





(II.vii. ix) exportar e importar las opciones;

Si utilizamos Netbeans en más de una computadora, es interesante exportar nuestras opciones de configuración con los cambios realizados en el cuadro Herramientas-Opciones y luego importarlas en la otra máquina.

Para exportar:

- (6) Ir a **Herramientas-Opciones**
- (7) Pulsar **Exportar** y elegir el nombre y ubicación del archivo ZIP en el que se guardarán las opciones.
- (8) Seleccionar también la parte del cuadro que queremos exportar
- (9) Aceptar todos los cuadros

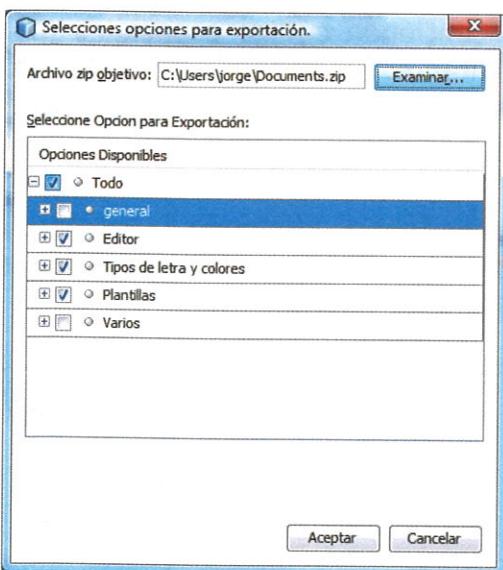


Ilustración 3-41, Cuadro de exportación de opciones de Netbeans

Para importar, hay que disponer del archivo ZIP creado durante la exportación y después volviendo a **Herramientas-Opciones**, elegir **Importar** y seleccionar el archivo. Se nos avisará de que se sobrescribirán las opciones que teníamos hasta ahora. Una vez aceptado el mensaje, Netbeans se reinicia con las nuevas opciones aplicadas.

c> Escribir el nuevo nombre y pulsar **Intro**.

- ◆ **Modificar los parámetros de un método.**
- ◆ **Mover clases.** Simplemente al arrastrar una clase de un paquete a otro en la ventana de proyectos ya nos preguntará sobre el cambio (que implicará muchas modificaciones en el código que Netbeans realizará automáticamente). También podemos elegir **Reestructurar-Mover** encima del elemento que deseamos mover. Esta operación hay que hacerla con cautela, ya que podemos tener problemas al realizarla por la cantidad de elementos involucrados en la posición antigua (no todo puede arreglarlo Netbeans)
- ◆ **Copiar clases.** Desde **Reestructurar-Copiar**. Simplemente pedirá el nuevo nombre y posición de la clase.
- ◆ **Eliminación segura.** Para que al borrar una clase se busque el código que la hacía referencia.
- ◆ Otras opciones de creación de código automático están en el menú Reestructurar, su uso será más evidente conforme conozcamos mejor el lenguaje Java.
- ◆ **Deshacer.** La opción situada en el menú Reestructurar permite deshacer el último cambio de reestructuración realizado.

