

**UT05 - IDE Entornos de  
Desarrollo Integrado**

# ÍNDICE

## Índice de contenido

1 ¿QUÉ ES UN IDE?.....	4
1.1 Componentes.....	4
2 El IDE ECLIPSE.....	4
2.1 Instalación.....	4
2.2 Acerca de Eclipse.....	5
2.3 El entorno de trabajo (workbench).....	5
2.3.1 Editores.....	8
2.3.2 Vistas.....	9
2.4 Proyectos.....	10
2.4.1 Navegando recursos.....	11
2.4.2 Copiar, pegar, buscar recursos.....	12
2.4.3 Tareas y marcadores.....	12
2.5 Organización de vistas.....	12
2.6 Perspectivas.....	13
2.7 Comparando ficheros.....	14
2.8 Historial de modificaciones.....	16
3 ECLIPSE y JAVA.....	17
3.1 Preparando Eclipse.....	17
3.2 Crear un proyecto de Java.....	19
3.3 Buscar elementos en el Explorador de Proyectos.....	22
3.4 Abriendo un editor de Java.....	23
3.5 Añadiendo un nuevo método.....	25
3.6 Usando el asistente de contenido.....	27
3.7 Identificando problemas en el código.....	28
3.8 Usando plantillas de código.....	29
3.9 Usando la historia local.....	30
3.10 Extrayendo un nuevo método.....	31
3.11 Creando una clase de Java.....	33
3.12 Renombrando elementos Java.....	38
3.13 Moviendo y copiando elementos Java.....	39
3.14 Navegar a las declaraciones Java de un elemento.....	40
3.15 Jerarquía de tipo.....	41
3.16 Buscando en el entorno de trabajo.....	46
3.17 Ejecutar programas Java.....	49
3.18 Depurar programas Java.....	52
3.19 Evaluar expresiones.....	56
3.20 Evaluar fragmentos o “snippets”.....	58
3.21 Perspectiva Java Browsing.....	59
4 BIBLIOGRAFÍA Y ENLACES.....	61

### Versión del documento

13/05/12 Creación del documento  
20/05/12 Java y Eclipse  
27/05/12 Editar archivos Java  
29/05/12 Navegando por Java  
03/06/12 Fin del doc.

### 1 ¿QUÉ ES UN IDE?

Un **entorno de desarrollo integrado**, llamado también **IDE** (sigla en inglés de *integrated development environment*), es un [programa informático](#) compuesto por un conjunto de herramientas de [programación](#). Puede dedicarse en exclusiva a un sólo [lenguaje de programación](#) o bien poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de [interfaz gráfica](#) (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. El lenguaje [Visual Basic](#), por ejemplo, puede ser usado dentro de las aplicaciones de [Microsoft Office](#), lo que hace posible escribir sentencias [Visual Basic](#) en forma de [macros](#) para [Microsoft Word](#).

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como [C++](#), [PHP](#), [Python](#), [Java](#), [C#](#), [Delphi](#), [Visual Basic](#), etc.

#### 1.1 Componentes

Los componentes del IDE pueden ser:

- Un [editor de texto](#)
- Un [compilador](#)
- Un [intérprete](#)
- Un [depurador](#)
- Posibilidad de ofrecer un sistema de [control de versiones](#).
- Posibles ayudas para la construcción de [interfaces gráficas de usuario](#).

Algunos entornos son compatibles con múltiples [lenguajes de programación](#), como [Eclipse](#) o [NetBeans](#), ambos basados en [Java](#); o [MonoDevelop](#), basado en [C#](#). También puede incorporarse la funcionalidad para lenguajes alternativos mediante el uso de [plugins](#). Por ejemplo, Eclipse y NetBeans tienen plugins para [C](#), [C++](#), [Ada](#), [Perl](#), [Python](#), [Ruby](#) y [PHP](#), entre otros.

Algunos de ellos son entornos [libres](#), como Code::Blocks, Eclipse, Lazarus, KDevelop y Netbeans.

### 2 EL IDE ECLIPSE

#### 2.1 Instalación

La herramienta puede descargarse de <http://www.eclipse.org/downloads/>

El descargable es un archivo comprimido eclipse-java-indigo-SR2-win32.zip

El siguiente paso es descomprimir el archivo en el directorio de destino. No es necesaria la instalación y puede iniciarse la herramienta haciendo doble click en .\eclipse\eclipse.exe

Antes, puede [descargarse un pack de idiomas para el español](#), aunque algunos plugin permanecerán en Inglés, al menos parte de la herramienta tendrá mensajes en castellano. Tras descargar el pack, debe descomprimirse en el mismo directorio donde se instaló la herramienta. [Instrucciones](#).

### **2.2 Acerca de Eclipse**

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.

Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.

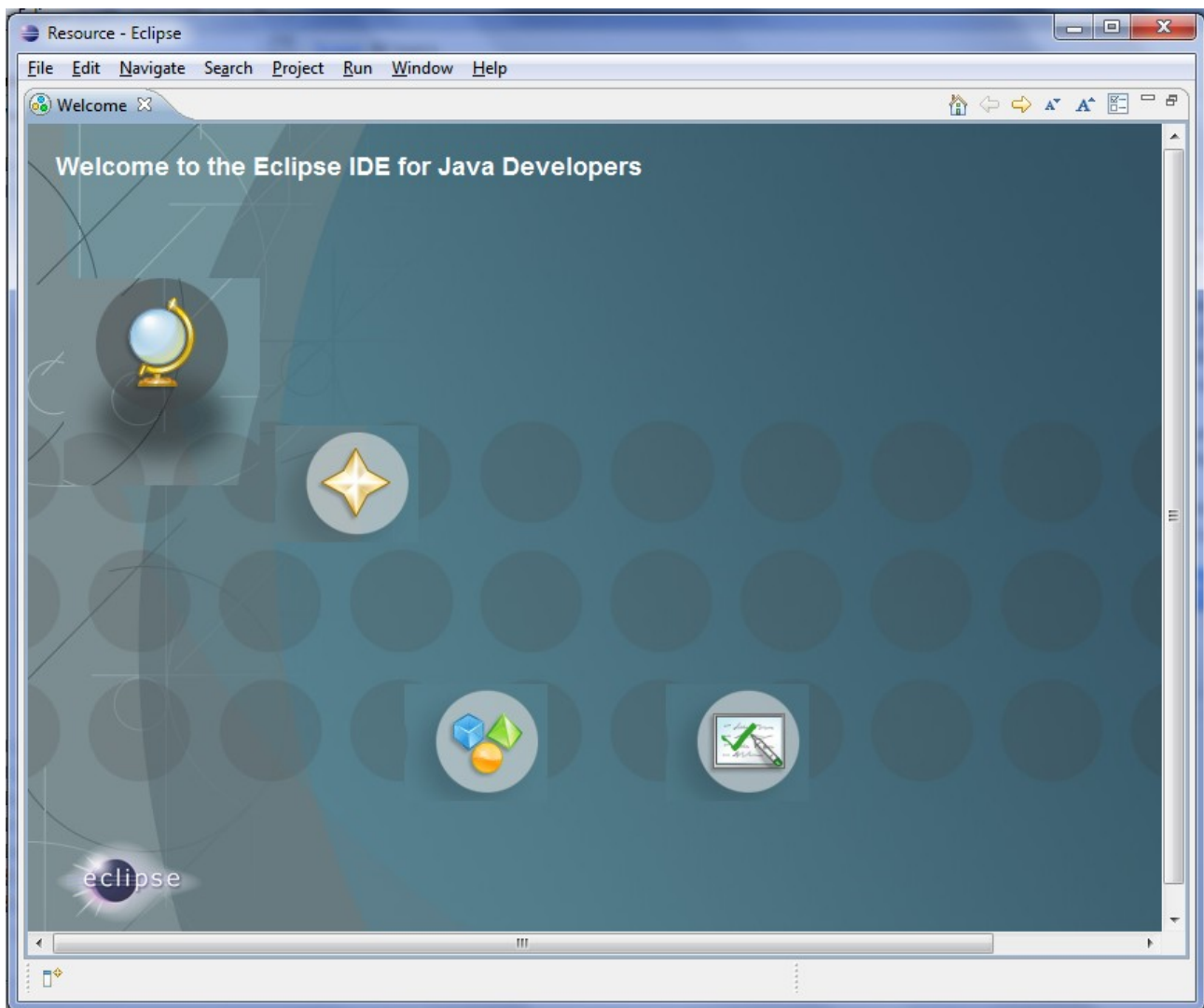
Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License. La Free Software Foundation ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con Licencia pública general de GNU (GNU GPL)

### **2.3 El entorno de trabajo (workbench)**

Lo primero que aparece es un diálogo pidiendo seleccionar el directorio donde se ubicará el espacio de trabajo. Será en ese directorio donde se guardarán los proyectos realizados.

Una vez indicado, aparece la pantalla siguiente:

## UT05 - IDE Entornos de Desarrollo Integrado



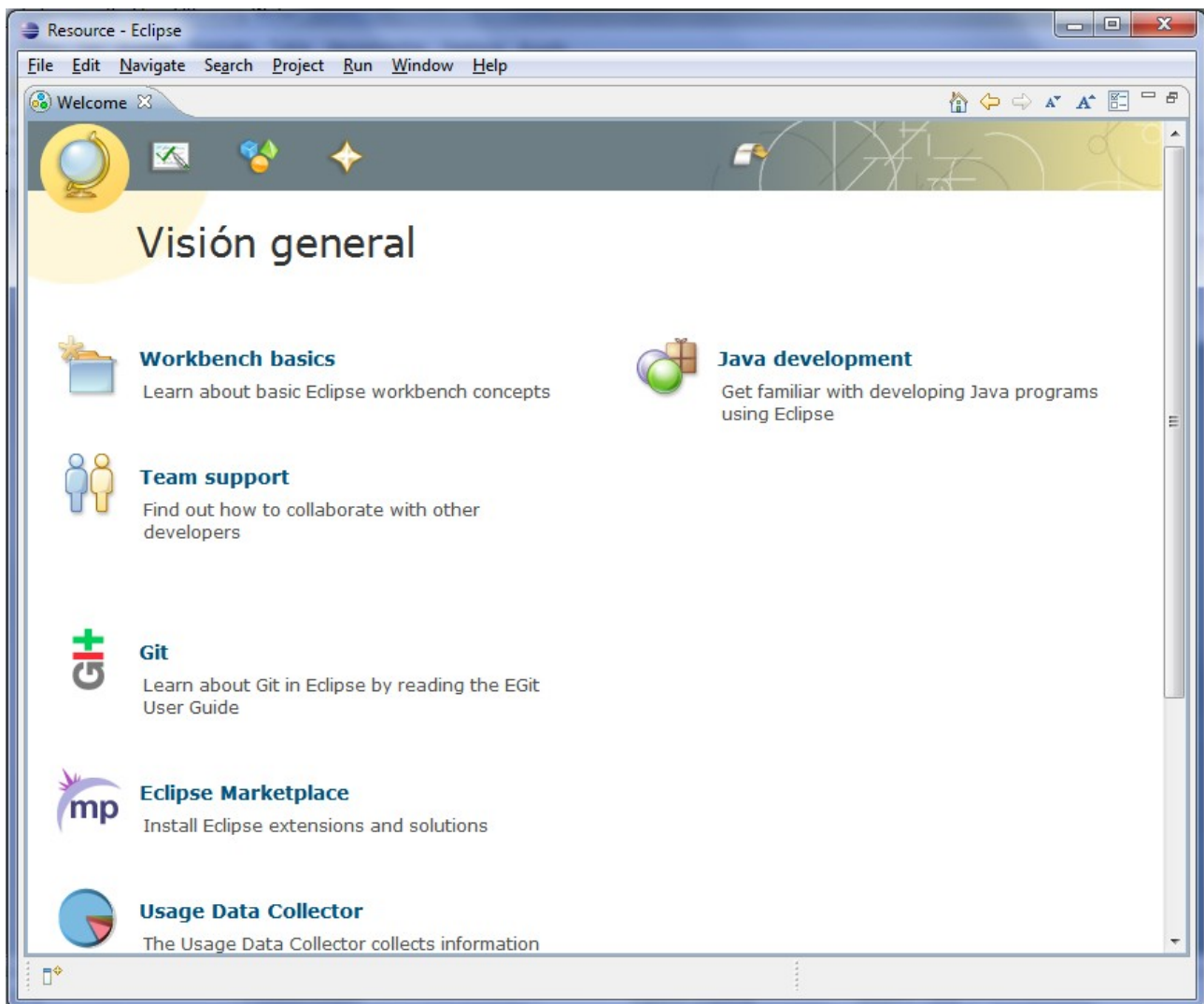
*Ilustración 1: Bienvenida*

Se trata de la pantalla de bienvenida, donde puede navegarse por tutoriales, ayudas, ejemplos, novedades, etc.

Pulsemos la bola del mundo que corresponde a “visión general”.

Aparece una nueva pantalla:

## UT05 - IDE Entornos de Desarrollo Integrado



*Ilustración 2: Visión general*

Por el momento, puede cerrarse la pantalla pulsando sobre la X a la derecha del título de la ventana “Welcome”. Puede recuperarse la ventana anterior, pulsando Help → Welcome en cualquier momento.

Lo que aparece después es el entorno de trabajo (*Workbench*).

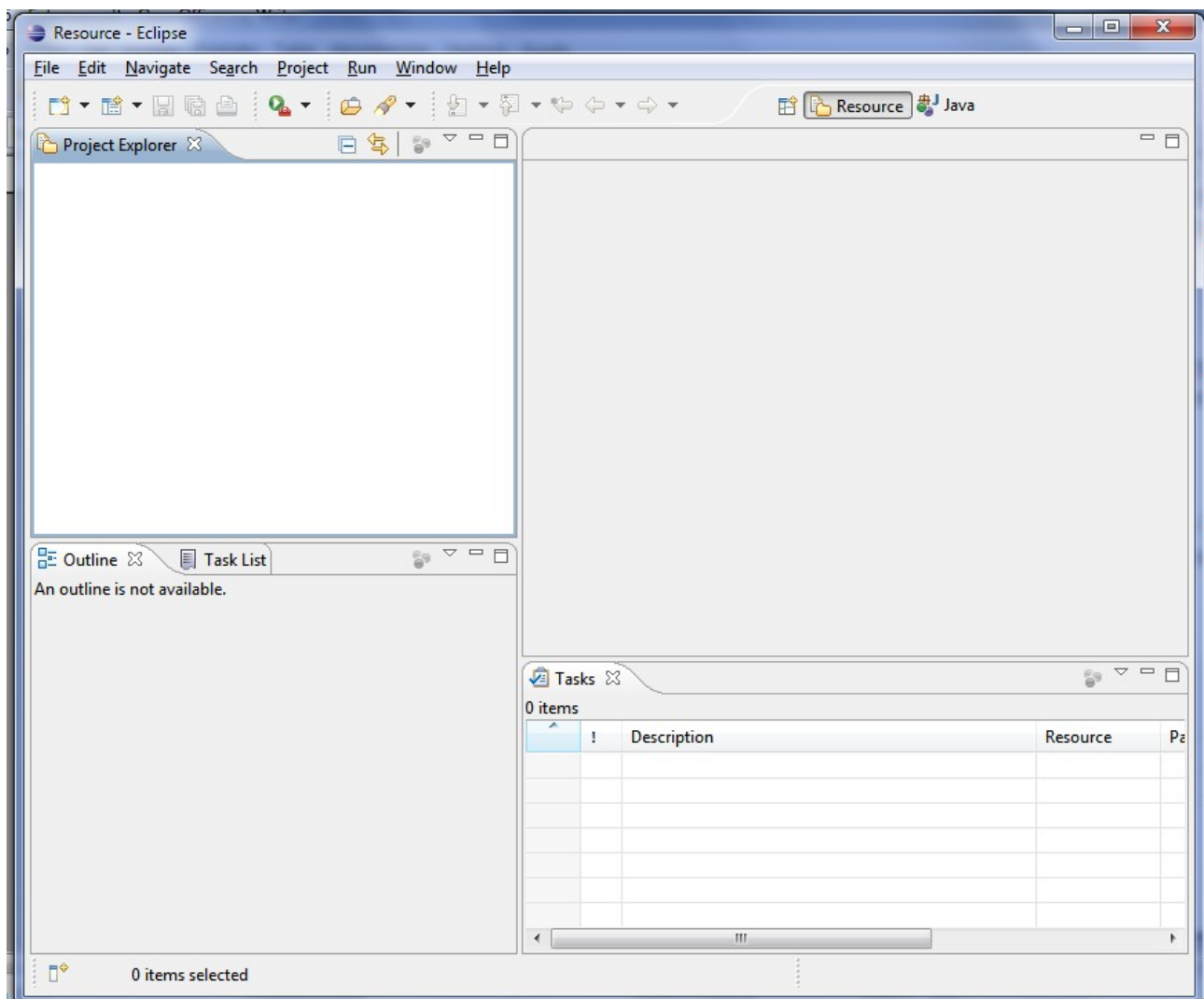
El entorno dispone de varias configuraciones visuales o perspectivas (*perspectives*). Arriba a la derecha puede cambiarse la perspectiva en uso (por defecto es java). Si quiere abrirse otra, hay un icono para hacerlo:



Seleccionamos la perspectiva “Resource” y veremos como la disposición de los marcos (partes) de trabajo cambia. Esas partes pueden ser vistas o editores. Las vistas son componentes visuales.

En cada momento sólo habrá una parte activa, mostrándose en azul su título.

## UT05 - IDE Entornos de Desarrollo Integrado

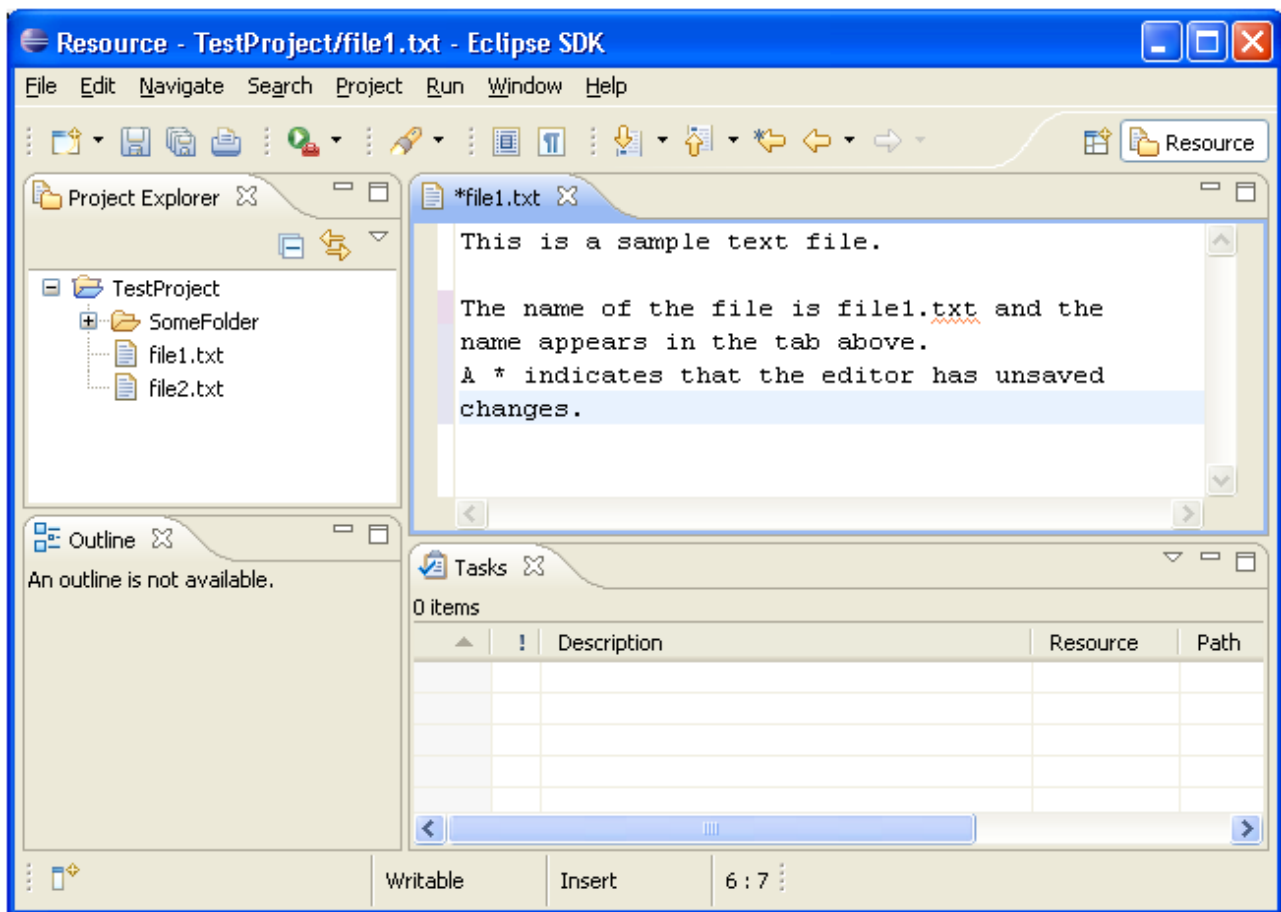


*Ilustración 3: vista "Project Explorer" activa*

### 2.3.1 Editores

Dependiendo del archivo a editar se usará el editor adecuado a la tarea.





*Ilustración 4: editor de texto*

En la barra de título, si aparece un asterisco “\*” al lado del nombre de archivo, se indica que el archivo en edición tiene cambios pendientes de grabarse en disco.

Si se abren varias partes con editores, éstas **pueden colocarse** de diversas maneras pulsando la barra de título y arrastrando a una nueva posición.

Puede navegarse a través de las diferentes editores por orden de utilización con los botones de flecha de navegación. O bien de modo circular con Ctrl+F6.

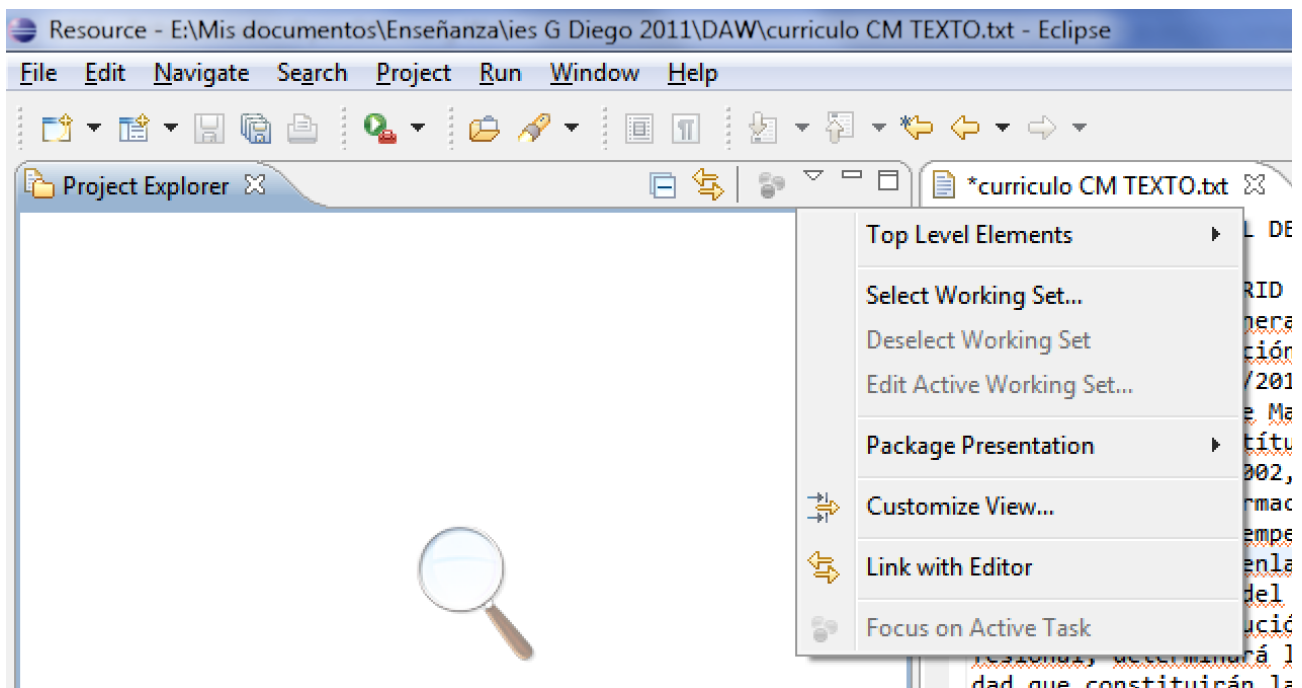
Puede intentarse abrir cualquier tipo de archivo. Si Eclipse no dispone de un editor propio, abrirá el que tenga asociado el Sistema Operativo.

