

Mi primera hora con Eclipse.

¿Qué es Eclipse?

En la web oficial de Eclipse (www.eclipse.org), se define como “An IDE for everything and nothing in particular” (un IDE para todo y para nada en particular). Eclipse es, en el fondo, únicamente un armazón (*workbench*) sobre el que se pueden montar herramientas de desarrollo para cualquier lenguaje, mediante la implementación de los plugins adecuados.

La arquitectura de plugins de Eclipse permite, además de integrar diversos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, etc.

El Proyecto Eclipse

El IDE Eclipse es, únicamente, una de las herramientas que se engloban bajo el denominado *Proyecto Eclipse*. El *Proyecto Eclipse* aúna tanto el desarrollo del IDE Eclipse como de algunos de los plugins más importantes (como el JDT, plugin para el lenguaje Java, o el CDT, plugin para el lenguaje C/C++).

Este proyecto también alcanza a las librerías que sirven como base para la construcción del IDE Eclipse (pero pueden ser utilizadas de forma completamente independiente), como por ejemplo, la librería de widgets SWT.

El Consorcio Eclipse

En su origen, el *Proyecto Eclipse* era un proyecto de desarrollo *OpenSource*, soportado y mantenido en su totalidad por IBM. Bajo la dirección de IBM, se fundó el *Consorcio Eclipse* al cual se unieron algunas empresas importantes como Rational, HP o Borland. Desde el día 2 de febrero de 2004, el *Consorcio Eclipse* es independiente de IBM y entre otras, está formado por las empresas: HP, QNX, IBM, Intel, SAP, Fujitsu, Hitachi, Novell, Oracle, Palm, Ericsson y RedHat, además de algunas universidades e institutos tecnológicos.

La librería SWT

El entorno de desarrollo Eclipse, incluyendo sus plugins, está desarrollado por completo en el lenguaje Java. Un problema habitual en herramientas Java (como NetBeans) es que son demasiado “pesadas”. Es decir, necesitan una máquina muy potente para poder ejecutarse de forma satisfactoria. En gran medida, estas necesidades vienen determinadas por el uso del API Swing para su interfaz gráfico.

Swing es una librería de widgets portable a cualquier plataforma que disponga de una máquina virtual Java pero a costa de no aprovechar las capacidades nativas del sistema donde se ejecuta, lo cual supone una ejecución sensiblemente más lenta que la de las aplicaciones nativas.

SWT es una librería de widgets equivalente a Swing en la cual, se aprovechan los widgets nativos del sistema sobre el que se ejecuta. El hecho de aprovechar los widgets nativos, permite que la ejecución de interfaces de usuario sea mucho más rápida y fluida que si se utilizase Swing y, además, siempre dispone del “Look and Feel” del sistema, sin necesidad de “emularlo”.

La contrapartida es que la librería SWT es nativa, es decir, es necesario disponer de una librería SWT específica para cada sistema operativo.

Existen versiones de SWT para los S.O. más habituales, incluyendo Windows, Linux, HP-UX, MacOS, etc.

Obtener, instalar y ejecutar Eclipse IDE.

El IDE Eclipse se puede obtener bajándolo directamente del sitio web oficial del *Proyecto Eclipse* - www.eclipse.org - o de cualquier “mirror” autorizado. Existen versiones instalables para cualquier plataforma que soporte la librería SWT, descargas que incluyen el código fuente y descargas que incluyen los plugins más habituales. Además, de este mismo sitio, se puede descargar la librería SWT independientemente y su SDK.

Como Eclipse está escrito en Java, es necesario, para su ejecución, que exista un JRE (Java Runtime Environment) instalado previamente en el sistema.

La instalación de Eclipse, es tan sencilla como descomprimir el archivo descargado en el directorio que se estime conveniente.

Obtener e instalar Plugins

La descarga básica del entorno Eclipse incluye algunos de los plugins más básicos, pero siempre es deseable obtener alguna funcionalidad extra. Para ello, es necesario instalar nuevos plugins.

En el apartado *Community* del sitio web oficial de Eclipse se pueden encontrar enlaces a cientos de plugins.

Advertencia.

Es importante escoger cuidadosamente los plugins que se van a instalar pues, la cantidad de plugins instalados, influye en el rendimiento del IDE Eclipse, en especial, en el tiempo de arranque inicial de la aplicación.

Para añadir un nuevo plugin, basta con descomprimir el archivo descargado en el subdirectorio “Plugins” de la carpeta donde está instalado Eclipse. La próxima vez que se ejecute Eclipse, automáticamente, se reconocerán y añadirán los nuevos plugins instalados.

Ejecutar Eclipse

Las versiones que se pueden descargar del sitio web de Eclipse vienen con un ejecutable que permite lanzar directamente el IDE Eclipse. Antes de ejecutar Eclipse es importante verificar que se tienen permisos de escritura en el directorio, ya que, la primera vez que se ejecuta, Eclipse tiene que crear las carpetas en las que guardará información sobre workspaces, logs, etc.

Truco Linux.

La carpeta en la que está el ejecutable de Eclipse no tiene por qué ser, forzosamente, la que tenga permisos de escritura para el usuario. Si que debe tenerlos el directorio desde el cuál se ejecuta Eclipse. Es decir, se puede llamar a Eclipse a través de un script que esté en cualquier directorio (este si debe tener permisos de escritura). Esta solución es muy útil para sistemas con varios usuarios y para organizar mejor el trabajo en varios workspaces independientes.

Un vistazo general al IDE.

La primera vez que se ejecuta Eclipse se puede ver una pantalla muy similar a la que se muestra en la Figura 1. Antes de enfrentarse a la dura tarea del programador, es interesante echar un primer vistazo al entorno para conocer sus características particulares, la forma que tiene de organizar el trabajo, las herramientas adicionales que ofrece, etc.

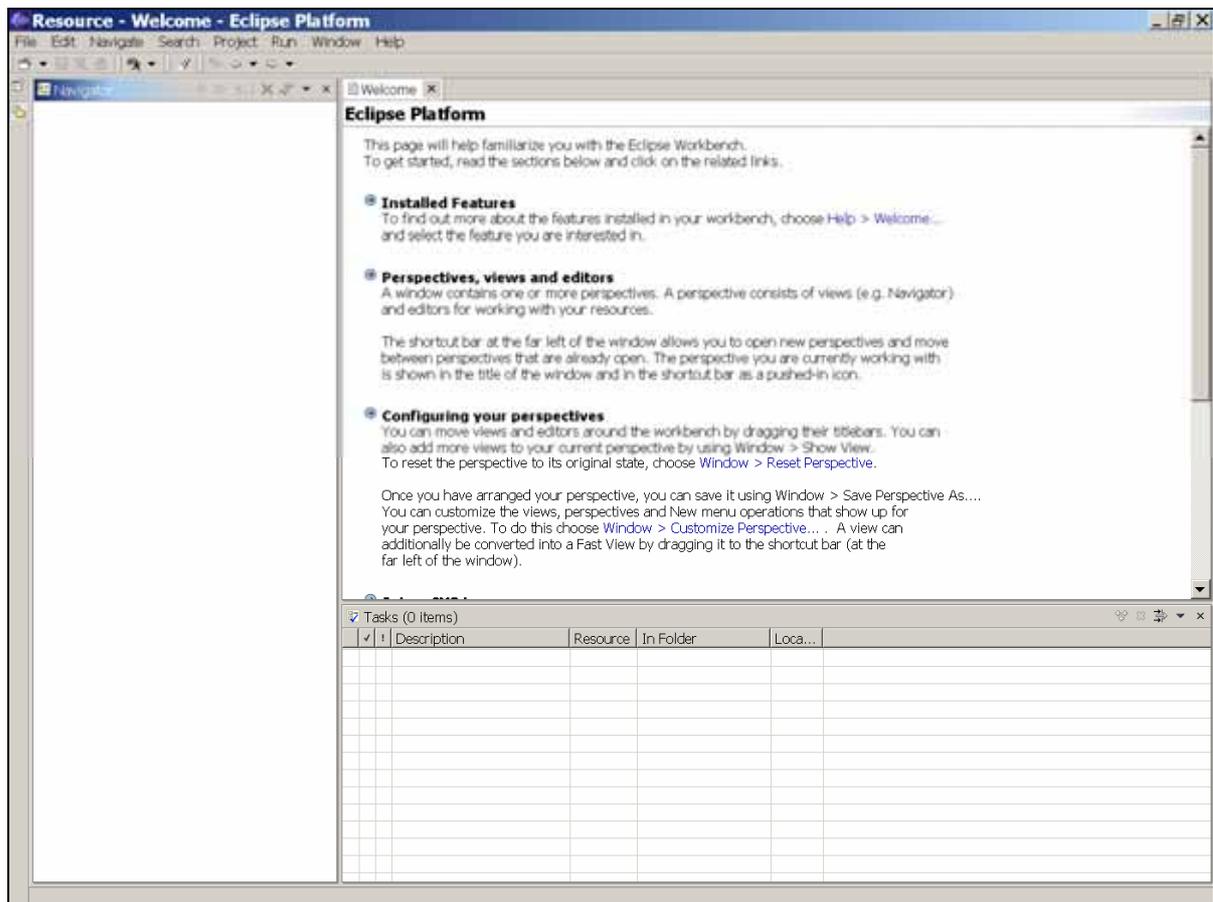


Figura 1. Eclipse IDE.

Editores

La ventana principal (la mas grande en la Figura 1), se llama “Editor”. Los Editores son el lugar donde se escribirán los programas. Es posible tener varios Editores abiertos a la

vez, apilados uno encima de otro. En la parte superior de la ventana de Editores, se mostrarán solapas que permiten acceder a cada uno de los Editores abiertos (o bien cerrarlos directamente).

Vistas

Además de los Editores, existen un segundo tipo de ventanas “secundarias”, que se llaman Vistas. Las Vistas sirven para cualquier cosa, desde navegar por un árbol de directorios, hasta mostrar el contenido de una consulta SQL. Se puede decir que las Vistas son ventanas auxiliares para mostrar información, requerir datos, etc.

Cada plugin puede definir Editores propios y todas las Vistas que sean necesarias. En la Figura 1, están abiertas dos ventanas de Vistas.

La Vista vertical de la izquierda, mostrará el árbol de directorios de los proyectos (cuando los haya).

La Vista horizontal inferior muestra una pequeña “agenda” de tareas pendientes que pueden ser introducidas por el usuario, de forma directa, o por Eclipse, en función de determinados eventos.

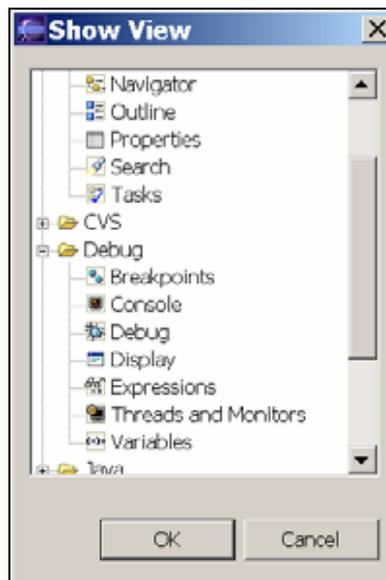


Figura 2. Ventana de selección de Vistas.

Para seleccionar qué Vistas se deben mostrar, se utiliza la opción “*Show View*” en el menú “*Window*” (ver Figura 2).

Barras de Herramientas

El tercero de los componentes del entorno son las barras de herramientas. Existen dos barras de herramientas: la barra de herramientas principal y la barra de Perspectivas.

La barra de herramientas principal contiene accesos directos a las operaciones mas usuales (guardar, abrir, etc.), botones que permiten lanzar la ejecución de herramientas externas y tareas relacionadas con el Editor activo (ejecutar un programa, depurar, etc.).

La barra de Perspectivas contiene accesos directos a las Perspectivas que se están utilizando en el proyecto. Una Perspectiva es un conjunto de ventanas (Editores y Vistas) relacionadas entre sí. Por ejemplo, existe una Perspectiva Java que facilita el

desarrollo de aplicaciones Java y que incluye, además del Editor, Vistas para navegar por las clases, los paquetes, etc.

La Perspectiva que está abierta en la Figura 1, es la llamada “*Resource Perspective*” y su función es navegar por el árbol de directorios de un proyecto y editar los ficheros que contiene utilizando el Editor mas adecuado.

Se puede seleccionar las perspectivas activas – las que se muestran en la Barra de Perspectivas – utilizando la opción “*Open Perspective*” del menú *Window*. Desde este mismo menú también es posible definir Perspectivas personalizadas.

Además de las barras de herramientas principales, cada Vista puede tener su propia barra de herramientas.

Programar con Eclipse.

Eclipse es un IDE que no está orientado específicamente hacia ningún lenguaje de programación en concreto. El uso de un determinado lenguaje, estará supeditado a la existencia de un plugin que le de soporte.

Con la versión estándar del entorno Eclipse se distribuye el plugin necesario para programar en lenguaje Java, su nombre es JDT.

Del sitio oficial de Eclipse se puede bajar también el plugin CDT para los lenguajes C/C++. Buscando un poco más en las bases de datos de plugins se pueden encontrar extensiones para lenguajes como Pascal o Python.

Mi primer programa en Java.

Como se acaba de comentar, el plugin JDT que da soporte al lenguaje Java viene incluido en la distribución estándar de la herramienta Eclipse.

Nuevo Proyecto Java

Para poder realizar un programa en Eclipse es necesario crear un proyecto. Un Proyecto agrupa a un conjunto de recursos relacionados entre sí (código fuente, diagramas de clases o documentación).