(II.vii.vii) plantillas generales

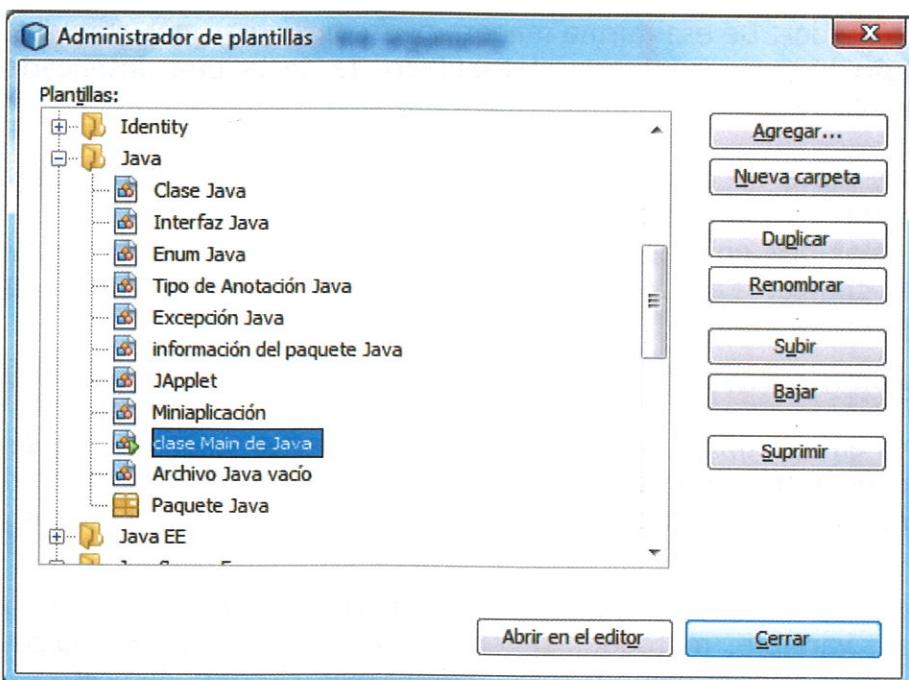


Ilustración 3-39, El cuadro de administración de plantillas

En cuanto escribamos el texto, Netbeans colorea todas las apariciones de ese texto en el archivo actual. Los botones anterior y siguiente tanto en la barra de búsqueda como en la barra superior del código nos permiten movernos por cada aparición del texto buscado en el código.

reemplazar texto

Si deseamos reemplazar texto, basta con pulsar **Ctrl+H** o bien **Edición-Reemplazar**. En este caso aparece un cuadro en el que hay que escribir lo que queremos buscar y el texto que le reemplaza. Podemos ir sustituyendo cada aparición del texto buscado uno a uno (botón **Encontrar** y botón **Sustituir**), o bien pulsar en **Sustituir todo** (estando muy seguros de lo que hacemos) para que se reemplacen de golpe todas las apariciones.

(II.vii.iv) dar formato al código

Desde el menú **Fuente** o desde el botón secundario del ratón, la opción **Formato** reformatea el código de modo que las tabulaciones, aperturas y cierres de llaves, etc. sean las habituales haciendo el código legible.

La forma de dar formato al código se configura en **Herramientas-Opciones** eligiendo **Editor** y luego la pestaña de **Formato**.

(II.vii.v) modificación de opciones del editor

Desde el cuadro **Herramientas-Opciones** eligiendo el botón **Editor** se puede configurar la forma de trabajar del editor de código de Netbeans. Entre ellas:

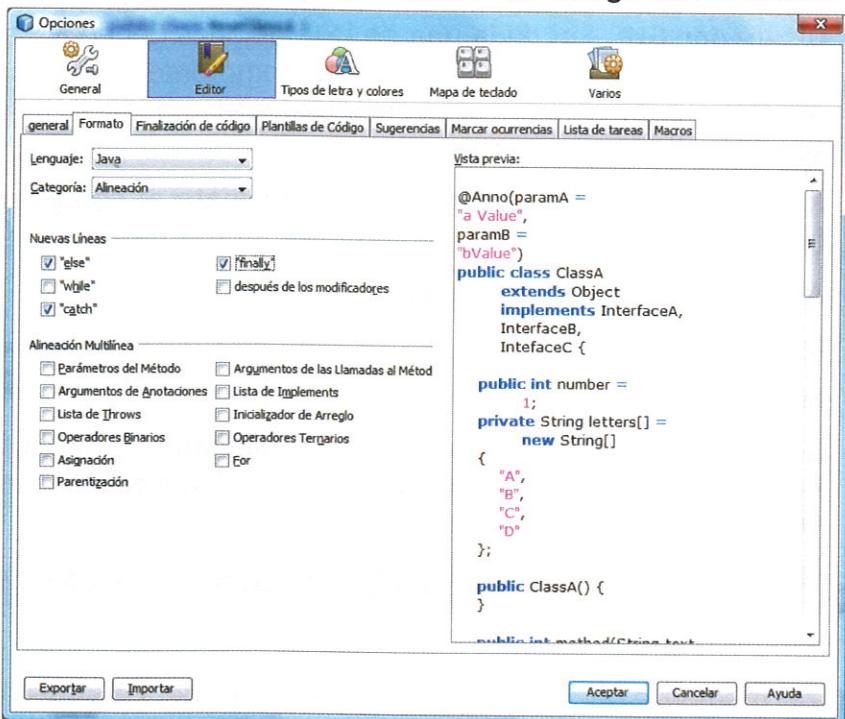


Ilustración 3-38, El cuadro de opciones del editor en el apartado de formato del código

- (8) Inmediatamente se abrirá la documentación en nuestro navegador predeterminado

La documentación **javadoc** se guarda en el directorio **javadoc** dentro del directorio **dist** que es el mismo que sirve para almacenar el archivo **jar** que contiene el proyecto listo para su distribución.

De esa forma la documentación también queda preparada para su distribución.

(II.vii) edición de código

(II.vii.i) aspecto de la ventana de código

Cuando se abre (con doble clic) un archivo que contiene código fuente, o bien cuando se crea, el contenido aparece en la zona dedicada al código. Si queremos maximizar la zona en la que se muestra el código, simplemente basta con hacer doble clic en la pestaña con el nombre del archivo abierto. Un nuevo doble clic en la misma zona devuelve al estado normal el tamaño.