## 2.3.2 Vistas

Pulsando el botón derecho sobre el título de una vista aparece una serie de opciones como mover, minimizar, cambiar el tamaño, etc.

Hay un botón en algunas vistas, llamado “pull down” donde se muestran opciones de visualización, filtro u ordenación de los contenidos de las vistas.

## UT05 - IDE Entornos de Desarrollo Integrado



*Ilustración 5: pull down menu*

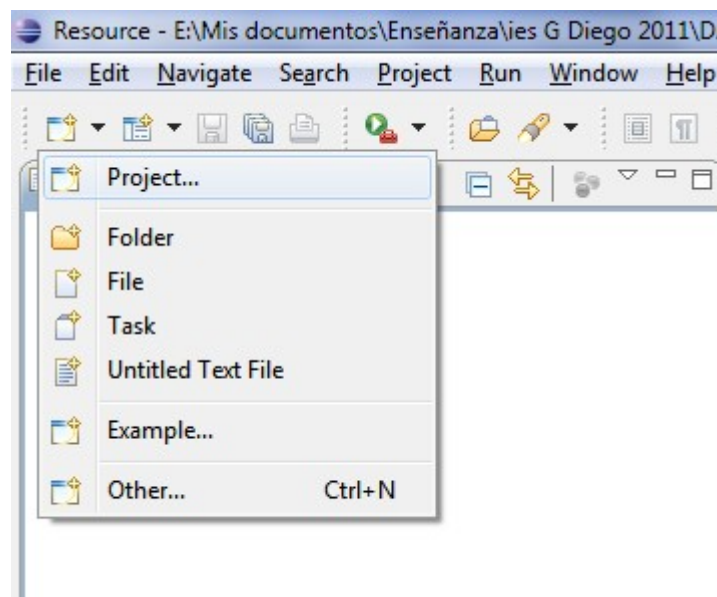
La lista de todas las vistas posibles aparecen en Window → Show View o pulsando en el icono de la esquina inferior izquierda de la pantalla.

Windows → reset perspective deja la perspectiva en su estado original.

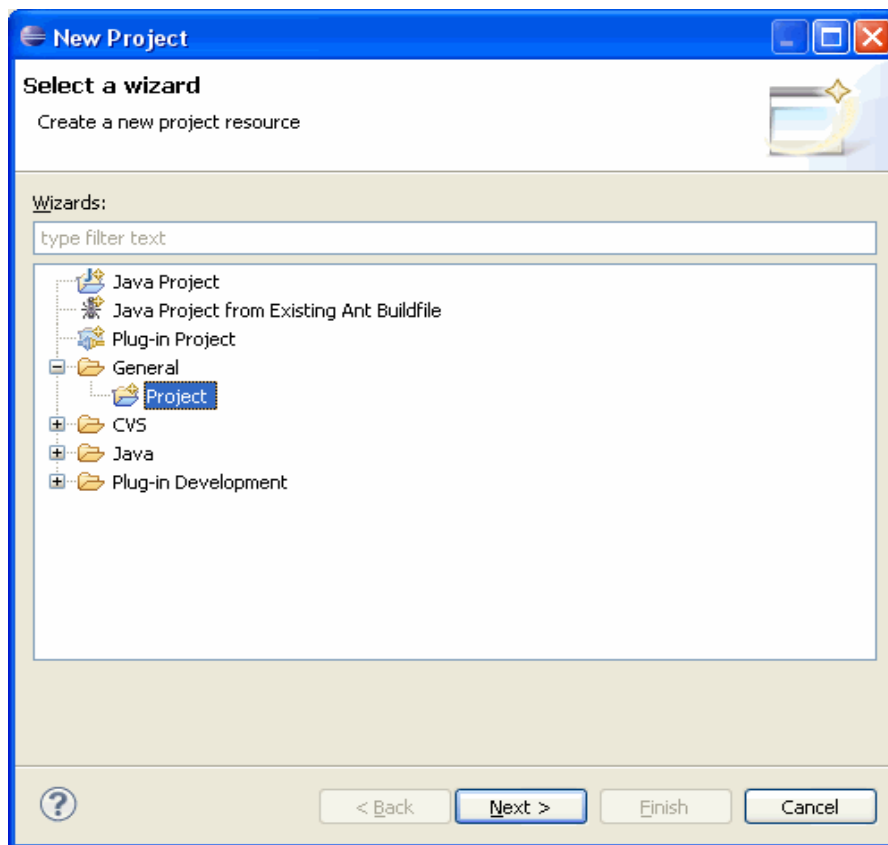
### 2.4 Proyectos

Veremos cómo crear proyectos. Un proyecto puede ser creado usando el menú File → New → Project. Una vez que el proyecto ha sido creado, una carpeta y un archivo serán también creados.

Otra opción es usar el botón de New de la barra de herramientas y seleccionar “Project”:




*Ilustración 6: Creando un proyecto nuevo*



*Ilustración 7: Creando un nuevo proyecto*

Una vez iniciado, seleccionamos General → Proyecto y Next. En el campo Project Name ponemos el nombre del proyecto y pulsamos Finish.

Dentro del proyecto podemos crear carpetas y archivos.

El botón  puede usarse para encadenar la vista de navegación y la selección de la ventana de editor activa. Si se pulsa, al cambiar de archivo en la vista de navegación, se cambia el foco en el archivo que se está editando hacia el que se selecciona en la vista de navegación.

### 2.4.1 Navegando recursos

Los proyectos, carpetas y archivos se llaman en conjunto “**recursos**”.

Un recurso puede abrirse seleccionándolo en la vista de navegación (Project Explorer) y haciendo doble click o bien desplegando con el botón derecho y eligiendo con qué se abre (puede elegirse incluso una aplicación externa). Las asociaciones de archivos y programas se pueden modificar en Window → Preferences → General → Editors → File associations


Cuando se ha trabajado con diversos archivos o recursos, es posible pasar de uno a otro con la herramienta Navigate → Go to → Resource

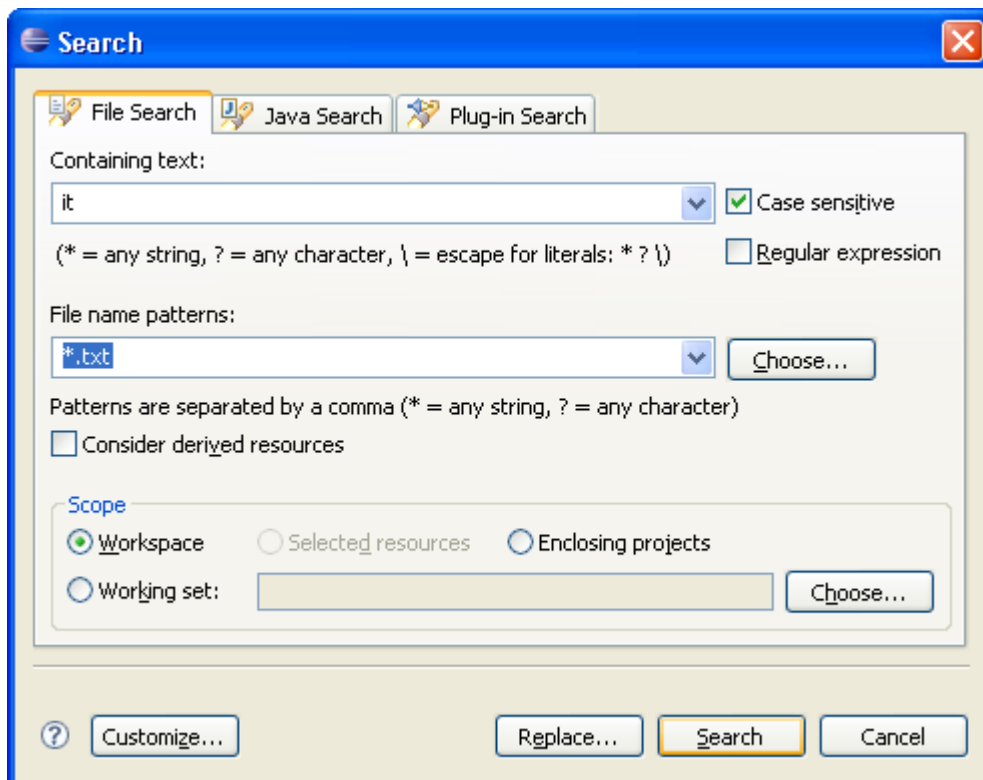
Todos los recursos se localizan físicamente en el directorio que se indicó al iniciar Eclipse.

Si no se recuerda la ubicación o se desea añadir otra ubicación o cambiarla, se debe hacer en File → Switch Workspace. IMPORTANTE: tras grabar la localización, se debe pulsar Cancel para cerrar el diálogo, o Eclipse se cerrará y se reabrirá en el nuevo espacio de trabajo.

## 2.4.2 Copiar, pegar, buscar recursos

Puede copiarse y pegarse o moverse recursos siguiendo los procedimientos habituales en las vistas de navegación.

Existe una herramienta de búsqueda de recursos accesible por botón  de la barra de herramientas.



*Ilustración 8: Herramienta de búsqueda de recursos*







Pueden hacerse búsquedas usando patrones o expresiones regulares tanto de nombre de recurso como de texto o elementos contenidos en ellos. En el tab “Java search” se permite la búsqueda de elementos lógicos de las clases, por ejemplo métodos, paquetes, constructores, etc.

## 2.4.3 Tareas y marcadores

Pueden añadirse marcas de posición (bookmarks: Edit → Add Bookmark o botón derecho en el archivo → add task) en uno o varios recursos, marcas para depuración, tareas (Edit → Add Task o botón derecho en el archivo → Add Bookmark) y problemas. El tipo de marca aparecerá a izquierda en el archivo y línea donde se ponga y pueden ser buscados con la herramienta de búsqueda (search) o con las vistas correspondientes: pueden verse todas las tareas, bookmarks y problemas en la vista correspondiente dedicada a ellas.

## 2.5 Organización de vistas

Las vistas pueden organizarse a partir de la perspectiva seleccionada arrastrando la barra de título y soltando en la ubicación deseada. Al mover la vista aparece uno de los siguientes símbolos y un recuadro que indica la nueva ubicación.

	Colocar arriba
	Colocar abajo
	Colocar a la derecha
	Colocar a la izquierda
	Colocar como pestaña
	Prohibido

Windows → Reset perspective retorna la apariencia a la colocación de vistas por defecto.

## 2.6 Perspectivas

Constituye un conjunto de vistas visibles con una particular disposición acorde con el objetivo de cada perspectiva.

Hasta ahora hemos trabajado con la perspectiva sencilla llamada “*Resources*”.

Puede **abrirse** una perspectiva desde el botón de la barra de herramientas o desde Window → Open Perspective

Al abrir la nueva perspectiva se abrirá en el entorno de trabajo (Workbench). Puede configurarse las Preferencias **para que se abra en un nuevo Workbench** (Window → Preferences → General → Editors → Perspectives).

El usuario puede disponer las vistas a su gusto y **crear una nueva perspectiva**, guardándola desde Window → Save Perspective As...

Además es posible configurar los iconos visibles en la barra de herramientas, las opciones de la barra de menús y sus desplegables y ... en **Window → Customize Perspective**

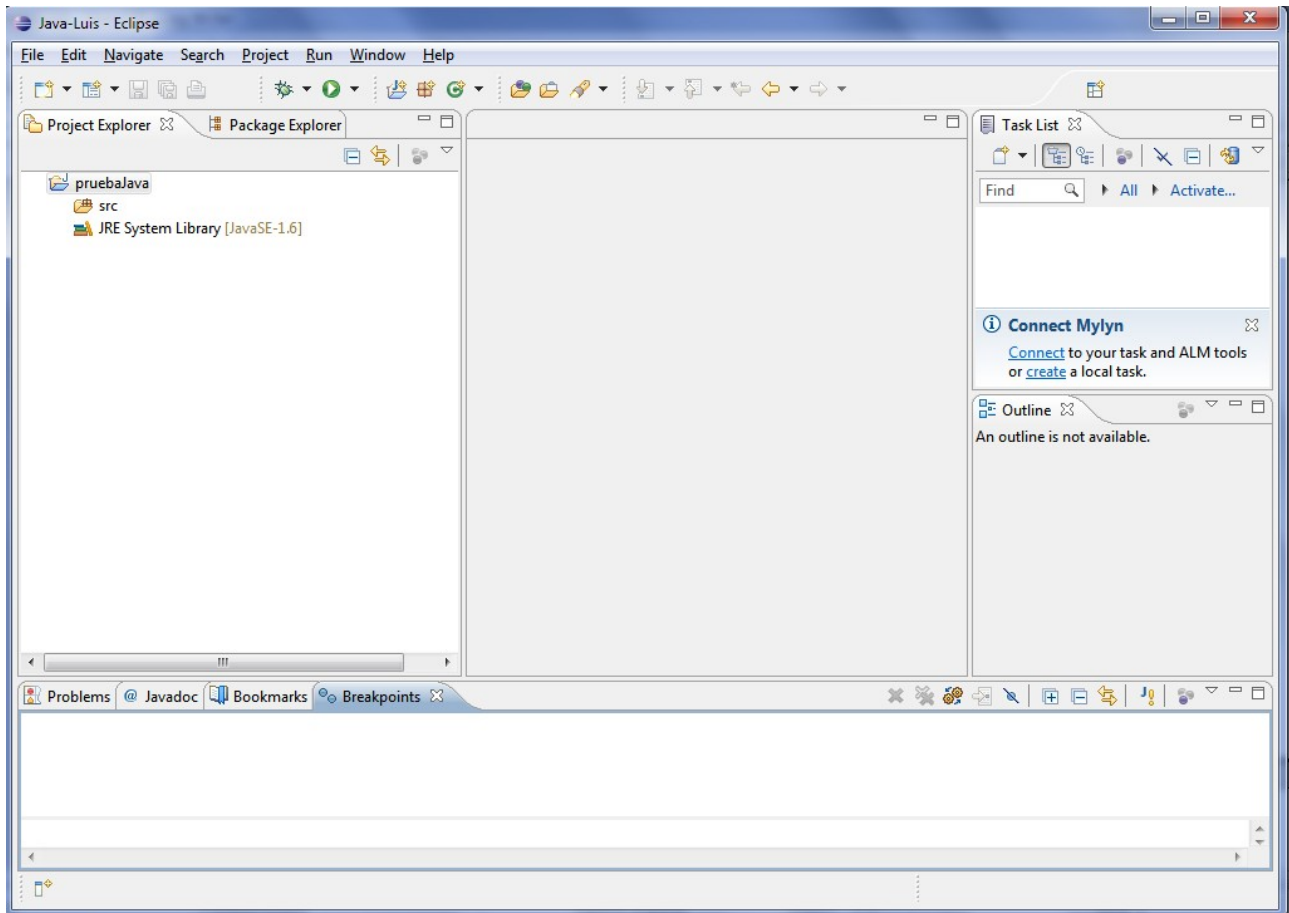
**Window → Reset Perspective** restaura la configuración de la perspectiva actual.

### Ejercicio 1

Dada la perspectiva “Java (default)” configúrala para que quede como la de la imagen de la Ilustración 1 haciendo las transformaciones siguientes:

- Quita la vista declaration
- Ensancha las vistas de la parte inferior par darles más espacio
- Añade el Package Explorer junto al Project Explorer
- Elimina el Icono “run last tool” y “new java project”

Guarda el nuevo layout como una perspectiva “java-alternativa”



*Ilustración 9: Perspectiva del Ejercicio 1*

### 2.7 Comparando ficheros

Selecciona la perspectiva Resources.

Crea un nuevo proyecto “compararFicheros”

Crea en él un ficheros de texto file1.txt

En file1.txt teclea:

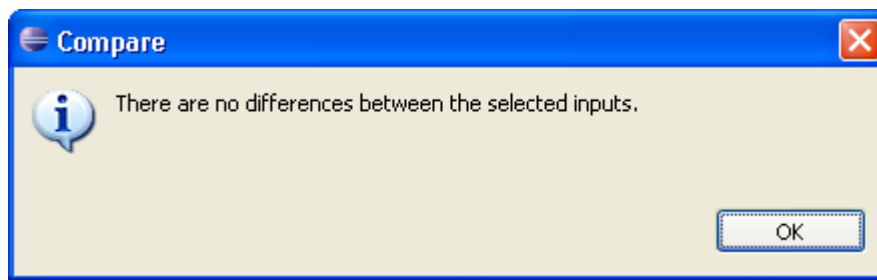
```
This is line 1.  
This is line 2.  
This is line 3.  
This is line 4.  
This is line 5.
```

Copia el fichero como file2.txt

Tendremos dos ficheros idénticos.

Para compararlos selecciónalos **en la ventana de navegación** y pulsa Compare With → Each Other en el menú contextual (botón derecho).

Debe aparecer la imagen:



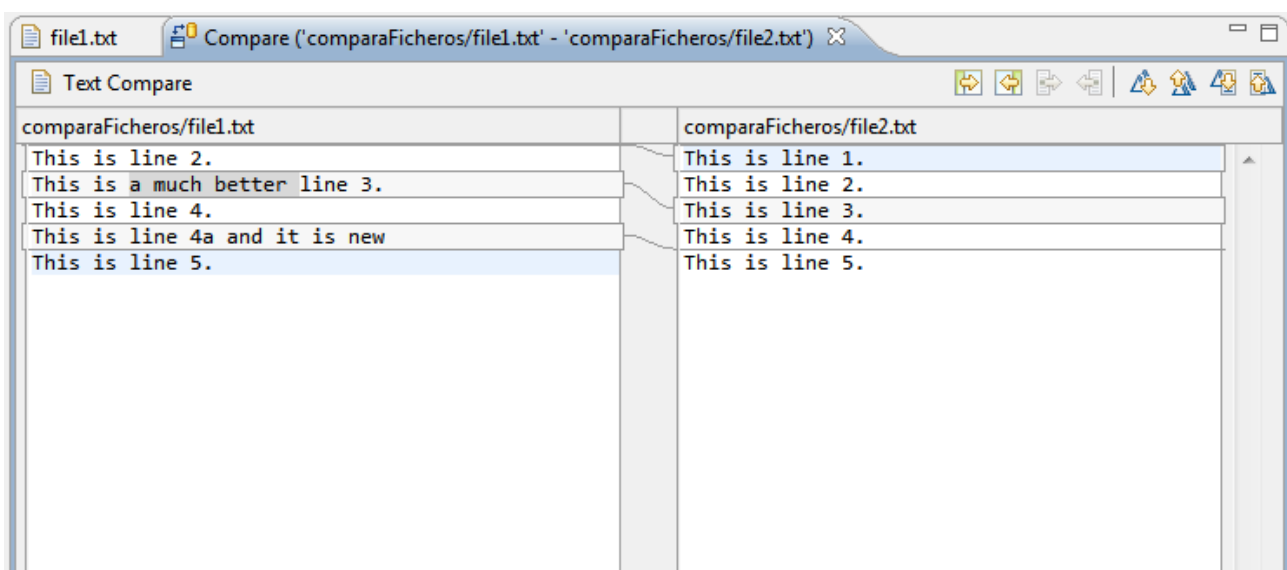
*Ilustración 10: ficheros iguales*

Ahora cambiemos varias líneas de file1.txt

```
This is line 2.  
This is a much better line 3.  
This is line 4.  
This is line 4a and it is new  
This is line 5.
```

Guarda los cambios y repite la comparación.

Aparece un editor especial de comparación:



*Ilustración 11: Comparación ficheros diferentes*

En el diagrama se conectan aquellas líneas que son diferentes en ambos ficheros.

La primera conexión indica que falta una línea en file1.txt

La segunda línea indica diferencias entre ambos ficheros.

La tercera línea indica que el texto en file1.txt no aparece en file2.txt


Para mejor visualización, puede seleccionarse con el cursor la conexión, que se mostrará resaltada.

Los iconos del editor de comparación son:




 : Siguiente cambio. Navega entre las diferencias.

## UT05 - IDE Entornos de Desarrollo Integrado

: Cambio anterior.

El resto de iconos sirve para mezclar las líneas de uno y otro fichero y resolver las diferencias:

 Por orden, significan:

- Copia el documento entero de izquierda a derecha.
- Copia el documento entero de derecha a izquierda.
- Copia cambio actual de izquierda a derecha.
- Copia cambio actual de derecha a izquierda.

Para borrar una línea, simplemente despliega el menú de contexto y elige Delete.

### Ejercicio 2

Toma los dos ficheros que te indique el profesor, incorpóralos a un nuevo proyecto (comparaEj2) y compáralos. Haz que en los dos ficheros, finalmente, queden iguales. Para ello:

- Si dos líneas son diferentes, la sea de mayor longitud y reemplaza la otra
- Incluye en el otro fichero las líneas que no estén
- Para cada cambio añade un bookmark
- Muestra la vista de bookmarks al finalizar colocándola junto a Tasks
- Ve a la segunda línea marcada del file2.txt a partir de su bookmark

## 2.8 Historial de modificaciones

Puede verse el historial de modificaciones de un fichero pulsando en el menú contextual Team → Show Local History

Se mostrará un diálogo como el siguiente:

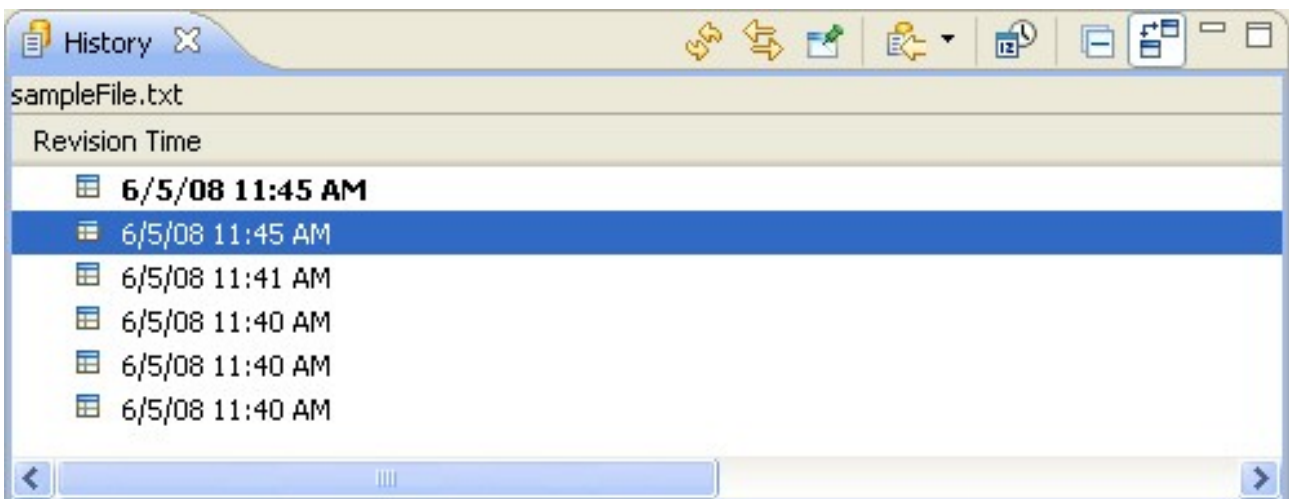


Ilustración 12: Historial de modificaciones

En él se muestra cada modificación hecha como una entrada en una línea diferente. La última entrada es el estado actual del fichero.

Puede verse el estado del fichero en cada entrada del historial haciendo doble click sobre la entrada.

Si se quieren resaltar los cambios del fichero en un instante dado con respecto al estado actual, en la



entrada correspondiente, en el menú contextual puede seleccionarse la opción “Compare current with Local”. Se abrirá la ventana de comparación de ficheros y podrán recuperarse selectivamente las líneas que se requiera.

La opción “Get contents” sustituye la versión actual por la que se haya seleccionado.

### 3 ECLIPSE y JAVA

#### 3.1 Preparando Eclipse

Vamos a verificar la instalación del JRE (Java Runtime Environment). Para que Eclipse funcione correctamente debemos tenerlo instalado. Para ello comprobamos Windows → Preferences → Java → Installed JREs

Por defecto estaremos usando el JRE usado para el entorno de desarrollo de Eclipse (para abrir Eclipse es necesario tener instalado un JRE). Sin embargo se recomienda usar un JDK (Java Development Kit) en lugar de un JRE para desarrollar en Java. El JDK está diseñado para el desarrollo y contendrá código fuente de la biblioteca de Java y facilidades para depurar.

Sal de Eclipse e instala el SDK de Java desde [aquí](#).

Cuando se termine de instalar, vuelve a iniciar Eclipse y ve a la opción Windows → Preferences → Java → Installed JREs. Pulsa el botón Search para localizar el nuevo SDK y selecciónalo de la lista mostrada.