Se puede crear un nuevo proyecto desde el menú (*File* → *New* → *Project*), desde la barra de herramientas principal o desde la vista “*Navigator*” (abriendo el menú pop-up con el botón derecho del ratón y la opción *New* → *Project*).

Cualquiera de estas tres opciones lanzará el *wizard* de creación de proyectos. Para iniciar un proyecto Java se debe seleccionar la opción *Java* → *Java Project*.

Después de indicar un nombre y una ubicación para el nuevo Proyecto se puede, opcionalmente, realizar algunas configuraciones como son:

- Crear un subdirectorio para almacenar el código y un subdirectorio diferente para almacenar las clases compiladas.
- Indicar las dependencias del nuevo proyecto respecto a proyectos anteriores (existentes en el mismo workspace).
- Indicar la ubicación de librerías (.jar) que necesita el proyecto y/o definir variables de entorno.
- Definir el orden de búsqueda de los *classpaths* que se manejan, principalmente para solucionar conflictos en caso de que haya clases con el mismo nombre cualificado.

Es siempre recomendable definir una carpeta (por ejemplo, de nombre *src*) para contener el código y otra (de nombre *bin*, por ejemplo) donde se dejarán los *.class* generados.

En la solapa *Libraries*, se pueden añadir todos los *.jar* que sean necesarios (con el botón “*Add External Jars...*”).

Todas estas configuraciones pueden modificarse en cualquier momento a través del menú contextual de la vista *Navigator*, en la opción *Properties* → *Java Build Path*.

Al crear el proyecto Java, Eclipse, de forma automática, abre la Perspectiva Java, que es la colección de vistas que define el plugin JDT para programar con Java. Esta Perspectiva está compuesta de las vistas: *Package Explorer* (que permite navegar por todos los paquetes –classpaths- a los que puede acceder el proyecto) y *Outline* (que muestra un esquema de la clase cuyo código se está visualizando en el Editor activo).

Además, si la Perspectiva Java está activa, se añaden a la barra de herramientas principal algunos botones extra que permiten acceder con rapidez a las funciones más usuales (ejecutar, depurar, crear clases, etc.)

Ejemplo.

A modo de ejemplo, crearemos un Proyecto Java, cuyo nombre será *Prueba*. Su ubicación será la carpeta por defecto y no requerirá ningún *.jar* extra en el classpath. Además, se configurará como carpeta para el código, el directorio *prueba/src* y para las clases compiladas, se utilizará la carpeta *prueba/bin*.

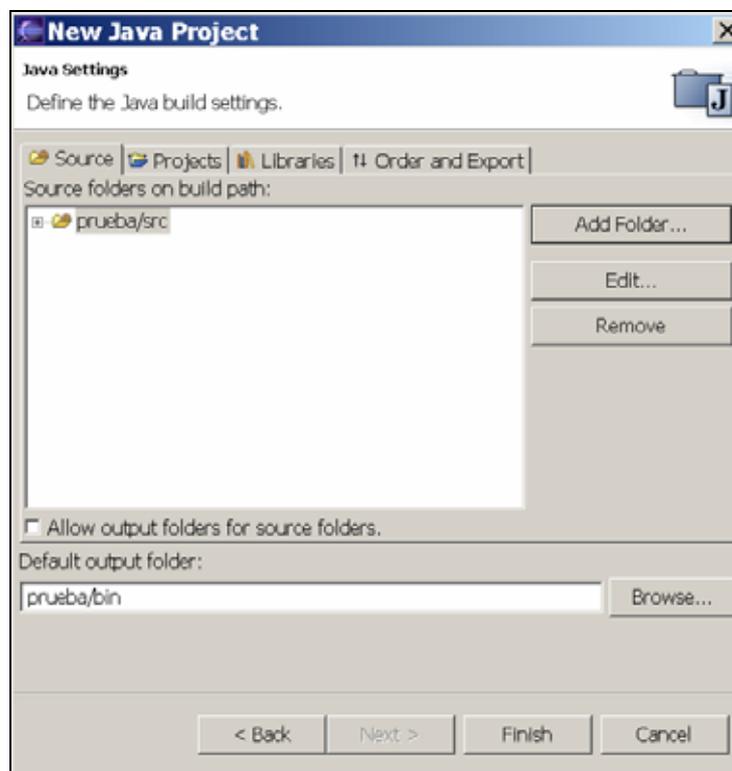


Figura 3. Nuevo proyecto Java.

Nuevas Clases

El modo más directo de crear una nueva clase (o interface) es utilizar el “*wizard de creación de clases*” que se puede lanzar, teniendo la Perspectiva Java activa, a través del botón correspondiente (Figura 4) en la barra de herramientas.



Figura 4. Botón creación clases.

El *wizard de creación de clases*, se compone de un único formulario en el que se indicarán las características de la nueva clase (o interface) que se quiere crear: nombre, superclase, interfaces que implementa, etc.

Ejemplo.

Aprovechando el proyecto Prueba que hemos creado en el apartado anterior, podemos ahora crear una nueva clase, utilizando el wizard. Esta clase se llamará *MiPrueba* y estará pertenecerá al paquete *es.prueba*.

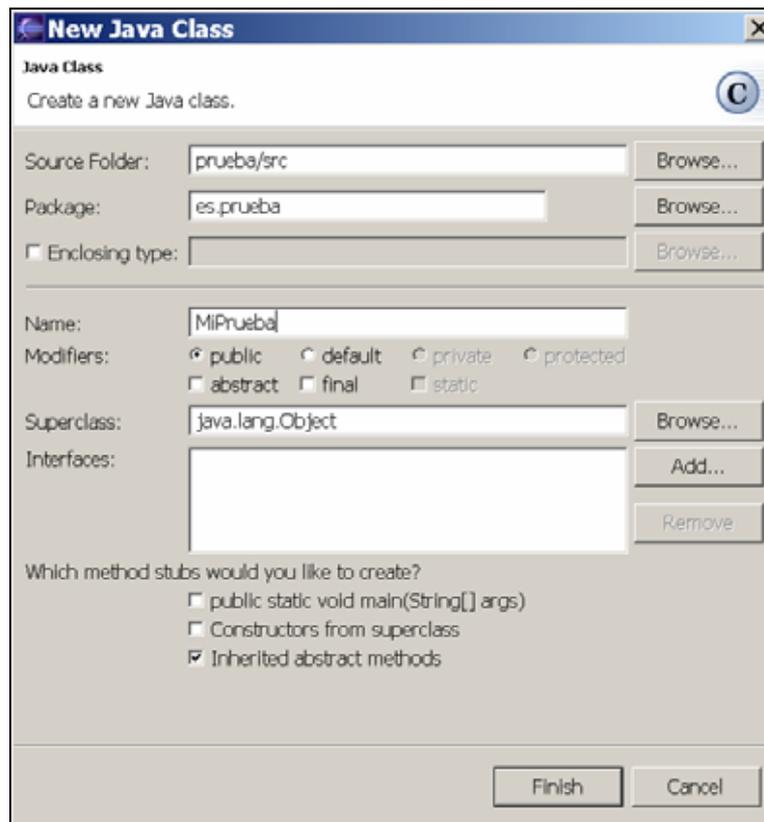


Figura 5. Nueva clase.

En ocasiones, especialmente cuando se trata de proyectos grandes, con muchos paquetes de nombres largos, el wizard de creación de clases puede ser una herramienta lenta y pesada.

Existen otros medios para crear clases. El más sencillo es, en la vista *Navigator*, crear un nuevo fichero java. Para ello, basta con seleccionar la carpeta donde se va a guardar

la nueva clase y, en el menú pop-up seleccionar la opción *New* → *File*. Es importante que el fichero que se cree tenga la extensión *.java*.

Truco.

Otra opción también sencilla es reutilizar el código ya escrito de otra clase. La forma más rápida es, sobre la misma vista *Navigator*, copiar un fichero (ctrl.+C) y pegarlo en otra carpeta (ctrl.+V). Cuando se abra la nueva clase en el Editor, Eclipse detectará y marcará los errores que tenga (por ejemplo, que la declaración **package** no se corresponda con la ubicación de la clase en el árbol de directorios del proyecto) y se encargará de proponer y ejecutar las soluciones adecuadas.

Si se quiere cambiar los imports, includes o la superclase, la opción más cómoda es dejar que el mecanismo *code completion* de Eclipse haga el trabajo.

En este apartado se ha hablado de cómo crear nuevas clases. La creación de nuevos interfaces es prácticamente igual a la de las clases, en todos los sentidos.

Programar con Eclipse

Cuando se crea una nueva clase se puede ver, en la ventana Editor, que algunas palabras están coloreadas de forma diferente. Este marcado de palabras es debido a que los Editores Java que implementa el plugin JDT, incluyen capacidad para realizar *syntax highlighting* (o reconocimiento sintáctico de palabras reservadas del lenguaje).

De esta forma, las palabras reservadas del lenguaje aparecerán escritas en negrita y en color Burdeos, los comentarios en verde y los comentarios de documentación (javadoc) en azul.

Corrector de Errores

Aparte de identificar las palabras reservadas del lenguaje, JDT puede detectar, y marcar sobre el código de un programa, los lugares donde se pueden producir errores de compilación. Esta característica funciona de forma muy parecida a los correctores ortográficos que tienen los procesadores de textos (ver Figura 5).

Cuando Eclipse detecta un error de compilación, se marcará la sentencia errónea, subrayándola con una línea ondulada roja (o amarilla, si en lugar de un error se trata de un *warning*).

Si el programador posiciona el puntero del ratón sobre la instrucción que produjo el fallo, se mostrará una breve explicación de por qué dicha instrucción se ha marcado como errónea.

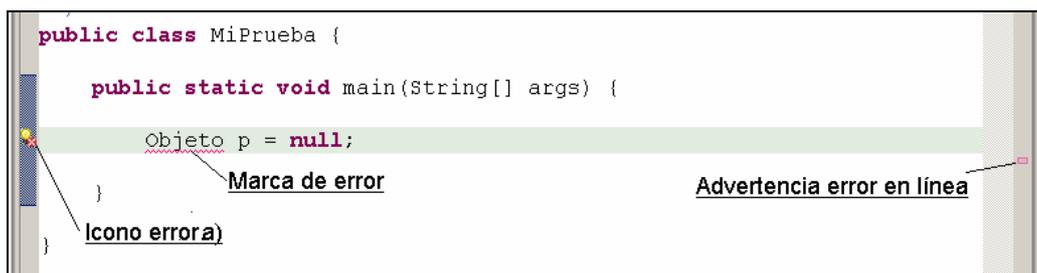


Figura 6. Corrector integrado.

Cada línea de código que contenga un error (o un warning), será también marcada con un icono de error (o warning) que aparecerá en la barra de desplazamiento izquierda del editor (identificado, en la Figura 6 como “a”). Si el programador pulsa una vez sobre dicha marca (ojo, una sola vez, no doble click) se desplegará un menú pop-up mediante el cuál, Eclipse mostrará posibles soluciones para los errores detectados. Si se acepta alguna de las sugerencias del menú pop-up, Eclipse se encargará de llevar a cabo dicha solución, de forma completamente automática.

Truco.

Es muy útil aprovechar la corrección de código para escribir código más rápidamente. Por ejemplo, si se necesita hacer un *casting* a una determinada variable, una forma rápida y cómoda es, deliberadamente, “olvidarse” de hacerlo. Cuando el mecanismo de corrección lo detecte, siempre se puede dejar que sea él mismo quien solucione automáticamente el problema, ahorrándonos la labor de escribir el *casting*, de escribir las sentencias **import** que puedan ser necesarias, etc.

Code Completion

Un entorno de desarrollo no puede considerarse como útil, en la práctica, si no dispone de la capacidad de completar automáticamente las sentencias que está escribiendo el programador. El mecanismo de “*code completion*” en Eclipse es muy similar al que implementan otros IDEs: cuando se deja de escribir durante un determinado intervalo de tiempo se muestran, si los hay, todos los términos (palabras reservadas, nombres de funciones, de variables, de campos, etc.) que empiecen por los caracteres escritos. Si se escriben determinados caracteres (como el punto, por ejemplo) se puede provocar la ejecución del mecanismo de “*code completion*” sin necesidad de esperar a que pase el tiempo establecido.

Truco.

Hay veces (generalmente cuando más falta hace) que el mecanismo de completado automático no se dispara cuando se espera que lo haga, o tarda demasiado. Otras veces, parece que se dispara siempre y acaba siendo pesado. La mejor opción es acostumbrarse a lanzarlo siempre manualmente (en Eclipse se utiliza la combinación de teclas ctrl.+ espacio).

Otra característica relacionada con el *code completion* es la asistencia a la escritura en llamadas a funciones. Automáticamente, cuando se van a escribir los parámetros que se pasan a un método, se muestra una caja de texto indicando los tipos que éstos pueden tener.

Templates

De forma similar a muchos otros entornos de desarrollo, Eclipse permite definir y utilizar *templates*. Los *templates* son plantillas de código (generalmente porciones de código de uso habitual y muy repetitivo) que se escriben automáticamente.

Los *templates* están compuestos de dos partes: un bloque de código (o de comentario), de uso frecuente, que se escribe automáticamente y una cadena que provoca la escritura del *template*. Las cadenas que disparan templates serán reconocidas por el sistema de *code completion*, con la diferencia de que, en lugar de terminar la escritura de la cadena, ésta será sustituida por el *template* que tiene asociado.

Ejemplo.