Es posible incluso ver dos archivos a la vez si abrimos ambos y después arrastramos la pestaña de uno de ellos a un lateral.

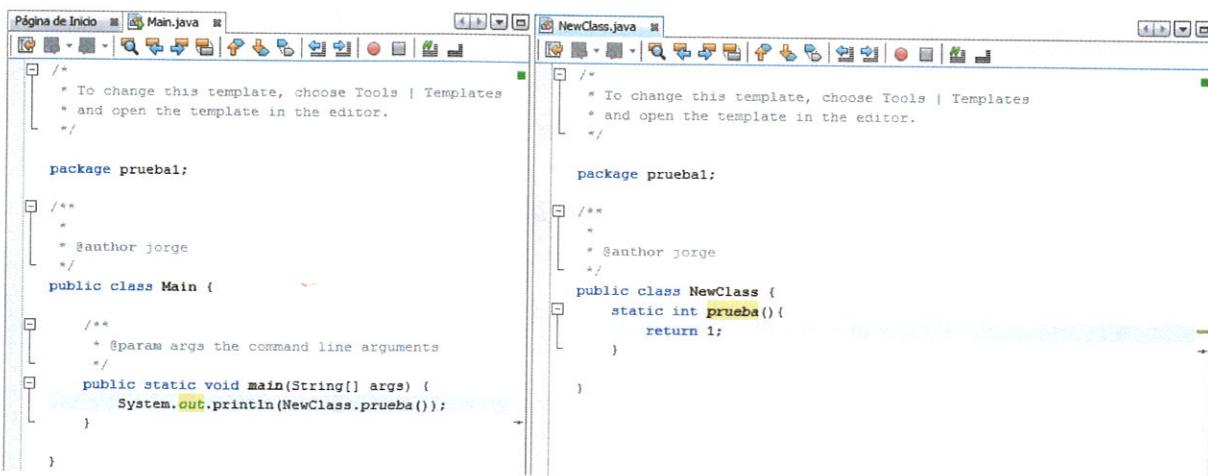


Ilustración 3-37, La ventana de código mostrando dos archivos a la vez

Esta última acción puede ser muy interesante si la hacemos con el propio archivo; es decir consiguiendo ver el mismo archivo en dos pestañas diferentes, a fin de examinar a la vez dos partes distintas del mismo. Para ello primero hay que duplicar el archivo basta con pulsar el botón derecho sobre la pestaña del archivo abierto y elegir **Clonar documento**. Después podremos arrastrar la nueva pestaña a la posición que queramos.

(II.v) compilar y ejecutar programas

(II.v.i) compilar

A diferencia de Eclipse, en Netbeans es necesario compilar cada clase para producir el archivo **class** precompilado correspondiente.

Para compilar basta con pulsar **F9** sobre el archivo a compilar, o bien pulsar el botón secundario del ratón y elegir **Compilar**.

(II.v.ii) ejecutar la clase principal del proyecto

Para probar el proyecto, debemos ejecutarlo. El proyecto principal (será aquel sobre el que marquemos la casilla al crearle) se puede compilar y ejecutar de golpe botón derecho sobre la imagen del proyecto y eligiendo **Ejecutar**, o simplemente pulsando la tecla **F6**. También podemos hacer lo mismo desde el menú **Ejecutar**.

Los otros proyectos no se pueden ejecutar con la tecla **F6** pero sí se pueden probar con **Alt+F6** o podemos pulsar el botón derecho sobre el proyecto y elegir **Ejecutar**.

Un proyecto puede tener varias clases ejecutables (varias clases con **main**) en ese caso para compilar un archivo concreto, basta con seleccionarle en la ventana de proyectos, pulsar el botón derecho y elegir **ejecutar archivo**; también podemos hacer lo mismo si estamos en el código que deseamos ejecutar y pulsamos **Mayúsculas+F6**.

El panel de **Salida** mostrará el resultado de la compilación.

(II.v.iii) preparar la distribución

Desde Netbeans podemos preparar el programa para su distribución e implantación en otras computadoras. Es una de las opciones más interesantes del entorno ya que nos facilita enormemente este cometido.

Para hacerlo basta seleccionar el proyecto que deseamos preparar y después elegir **Ejecutar-Limpiar y generar Main Project**.

