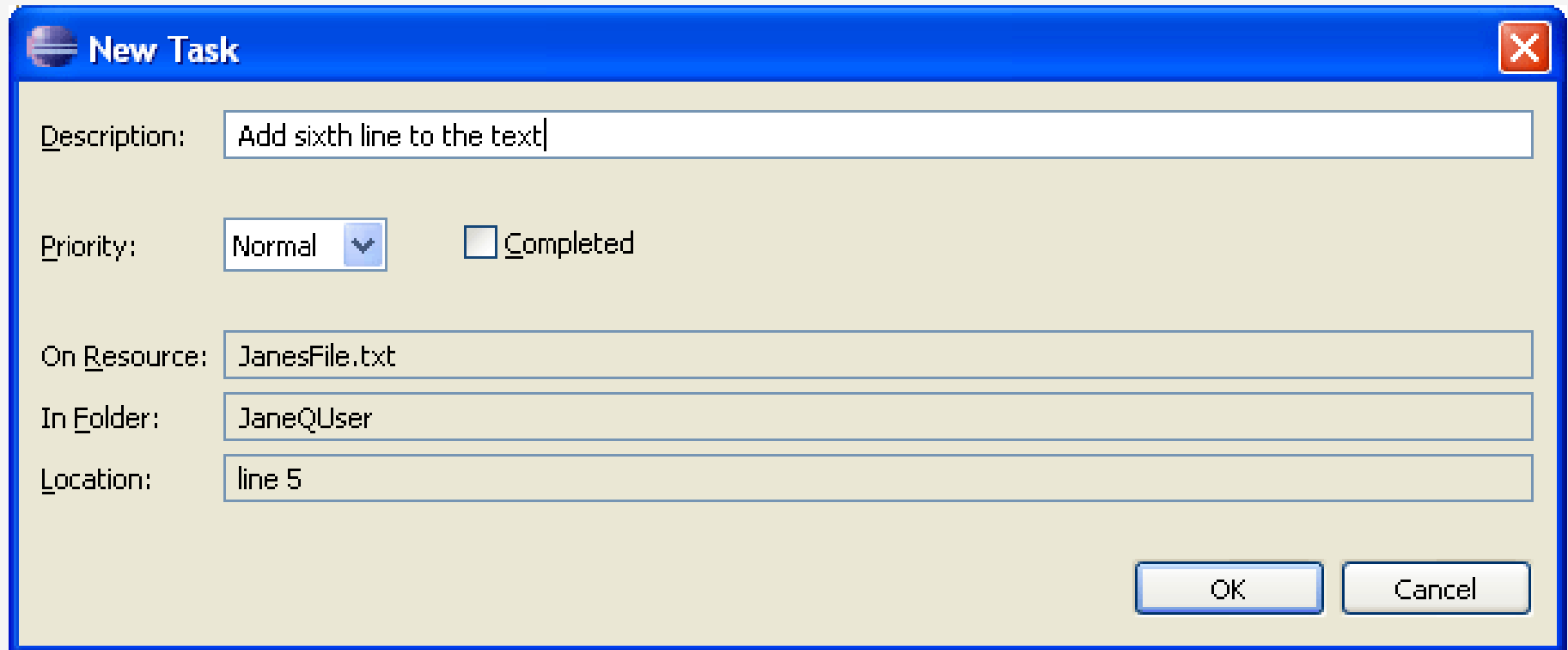
The screenshot shows a software window titled 'Tasks'. The window has a toolbar with several icons. A red box highlights the 'Add' icon, which is a blue document with a plus sign. To the right of the 'Add' icon are a red 'X' (delete), a double-headed arrow (move), a dropdown arrow, a minimize button,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

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

# Création et gestion de tâches



# Création et gestion de tâches



The image shows a 'New Task' dialog box with a blue title bar and a close button (red X) in the top right corner. The dialog contains several input fields and a checkbox:

- Description:** A text box containing the text 'Add sixth line to the text|'.
- Priority:** A dropdown menu set to 'Normal' and a checkbox labeled 'Completed' which is currently unchecked.
- On Resource:** A text box containing 'JanesFile.txt'.
- In Folder:** A text box containing 'JaneQUser'.
- Location:** A text box containing 'line 5'.

At the bottom right of the dialog, there are two buttons: 'OK' and 'Cancel'.

# Création et gestion de tâches

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