En el ejemplo inferior, se pretende escribir un bucle *for* que itere un *array*. Se trata de un tipo de construcción muy común, por ello, es firme candidata a ser asociada a un *template*.

```
public static void main(String[] args) {  
    for  
}
```

Si en el código anterior se pulsa la combinación ctrl.+ espacio, y se selecciona la opción “*for – iterate over array*”, el resultado que se obtiene es el siguiente:

```
public static void main(String[] args) {  
    for (int i = 0; i < args.length; i++) {  
    }  
}
```

El plugin JDT, por defecto, define una buena cantidad de *templates*, tanto para construcciones de código, como para la escritura de javadoc pero, de todas formas, es posible definir nuevos *templates* personalizados (o modificar los existentes).

A la ventana de configuración de templates se accede a través del menú principal en la opción *Window* → *Preferences* → *Java* → *Editor* → *Templates*.

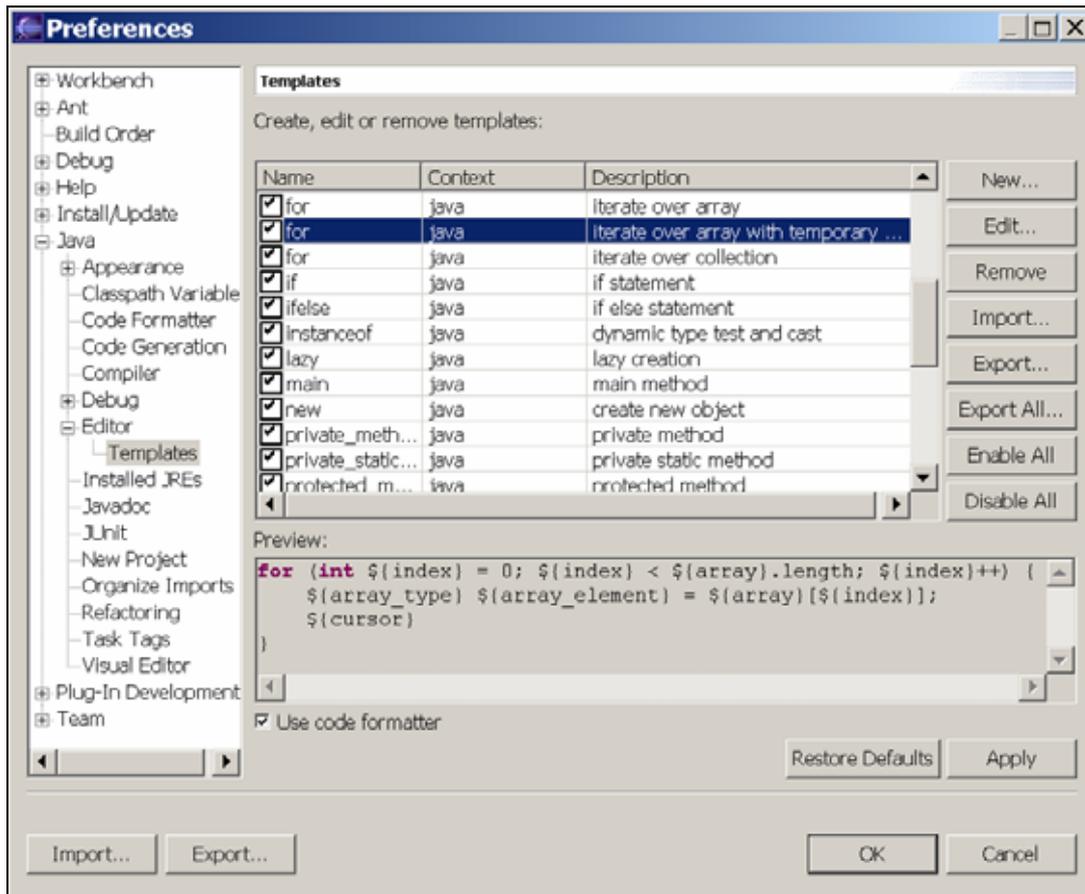


Figura 7. Configuración de templates.

Code Formatting

Todos los programadores sabemos lo importante que es disponer de un código ordenado, legible y fácil de entender. Al menos en teoría. En la práctica no suele encontrarse nunca tiempo, ni ganas, para conservar el aspecto del código.

Eclipse incorpora una herramienta para realizar automáticamente el formateo del código de acuerdo a unos criterios preestablecidos.

Para formatear el código que muestra el Editor activo, basta con seleccionar la entrada *Source* → *Format* del menú contextual que aparece al pulsar con el botón derecho del ratón sobre el propio Editor.

Ejemplo.

Veamos cómo funciona el *formateador de código* para el ejemplo que se muestra a continuación:

```
public static void main(String[] args) {  
    System.out.println("Esta es una línea muuuy larga y además con varias instrucciones"); int i =0; i++;  
}
```

Si se ejecuta el formateo automático, el resultado obtenido es el siguiente:

```
public static void main(String[] args) {  
    System.out.println(  
        "Esta es una línea muuuy larga y además con varias instrucciones");  
    int i = 0;  
    i++;  
}
```

El *formateador de código* puede configurarse en la ventana de *Preferences* del entorno (opción *Window* → *Preferences* del menú principal), en el apartado *Java* → *Code Formatter*. En esta ventana, entre otras cosas, se puede indicar la longitud de las líneas de texto, la indentación a aplicar, el formato para las asignaciones, etc...

Truco.

El *formateador de código* permite despreocuparse casi por completo del aspecto del código. Es decir, se puede, sin problemas escribir líneas inusualmente largas, varias sentencias en una misma línea, etc. En resumen, se pueden hacer todas esas cosas que a los programadores nos encantan y que están completamente prohibidas (pero que son tremendamente cómodas a la hora de programar).

Eso sí, es muy importante que el *formateador de código* esté configurado de acuerdo a las convenciones de presentación que se quieran aplicar.

Manipulación del Código

La capacidad de formato automáticamente a los programas, es sólo una de las posibilidades de manejo de la estructura del código que soporta Eclipse.

Otras posibilidades, englobadas bajo la entrada *Source* del menú contextual del Editor son:

- *Comment* y *Uncomment*. Estas dos opciones permiten seleccionar un trozo de código y comentarlo (o descomentarlo) de una vez. Los comentarios que se establecen de esta forma, son comentario de “tipo línea” (*//...*) por lo tanto, no se ven afectados en caso de que existan previamente bloques de comentarios en el código seleccionado.
- *Add Javadoc Comment*. Escribe un bloque de comentarios javadoc para el elemento seleccionado. Por ejemplo, si se coloca el cursor sobre un método y se ejecuta la operación *Source* → *Add Javadoc Comment*, se creará, sobre ese método, el esqueleto predefinido para la documentación javadoc, que contendrá

etiquetas para cada uno de los parámetros (@param), para el resultado (@return), para las excepciones (@throws), etc...

- *Add import*. Escribe las sentencias **import** para la clase sobre la que esté posicionado el cursor (o sobre la más próxima si no está sobre ninguna).
- *Organize Imports*. Agrupa las sentencias import en función de la ubicación de las clases (o paquetes) referenciados, en la jerarquía global de paquetes del proyecto.
- Opciones de generación automática de “esqueletos” de código. Estas opciones permiten generar, automáticamente, el código necesario para definir métodos get y set (*Source* → *Generate Setter and Getter...*) para los atributos de la clase, extender constructores y otros métodos definidos en una superclase o en un interfaz, etc.

Todas estas opciones de manipulación de código pueden configurarse en las entradas del menú principal *Window* → *Preferences* → *Java* → *Organize Imports* y *Window* → *Preferences* → *Java* → *Code Generation*.

Refactoring

En el punto anterior, se comentaban algunas de las facilidades que ofrece Eclipse para crear y manipular bloques de código de una forma fácil y cómoda, evitando el tedio de tener que realizar todo el trabajo a mano.

Todas las operaciones de manejo de código explicadas trabajan, únicamente, con código escrito sobre un mismo fichero (o perteneciente a una misma clase). Si las modificaciones que se quieren realizar deben involucrar a varias clases, escritas en varios ficheros diferentes, todos ellos pertenecientes al mismo proyecto, entonces se pueden utilizar las herramientas de *Refactorización*.

Las herramientas de *Refactoring* son especialmente útiles cuando se trata de realizar modificaciones, o actualizaciones, en el código, que afectan a varios elementos del diseño.

En Eclipse, se puede acceder a las operaciones de *Refactoring* a través de la opción *Refactor* en el menú principal o en el menú pop-up del Editor.

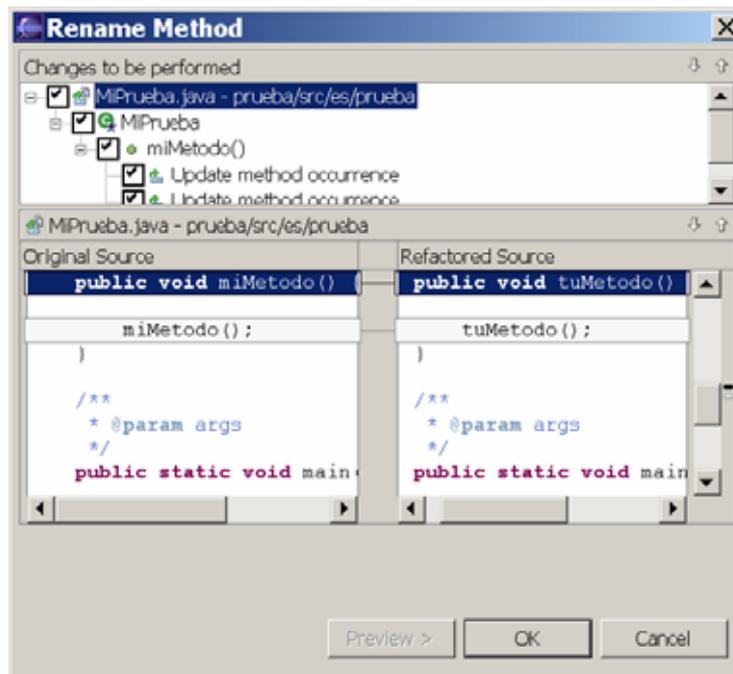
Ejemplo.

En este ejemplo, se utilizarán las operaciones de *Refactoring* para cambiar el nombre de un método. Esta modificación no solamente afecta a la clase que lo implementa. También afectará a todas las clases que realicen llamadas a dicho método, a las subclases que lo redefinan, etc.

```
public void miMetodo() {  
  
    miMetodo();  
}
```

Para realizar el cambio de nombre, habrá que seleccionar el nombre del método a modificar y lanzar la operación *Refactor* → *Rename...* del menú contextual. Aparece el diálogo que solicita un nuevo nombre y pide confirmación para actualizar también las referencias que se hagan, en el proyecto, al método que se modifica.

Si se pulsa el botón “*Preview >*”, se muestra una comparación del antes y el después de cada porción de código que se va a modificar.



Una vez aceptados los cambios, el resultado final es:

```
public void tuMetodo() {  
  
    tuMetodo();  
}
```

La herramienta de *Refactoring* que incluye Eclipse permite realizar muchas otras refactorizaciones (aparte de los cambios de nombre). Entre otras están: cambiar los parámetros de un método, mover un método a una subclase o superclase, extraer un interfaz, convertir una variable local en un atributo, etc.

Compilar

Una de las características más curiosas del IDE Eclipse es el modo en que se compilan los proyectos. No existe en Eclipse ningún botón que permita compilar individualmente un fichero concreto. La compilación es una tarea que se lanza automáticamente al guardar los cambios realizados en el código. Por esta razón es prácticamente innecesario controlar manualmente la compilación de los proyectos.

En caso de necesidad, existe una opción en la entrada *Project* del menú principal, llamada “*Rebuild Project*” que permite lanzar todo el proceso de compilación completo (también existe la entrada “*Rebuild All*” para re-compilar todos los proyectos abiertos).

Truco.

En general, el proceso de “compilación bajo demanda” de Eclipse funciona bastante bien. Para agilizar el proceso de compilación, y que éste no interrumpa al programador cada vez que salva los cambios realizados, Eclipse mantiene un sistema de caché de código que permite reducir las tareas de compilación a lo estrictamente necesario. Este sistema de cachés a veces, no detecta determinados cambios en el código (por ejemplo, cuando se cambia el contenido de cadenas de texto) y esto puede acarrear que el proceso de compilación no se ejecute siempre que sea necesario.

Es recomendable, si se actualizan solamente cadenas de texto, o si se tienen modificaciones en varias clases con muchas referencias entre si, lanzar el proceso de *Rebuild* de forma manual, para garantizar que el código se compila correctamente.

Cuando se compila un proyecto completo, Eclipse utiliza una secuencia de instrucciones de construcción (build) predefinidas en función de la configuración del proyecto (carpetas de código, carpeta destino, JDK a utilizar, classpaths definidos, etc.)

Sin embargo, en muchas ocasiones, el proceso de construcción de un proyecto incluye muchas otras operaciones además de la mera compilación del código (crear una base de datos, inicializar determinados ficheros, crear archivos de despliegue, etc.)

Es posible configurar el compilador de Eclipse para que realice cualquier proceso de construcción del proyecto de forma automática. Para ello, se utilizan scripts Ant [<http://ant.apache.org>].

Ant es una evolución de los clásicos Makefiles, que utiliza scripts escritos sobre XML. No es el objetivo de este documento explicar la herramienta Ant, pero sí cómo integrarla en el entorno Eclipse.

Si se quiere utilizar una herramienta de compilación (en este caso Ant) diferente del constructor (Builder) definido por defecto por la herramienta Eclipse, debe ejecutarse la opción “*Project* → *Properties* → *External Tool Builders*”. En esta ventana, se pueden añadir, modificar o eliminar los diferentes mecanismos de compilación y construcción disponibles para el proyecto.