Con esta instrucción Netbeans realiza las siguientes tareas:

- (5) Crear una carpeta llamada **dist** en el directorio en el que se almacena el proyecto (esta carpeta es visible desde el panel **Archivos**)
- (6) Comprime todos los archivos compilados en un archivo de tipo **jar** dentro de la propia carpeta **dist**. Los archivos jar son archivos de tipo **zip** que contienen clases java y otros archivos necesarios para la correcta ejecución de una aplicación java.
- (7) Crea un archivo llamado **readme.txt** con información sobre como ejecutar el proyecto (normalmente será **java -jar nombreArchivoJar**)

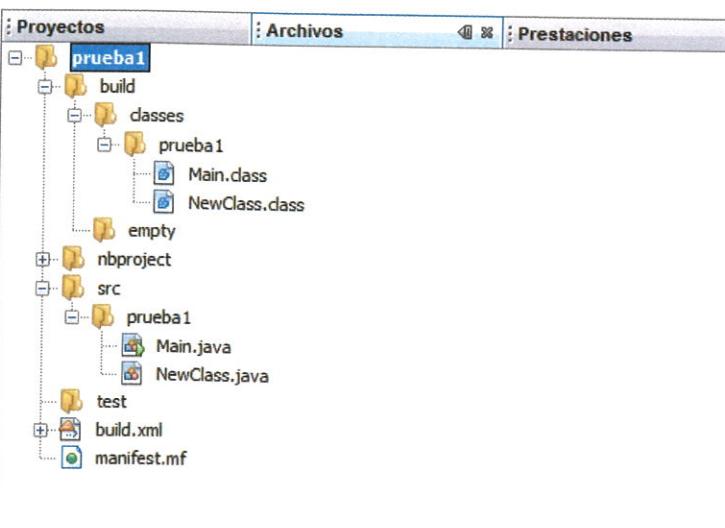


Ilustración 3-33, Ventana de archivos del proyecto

La carpeta **build** contiene los archivos compilados (los **class**). La carpeta **src** el código fuente. El resto son archivos por Netbeans para comprobar la configuración del proyecto o los archivos necesarios para la correcta interpretación del código en otros sistemas (en cualquier caso no hay que borrarlos).

(II.iv.vi) propiedades del proyecto

Pulsando el botón derecho sobre un proyecto, podemos elegir **Propiedades**. El panel de propiedades nos permite modificar la configuración del proyecto. Por ejemplo podremos indicar una nueva clase principal en el proyecto, si fuera necesario o bien indicar los argumentos o parámetros que enviaremos al programa