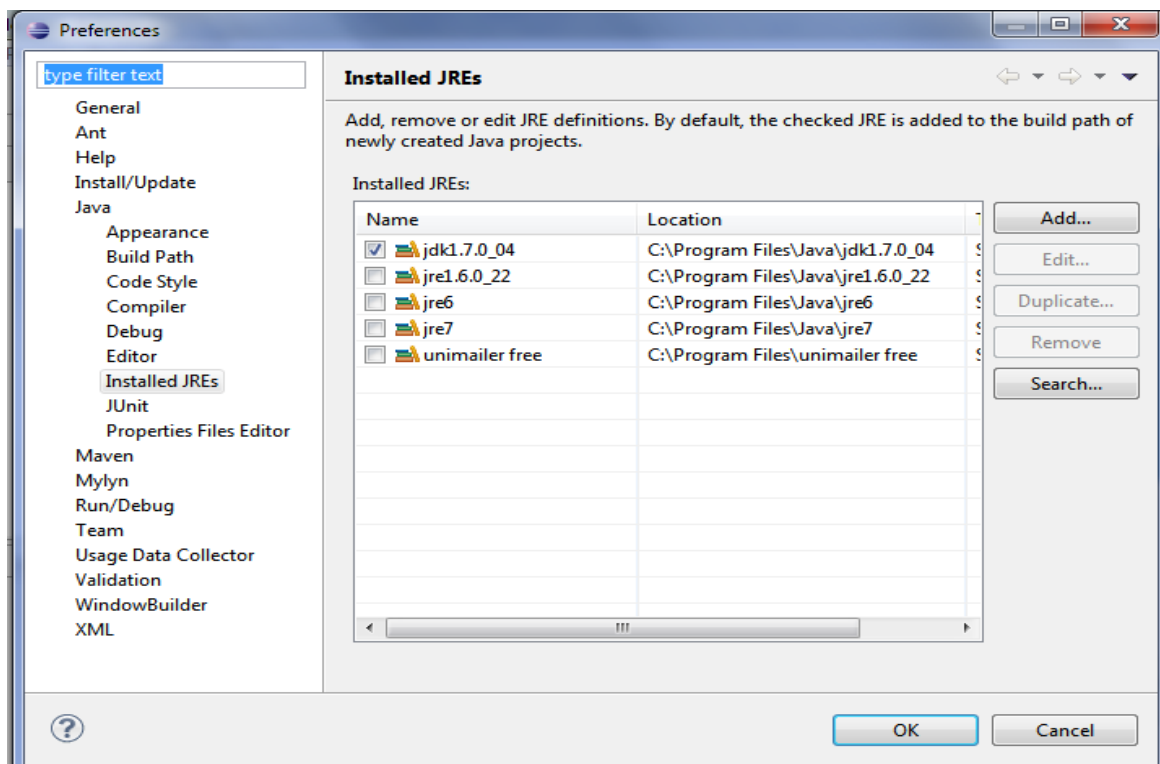
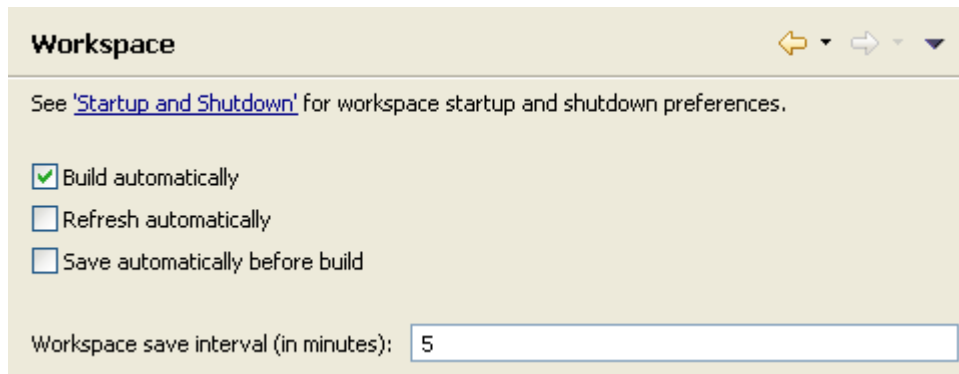


Ilustración 13: Selección JDK

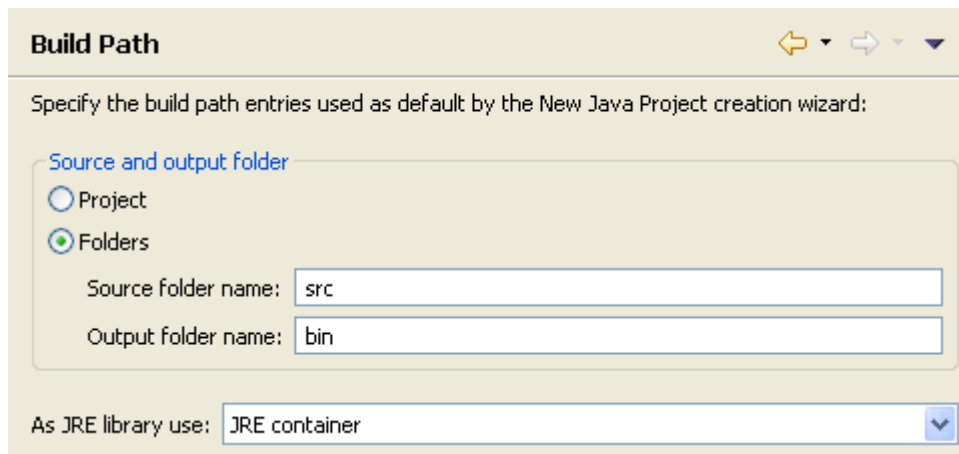
## UT05 - IDE Entornos de Desarrollo Integrado

En General → Workspace confirma que está activada la selección Build automatically.



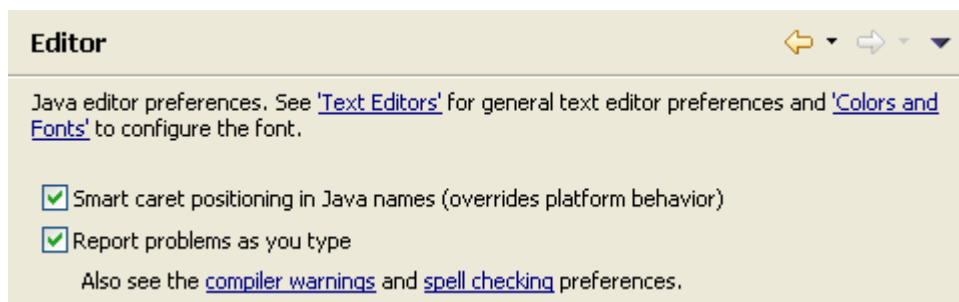
*Ilustración 14: Build automático*

En Java → Build Path la selección debe ser:



*Ilustración 15: Path*

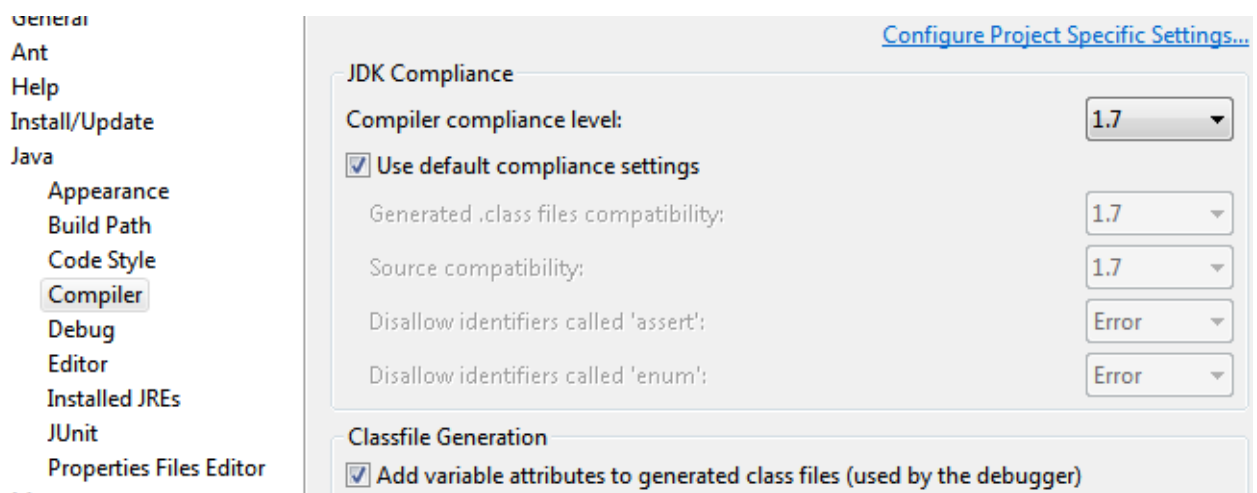
En Java → Editor marca “**Report problems as you type**” para que los errores se muestren según se teclea.



*Ilustración 16: Reportar errores mientras se teclea*

En Java > Compiler selecciona 1.7 Pulsa OK para grabar todas las preferencias.

## UT05 - IDE Entornos de Desarrollo Integrado



*Ilustración 17: Versión*

Pulsa OK para grabar las opciones anteriores.

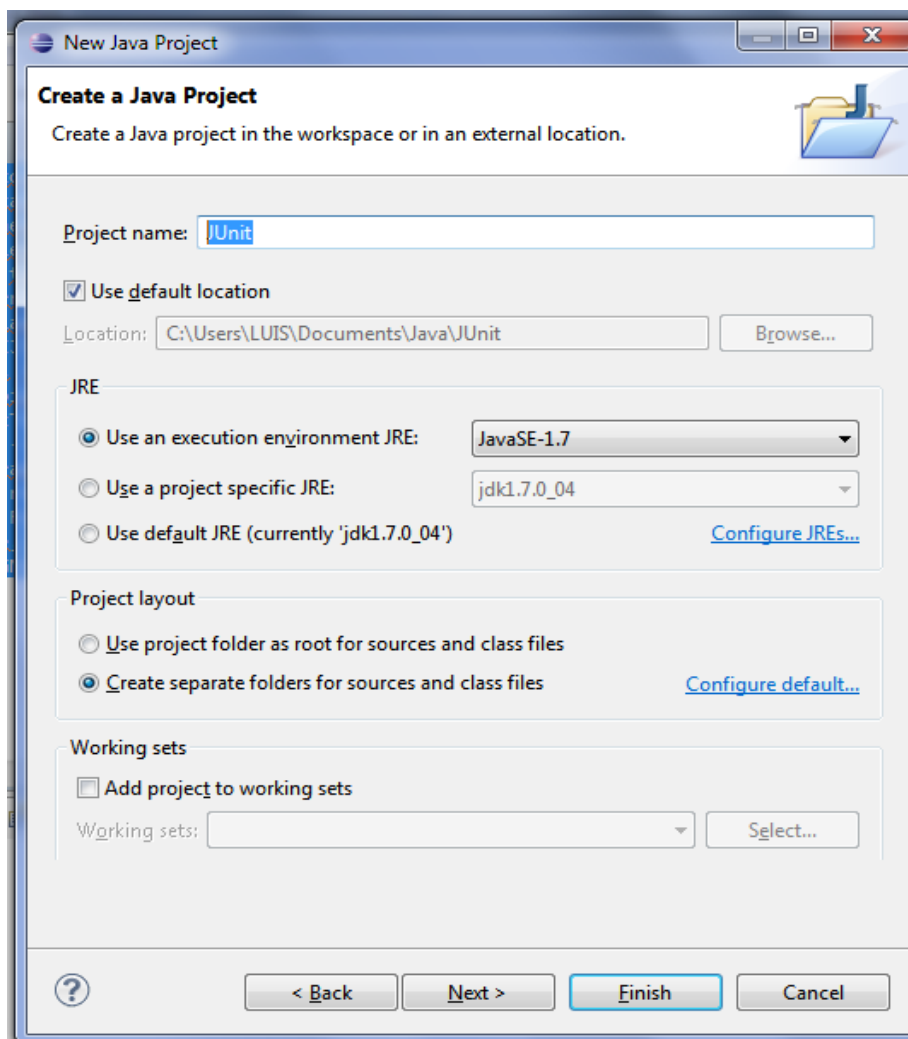
### **3.2 Crear un proyecto de Java**

Descarga el [proyecto de prueba](#) desde la página de Eclipse.

Desde Eclipse pulsa **File > New > Project**, selecciona Java Project y Next para iniciar el Wizard.

En el diálogo, rellena: Nombre del Proyecto: JUnit

## UT05 - IDE Entornos de Desarrollo Integrado



*Ilustración 18: Nombre de proyecto*

Pulsa “Finish”

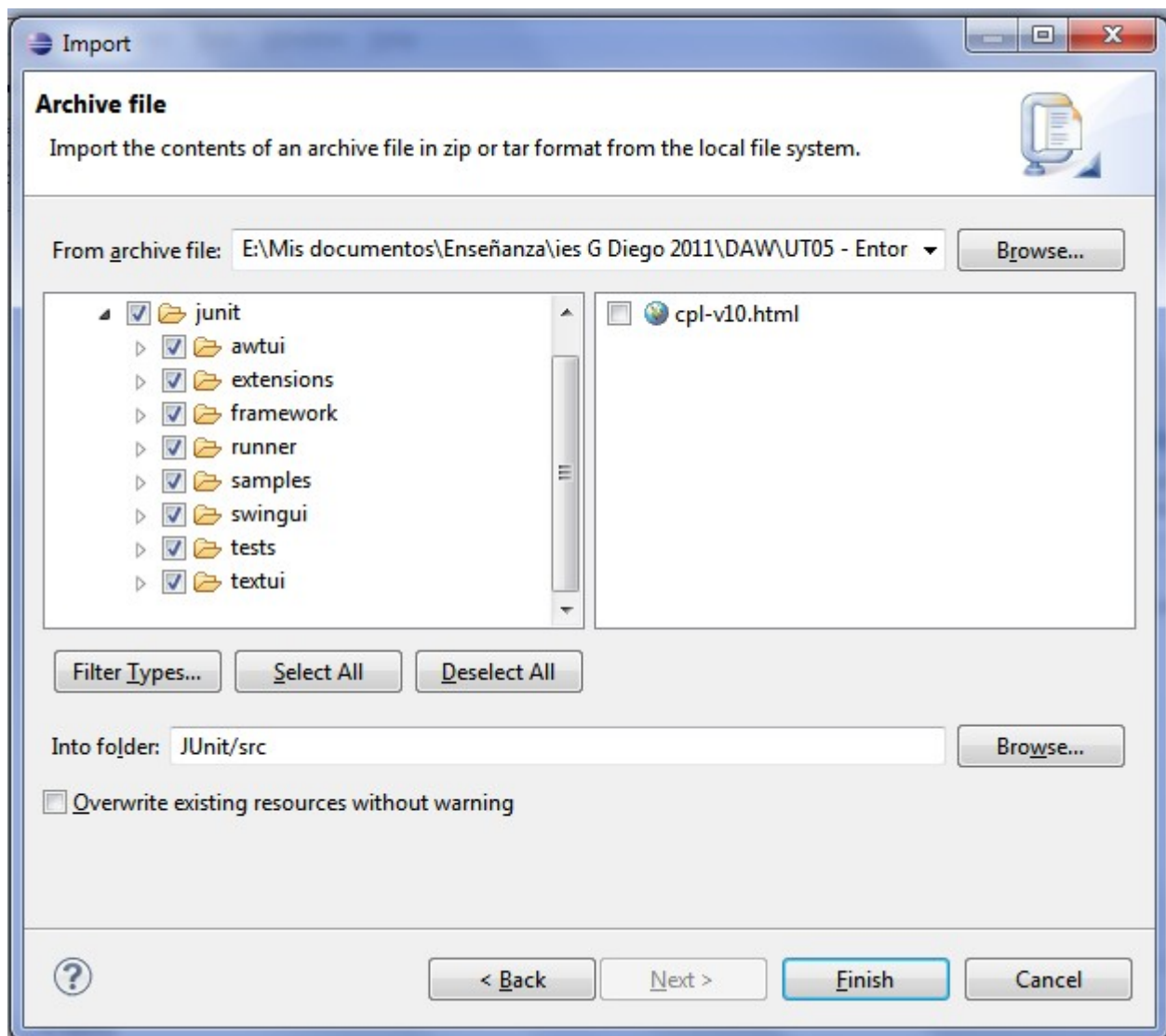
En el explorador de proyectos, en proyecto, selecciona la carpeta “src” y en el menú contextual, “Import”.

General → Archive file → Next

Pulsa “Browse” para buscar los archivos y selecciona el zip descargado.

Debes seleccionar las opciones de la Ilustración 19

## UT05 - IDE Entornos de Desarrollo Integrado

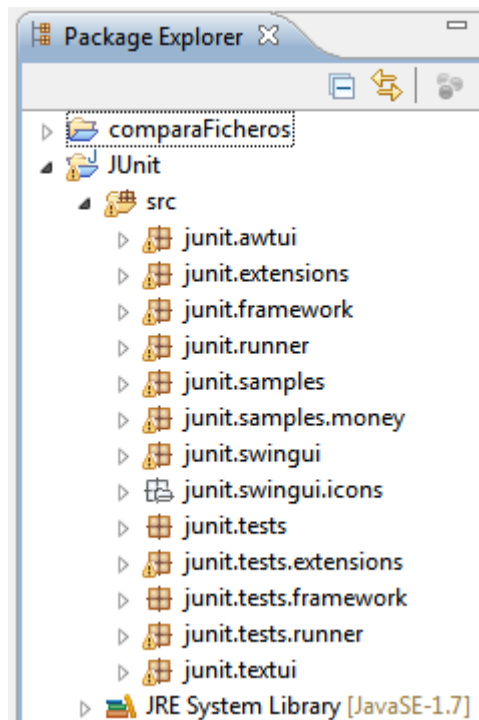


*Ilustración 19: Importar ficheros*

Pulsa “Finish”.

Los recursos importados se compilan mientras se importan en el entorno de trabajo. Es debido a la opción “Build automatically” chequeada antes en las preferencias.

En el Explorador de Paquetes, expande el proyecto Junit y la carpeta src para ver lo paquetes.



*Ilustración 20: Proyecto JUnit*

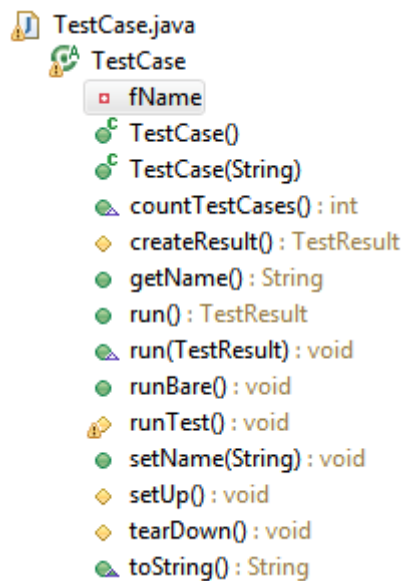
### **3.3 Buscar elementos en el Explorador de Proyectos**

En la ventana de la Ilustración 20, despliega junit.framework, para ver los ficheros contenidos en el paquete.

Expande ahora el fichero TestCase.java. El Explorador muestra elementos específicos de Java dentro del fichero.

Clases, grado de privacidad y sus miembros (métodos y atributos o campos) aparecen en el árbol.

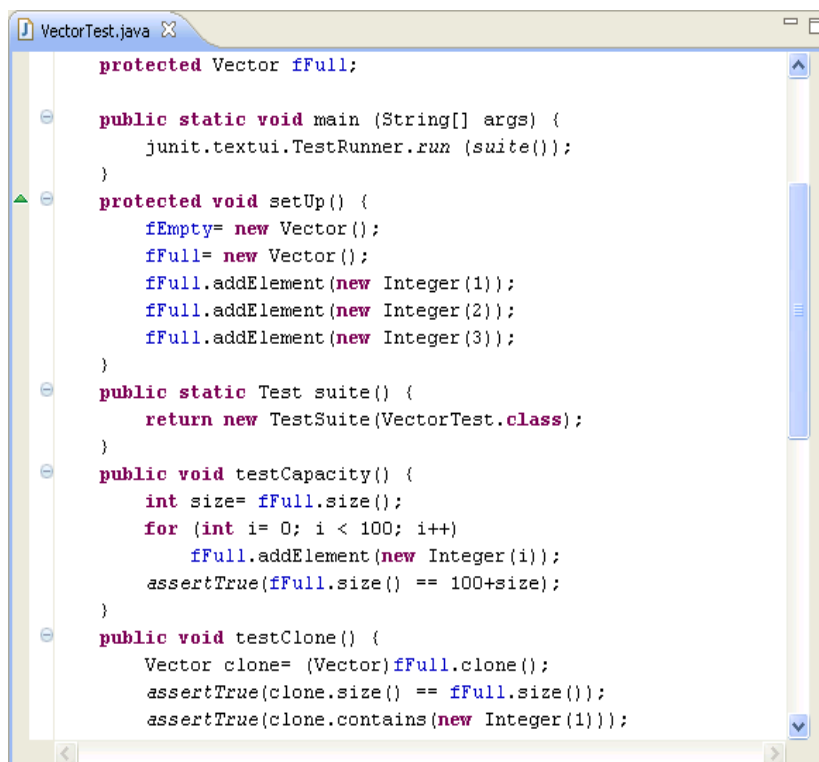
Cada elemento aparece precedido por un símbolo gráfico que identifica su grado de privacidad y el tipo de elemento de que se trata.



*Ilustración 21: Tipos de elementos*

### 3.4 Abriendo un editor de Java

Selecciona el paquete `junit.samples` y selecciona el fichero `VectorTest.java`. Puedes abrir `VectorTest.java` en el editor Java haciendo doble click en él. En general, hacer doble click sirve para abrir cualquier elemento.



*Ilustración 22: Coloreado de elementos según sintaxis*

Lo primero que se aprecia es el coloreado según sintaxis o Syntax Highlighting. Diferentes tipos de elementos en el código de Java se muestran en colores específicos. Ejemplos de elementos que se

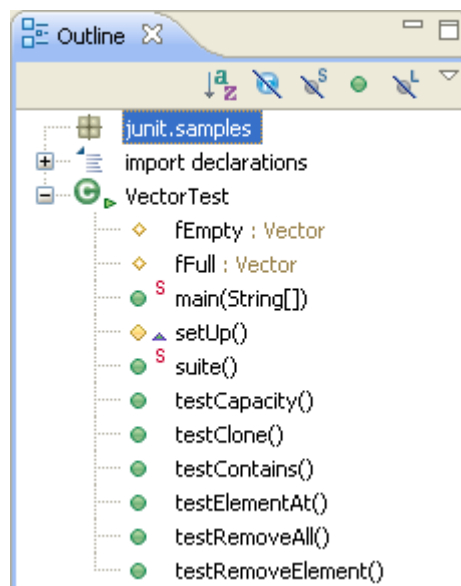
## UT05 - IDE Entornos de Desarrollo Integrado

colorean de modo diferente son:

- Comentarios comunes
- Comentarios Javadoc
- Palabras clave del lenguaje Java
- Strings

A la derecha aparece la vista Outline, de contenido similar al explorador de paquetes, muestra los elementos con una serie de símbolos que identifican su naturaleza.

Se muestra, por ejemplo, si un elemento Java es static (**S**), abstract (**A**), o final (**F**).



*Ilustración 23: Vista Outline*

Otros iconos como muestran si un elemento sobrescribe (overrides) un método de una clase base, o cuando implementa un método de un interface.

En la imagen, el método `countTestCases()` implementa el método del mismo nombre definido en el interfaz `Test`. El símbolo del triángulo vacío lo indica así en la vista Outline.