Para añadir un nuevo script Ant, habrá que pulsar el botón “*New...*” y seleccionar la opción “*Ant Build*” que, nos llevará directamente al wizard de configuración de scripts Ant.

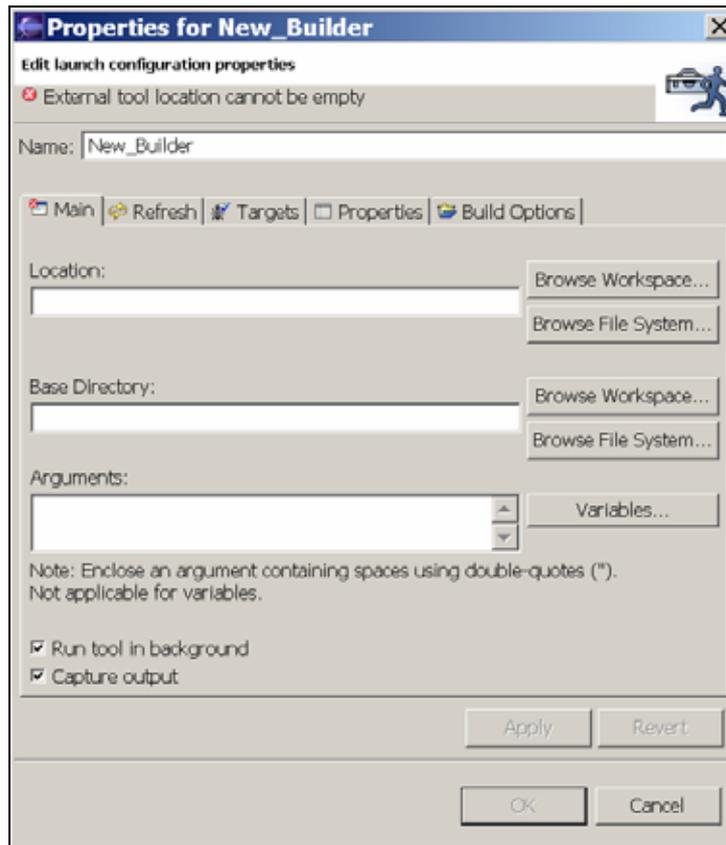


Figura 8. Wizard Ant.

Como se puede ver en la Figura 8, el wizard Ant es una herramienta sencilla. En la solapa principal (la que se muestra en la Figura 8) es necesario indicar la localización del script Ant que se quiere utilizar para compilar el proyecto, así como el directorio base y los argumentos. El resto de las solapas sirven para definir el modo en que Eclipse va a gestionar las llamadas al script.

Ejecutar

Una vez compilado correctamente, ejecutar el proyecto es la parte más sencilla (si el proyecto está correctamente programado claro). Prácticamente todas las opciones de ejecución se pueden manejar desde el botón *Run* de la barra de herramientas principal (ver Figura 9).



Figura 9. Botón ejecución.

El botón *Run* puede utilizarse de dos formas: bien pinchando el propio botón, en este caso, se repetirá la última ejecución realizada, o bien pinchado sobre la flecha a su lado lo cual permitirá ver el menú de ejecución.

El menú de ejecución, a su vez tiene dos partes. La entrada “*Run As*” permite ejecutar directamente la clase que se está mostrando en la ventana del Editor activo, utilizando la configuración de ejecución por defecto.

La entrada “*Run...*”, permitirá definir nuevas configuraciones de ejecución.

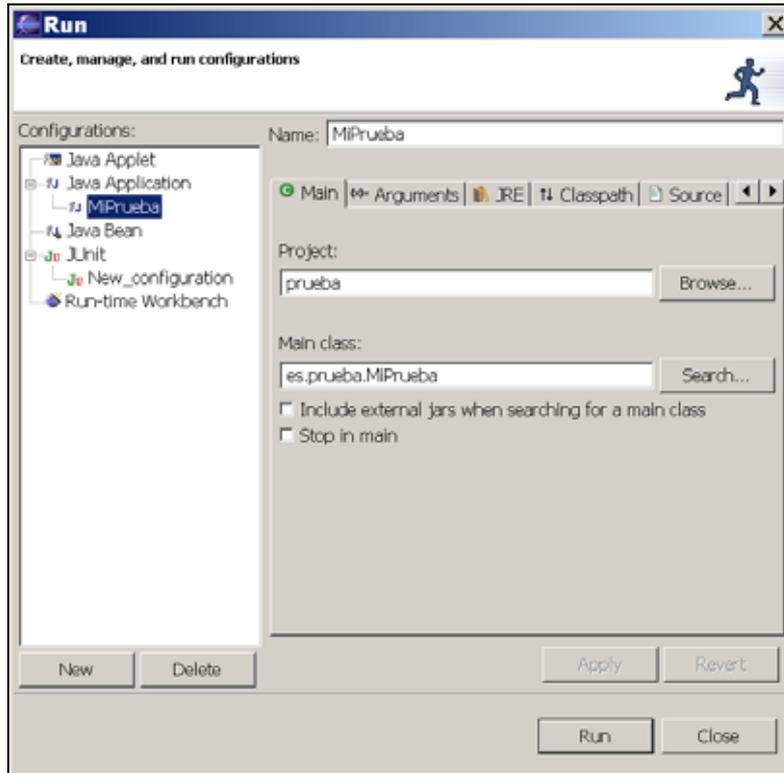


Figura 10. Ventana de configuraciones de ejecución.

Una configuración de ejecución es un conjunto de parámetros que se tendrán en cuenta a la hora de lanzar una ejecución de un programa. Algunos de estos parámetros pueden ser: un classpath determinado, la versión del JRE que se utilizará o los propios parámetros que se pasarán a la clase que se va a ejecutar.

Los parámetros que se pueden definir, y sus valores por defecto, vendrán determinados por el tipo de programa que se va a ejecutar. No tendrá los mismos parámetros una aplicación Java que un applet, por ejemplo.

Depurar Aplicaciones

La principal diferencia entre un simple editor y un buen entorno de desarrollo es que éste integre, o no, una buena herramienta visual para depurar los programas escritos.

Eclipse incluye un depurador potente, sencillo y muy cómodo de utilizar.



Figura 11. Botón depurador.

Lanzar el depurador es una tarea exactamente igual que ejecutar un programa, solo que en lugar de utilizar el botón de ejecución, se utiliza el botón de depuración (ver Figura 11). Estos dos botones, y los menús que despliegan, tienen un comportamiento exactamente idéntico (salvo por el hecho de que el botón de depuración provoca la ejecución paso a paso de los programas).

Nota.

Cuando se lanza el proceso de depuración, siempre se realiza una compilación y construcción completa del código.

Cuando el depurador entra en acción, de forma automática, se abre la *Perspectiva Depuración* (Figura 12), en la que se muestra toda la información relativa al programa que se está depurando.

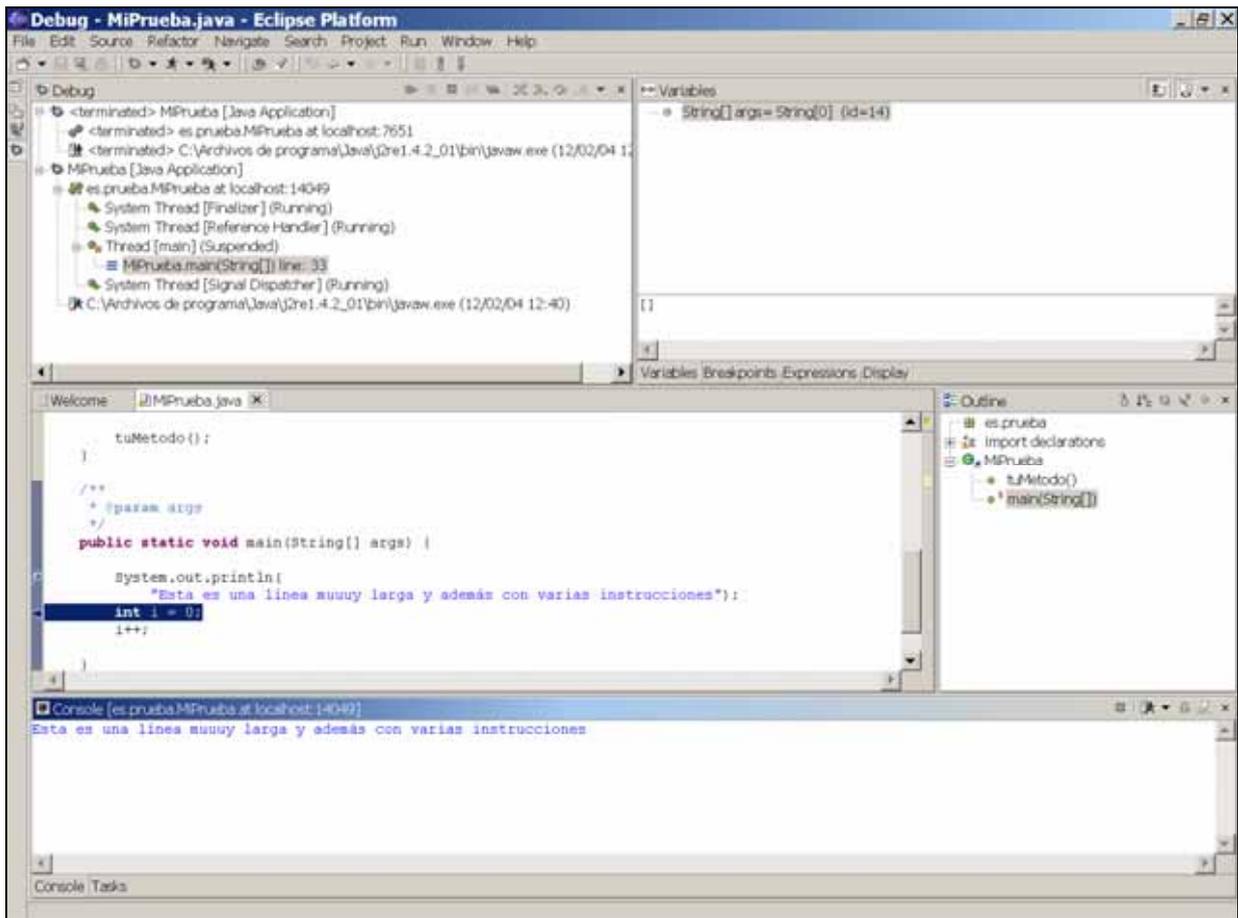


Figura 12. Perspectiva Debug.

Vista Editor

La ventana más importante, situada en el centro de la pantalla, sigue siendo el Editor. Sobre el Editor se irá marcando la traza de ejecución del programa con una pequeña flecha azul, situada sobre el margen izquierdo. La línea sobre la que esté dicha flecha, será la próxima en ejecutarse.

Cuando la Perspectiva Debug está activa, el menú contextual del Editor cambia para mostrar opciones de depuración, por ejemplo, ejecutar el programa hasta la línea que tiene el cursor, inspeccionar una variable (o una expresión seleccionada), etc.

Vistas de Inspección

En la parte superior izquierda, se puede ver la ventana *Debug*. En esta ventana es donde se controla la ejecución del programa que se está depurando ya que contiene la barra de botones de ejecución. En esta barra están los clásicos botones para detener la depuración, ejecutar hasta el final, ejecutar paso a paso, etc.

Truco.

Es más útil controlar la ejecución del programa utilizando las teclas. Con F5 se ejecuta paso a paso el programa y con F6 se ejecuta una función sin entrar a ejecutarla paso a paso.

Además, esta ventana también muestra información a cerca de los hilos (threads) activos y de los procesos de depuración realizados con anterioridad.

Atención.

Muchas veces, por prisa o por descuido, se lanzan nuevos procesos de depuración sin detener los anteriores (sobre todo, cuando se entra en una vorágine de cambios con el objetivo de solucionar un bug que se resiste más de lo esperado). Si los procesos abiertos se apilan demasiado, se puede agotar la memoria. Para evitarlo, de vez en cuando, se puede echar una ojeada a la perspectiva *Debug* y comprobar que todos los hilos tengan el estado [terminated], y si no lo tienen, finalizarlos (con el botón que detiene la depuración y el hilo previamente seleccionado en la vista *Debug*).

La vista de inspección (a la derecha de la vista *Debug*), permite ver los valores de los diferentes elementos (variables, breakpoints, expresiones...) que intervienen en el programa, en un instante de ejecución determinado.

Las dos vistas más interesantes son la vista de inspección de variables, que muestra los valores que toman todas las variables (atributos, campos, etc.) cuyo ámbito alcanza a la línea que se está ejecutando en un momento dado y la lista de inspección de breakpoints.

Establecer un breakpoint es tan sencillo como hacer doble clic en el margen izquierdo del Editor del código, a la altura de la línea sobre la que se quiere detener la ejecución. El breakpoint creado quedará identificado por un punto azul (ver Figura 12) sobre la línea.

La vista de inspección de breakpoints permite, además de ver todos los breakpoints definidos, configurar sus propiedades. A través del menú pop-up de la vista se puede activar o desactivar un breakpoint, eliminarlo, configurarlo para que detenga la ejecución cuando se pase por él un determinado número de veces o cuando lo haga un hilo en concreto, etc.

Vista Consola

Por último, en la parte inferior de la Perspectiva Debug, se muestra la consola. La consola es la vista sobre la cual se redirecciona tanto la entrada como la salida estándar, del programa que se está depurando (o ejecutando).

Documentación

En este apartado se hablará de todos los aspectos relativos a la documentación en el entorno Eclipse, tanto de la incorporación de archivos de documentación para ser utilizados durante la programación, como de la generación de la documentación de la aplicación.