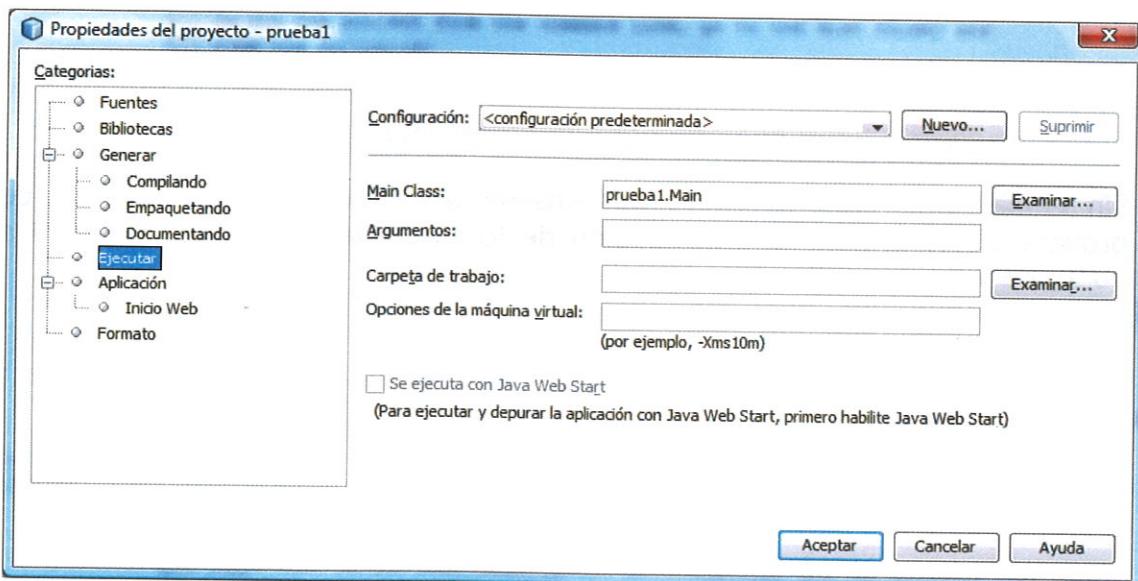


Ilustración 3-34, Cuadro de propiedades abierto por las opciones de ejecución

(II.iv.iii) crear paquetes

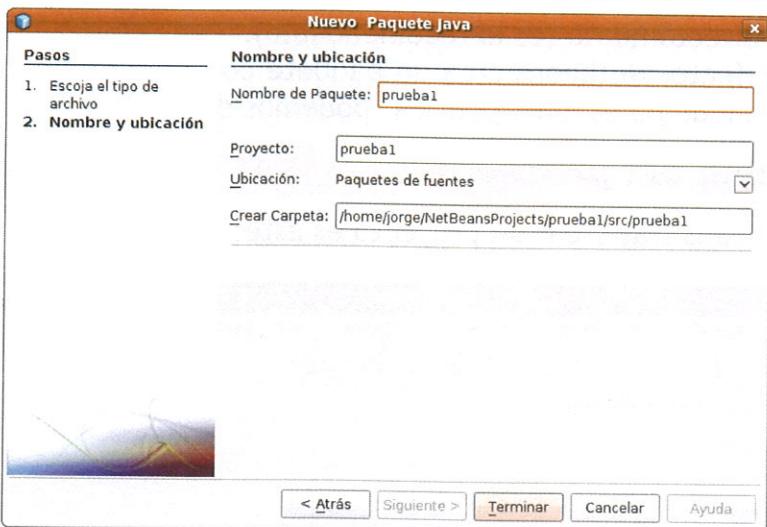


Ilustración 3-30, Cuadro de creación de nuevos paquetes

Las clases se organizan en paquetes. Es recomendable que cada proyecto tenga al menos un paquete, de otro modo las clases se crearían en el paquete por defecto y eso no es aconsejable.

Pulsando el botón derecho en la zona del proyecto donde queremos crear el paquete (normalmente dentro de **paquetes de fuentes**) podremos elegir **Nuevo-Paquete**. Tras llenar el cuadro siguiente (donde simplemente se elige el nombre para el paquete, siempre en minúsculas) tendremos un nuevo paquete.

Más adelante en este manual se explicara con detalle la utilidad de los paquetes

(II.iv.iv) crear nuevas clases

Simplemente se crean pulsando el botón derecho en el paquete en el que deseamos almacenar la clase y eligiendo **Nuevo-Clase**.

Al principio lo normal es que nuestros proyectos sólo contengan la clase principal. A medida que profundicemos en el aprendizaje de Java, necesitaremos más clases (y más paquetes) en cada proyecto.

(II.iv) proyectos de Netbeans

(II.iv.i) crear proyectos

Una vez aceptada la pantalla de bienvenida de Netbeans, podremos crear proyectos desde **Archivo-Proyecto nuevo** o desde el botón  en la barra de herramientas.



Ilustración 3-27, Selección del tipo de proyecto en Netbeans

En la ventana **Nuevo proyecto** hay que seleccionar aplicación Java. Después podremos elegir el nombre y las opciones del proyecto:

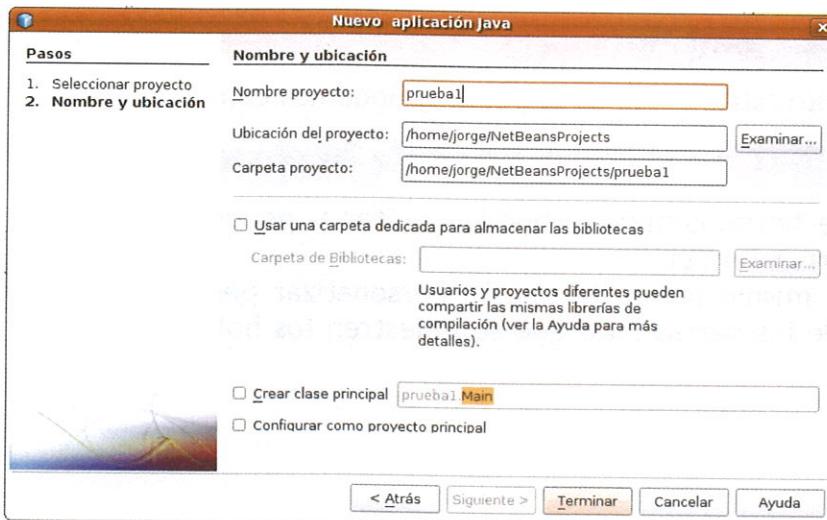


Ilustración 3-28, Selección de las opciones del proyecto

Dentro de las opciones podemos elegir crear la clase principal. La clase principal del proyecto es la que posee el método main, por lo tanto será la

(II.ii.i) comprobación de la plataforma Java instalada

En nuestro ordenador podemos tener varias instalaciones del SDK de Java. Para saber cuáles ha detectado y con cuáles trabaja Netbeans, podemos elegir **Herramientas-Plataformas Java**.

Desde este cuadro podremos configurar la plataforma que se utilice con Netbeans.