## UT05 - IDE Entornos de Desarrollo Integrado

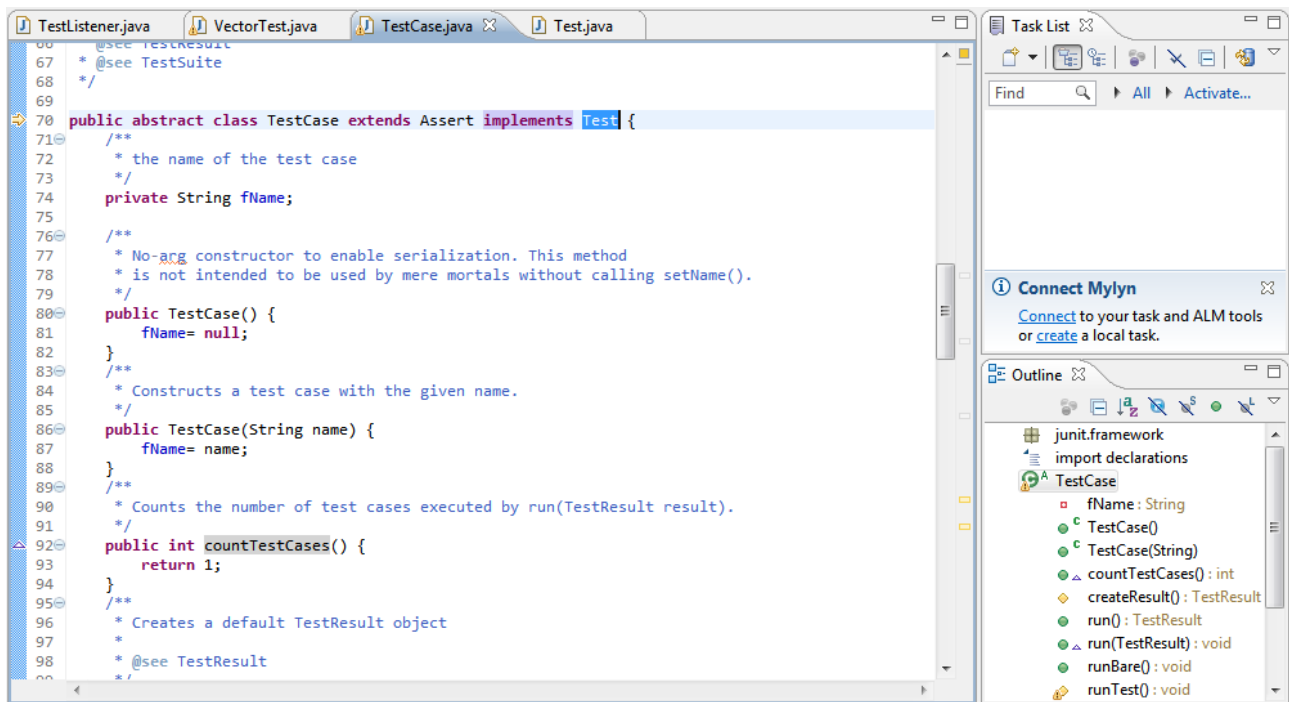


Ilustración 24: Implementación de un método de la Interfaz “Test”

El símbolo “overrides” es el mismo triángulo relleno.

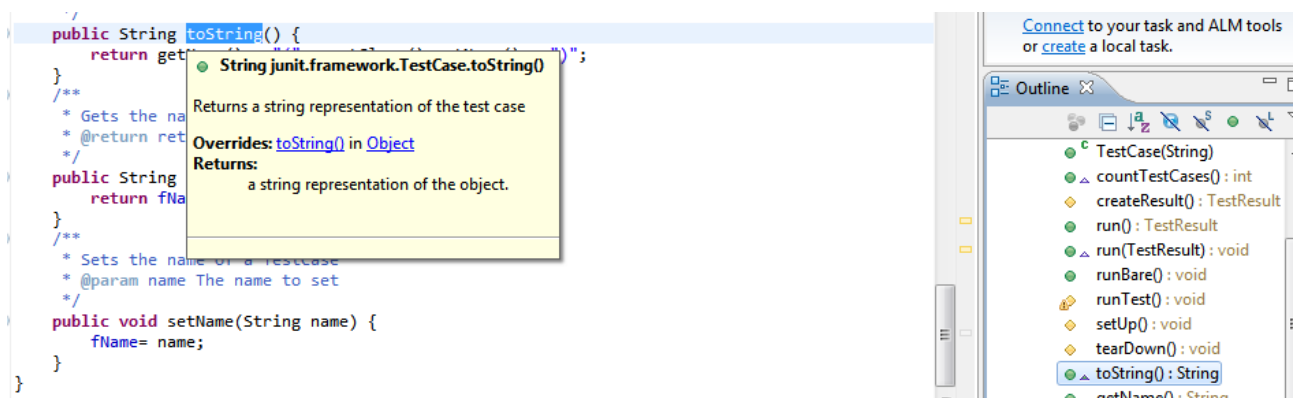


Ilustración 25: Sobreescribe (overrides)

Si se pincha en el editor, Outline muestra en qué elemento nos encontramos.

Los botones **Hide Fields**, **Hide Static Members**, y **Hide Non-Public Members** muestran u ocultan ciertos elementos de la vista Outline.

Muestra todo y comprueba que, pinchando en un elemento de la vista Outline, se selecciona ese elemento en el editor.

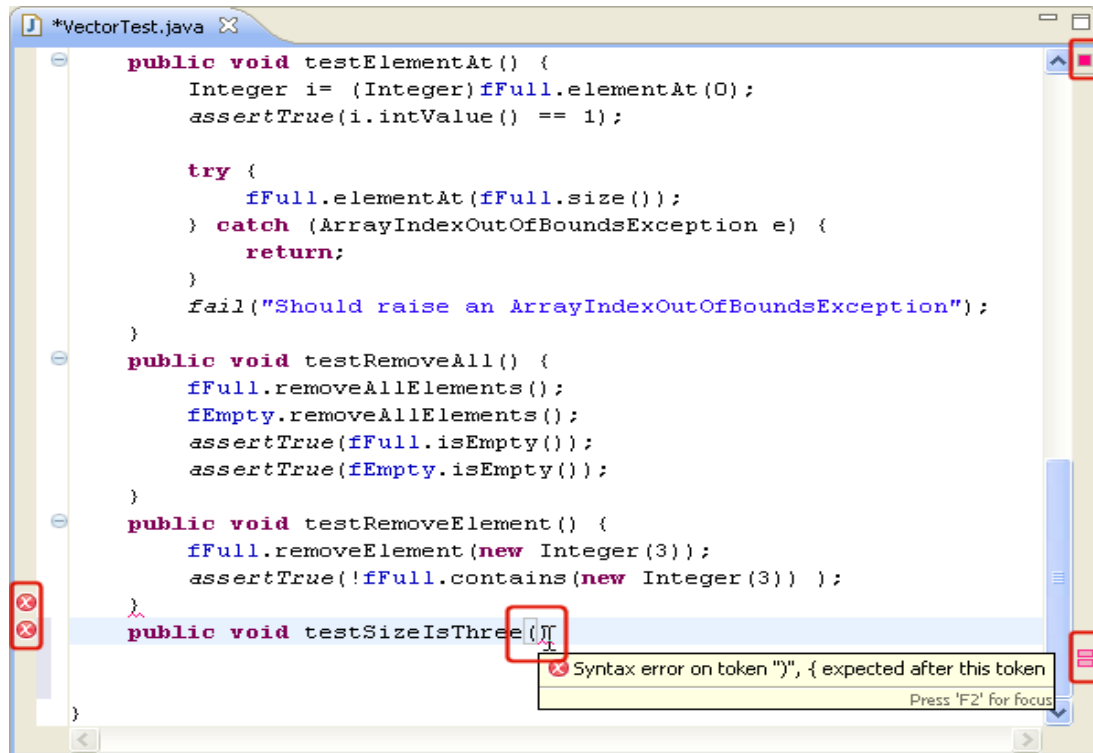
Presionando Ctrl-O o bien Navigate → Quick Outline, muestra una ventana similar sobre el editor Java. Volviendo a presionar Ctrl-O se muestran todos los campos heredados.

### 3.5 Añadiendo un nuevo método

Vamos a añadir un nuevo método al final del archivo *VectorTest.java*.

## UT05 - IDE Entornos de Desarrollo Integrado

Teclea `public void testSizeIsThree()` antes de la última llave.



*Ilustración 26: Añadiendo un método*

Mientras tecleas, verás que lo tecleado es interpretado como erróneo, con las siguientes “consecuencias visuales”:

- Iconos de error aparecen en rojo a la izquierda y comentarios de error a la derecha del editor de Java.
- También se marca en rojo en el texto el punto exacto donde se produce el error con un subrayado en zigzag.
- Un cuadrado rojo se activa en la esquina superior derecha de la vista del editor.

Al pasar el ratón por esos puntos, se muestra información relativa a los errores.

Si no se desea que se muestren estos errores al teclear, se debe configurar en Preferencias → Java → Editor, desmarcando **Report problems as you type**.

Si se graba el archivo, el código es compilado de modo automático y los errores se reflejarán también en el explorador de proyectos en todos los elementos que lo contienen.

## UT05 - IDE Entornos de Desarrollo Integrado

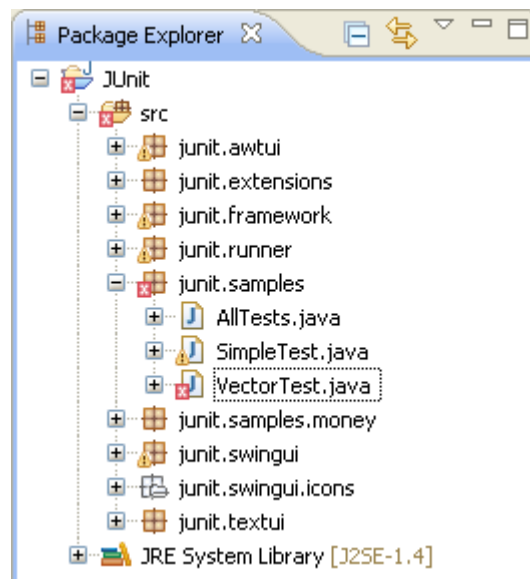


Ilustración 27: Proyecto con errores

Si se añaden las llaves y un comentario, al grabar los errores desaparecen.

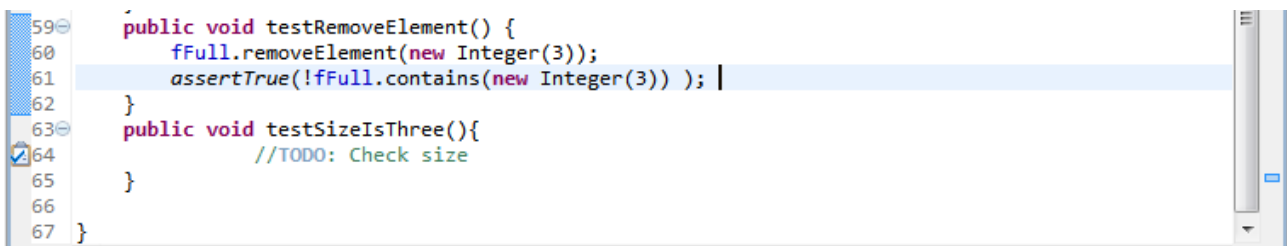


Ilustración 28: Nuevo método finalizado

### 3.6 Usando el asistente de contenido

Abre `junit.samples/VectorTest.java` y selecciona el método `testSizeIsThree()` en la vista Outline.

Reemplaza el comentario `//TODO` por las líneas:

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

Con el cursor al final de la palabra `assert`, pulsa `Ctrl+Espacio` para activar el asistente de contenido. Se listarán las propuestas del asistente para completar el código inacabado. Se puede seguir tecleando código, y el asistente reducirá la lista de opciones. Un click sencillo en la lista nos abre la ayuda Javadoc para ese ítem.

Selecciona `assertTrue(boolean)` fde la lista y pulsa `Enter`. El código `assertTrue(boolean)` se inserta.

Completa la línea como sigue:

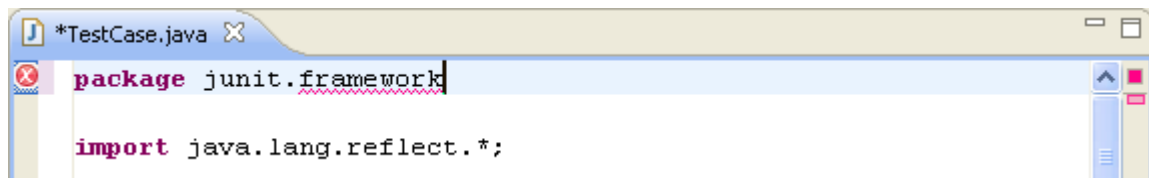
```
assertTrue(v.size() == fFull.size());
```

Graba el archivo.

## 3.7 Identificando problemas en el código

Abre `junit.framework.TestCase.java`

Borra el punto y coma al final de la primera línea:

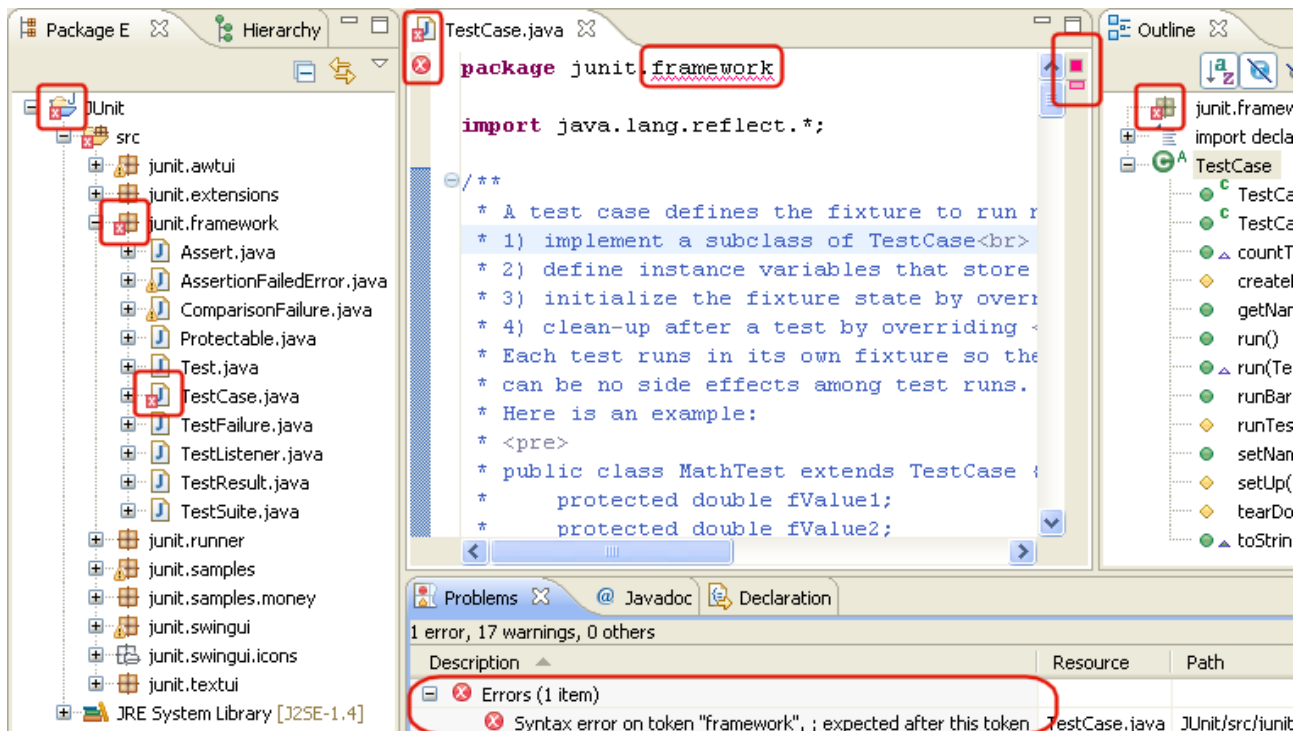


```
package junit.framework;

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

*Ilustración 29: Borrando el punto y coma*

Al guardar, se muestran los errores con símbolos visuales, como los vistos antes.



*Ilustración 30: Errores y símbolos*

Observa que en la vista “Problems” se muestra el error.

Cierra el editor.

Desde la vista Problems es posible dirigirse al punto señalado haciendo doble click en la línea correspondiente o eligiendo “Go to” en el menú de contexto.

Corrige el problema anterior y graba.

Ahora, desde la vista Outline ve a `getName()`. Cambia el código de la línea

```
return fName;
```

por

```
return fTestName;
```

Aparecerá un icono a la derecha con una bombilla. Eso significa que el editor tiene algunas propuestas para arreglar el error. Pula el icono de la bombilla y aparecerán las opciones propuestas.

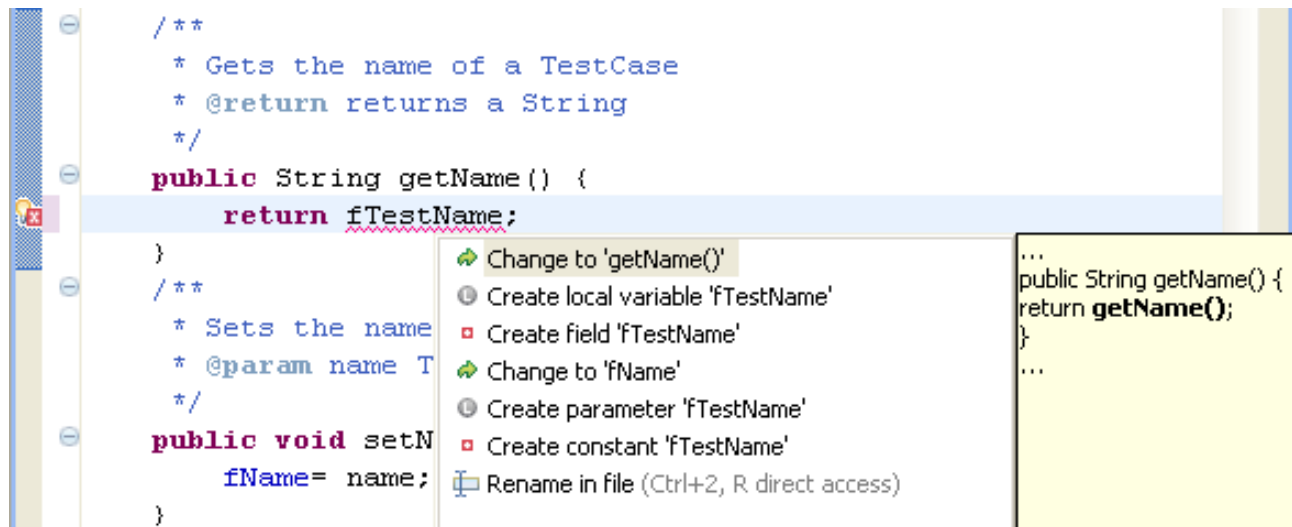


Ilustración 31: Opciones de solución de errores

Para reparar el error, pulsa Change to fName.

Puede configurarse cómo se indican los errores en General → Editors → Text Editors → Annotations de las preferencias.

### 3.8 Usando plantillas de código

Veremos cómo usar una plantilla de código para un bucle sencillo.

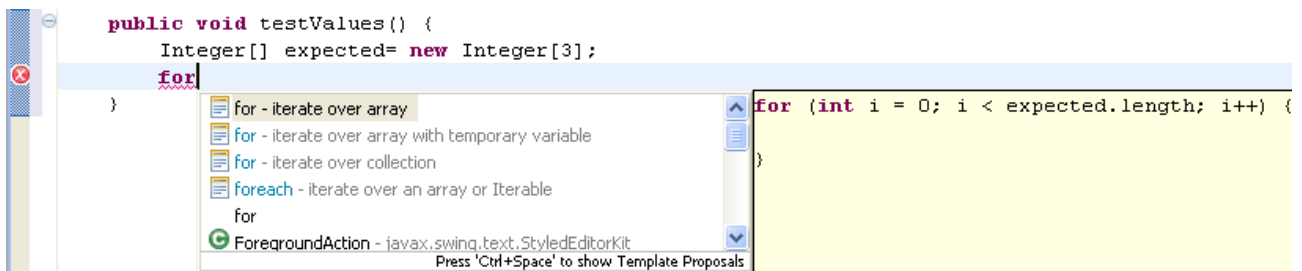
Abre `junit.samples.VectorTest.java`.

Inicia un nuevo método tecleando:

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

Con el cursor al final del `for`, pulsa CTRL+Espace para el asistente de contenido. Verás una lista de plantillas para bucles `for`. Al seleccionar una de las opciones, se mostrará el código que se insertará si se elige esa opción. Para recorridos sobre arrays, El nombre del array se adivina de modo automático.

## UT05 - IDE Entornos de Desarrollo Integrado



*Ilustración 32: plantilla para for*

Elige `for - iterate over array` y pulsa Enter (o haz doble click). La plantilla se inserta en tu código.

A continuación cambiaremos el nombre de la variable de iteración. Verás que ésta queda seleccionada automáticamente tras insertar el código de la plantilla. Pulsa la letra “e” para renombrar. Todas las referencias a “i” cambiarán por “e” de modo automático.

Pulsando “tab” la selección se mueve a la siguiente variable de la plantilla, que puede ser editado o ignorado pulsando nuevamente “tab”.

Completa el bucle como sigue:

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

Usa el editor de contenido para completar el código por `toArray - convert collection to array` y navega por la plantilla para que quede:

```
Integer[] actual= (Integer[]) fFull.toArray(new Integer[fFull.size()])
```

Luego añade punto y coma y el código siguiente:

```
assertEquals(expected.length, actual.length);  
for (int i= 0; i < actual.length; i++)  
    assertEquals(expected[i], actual[i]);
```

Guarda el archivo.

### 3.9 Usando la historia local

Es posible restaurar un cambio hecho en un método desde la vista Outline.

Haz un cambio en un método de un archivo.

Graba el archivo.

Selecciona el método en la vista Outline y en el menú contextual pulsa **Replace With > Element from Local History**. Haz doble click en el elemento en cuestión.

Se mostrará un editor de comparación en el que se pueden ver las diferencias. Si deseas reemplazarlo, pulsa esa opción.

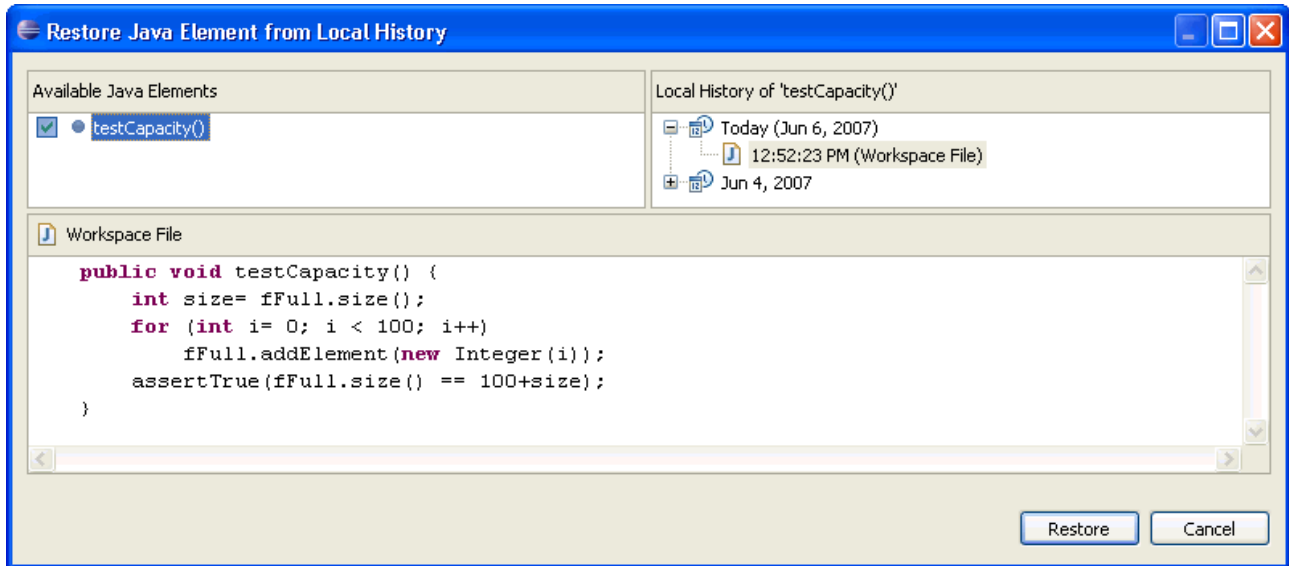
También es posible restaurar elementos (métodos o atributos) borrados.

## UT05 - IDE Entornos de Desarrollo Integrado

Borra un método completo en la vista Outline.

En el menú de contexto de la vista Outline **selecciona la clase** y de su menú de contexto elige **Restore from Local History...**

Aparecerá una ventana como la siguiente. Podrás elegir el código que se restaurará eligiéndolo y pulsando Restore.



### 3.10 Extrayendo un nuevo método

En este apartado mejoraremos el código de un constructor.

Abre en *junit.framework*, el archivo *TestSuite.java* en el editor

Lo que vamos a hacer es crear un nuevo método a partir de un bloque de código existente de modo automático.

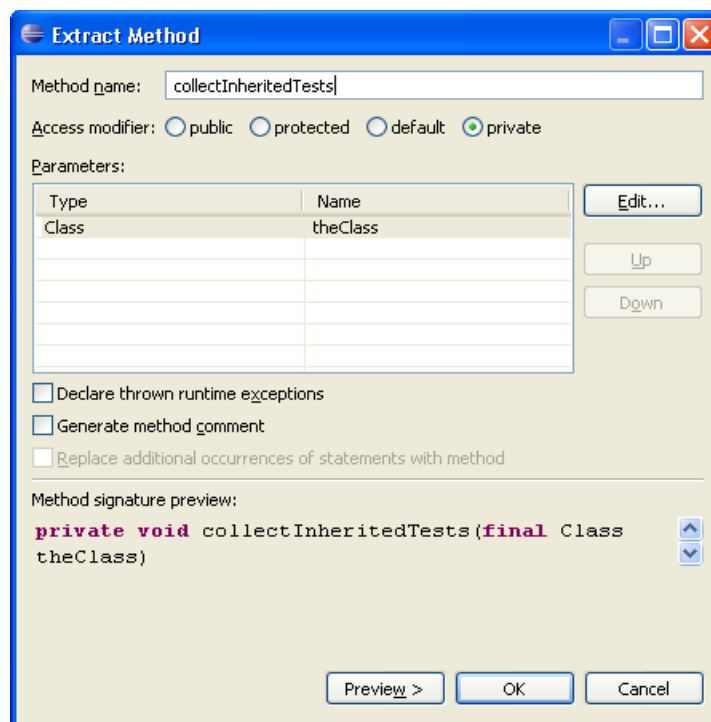
El nuevo método se llamará *collectInheritedTests*.

Selecciona el código siguiente:

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

Un vez seleccionado, en el menú de contexto pulsa **Refactor > Extract Method**

Se abrirá un diálogo como el siguiente:



*Ilustración 33: Extrayendo un método*

Para anticipar qué va a ocurrir, puedes ver el código resultante pulsando preview:



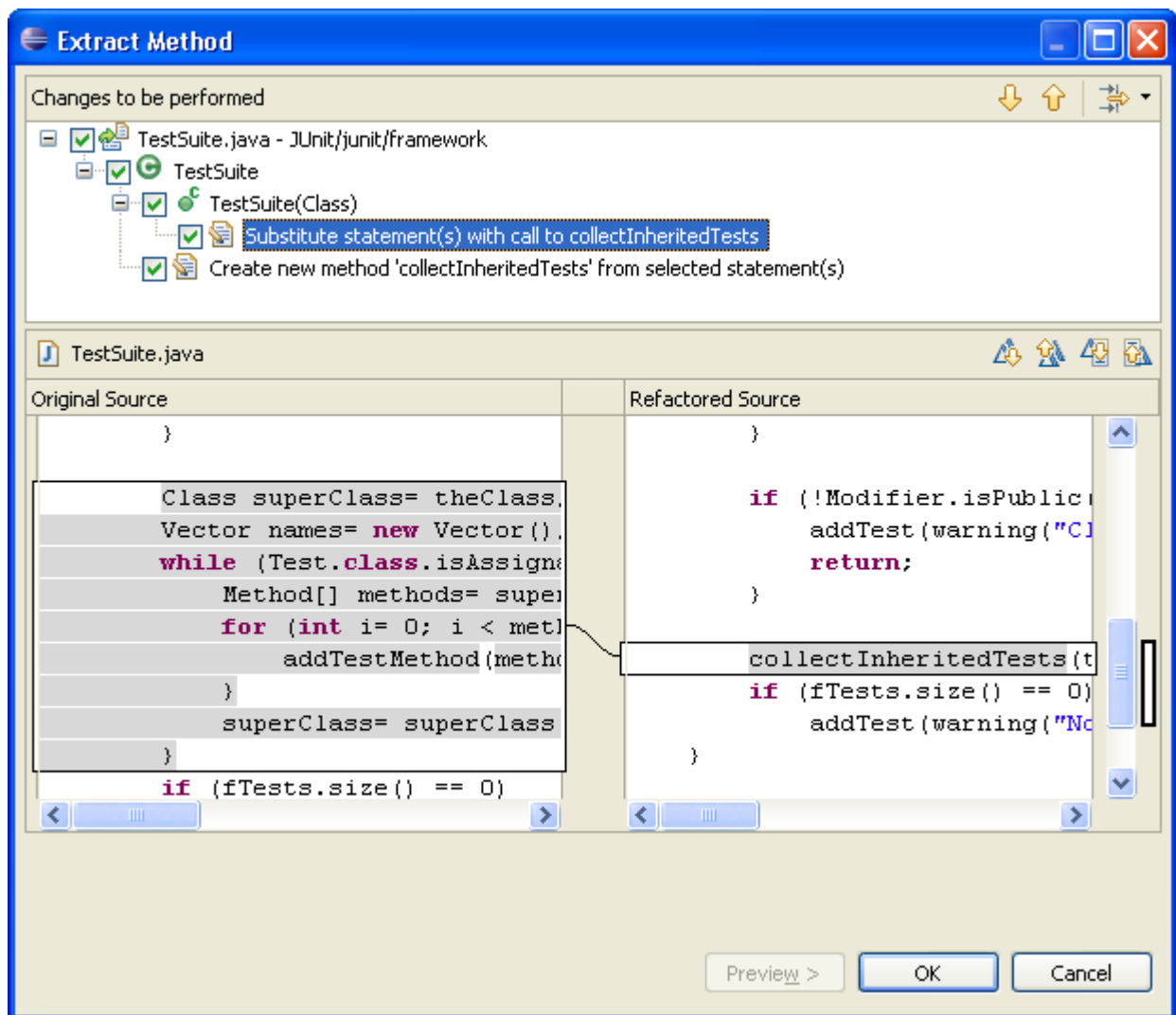


Ilustración 34: preview

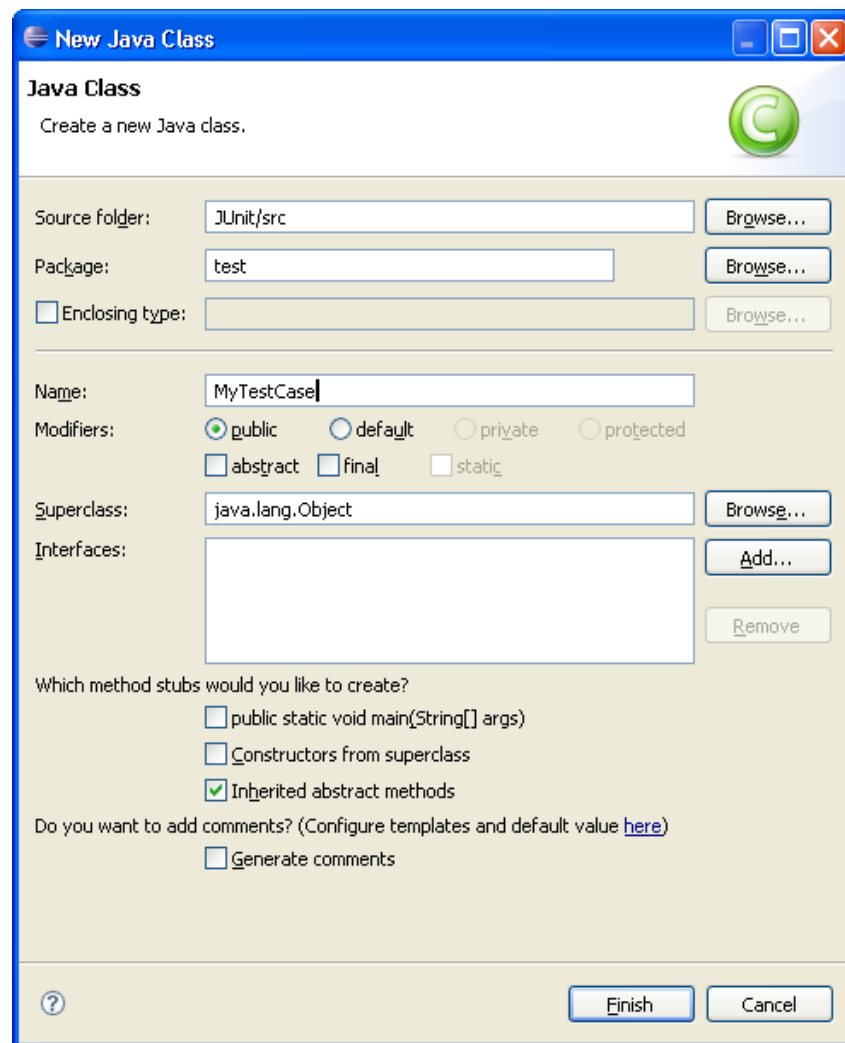
### 3.11 Creando una clase de Java

En la vista del Explorador de Paquetes, selecciona el proyecto JUnit. Haz click en el botón **New Java Package** de la barra de herramientas, o selecciona **New > Package** del menú de contexto del proyecto.

En el campo **Name**, teclea *test* con nombre del nuevo paquete. Luego **Finish**.

Pulsa el botón **New Java Class** de la barra de herramientas.

Asegúrate de que *JUnit* aparece en el campo **Source Folder** y de que *test* aparece en el campo **Package**. En el campo **Name**, teclea *MyTestCase*.

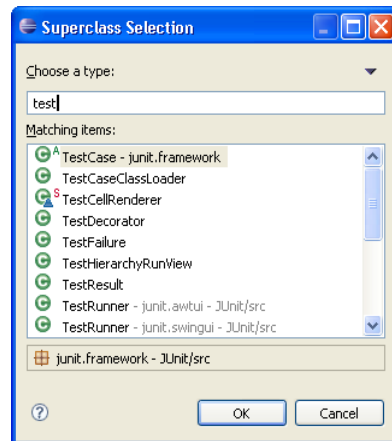


*Ilustración 35: crear clase MyTestCase*

Pulsa el botón Browse cerca del campo Superclass.

Se abrirá un diálogo como el siguiente:

## UT05 - IDE Entornos de Desarrollo Integrado

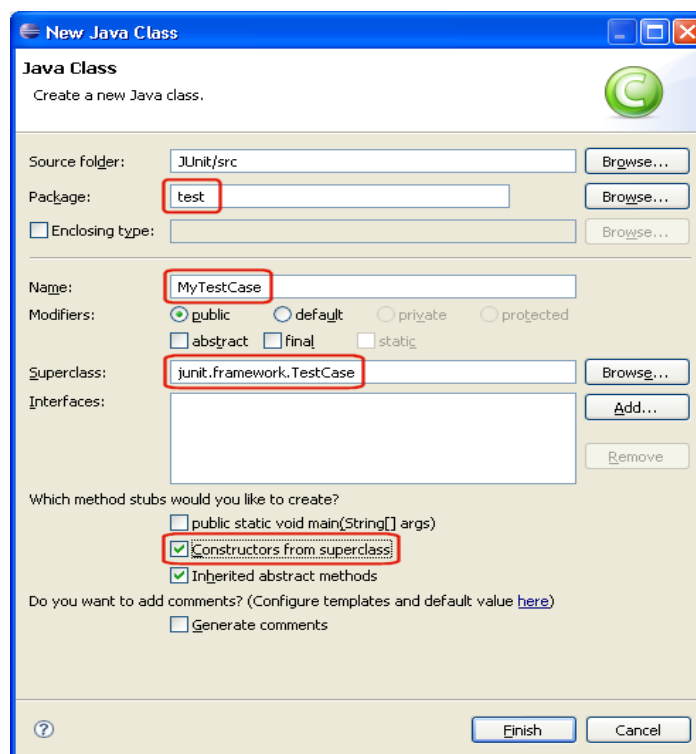


*Ilustración 36: Elegir superclase*

En el campo **Choose a type** escribe test, para reducir la lista de casos de superclases disponibles.

Selecciona *TestCase* class y pulsa **OK**

Marca el checkbox **Constructors from superclass**



*Ilustración 37: diálogo para crear la clase relleno*

Pulsa **Finish**.

Se generará una nueva clase con una plantilla de código creada.

## UT05 - IDE Entornos de Desarrollo Integrado

```
package test;

import junit.framework.TestCase;

public class MyTestCase extends TestCase {

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

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

}
```

Ahora crearemos algunos métodos desde la superclase. Selecciona la clase en la vista Outlook. En el menú contextual, elige Source → Override/implement Methods

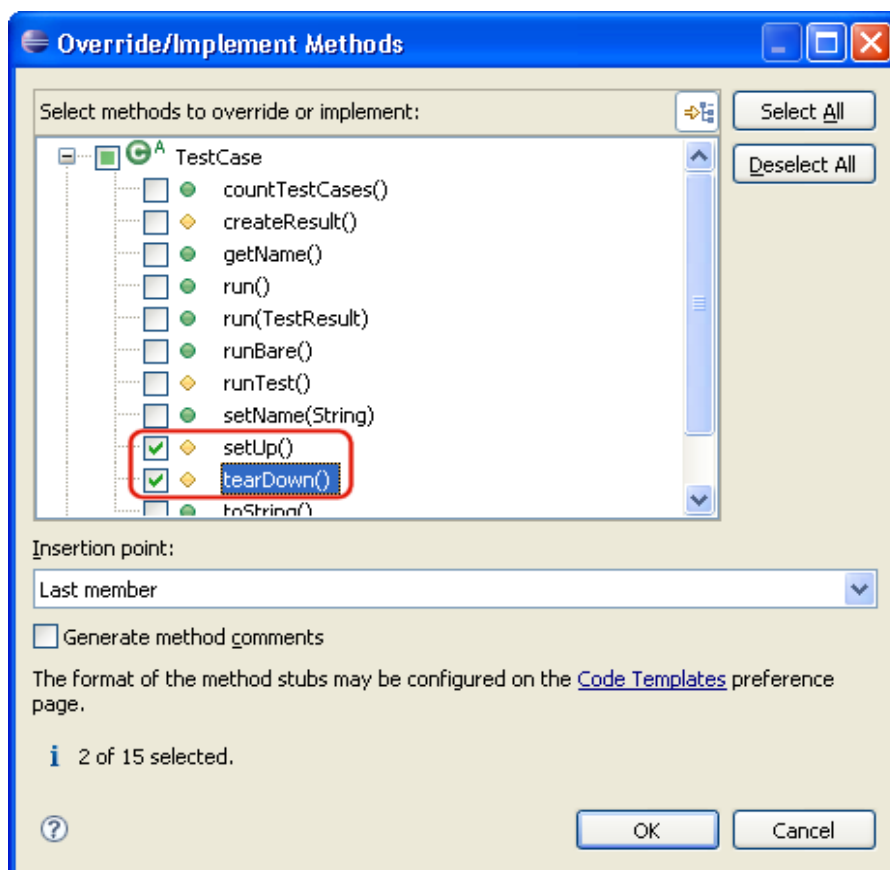


Ilustración 38: Diálogo Override/Implement

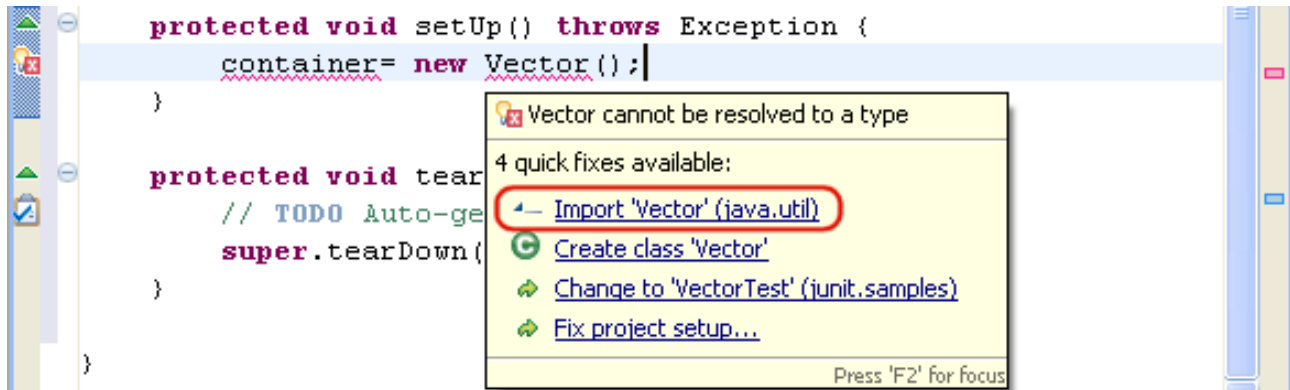
Marca los métodos setUp y tearDown. Pulsa OK. Los dos métodos se añadirán a la clase.

Cambia el cuerpo de setUp() y escribe en él

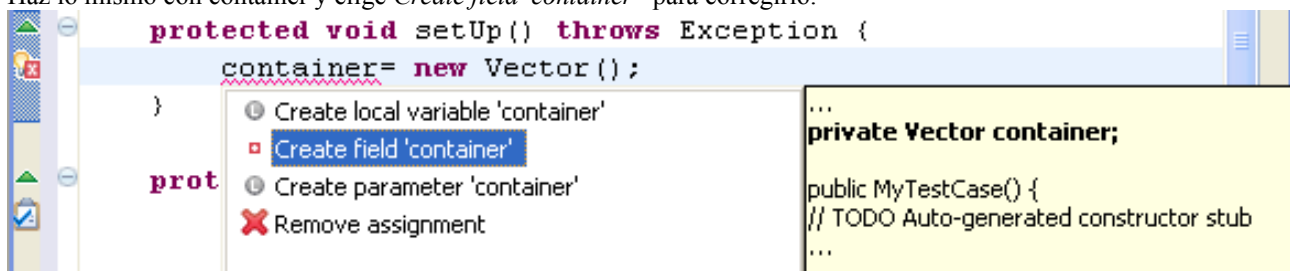
## UT05 - IDE Entornos de Desarrollo Integrado

```
container= new Vector();
```

Aparecen algunos errores. Pone el cursor sobre Vector. Un mensaje aparece, mostrando el mensaje de error y posibles soluciones. Elige *Import 'Vector' (java.util)*. Se añadirá la línea de import en el código, y el error se corregirá.



Haz lo mismo con container y elige *Create field 'container'* para corregirlo.



Más arriba en el código aparecerá la línea con la definición:

```
private Vector container;
```

También puede usarse la combinación Ctrl+1, que te lleva al siguiente error y propone soluciones.

Ahora crearemos de modo automático dos métodos típicos para dar valor a un atributo y para leer el valor de un atributo.

Selecciona la clase en Outliner. Elige del menú de contexto **Source > Generate Getters and Setters**

Se nos sugiere crear getContainer y setContainer. Selecciona ambos y pulsa OK. Se crearán, al final del código, los métodos:<sup>1</sup>