Configurar el acceso a JavaDocs

El primer valor que se debe configurar es la ubicación de la documentación de las librerías Java estándar. Esta configuración se define para todo el entorno y será accesible para cualquier proyecto con el que se trabaje.

La configuración de la documentación estándar se realiza en la ventana de configuración de los posibles JRE que puede utilizar Eclipse, a la cual se accede desde la opción “*Window → Preferences → Java → Installed JREs*” del menú principal.

En esta ventana, se puede ver una lista de los JREs disponibles, para configurar la documentación de alguno de ellos, basta con seleccionarlo y utilizar el botón “*Edit...*”. En el wizard que aparece (Figura 13) habrá una caja de texto donde se puede introducir la dirección de la carpeta de documentación correspondiente.

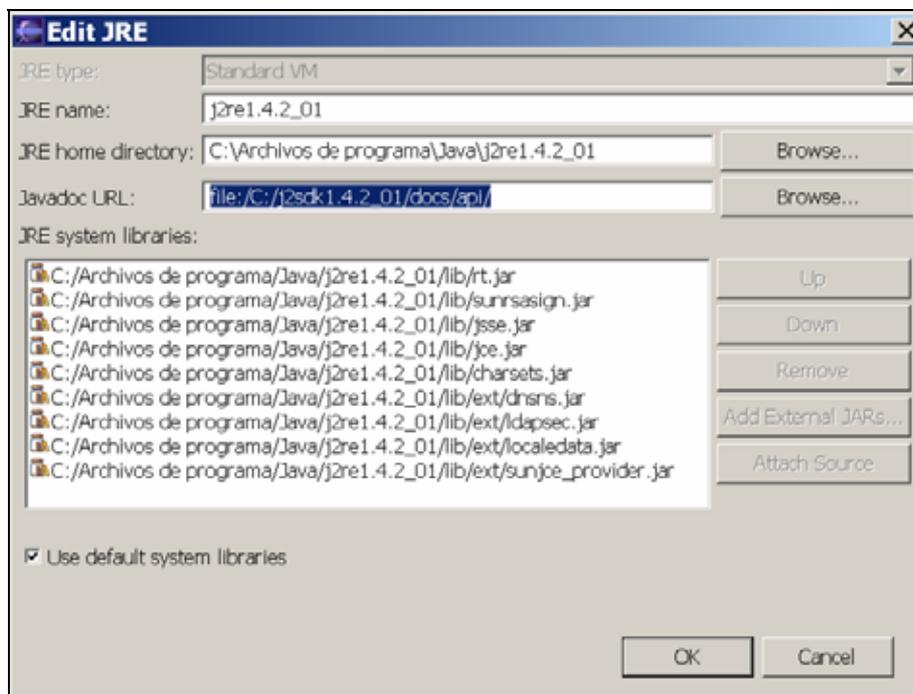


Figura 13. Configuración de un JRE.

Además de las librerías estándar de Java, es frecuente que en los proyectos se utilicen otras muchas librerías, que tengan su propia documentación. Eclipse permite integrar en

su sistema de ayuda cualquier JavaDoc relacionado con las librerías que se utilicen en un proyecto.

La configuración de la documentación de las librerías que utiliza el proyecto se realiza desde la ventana de configuración de las propias librerías *en Project → Properties → Java Build Path → Librerías*.

En esta ventana, cada librería (.jar) en el classpath del proyecto, aparece como una entrada que tiene dos propiedades (se muestran pulsando el signo “+” al lado de su nombre): la ubicación de su documentación y la ubicación de su código.

Para poder utilizar la documentación de una librería, basta con escribir su ubicación en el lugar correspondiente.

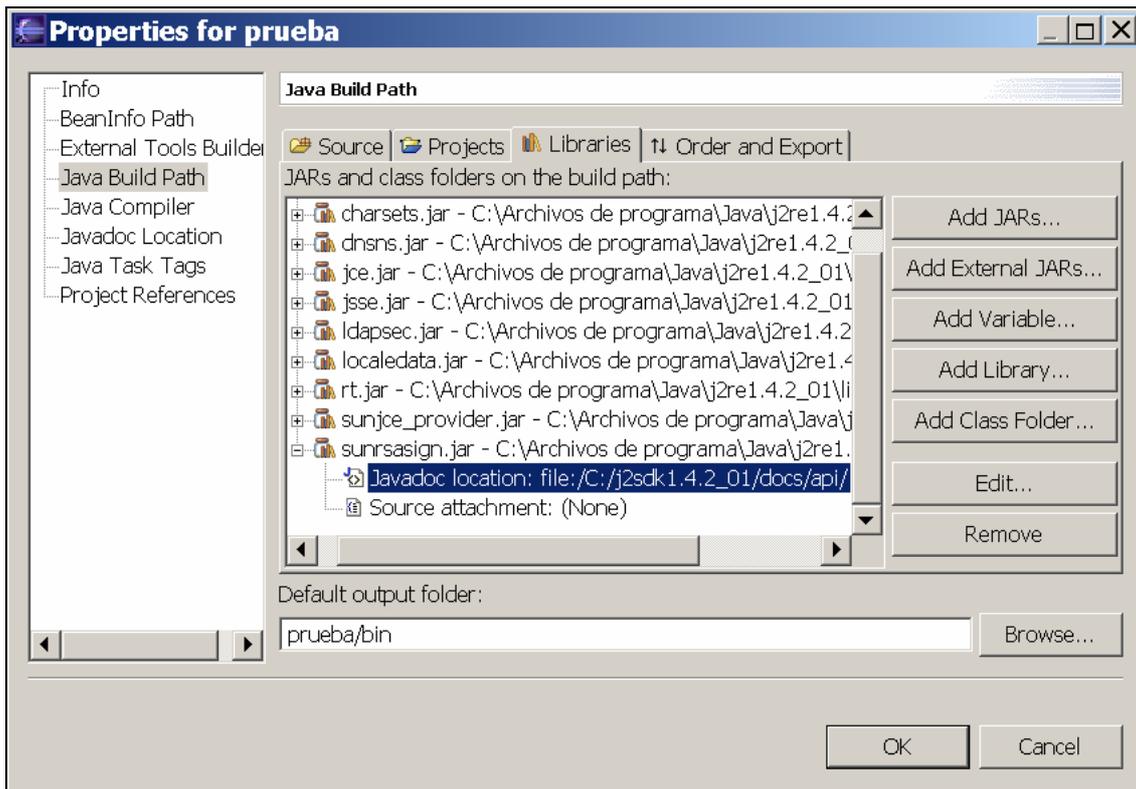


Figura 14. Configuración de la documentación de una librería.

En la Figura 14 se muestra el lugar, en la ventana de Propiedades del proyecto, donde se puede configurar la documentación de una librería concreta.

Por último, también es interesante poder acceder y consultar la propia documentación JavaDoc del proyecto que se está implementando. La configuración de este tipo de documentación también se realiza desde la ventana de propiedades del proyecto (*Project → Properties*), en el apartado “*Javadoc Location*” (Figura 15).

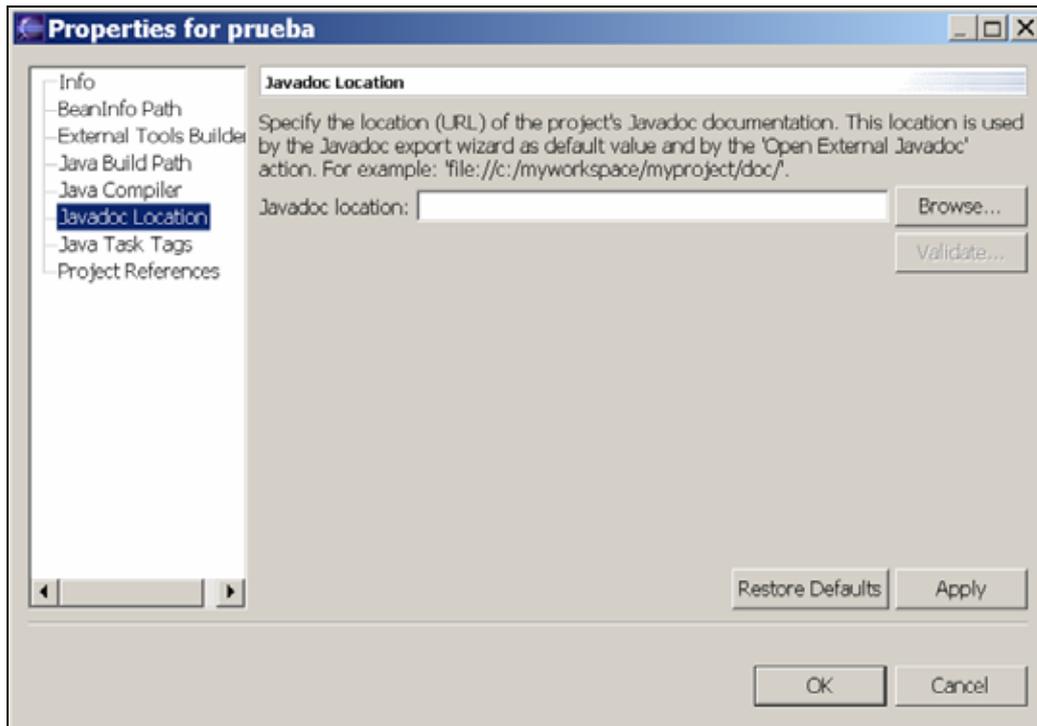


Figura 15. Configuración del Javadoc del proyecto.

Utilizar la documentación

Una vez que todas las posibles fuentes de documentación del proyecto han sido configuradas acceder a ellas es lo más sencillo, basta con seleccionar, en el Editor, el elemento que se quiere consultar y pulsar F1. La ayuda se desplegará, en formato HTML, en el navegador integrado de Eclipse (Figura 16).

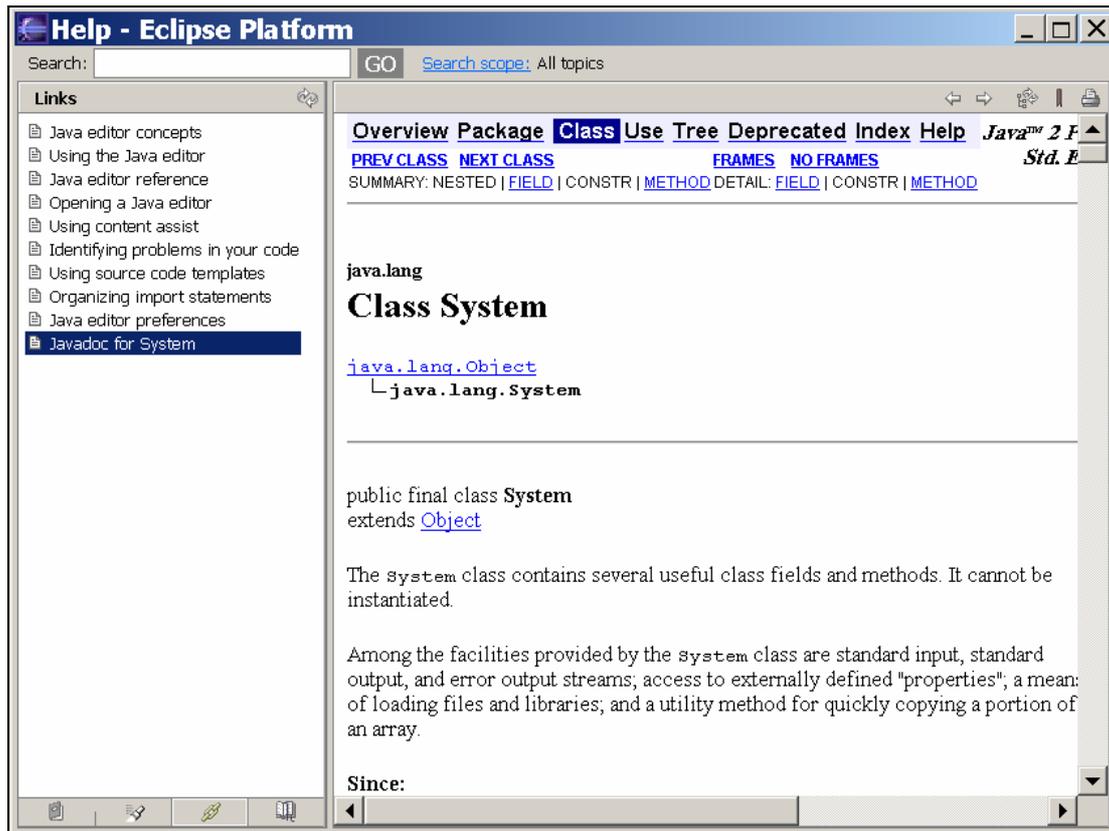


Figura 16. Ayuda en línea de Eclipse.

Generar JavaDoc del proyecto

Además de poder consultar las diferentes fuentes de documentación javadoc que maneja el proyecto, Eclipse permite, de una forma muy sencilla generar, automáticamente, la documentación del propio proyecto.

Antes de poder crear los ficheros de documentación, es necesario configurar la herramienta JavaDoc que Eclipse debe utilizar. Para ello basta con escribir la ubicación del ejecutable javadoc en la opción “*Window → Preferences → Java → Javadoc*” accesible desde el menú principal (Figura 17).