(II.iii) aspecto inicial de Netbeans

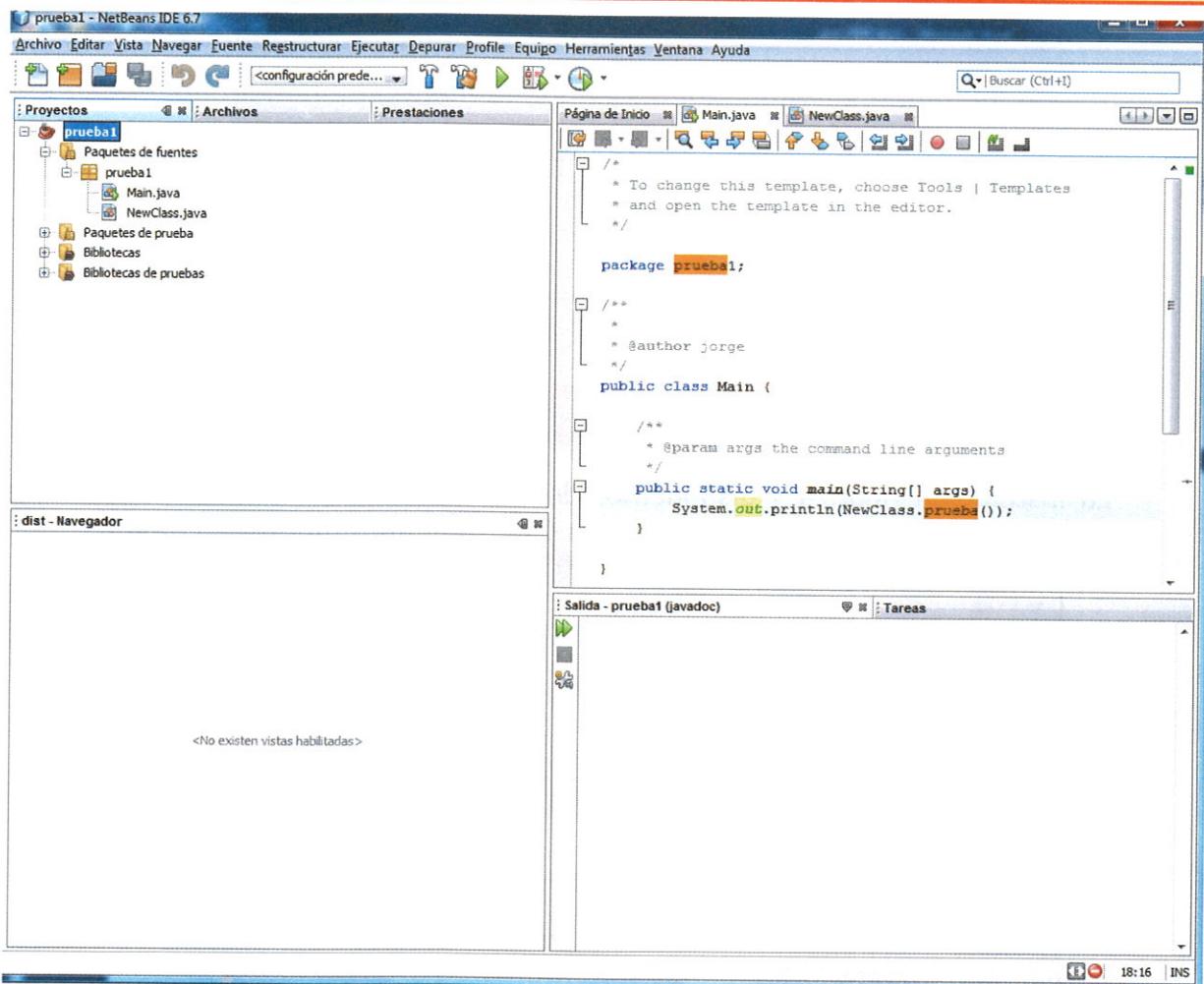


Ilustración 3-26, Aspecto habitual de Netbeans

Una vez cerrada la ventana de bienvenida de Netbeans, el aspecto de la misma es similar al de Eclipse. Los elementos fundamentales son:

- ◆ **Menú principal.** Se trata del menú superior de la aplicación al estilo de cualquier aplicación gráfica
- ◆ **Barras de herramientas.** Situadas debajo del menú principal, en ella se colocan los botones más útiles.