```
public Vector getContainer() {
    return container;
}

public void setContainer(Vector container) {
    this.container = container;
}
```

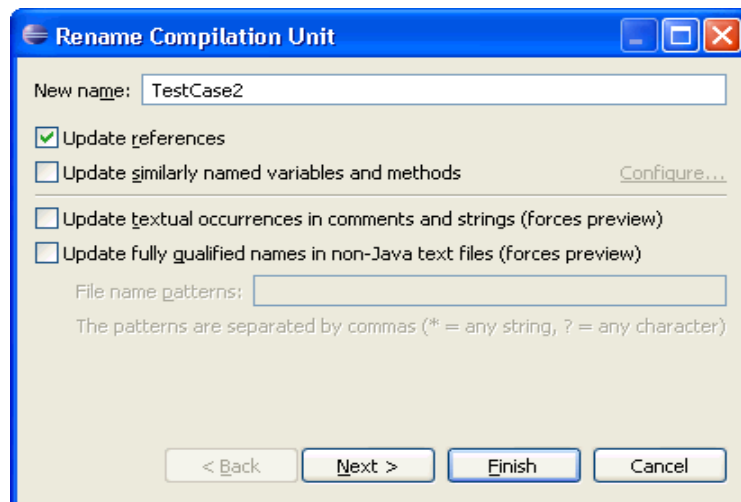
<sup>1</sup>El formato del código generado puede configurarse en Java → CodeStyle → Formatter y en Java → Code Style puede ponerse prefijos o sufijos para los campos (atributos). Por ejemplo "cSexo".

### 3.12 Renombrando elementos Java

Usaremos la refactorización. Las acciones de refactorización (Refactoring) cambian la estructura del código sin afectar a su semántica (comportamiento)

En el explorador de Paquetes, seleccionamos *junit.framework.TestCase.java*

En el menú de contexto, Refactor → Rename

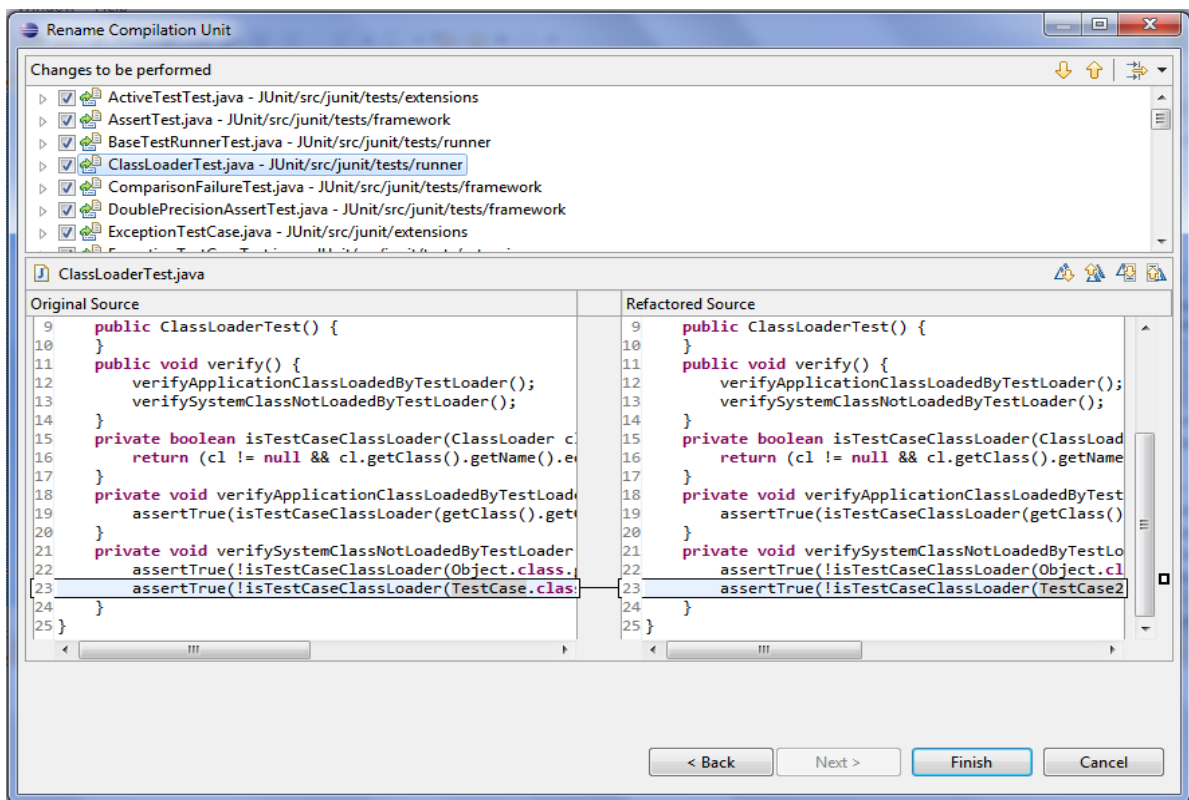


*Ilustración 39: refactorizando*

En el campo **New Name** teclea "*TestCase2*".

Pulsa Next para ver los posibles cambios que se generan.

Obtenemos una pantalla de comparación de código y la lista de ficheros afectados. Puede navegarse para ver en cada caso qué cambios van a resultar.



*Ilustración 40: comparando cambios*

Se pueden deseleccionar cambios si se desea, aunque lo típico es aceptar todos, pulsando **Finish**.

Puede deshacerse todo lo hecho en **Edit > Undo Rename Compilation Unit**

### 3.13 Moviendo y copiando elementos Java

Desde el explorador de paquetes, seleccionemos el archivo con la clase que creamos antes, *MyTestCase.java* y arrastremosla a *junit.samples*. Esta acción es similar a hacer **Refactor > Move** desde el menú de contexto. Mover el archivo exigirá rehacer las referencias, por ejemplo los import o las sentencias package.

Edit → Undo Move deshace el cambio.

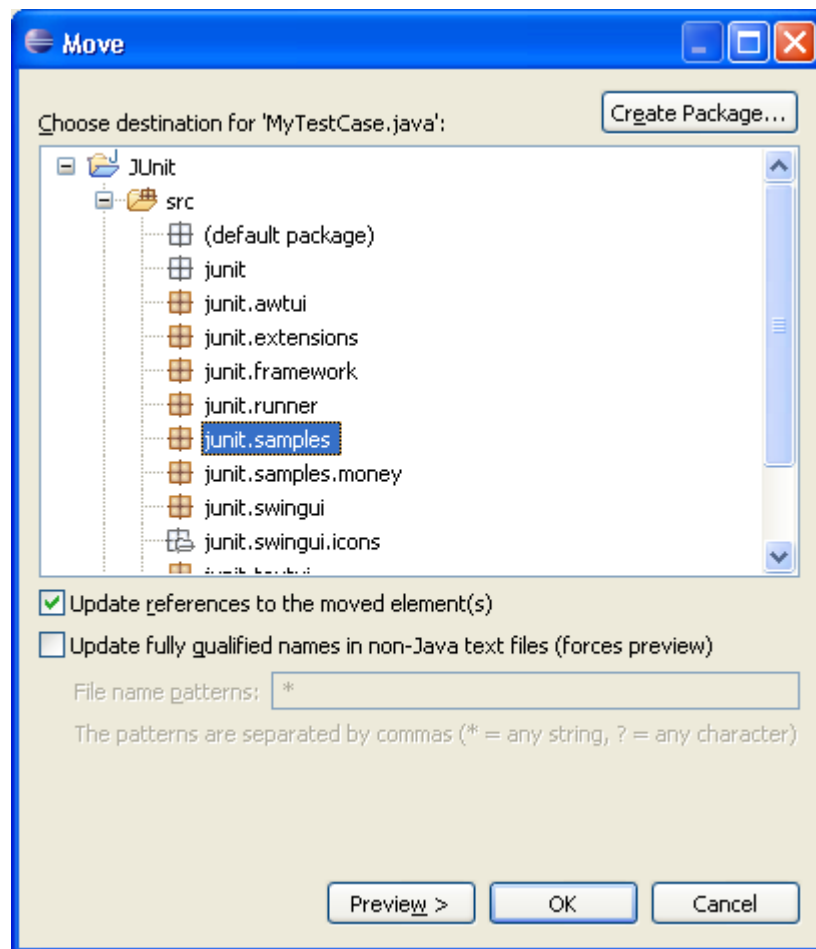
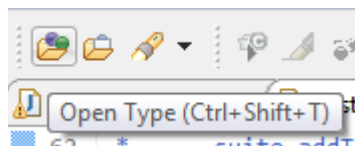


Ilustración 41: Refactor → Move

### 3.14 Navegar a las declaraciones Java de un elemento

Pulsando **Ctrl+Shift+T** se abre el diálogo llamado **Open Type**. También eligiendo **Navigate** → **Open Type** o haciendo click en su icono de la barra de herramientas.



Teclea `Money` y luego selecciona `MoneyTest` y pulsa **Enter**, lo que abrirá el “tipo” en el editor de Java.

Si en la primera línea seleccionamos la superclase `TestCase`, podemos ir a su declaración (la de la superclase) de varias maneras:

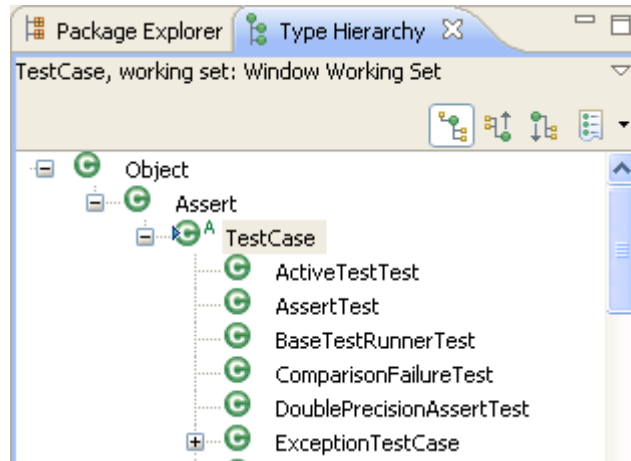
- **Navigate > Open Declaration**
- **F3**
- Seleccionando `TestCase` y eligiendo la opción del Menú contextual **Open Declaration**

Este comando funciona igual en métodos y campos (atributos).



### 3.15 Jerarquía de tipo

El comando **Navigate > Open Type Hierarchy** abre la jerarquía de clases de la que depende una clase dada. Prueba a hacerlo en la clase `TestCase`.



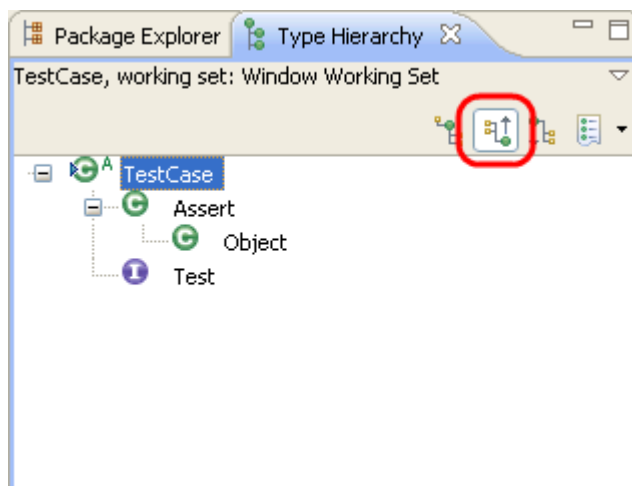
*Ilustración 42: Jerarquía de tipo*

Puede ejecutarse también seleccionando el fichero en el explorador de paquetes o seleccionando el nombre de clase en el código y pulsando F4 o eligiendo la opción del menú de contexto.

La clase para la que se abrió la jerarquía se marca con una flechita verde delante del icono.

Los botones de la vista permiten distintas opciones:

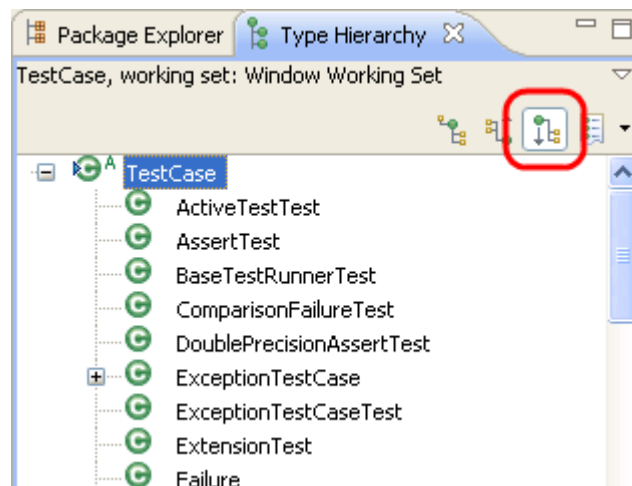
1. **Show the Type Hierarchy** para ver la jerarquía de clases, incluyendo las clases base y las subclases. La imagen anterior lo muestra.
2. **Show the Supertype Hierarchy** muestra los elementos del padre del tipo en cuestión, incluyendo los interfaces implementados.



*Ilustración 43: Jerarquía de supertipo de TestCase*

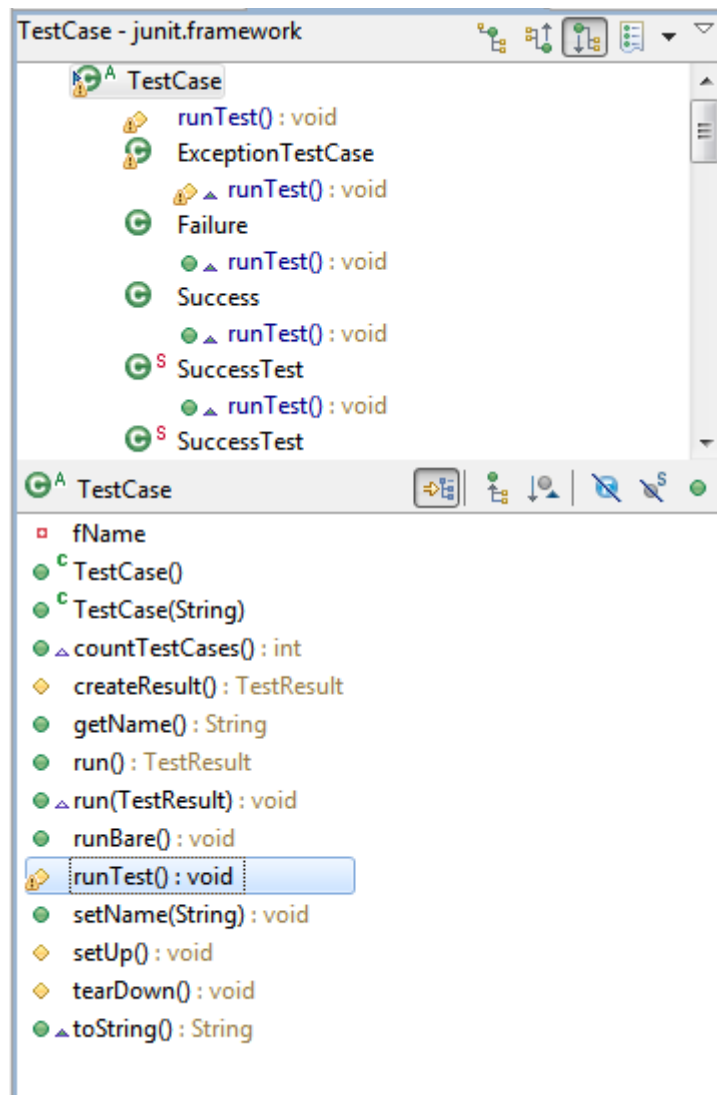
3. **Show the Subtype Hierarchy** solo muestra los subtipos del tipo seleccionado.

## UT05 - IDE Entornos de Desarrollo Integrado



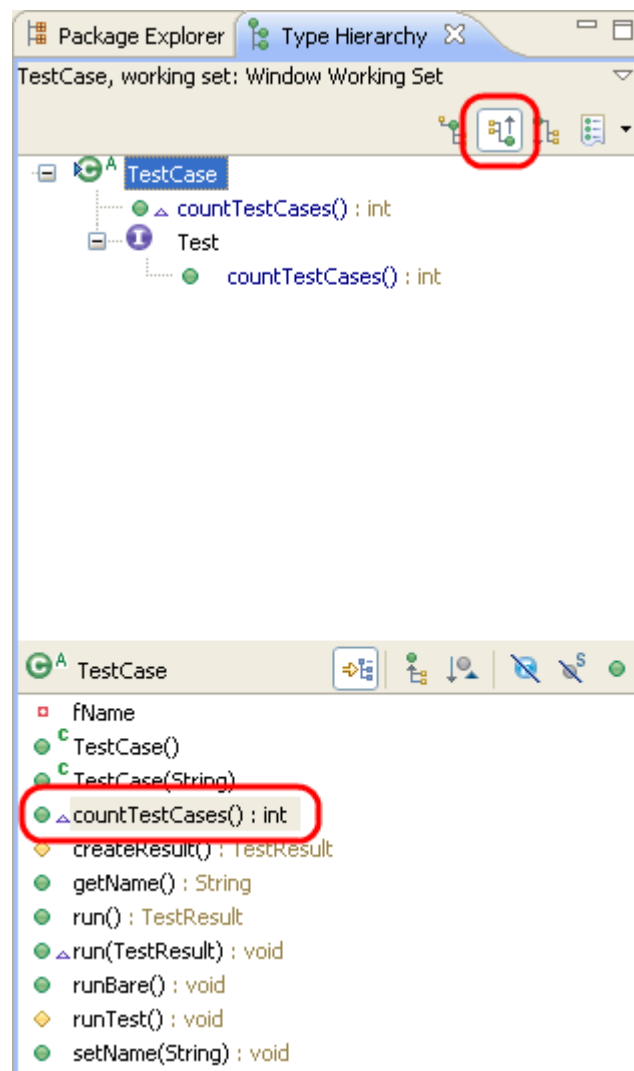
*Ilustración 44: Jerarquía de subtipo*

4. Si se pulsa el botón “Lock View and **Show Members in Hierarchy**” y luego se selecciona, por ejemplo, el método `runTest()` en la parte inferior, en la superior se visualizarán todos los lugares donde se implementa.



*Ilustración 45: Muestra miembros en la jerarquía.*

- Si se pulsa **Show the Supertype Hierarchy** y luego, en el panel de miembros (el inferior) se selecciona `countTestCases()`, se mostrará todos los sitios donde es declarado.



*Ilustración 46: declaraciones*

6. Selecciona el interfaz Test y, en su menú de contexto, elige **Focus On 'Test'** . Se presentará Test en la vista de jerarquía de tipos.
7. Selecciona el paquete junit.framework en el explorador de paquetes. Abre **Open Type Hierarchy** desde su menú de contexto. Se abrirá una jerarquía con todas las clases del paquete. Las clases que se muestran en blanco, pertenecen a otros paquetes, pero es necesario incluirlas para mostrar toda la jerarquía.

## UT05 - IDE Entornos de Desarrollo Integrado

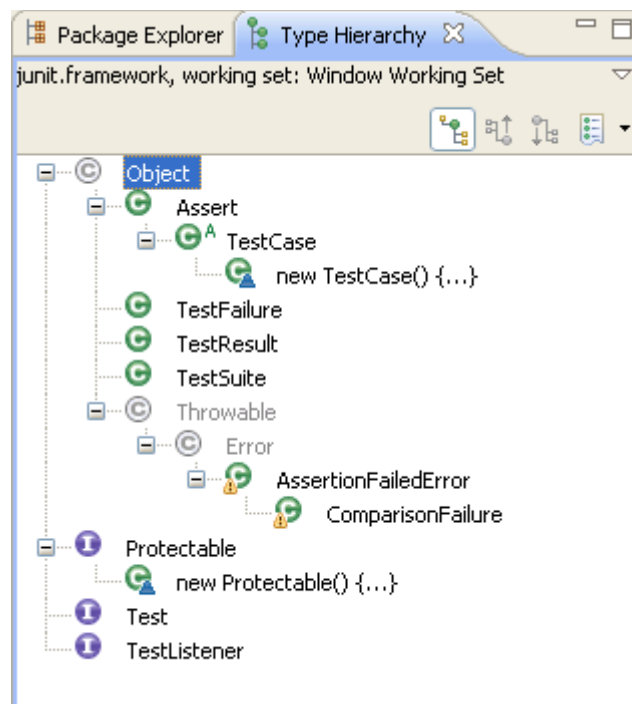


Ilustración 47: Clases de paquete framework

Usa jeraquías de tipo previas (**Previous Type Hierarchies**) para ir a un tipo elegido previamente. Haz click en la flecha junto al botón para ver una lista de elementos o haz click en el botón para editar la lista histórica.

Puede echarse un vistazo rápido a la jerarquía pulsando **Navigate → Quick Type Hierarchy (Ctrl+T)**. Pulsando repetidas veces Ctrl+T pasará de un tipo de vista a otro.

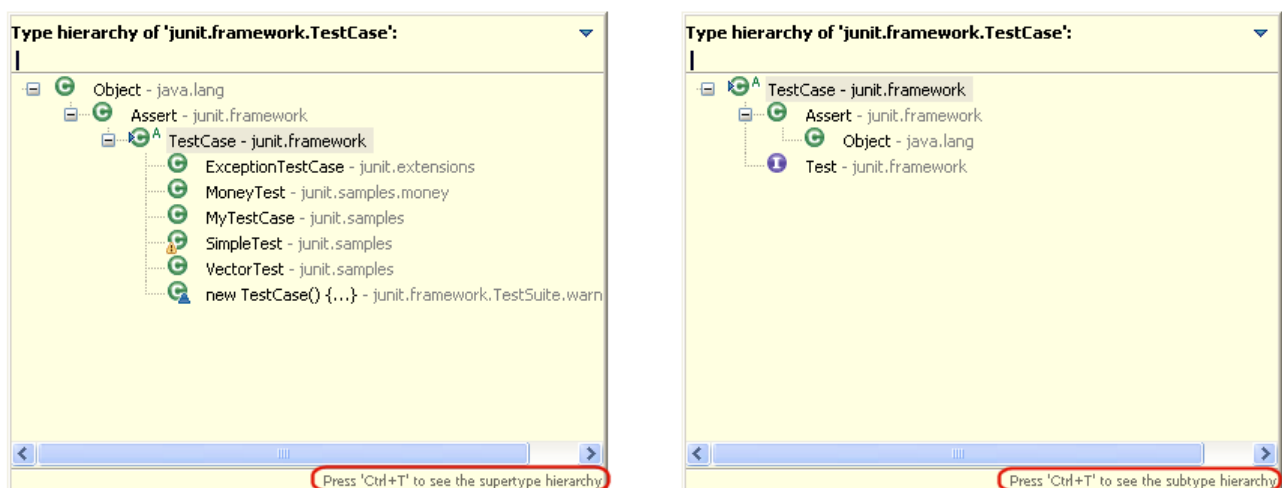


Ilustración 48: Vista rápida de la jerarquía

En TestCase.java busca el método runBare() y selecciona la llamada a setUp. Pulsa Ctrl+T. Se mostrarán todos los lugares donde se implementa o define TestCase.setUp().

## UT05 - IDE Entornos de Desarrollo Integrado

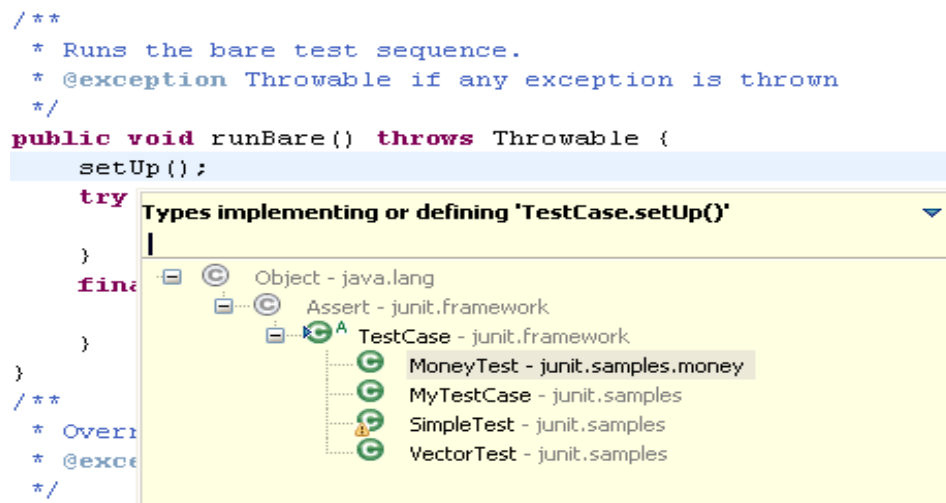


Ilustración 49: Buscar implementaciones o definiciones

### 3.16 Buscando en el entorno de trabajo

Vamos a aprender a buscar elementos Java en el entorno de trabajo.

En la barra de herramientas arranca la búsqueda desde su botón. Selecciona la pestaña de Java.

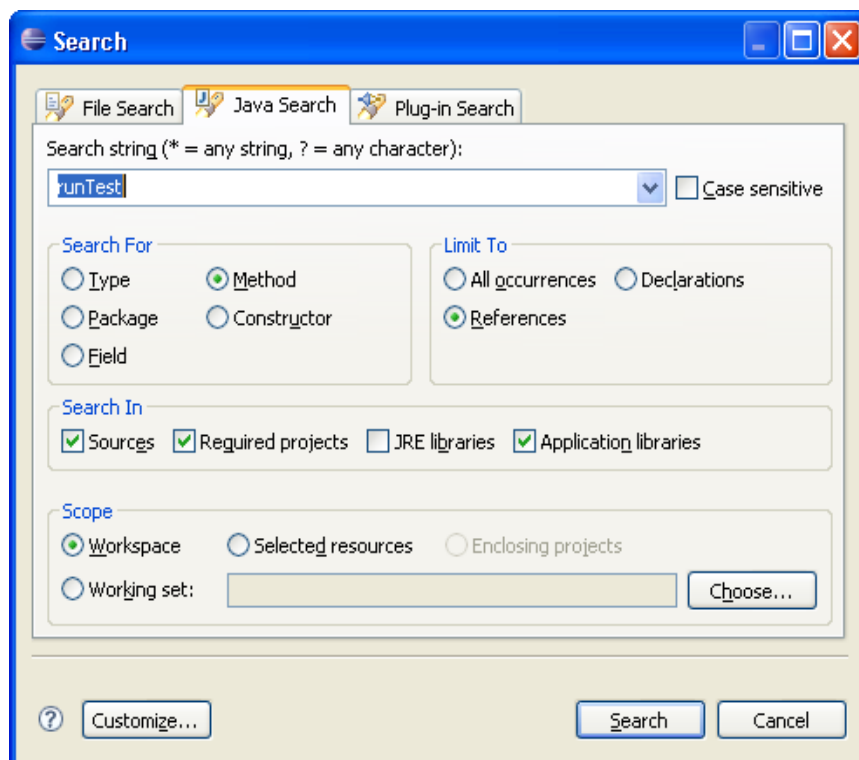
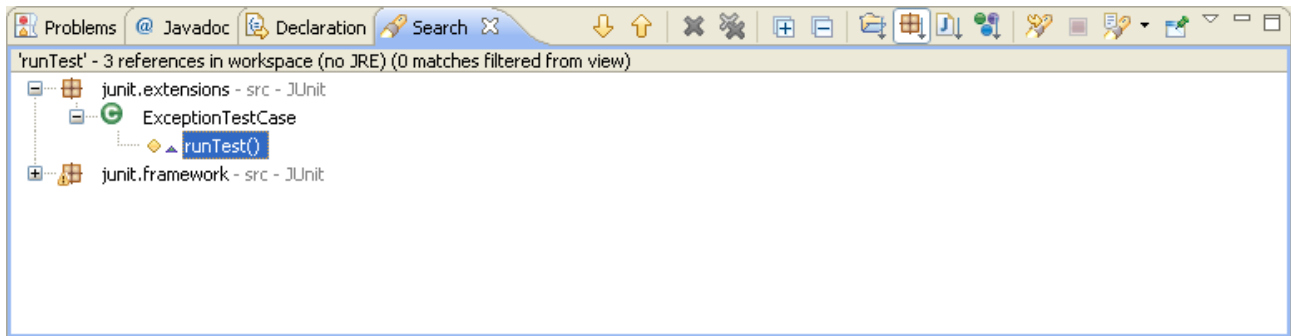


Ilustración 50: Búsqueda en Java

En el campo **Search string**, teclea `runTest`. En el área **Search For**, selecciona **Method**, y en **Limit To**, selecciona **References**. Verifica que el ámbito (Scope) es **Workspace**. Pulsa **Search**.

## UT05 - IDE Entornos de Desarrollo Integrado

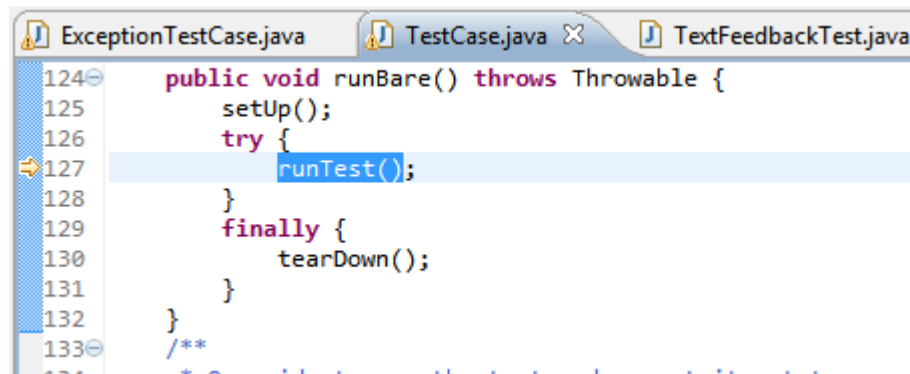
En la perspectiva Java, la vista de búsqueda muestra los resultados.



*Ilustración 51: Resultados de búsqueda runTest()*

Puede navegarse con los botones de flechas de la vista de búsqueda entre los resultados. Los ficheros donde aparece el método se muestran en el entorno de trabajo. También puede hacerse un doble click sobre ellos.

Una marca de búsqueda señala la posición del texto en los ficheros.



*Ilustración 52: marca de búsqueda de runTest()*

También puede hacerse una búsqueda en el explorador de proyectos o en la vista Outline.

Abre el archivo `junit.framework.Assert.java` desde el explorador de paquetes.

Selecciona en la vista Outline el elemento **fail(string)**.

Pulsa en el menú de contexto la opción **References** → **Workspace**. Se mostrarán todas las invocaciones del método **fail** en los archivos abiertos en el entorno de trabajo.

## UT05 - IDE Entornos de Desarrollo Integrado

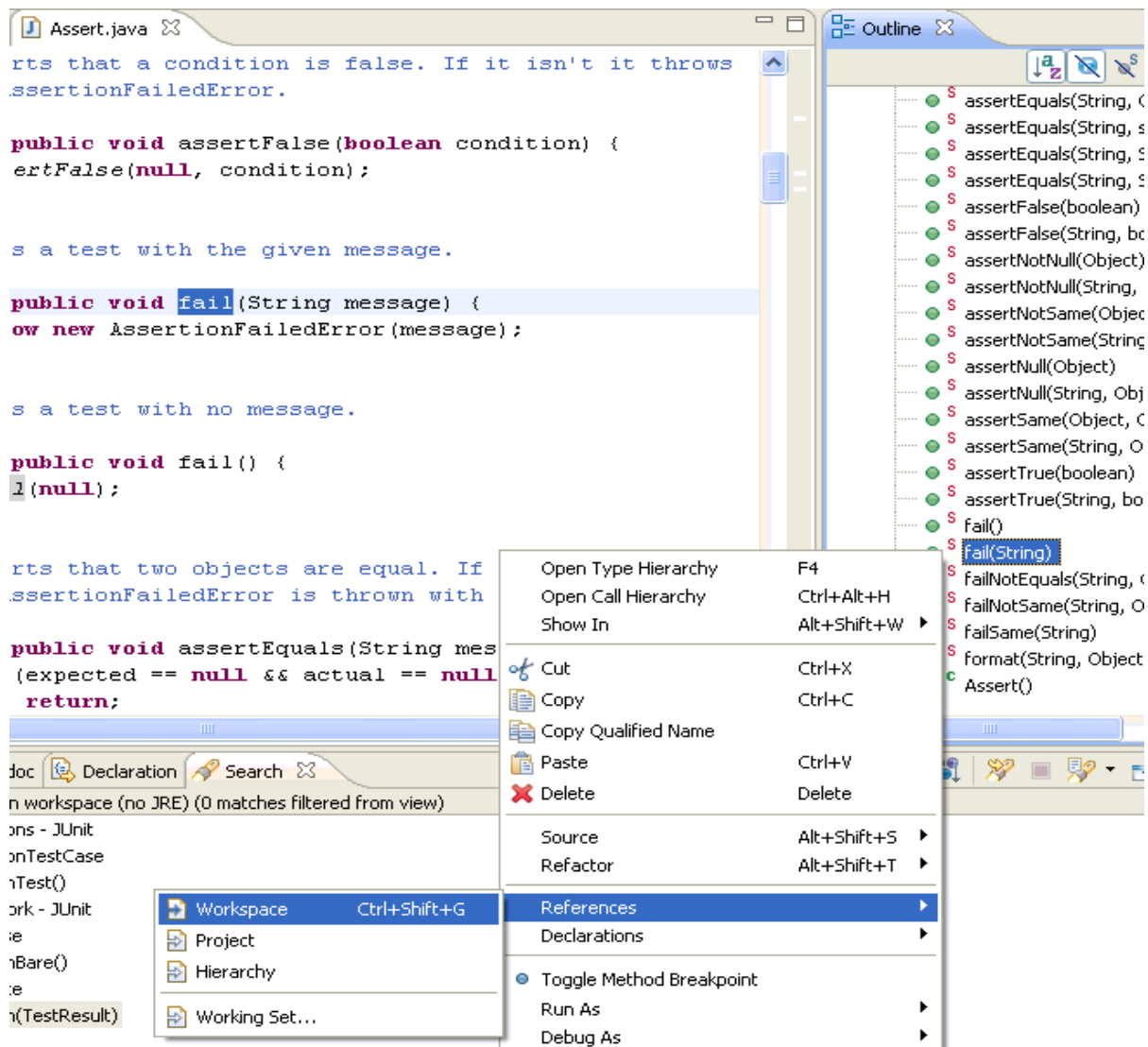


Ilustración 53: búsqueda desde Outline

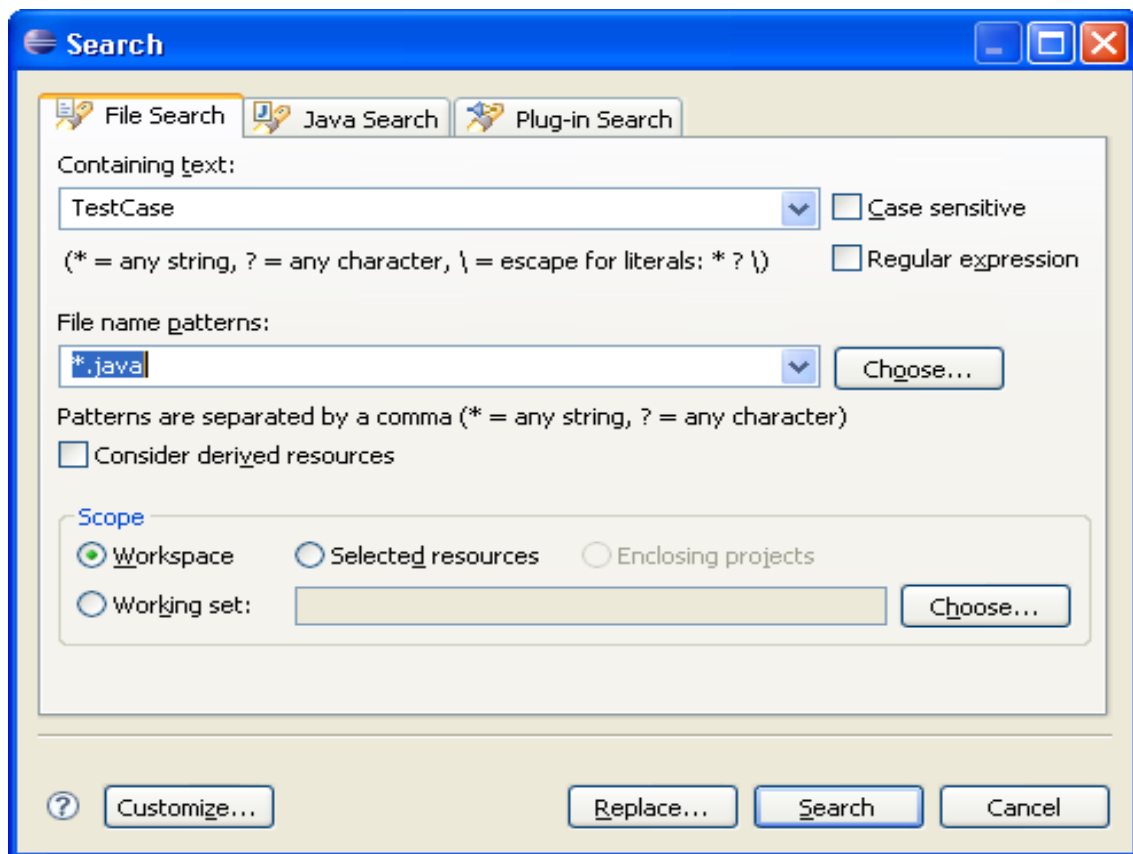
La opción References → workspace también está disponible seleccionando el nombre de un método en el editor y eligiendo esa opción desde el menú de contexto.

Las opciones de búsqueda de textos y archivos se encuentran en la primera pestaña File Search.

De ese modo puede buscarse un texto o determinados archivos que sigan un patrón o un determinado texto dentro de determinados archivos.

En la siguiente imagen se busca el texto `testCase`. Se buscará sólo en los archivos que terminan en `.java` y se encuentren en el entorno de trabajo.





*Ilustración 54: búsqueda de texto*

## Ejercicio 3

Explora las opciones de la barra de herramientas de la vista de búsqueda y documenta para cada icono su función.

## 3.17 Ejecutar programas Java

En la vista del Package Explorer, busca junit.textui/TestRunner.java y haz doble click para abrirlo en un editor.

En la vista de Outline, se ve que la clase TestRunner tiene un icono distinto, que indica que la misma posee un método main.

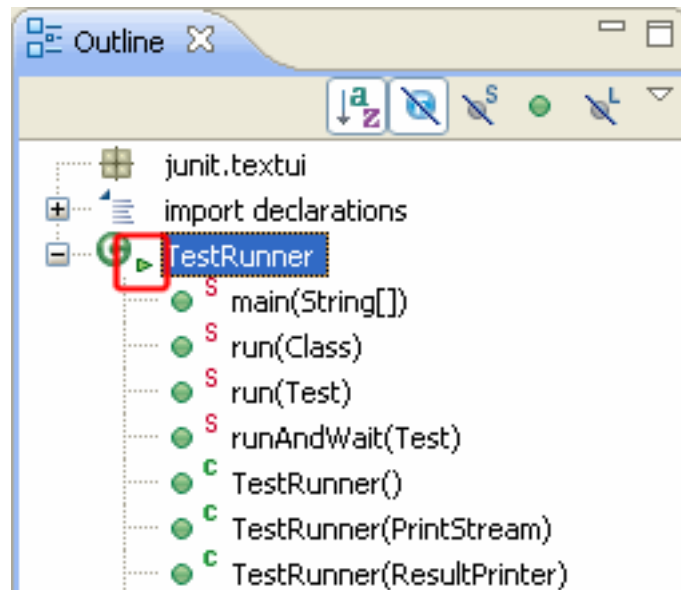
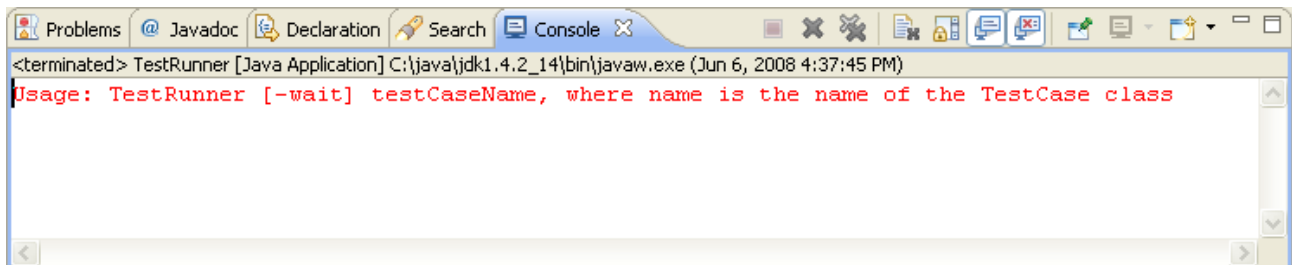


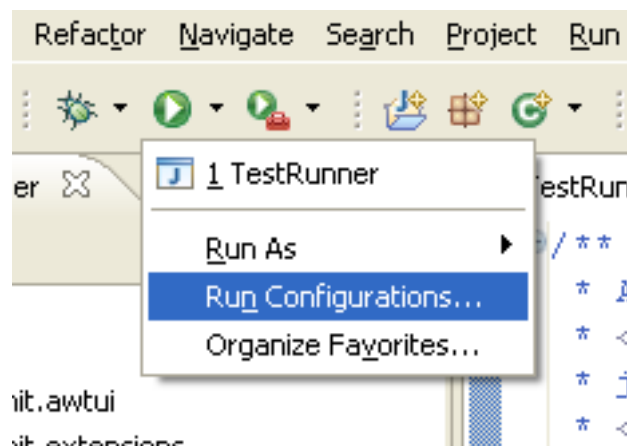
Ilustración 55: Clase con Main

Del menú de contexto en TestRunner.java en el explorador de paquetes selecciona **Run As > Java Application**. Se lanzará la clase seleccionada como una aplicación local de Java. La opción **Run As** está también disponible en otros lugares como en la vista Outline.

Aparecerá la vista de Consola (Console) diciendo que el programa necesita un argumento para ejecutarse. El modo de ejecución que hemos usado no permite la introducción de argumentos.



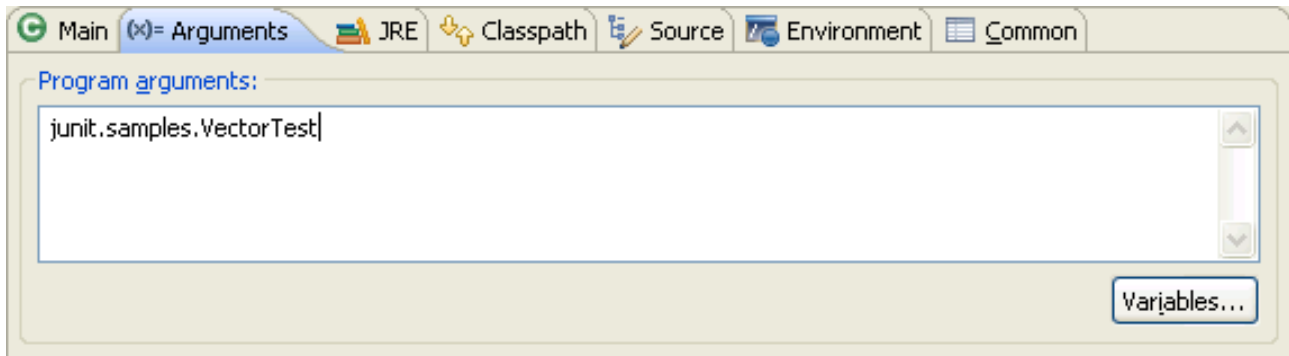
Para especificar argumentos, selecciona del desplegable del botón **Run**, la opción **Run Configurations...**



## UT05 - IDE Entornos de Desarrollo Integrado

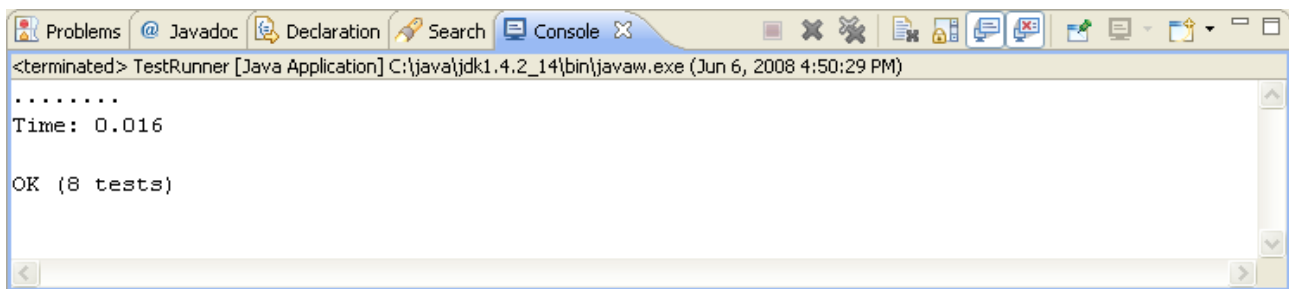
El diálogo se abre con la configuración de lanzamiento de TestRunner seleccionada. Se permite configurar como se lanza un programa, argumentos, Path de las clases y otra opciones. (Una configuración por defecto se creó al usar Run > Java Application).

Selecciona Java Application → TestRunner y abre la pestaña de argumentos.



Teclea *junit.samples.VectorTest* en el área de argumentos. Pulsa Run.

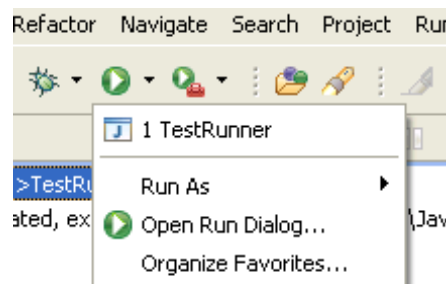
Esta vez se ejecutará correctamente, indicando el número de tests ejecutados.



Cambia a la perspectiva Debug. Se incluye ahora una entrada para la última ejecución del programa realizada.

Por defecto la vista debug borra las ejecuciones anteriores cuando se lanza una nueva. Esto puede configurarse en Run/Debug → Launching en Preferencias.

Se puede reejecutar un proceso seleccionando **Relaunch** desde el menú de contexto en la vista Debug. El histórico de aplicaciones ejecutadas se muestra también en el desplegable del botón Run. Pulsa en él para ver las aplicaciones que has ejecutado.



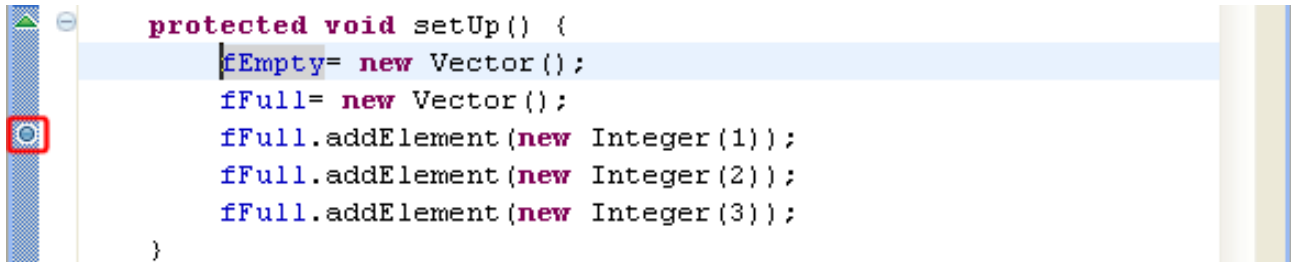
### 3.18 Depurar programas Java

Haz doble click en *junit.samples/VectorTest.java* para abrirlo en un editor.

Pon el cursor en la regla vertical del lado izquierdo del área de edición en la siguiente línea del método `setUp()`:

```
fFull.addElement (new Integer(1));
```

Haciendo doble click en la regla se pone un “BreakPoint”.

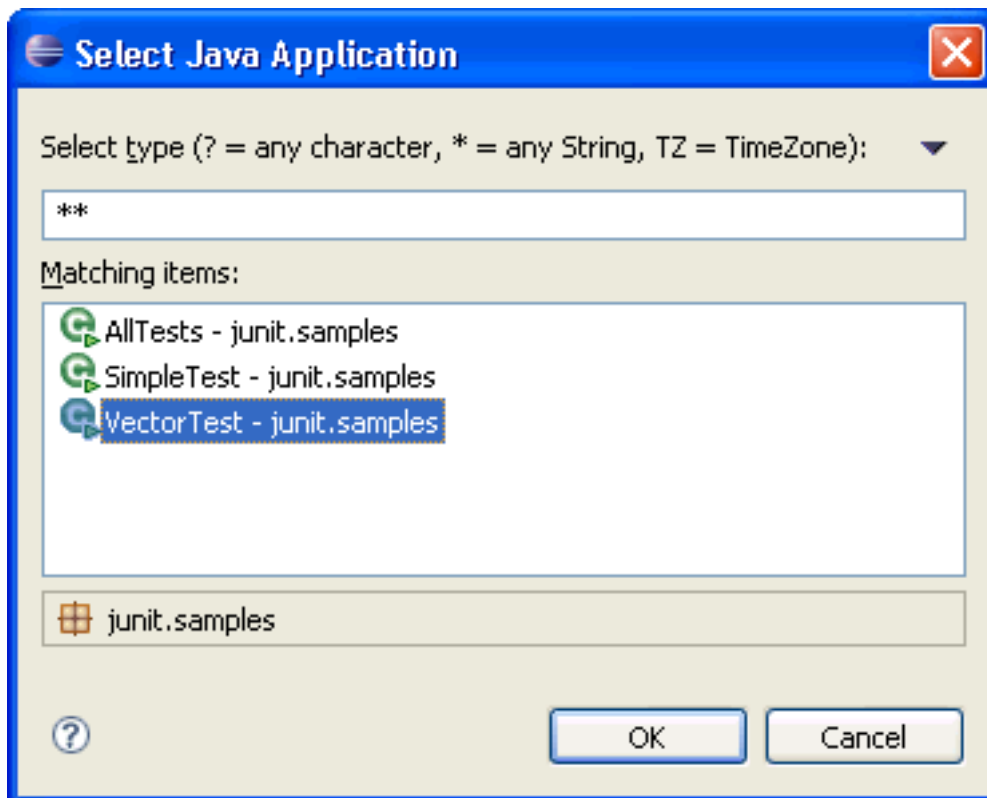


El color del breakpoint indica su estado. Azul significa que ha sido puesto pero no instalado.

Una vez que la clase es cargada por la máquina virtual, pasará a ser instalado y una marca de check aparecerá sobre el icono del breakpoint.

En el Explorador de paquetes, selecciona el paquete *junit.Samples* y en *Debug As, Java Application*.

Se abrirá un diálogo para elegir de entre todos los tipos que tengan un método `main()`.

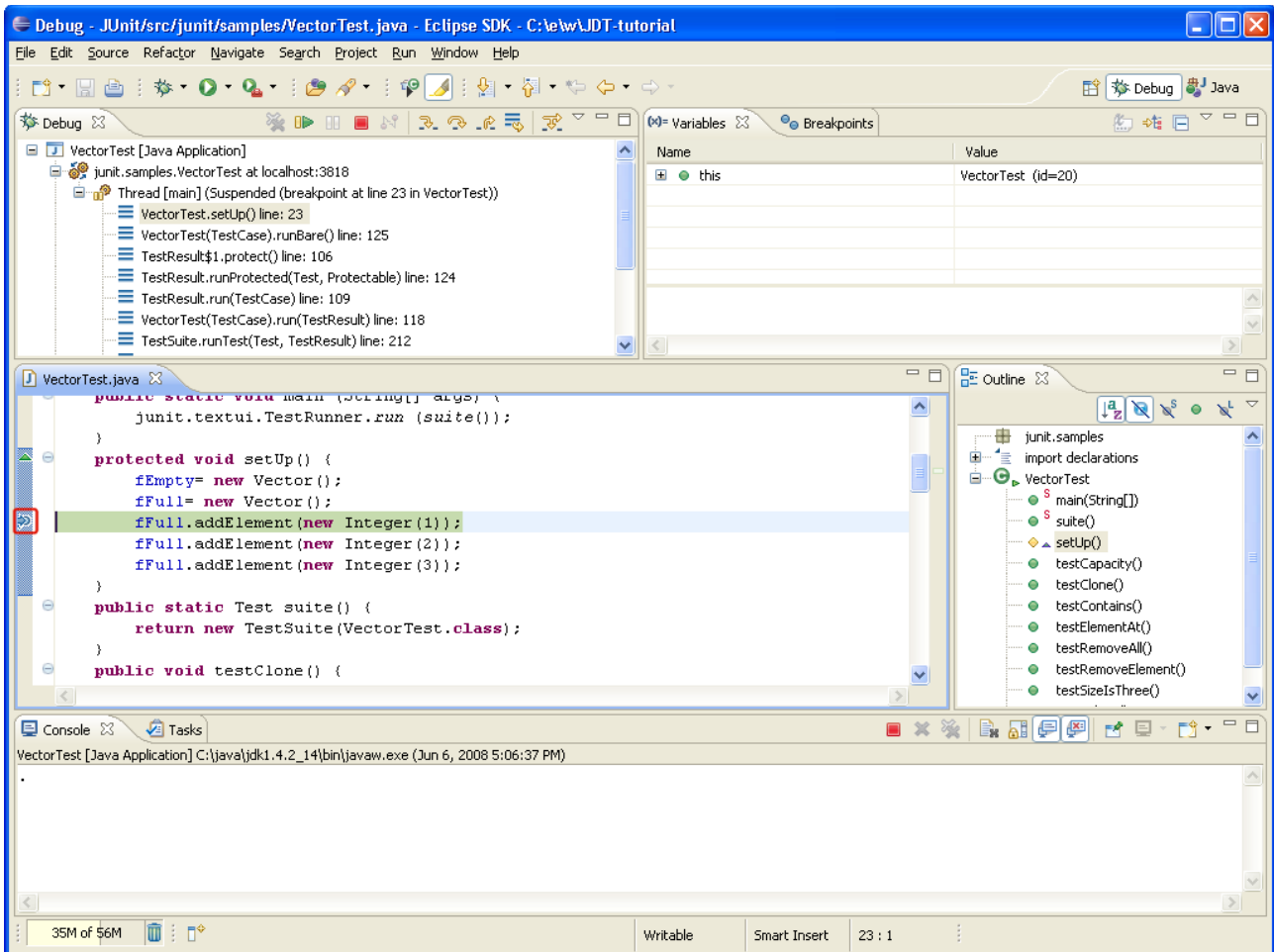


## UT05 - IDE Entornos de Desarrollo Integrado

Elige VectorTest y pulsa OK.

El programa se ejecutará hasta alcanzar el breakpoint puesto anteriormente. Una vez se alcanza el punto, la ejecución se suspende y se da la opción de abrir la perspectiva Debug. Contesta Yes.

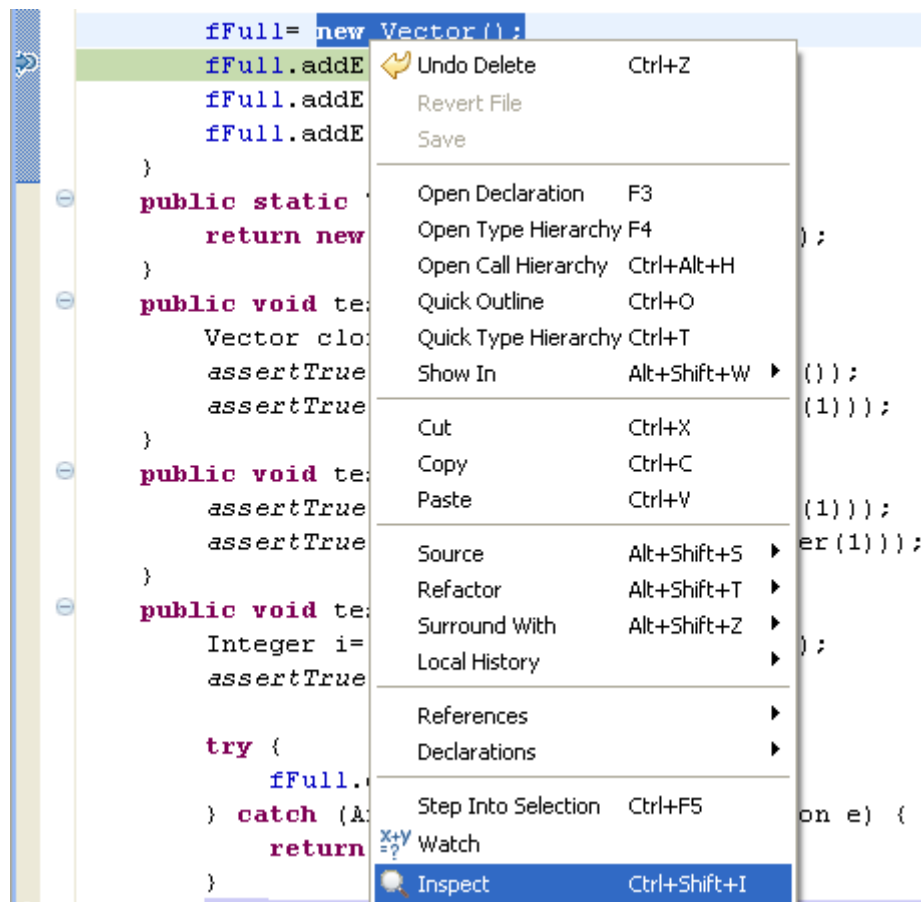
Fijate que el proceso está aún activo (not terminated) en la vista Debug. Otros hilos (threads) podrían aún estar ejecutándose.



Ahora el símbolo de breakpoint tiene un marca de check, porque la clase VectorTest está cargada en la máquina virtual.

Selecciona la instrucción `new Vector();` de la línea anterior al breakpoint y elige **Inspect**.

## UT05 - IDE Entornos de Desarrollo Integrado



La expresión será evaluada en el contexto de marco de pila actual. Y aparece un pop-up que muestra los resultados.

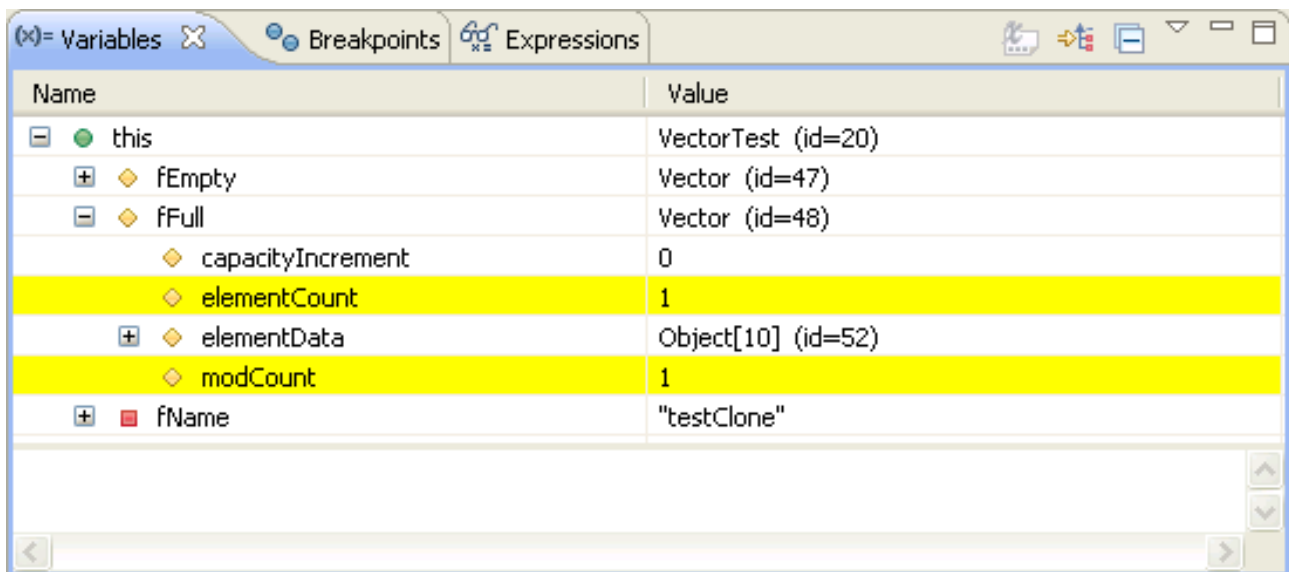
Se puede enviar un resultado a la vista Expressions pulsando Ctrl+Shift+I en el pop-up.

Las expresiones que se evalúen al depurar se mostrarán en esta vista Expressions. Puede borrarse una expresión que tras haber trabajado con ella eligiendo **Remove** desde el menú de contexto.

La vista de Variables (disponible en una pestaña al lado de Expressions) muestra los valores de las variables en el marco de pila seleccionado.

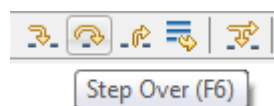
Expande el árbol de “this.fFull” en la vista Variables hasta que veas “elementCount”.

## UT05 - IDE Entornos de Desarrollo Integrado



Name	Value
this	VectorTest (id=20)
+ fEmpty	Vector (id=47)
- fFull	Vector (id=48)
+ capacityIncrement	0
+ elementCount	1
+ elementData	Object[10] (id=52)
+ modCount	1
+ fName	"testClone"

Las variables que se muestran cambiarán a la vez que se avance en la vista Debug. Para avanzar en la ejecución, pulsa el botón Step Over (F6) en la vista Debug.

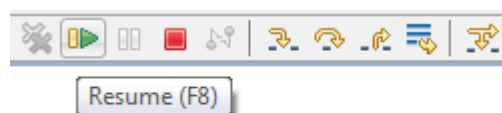


La ejecución continuará en la siguiente línea del método (o si es la última, saldrá del método y continuará desde el método llamante).

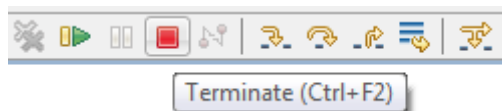
En la vista de variables se puede elegir ver una serie de tipos y estructuras lógicas. Se ocultan ciertos detalles de la implementación y se muestran simplemente como arrays o campos.

Step Into (F5), en el caso de que la línea de ejecución sea un método, pasará a mostrar la ejecución de ese método (Step Over no entra dentro de un método que esté seleccionado para ejecutar). Step Return (F7) sale del método actual.

El botón Resume continua la ejecución hasta el siguiente Breakpoint.



**Terminate (Ctrl+F2)** termina la ejecución.

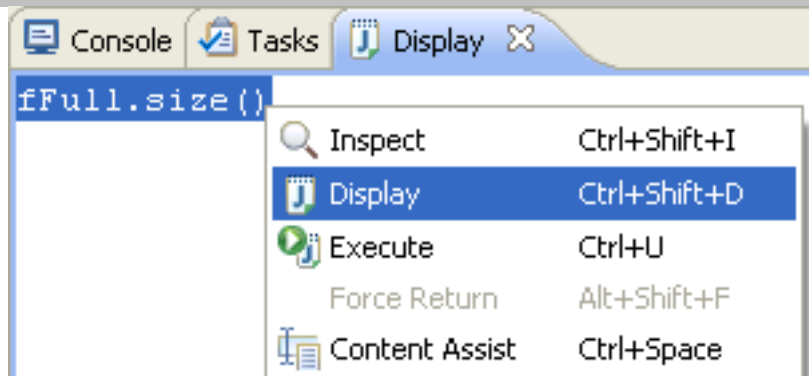


## 3.19 Evaluar expresiones

Depura de nuevo *junit.samples.VectorTest.java* hasta el breakpoint del método `setUp()` y selecciona **Step Over** dos veces para rellenar `fFull`.

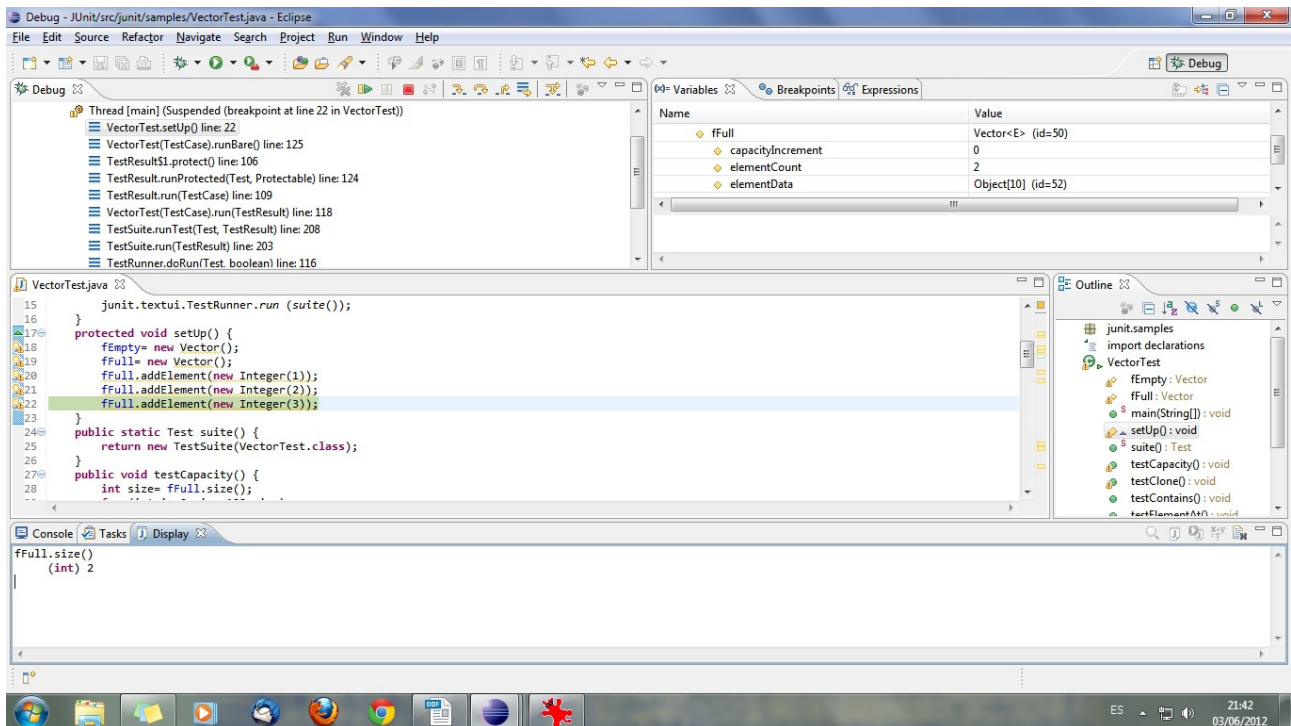
Abre la vista Display seleccionando **Window > Show View > Display** y teclea la siguiente línea en la vista:

```
fFull.size()
```



Selecciona el texto que acabas de teclear y desde su menú de contexto, selecciona **Display**. (Puedes también elegir **Display Result of Evaluating Selected Text** desde la barra de herramientas de la vista Display)

La expresión es evaluada y el resultado es mostrado en la vista Display.



Teclea lo siguiente en una nueva línea de Display:

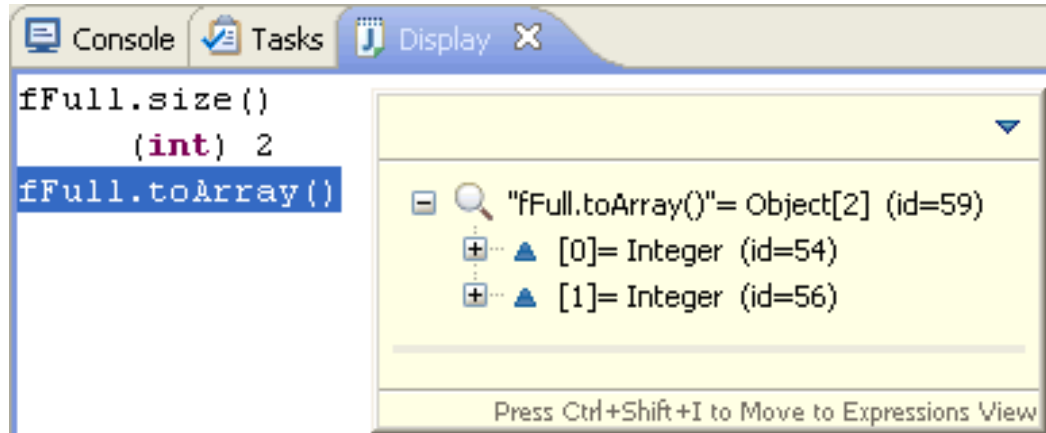


## UT05 - IDE Entornos de Desarrollo Integrado

```
fFull.toArray()
```

Esta vez selecciona Inspect desde el menú de contexto o desde su botón.

Se muestra una ventana de pop-up con el valor de la expresión evaluada:

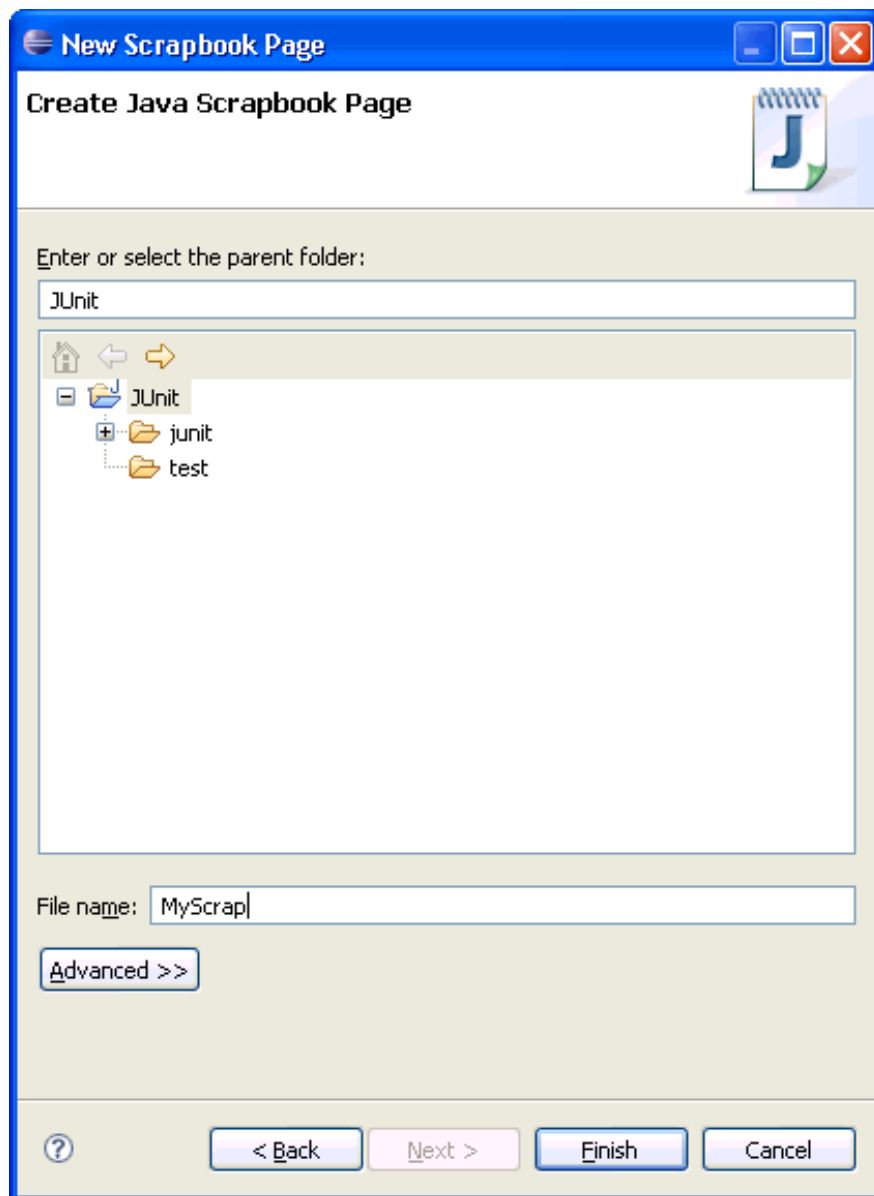


### 3.20 Evaluar fragmentos o “snippets”

Puede también evaluarse expresiones Java antes de ponerlas en el código usando el block de garabatos o *scrapbook* de Eclipse.

En File elige: **New > Other > Java > Java Run/Debug > Scrapbook Page**

Se pedirá una carpeta destino para la página.



Selecciona el directorio raíz de Junit.

En el campo nombre de fichero pon MyScrap.

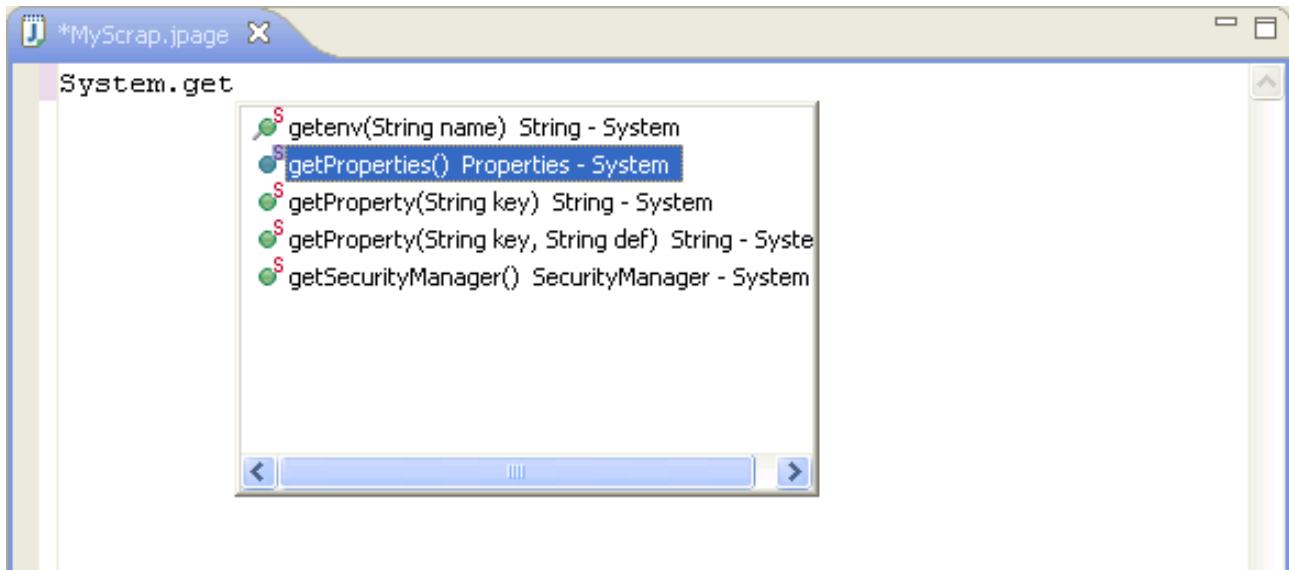
Finish.

Una página scrapbook es creada con la extensión *jpage*.

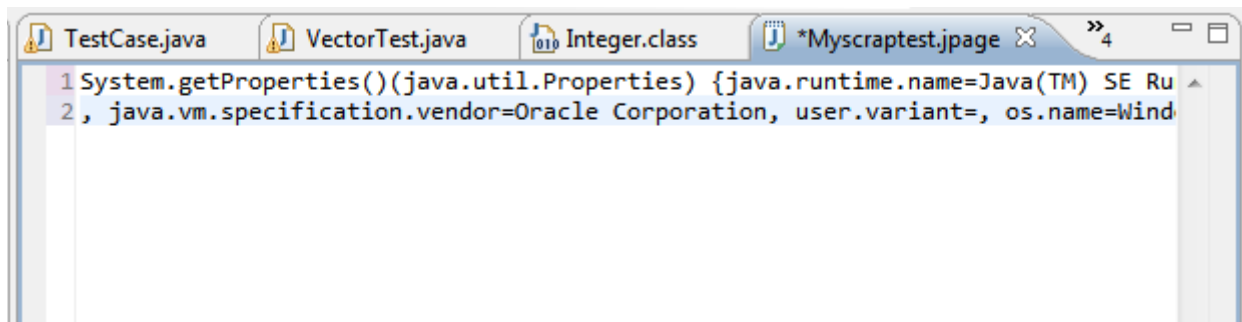
En el editor teclea `System.get` y usa el asistente de contenidos (Ctrl+Space) para completar

## UT05 - IDE Entornos de Desarrollo Integrado

el snippet como `System.getProperties()`.



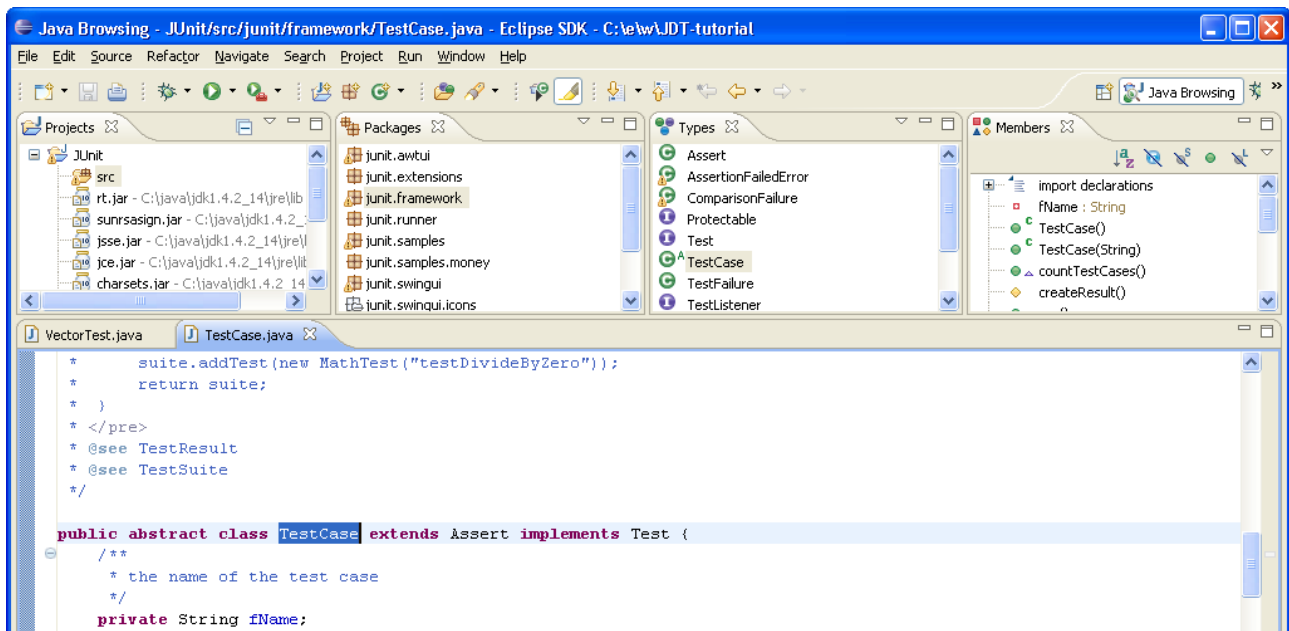
Selecciona **Display**.



### 3.21 Perspectiva Java Browsing

Es una manera de presentar información relativa a tu proyecto java de manera distinta e interrelacionada, usando distintas vistas para ver la misma información.

## UT05 - IDE Entornos de Desarrollo Integrado



Seleccionar un elemento en la vista Projects muestra sus paquetes en la vista **Packages**.

- La vista **Types** muestra los tipos contenidos en el paquete seleccionado en la vista **Packages**
- Las vista **Members** muestra los miembros de un tipo seleccionado. Es comparable a **Outline**.
- Si se selecciona un elemento en **Members** se muestra el elemento en el editor.

Las cuatro vistas están por defecto enlazadas con el editor activo, lo que significa que las vistas ajustarán su contenido y su selección de acuerdo al fichero presentado en el editor activo.

## 4 BIBLIOGRAFÍA Y ENLACES

- ✓ Wikipedia: [http://es.wikipedia.org/wiki/Entorno\\_de\\_desarrollo\\_integrado](http://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado)
- ✓ Ayudas de Eclipse.