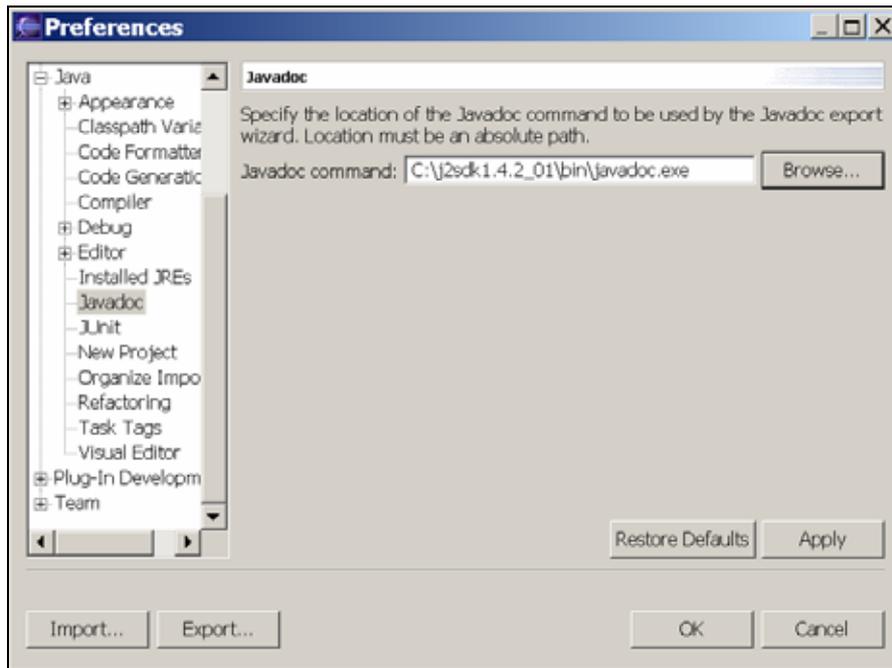


Figura 17. Configuración del ejecutable javadoc.

Para generar la documentación del proyecto, en Eclipse, se utiliza el wizard de “*exportación de javadoc*”. Se puede acceder a este wizard seleccionado, en la Vista *Resources*, el proyecto y, a través de la entrada “*Export...*” del menú pop-up, escogiendo la opción “*Javadoc*”.

El wizard está compuesto de tres ventanas cuya utilización es bien sencilla. En la primera de ellas se indicarán las clases y paquetes para los cuales se va a generar documentación, el nivel de protección de los miembros cuya documentación se escribirá y el Doclet que se va a utilizar (Figura 18).

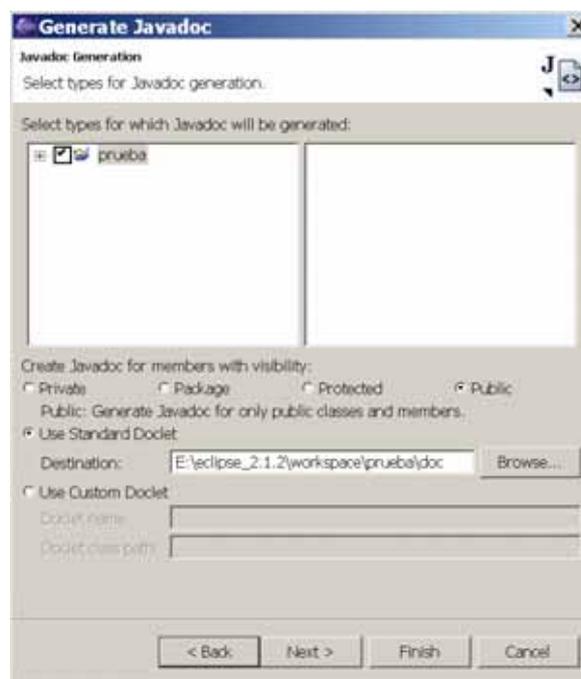


Figura 18. Exportar Javadoc (I)

En la segunda de las pantallas del wizard se configura el título de la documentación, las opciones de generación, el las referencias (enlaces) que se deben generar y la hoja de estilos a aplicar (Figura 19).

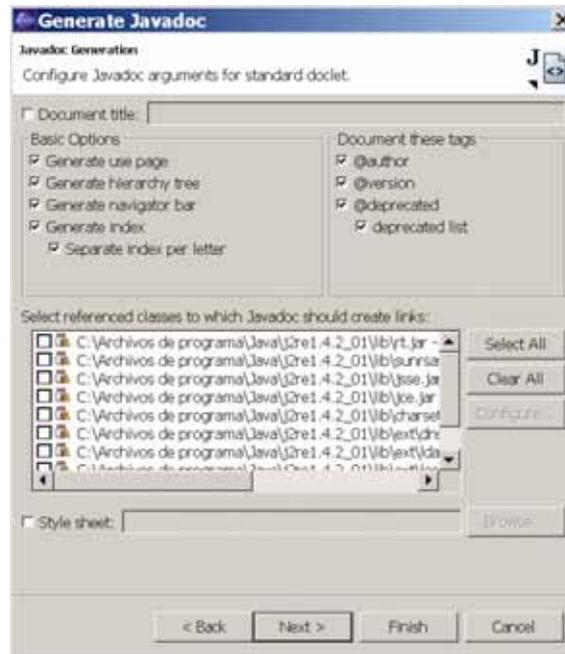


Figura 19. Exportar Javadoc (II)

La tercera, y última, pantalla se destina a la configuración de la página web que se utilizará como portada y a la adición de cualquier otro parámetro que se le quiera pasar al Doclet que generará la documentación (Figura 20).



Figura 20. Exportar Javadoc (III)

Pruebas

Eclipse facilita la tarea de crear y ejecutar pruebas unitarias utilizando el framework JUnit [www.junit.org].

Para poder utilizar el framework JUnit, es necesario colocar la librería *junit.jar* en el classpath del proyecto.

El plugin JDT incluye un wizard para la creación de casos de prueba (JUnit Test Cases) muy similar al propio wizard de creación de clases, explicado en este mismo documento.

Crear un nuevo TestCase, como decía, es muy similar a crear una nueva clase. De igual forma, se puede utilizar el botón de creación de clases (en la barra de herramientas principal, con la Perspectiva Java activa). En el menú desplegable que se muestra, en lugar de seleccionar Class o Interface, como se había explicado, se debe seleccionar la opción TestCase, lo cual mostrará el wizard JUnit.

Este wizard se compone de dos pantallas. La primera de ellas (Figura 21) sirve para realizar una configuración general de la clase de prueba (TestCase), indicando, entre otras cosas: su nombre, el paquete al que pertenecerá, su superclase, los métodos que debe sobrescribir (por ejemplo, Setup(), el main(String[]) , etc.) o la clase sobre la cuál se va a realizar las pruebas.

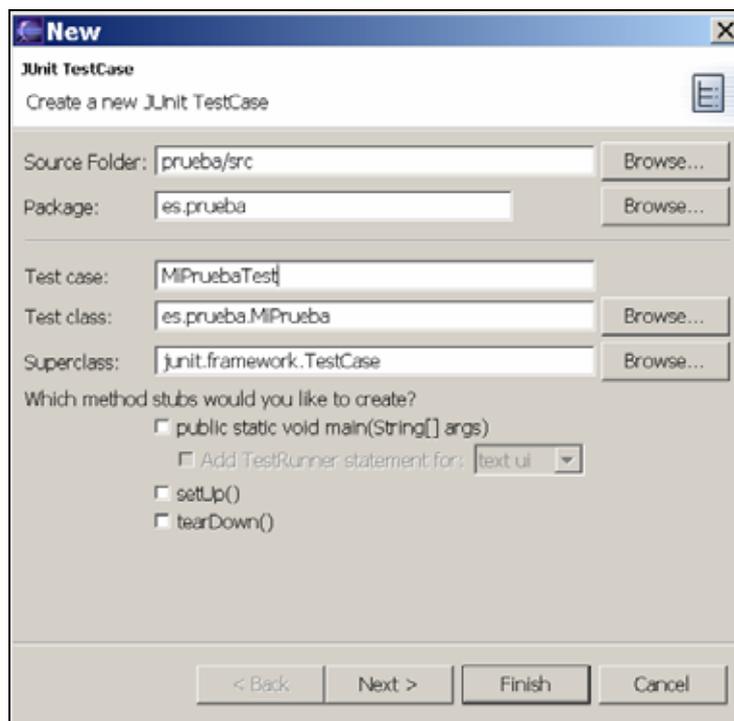


Figura 21. Wizard JUnit (I)

En la segunda pantalla (Figura 22) del wizard JUnit se seleccionarán los métodos para los cuáles se deben generar casos de prueba.

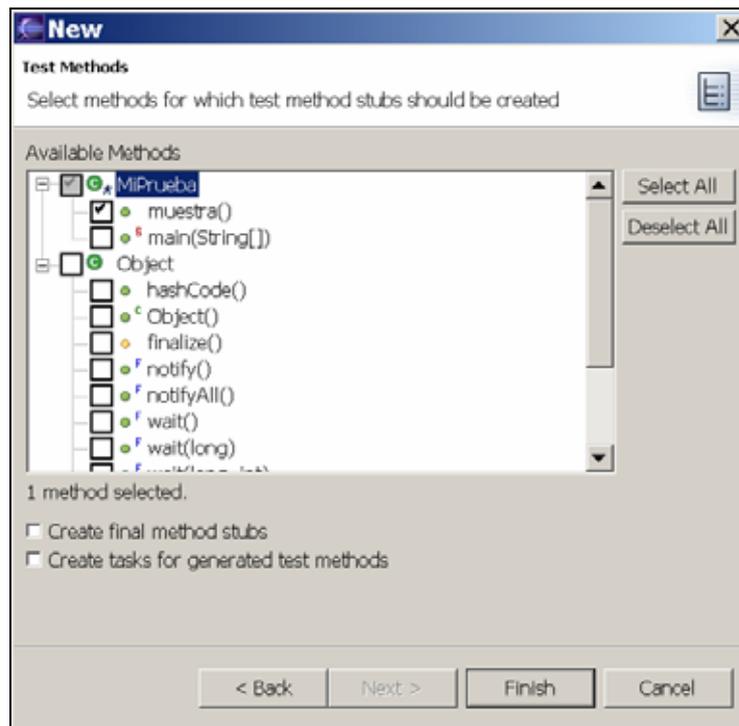


Figura 22. Wizard JUnit (II)

Ejemplo.

A modo de ejemplo, se generará la clase de prueba (TestCase) correspondiente a la clase MiPrueba() que se ha ido definiendo en los ejemplos anteriores.

El código de esta clase será el siguiente:

```
public class MiPrueba {

    public void muestra() {

        System.out.println("Mostrando");
    }

    /**
     * @param args
     */
    public static void main(String[] args) {

        System.out.println(
            "Esta es una línea muuuy larga y además con varias instrucciones");
        int i = 0;
        i++;

    }

}
```

Para crear el caso de prueba, se utilizará el wizard JUnit tal cual se acaba de explicar. En las Figuras 21 y 22, se pueden ver los datos de configuración del caso de prueba de este ejemplo.

El código del Caso de Prueba generado es el siguiente:

```
public class MiPruebaTest extends TestCase {

    /**
     * Constructor for MiPruebaTest.
     * @param arg0
     */
    public MiPruebaTest(String arg0) {
        super(arg0);
    }

    public void testMuestra() {

    }

}
```

Trabajo en equipo con Eclipse.

El IDE Eclipse integra un cliente para el sistema de gestión de versiones CVS.

Tanto el acceso a repositorios compartidos de código, como la navegación por los mismos, en Eclipse, se realiza a través de la Perspectiva CVS (Figura 23), a la cual se puede acceder mediante el menú principal (*Window* → *Open Perspective* → *Other* → *CVS*).

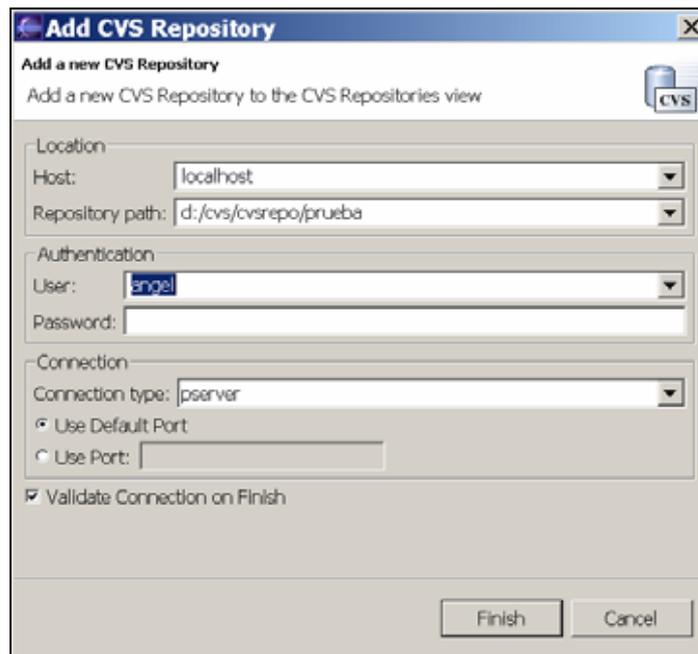


Figura 24. Conexión con Repositorio CVS.

Establecer una conexión CVS es tan sencillo como rellenar los campos del wizard, proporcionando la información necesaria. Se puede ver en la Figura 24 que la información necesaria es: localización del repositorio, autenticación del usuario en dicho repositorio y método de conexión.