Tanto en Windows como en Linux se descarga un archivo ejecutable que se encarga de la instalación:

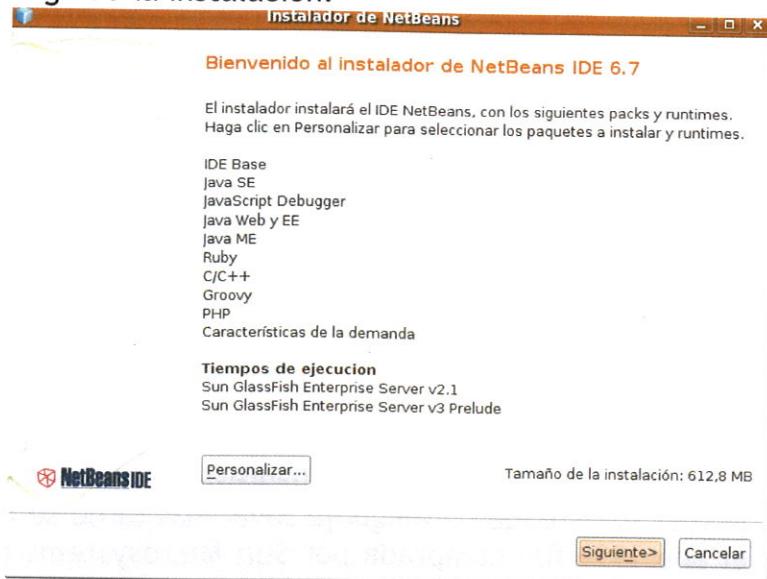


Ilustración 3-22, Pantalla inicial de instalación de Netbeans

Seleccionamos los componentes a instalar (para Java, al menos hay que instalar el IDE Base y Java SE).

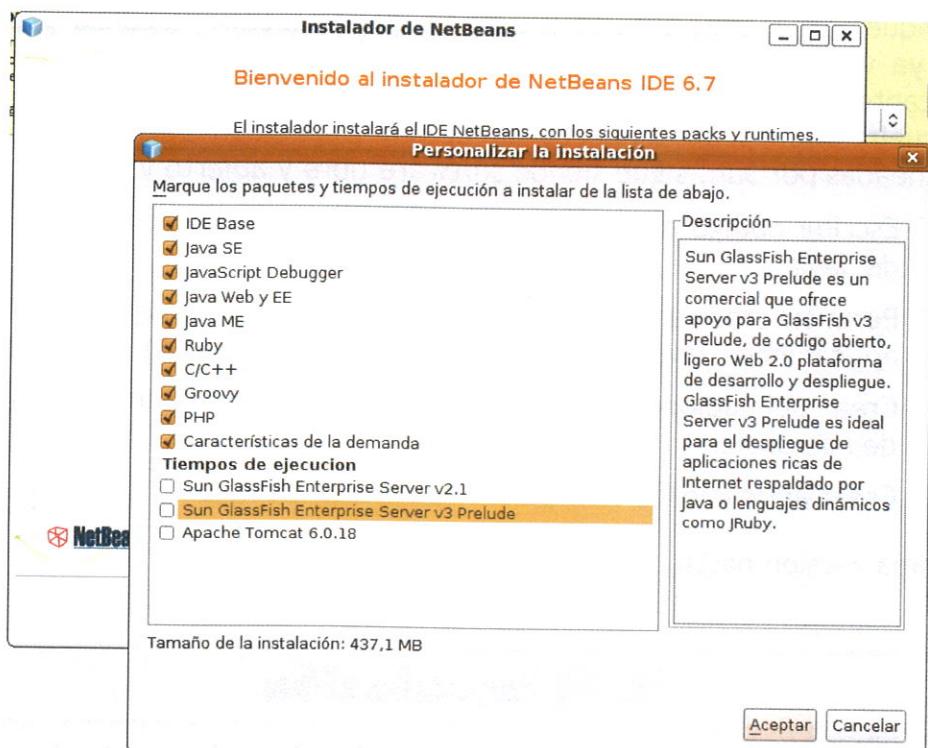


Ilustración 3-23, Selección de los componentes Netbeans a instalar



(I.viii) creación de javadoc con Eclipse

En el caso de la generación de la documentación Javadoc con Eclipse, es muy sencillo. Basta con seleccionar las clases (o el proyecto) del que se desea realizar Javadoc y elegir **Proyecto-Generar javadoc**.