Compartir Proyecto

Antes de empezar a trabajar con un repositorio compartido, es necesario asociar el proyecto CVS almacenado, con el proyecto Eclipse con el que se va a trabajar. Existen dos vías para realizar esta asociación:

- a) El proyecto Eclipse no existe en el repositorio y se quiere introducir en el mismo. Para ello es necesario, en cualquier vista (por ejemplo, en la vista *Navigator*) seleccionar el proyecto y, a través del menú pop-up, escoger la opción "*Team → Share Project...*". Se mostrará, entonces, una ventana donde se debe especificar el repositorio en el que se va a almacenar el proyecto, o bien, indicar que se debe crear uno nuevo.
Si se opta por esta última opción, se pasará a la ventana de creación de conexiones con repositorios CVS (Figura 24).
- b) Se quiere extraer un proyecto del repositorio CVS. En este caso, la compartición del proyecto se realiza desde la vista "*CVS Repositories*" de la *Perspectiva CVS*. Se seleccionará la carpeta correspondiente al proyecto con el que se quiere trabajar y, a través del menú pop-up, se escogerá la opción "*Check Out As Project*", si se quiere extraer como un proyecto genérico, o la opción "*Check Out As...*", si lo que se pretende es trabajar con un tipo de proyecto concreto (como un proyecto Java por ejemplo).

Ejecutar comandos CVS

Todas las órdenes que puede recibir el programa de línea de comandos `cv`s son accesibles desde el cliente implementado por Eclipse. Estas órdenes se agrupan en el submenú “*Team*” del menú contextual (pop-up) de cualquier vista que permita navegar por los ficheros que contiene el proyecto (vistas “*Navigator*” y “*Package Explorer*”, por ejemplo).

Estas órdenes son las conocidas: `commit`, `update`, `branch`, `merge`, `tag` as `version` ... Su función es la misma que tienen sus equivalentes en el programa `cv`s. Por ejemplo, la orden `commit`, actualiza el contenido del repositorio guardando los cambios. La orden `update`, realizará la tarea contraria, actualizando el proyecto local con los datos del repositorio.

Control Cambios

Eclipse mantiene, de forma local, un registro de los cambios realizados en cualquier archivo del proyecto. Es posible comprobar los cambios que se han ido realizando y, en caso de error, volver a versiones anteriores.

El acceso a la pantalla de control de versiones local (Figura 25) se realiza a través del menú contextual (pop up) en cualquier vista que permita ver los ficheros del proyecto (*Navigator* o *Package Explorer*) en la entrada “*Replace With → Local History...*”, teniendo seleccionado el fichero cuyo historial se quiere comparar.

Esta última opción permite ver los cambios realizados y, en caso de necesidad, volver a una situación anterior. Si la intención es únicamente comprobar el código en versiones anteriores, y no hacer ningún cambio, se puede utilizar la orden “*Compare With*” del menú pop up.

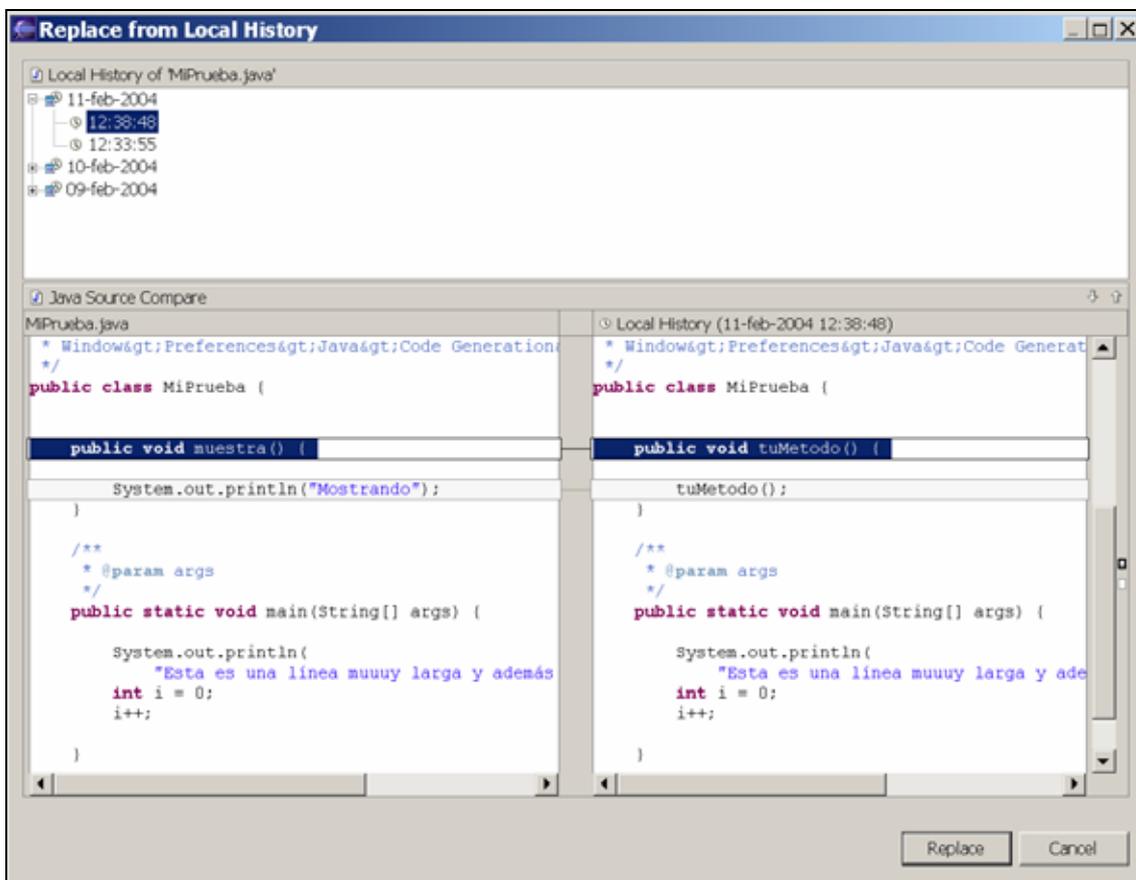


Figura 25. Historial local de cambios.

El interfaz de comparación y recuperación de versiones (Figura 25), puede utilizarse también sobre los ficheros del proyecto que están contenidos en el repositorio CVS, es decir, que no sólo es útil para el control de cambios en las versiones locales del proyecto. Para lanzar la ventana de control de cambios (Figura 25) para un fichero del repositorio CVS, desde la vista “*CVS Repositories*”, con el fichero a comparar seleccionado, se debe llamar a la opción “*Compare With...*” del menú contextual.

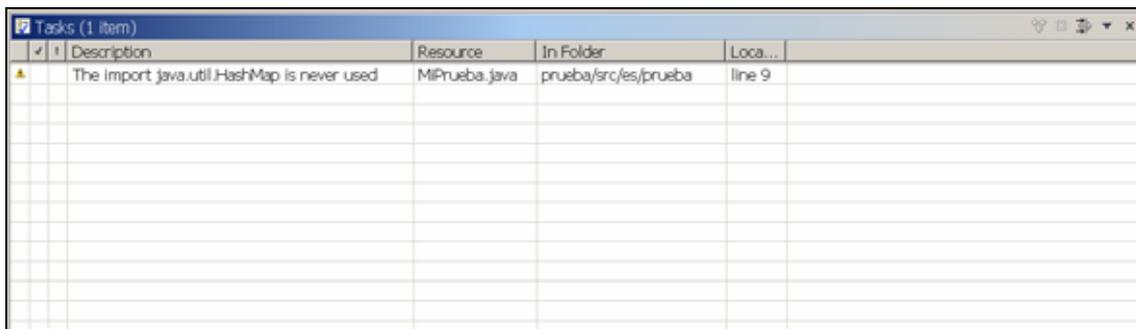
Otras herramientas de interés.

En la distribución del IDE Eclipse, además de las herramientas de programación otras utilidades estándar para realizar tareas habituales tales como gestionar tareas, realizar búsquedas sobre el código fuente, simplificar la navegación, etc.

Gestor de Tareas.

El entorno Eclipse incluye una vista, de nombre *Tasks*, que sirve para gestionar las tareas pendientes en un proyecto.

La vista *Tasks* (Figura 26), actúa también como B.O.E (Boletín Oficial de Eclipse), ya que es el medio a través del cual, Eclipse notifica cualquier error, advertencia, etc. que detecte a la hora de compilar, generar código o de realizar cualquier tarea de forma automática.



| Description | Resource | In Folder | Loca... |
|--|---------------|----------------------|---------|
| The import java.util.HashMap is never used | MIPrueba.java | prueba/src/es/prueba | line 9 |

Figura 26

Cuando Eclipse compile una clase, si detecta algún error de compilación, o algún warning, este evento se notificará, en la vista *Tasks*, como una tarea más. Lo mismo ocurre cuando se escribe, de forma automática, esqueletos de código o documentación que deben ser rellenados por el programador.

Además de las tareas introducidas automáticamente por Eclipse, es posible que el programador introduzca las suyas propias. Existen dos formas de introducir tareas:

- Mediante el menú contextual de la vista *Tasks*, seleccionando la opción “*New Task*”.
- Utilizando etiquetas (*Tags*) que Eclipse reconocerá como entradas para el gestor de tareas (Figura 27). Las *Tags* son etiquetas, palabras, que cuando se escriben dentro de un comentario, Eclipse las asocia directamente con una tarea. Por defecto se define la etiqueta *TODO* (To Do, “Por hacer” en inglés), pero es posible definir etiquetas personalizadas en la ventana de configuración del proyecto (*Project* → *Properties*), en la opción “*Java Task Tags*”.

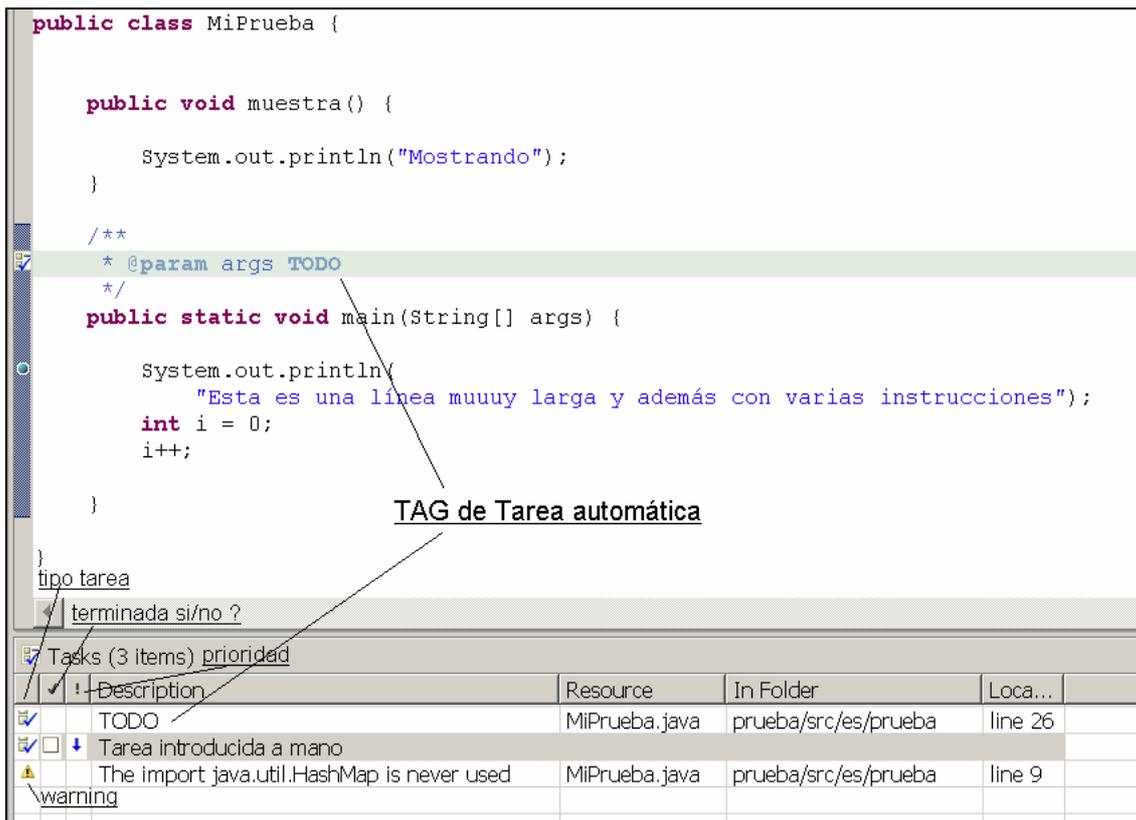


Figura 27

En el menú contextual (pop up) de la vista *Tasks*, además de introducir nuevas tareas, es posible purgar las tareas completadas, establecer filtros para que solamente se muestren las apropiadas, etc.

Búsqueda semántica.

Prácticamente todos (por no decir todos) los editores y entornos de desarrollo disponen de un mecanismo de búsqueda más o menos completo. La diferencia del motor de búsquedas que implementa Eclipse, respecto al de otros entornos, es que éste no se limita únicamente a buscar coincidencias sintácticas en los archivos del proyecto (como hacen la mayoría) sino que dispone de opciones de búsqueda semántica.

Las opciones de búsqueda semántica son las siguientes:

- Buscar los lugares del código donde se hace referencia a un tipo, o clase, determinados.
- Buscar la definición de una clase concreta.
- Buscar las clases que implementan un interfaz definido.
- Buscar los lugares en los cuales se accede para lectura, o escritura, a una variable, o atributo, determinados.

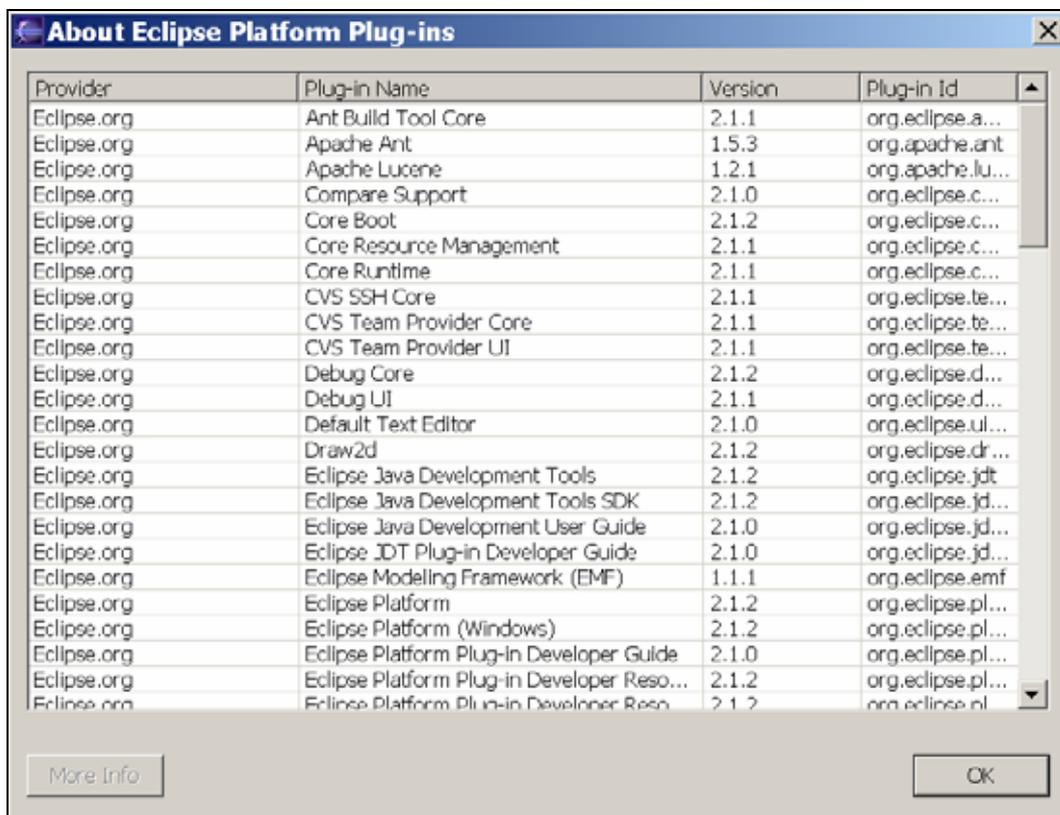
Todas estas opciones están accesibles en el menú principal, en la entrada *Search*.

Algunos Plugins interesantes.

Todas las opciones que se han comentado hasta ahora, en este documento, están implementadas en la versión estándar del IDE Eclipse.

Eclipse permite extender esta funcionalidad básica gracias a su arquitectura de plugins. Instalar un plugin es muy sencillo, basta con descomprimir el archivo del plugin (generalmente un .zip) en la carpeta *Eclipse/Plugins*. Una vez “instalado” el plugin de esta manera, la siguiente vez que se ejecute Eclipse, éste se encargará de realizar la carga de todos los nuevos plugins añadidos.

Es posible saber, qué plugins están activos en un momento determinado., para ello, es necesario lanzar la ayuda de Eclipse (desde el menú principal, seleccionado la entrada “*Help* → *About Eclipse Platform*”). Desde la ventana que se despliega, se puede ver la lista de plugins (Figura 28) pulsando el botón “*Plug-in Details*”.



| Provider | Plug-in Name | Version | Plug-in Id |
|-------------|--|---------|--------------------|
| Eclipse.org | Ant Build Tool Core | 2.1.1 | org.eclipse.a... |
| Eclipse.org | Apache Ant | 1.5.3 | org.apache.ant... |
| Eclipse.org | Apache Lucene | 1.2.1 | org.apache.lu... |
| Eclipse.org | Compare Support | 2.1.0 | org.eclipse.c... |
| Eclipse.org | Core Boot | 2.1.2 | org.eclipse.c... |
| Eclipse.org | Core Resource Management | 2.1.1 | org.eclipse.c... |
| Eclipse.org | Core Runtime | 2.1.1 | org.eclipse.c... |
| Eclipse.org | CVS SSH Core | 2.1.1 | org.eclipse.te... |
| Eclipse.org | CVS Team Provider Core | 2.1.1 | org.eclipse.te... |
| Eclipse.org | CVS Team Provider UI | 2.1.1 | org.eclipse.te... |
| Eclipse.org | Debug Core | 2.1.2 | org.eclipse.d... |
| Eclipse.org | Debug UI | 2.1.1 | org.eclipse.d... |
| Eclipse.org | Default Text Editor | 2.1.0 | org.eclipse.ul... |
| Eclipse.org | Draw2d | 2.1.2 | org.eclipse.dr... |
| Eclipse.org | Eclipse Java Development Tools | 2.1.2 | org.eclipse.jdt... |
| Eclipse.org | Eclipse Java Development Tools SDK | 2.1.2 | org.eclipse.jdt... |
| Eclipse.org | Eclipse Java Development User Guide | 2.1.0 | org.eclipse.jd... |
| Eclipse.org | Eclipse JDT Plug-in Developer Guide | 2.1.0 | org.eclipse.jd... |
| Eclipse.org | Eclipse Modeling Framework (EMF) | 1.1.1 | org.eclipse.emf... |
| Eclipse.org | Eclipse Platform | 2.1.2 | org.eclipse.pl... |
| Eclipse.org | Eclipse Platform (Windows) | 2.1.2 | org.eclipse.pl... |
| Eclipse.org | Eclipse Platform Plug-in Developer Guide | 2.1.0 | org.eclipse.pl... |
| Eclipse.org | Eclipse Platform Plug-in Developer Reso... | 2.1.2 | org.eclipse.pl... |
| Eclipse.org | Eclipse Platform Plug-in Developer Reso... | 2.1.2 | org.eclipse.pl... |

Figura 28

Plugins

Para terminar, se comentan algunos de los plugins más populares de la plataforma Eclipse:

CDT

CDT es el equivalente, para los lenguajes C y C++, al plugin JDT. Prácticamente todo lo escrito, en este documento, referido JDT es aplicable a CDT. Las diferencias más

importantes están en la gestión de la documentación Javadoc (que es específica de la plataforma Java) y en el uso de librerías JUnit (también específicas de Java). Otra diferencia importante es que CDT no puede compilar automáticamente el código, es necesario indicar un fichero Makefile para ello (si no se indica ningún Makefile y existe alguno en el directorio principal del proyecto, lo utilizará). CDT es un plugin oficial de la plataforma Eclipse y puede ser obtenido de la dirección www.eclipse.org.

VisualEditor

Hasta hace muy poco, la mayor pega que se le achacaba al IDE Eclipse era que no disponía de un editor gráfico de interfaces de usuario. Pues bien, esta pega ya no existe. El Proyecto Eclipse dispone de un plugin, llamado Visual Editor, que permite, de forma sencilla y completamente visual, diseñar interfaces gráficos de usuario (Figura 29).

VE es un plugin oficial de Eclipse y puede obtenerse de la dirección www.eclipse.org.

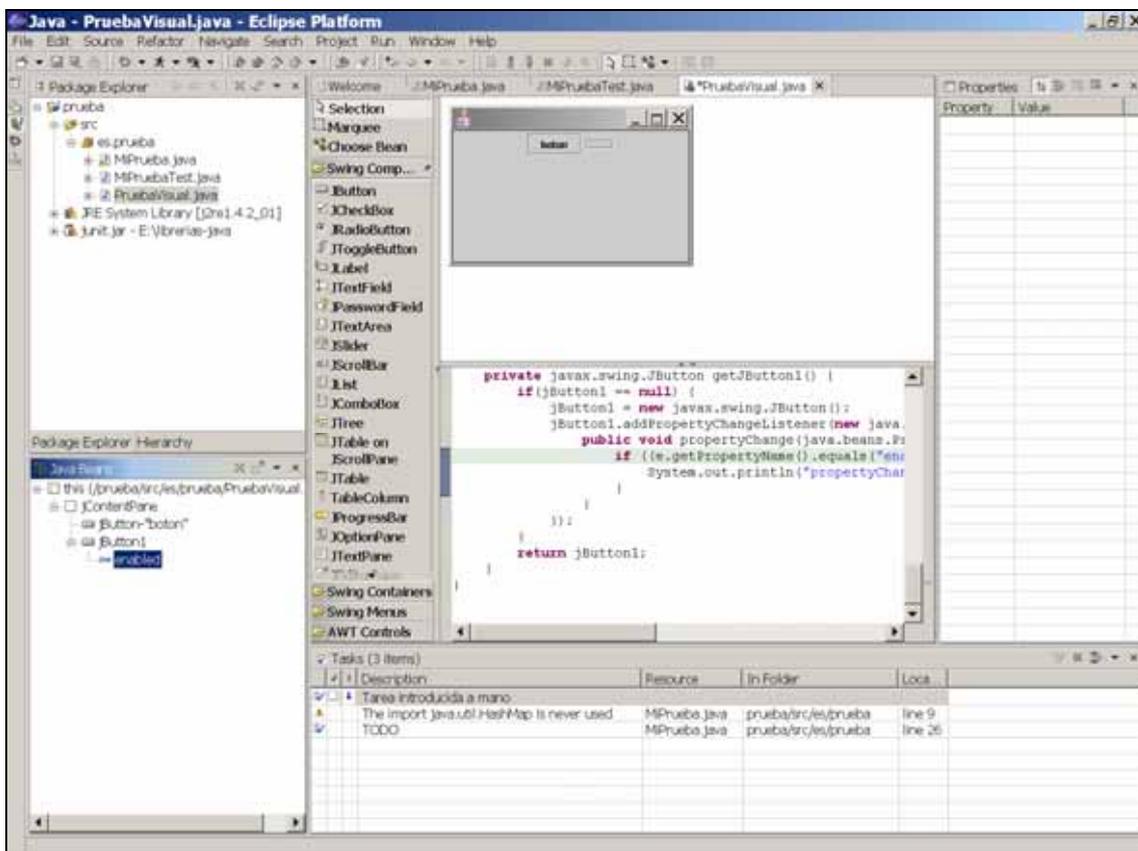


Figura 29. Plugin VE desplegado.

Omondo UML

Los plugins disponibles para Eclipse no se tienen por qué limitar obligatoriamente a la programación. Existen plugins que permiten integrar otras partes del proceso de desarrollo de aplicaciones como, por ejemplo, el diseño. El plugin UML (Figura 30) permite unificar diseño e implementación en una sola herramienta. Cualquier actualización realizada sobre el modelo UML, se verá reflejada en el código fuente de todas las clases a las que dicha modificación afecte, y a la inversa, cualquier cambio en el código se verá reflejado en los diagramas de clases UML.

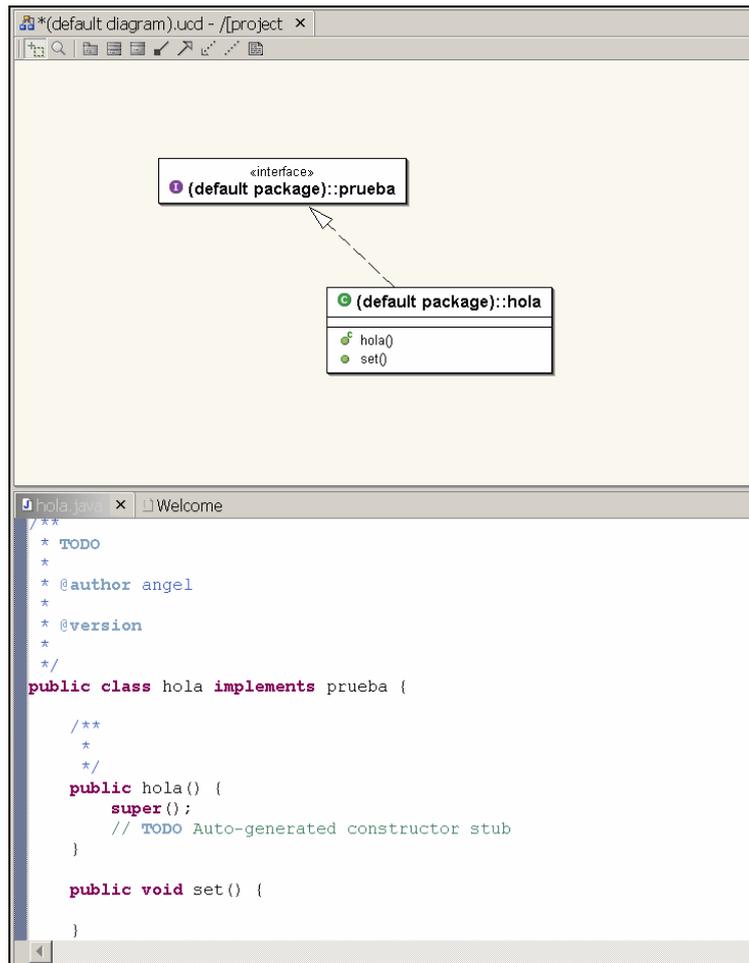


Figura 30. Plugin Omondo UML.

Jalopy

A pesar de que Eclipse incluye una herramienta para formatear el código fuente, es posible, que sus opciones estén algo limitadas para determinados proyectos. Jalopy es un plugin que permite formatear el código, añadir comentarios, etc. pero es mucho más flexible que la herramienta de formateo estándar.

Lomboz

Lomboz es un plugin que facilita el desarrollo de aplicaciones J2EE. Incluye reconocimiento sintáctico para páginas JSP y HTML, capacidad de depurar aplicaciones web (incluyendo páginas JSP), lanzamiento automático de los servidores de aplicaciones más habituales, wizards para crear EJBs y un largo etcétera.

El plugin Lomboz, que es OpenSource desde hace relativamente poco tiempo, se puede obtener en www.objectlearn.com.