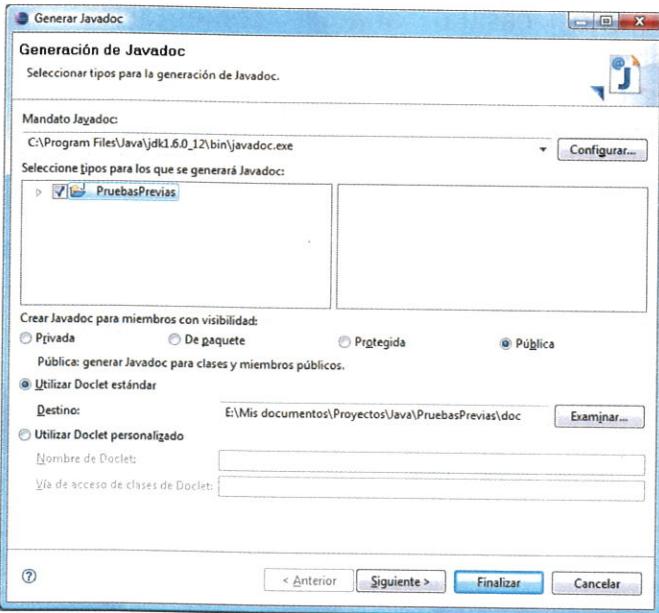


Ilustración 3-21, cuadro de generación de Javadoc en Eclipse

Desde el cuadro que aparece podemos:

- ◆ En el apartado **mandato javadoc**, asegurar que aparece una ruta válida al programa generador de documentos javadoc del kit de desarrollo de Java (**SDK**).
- ◆ En el apartado siguiente comprobar que están marcadas todas las clases que deseamos documentar
- ◆ Elegir la visibilidad de lo que se desea documentar (se entenderá mejor cuando conozcamos la programación orientada a objetos de Java)
- ◆ En los últimos apartados elegir la ruta en la que se creará la documentación (normalmente es la subcarpeta **doc** del proyecto actual).

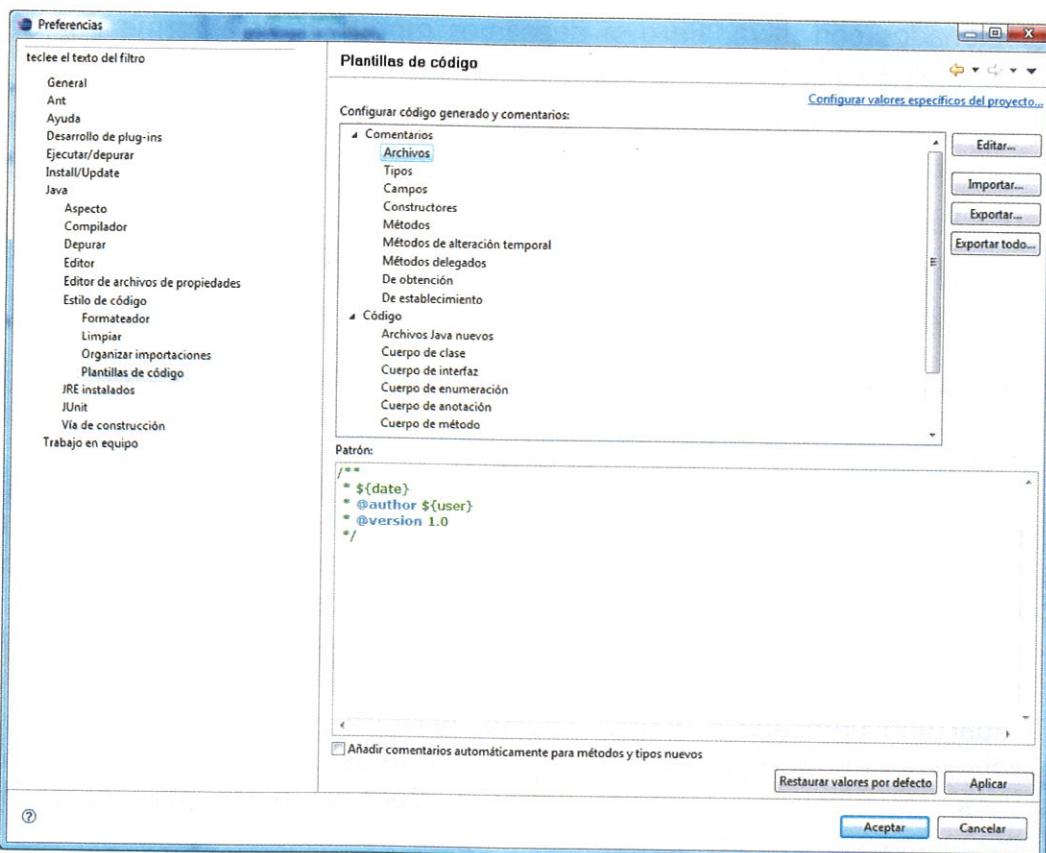


Ilustración 3-19, Cuadro de preferencias en el apartado Plantillas de código

comentarios

Apartado	Uso
Archivos	Aparece al inicio de cada archivo creado con Eclipse. Normalmente aparece la fecha y nombre del autor. Se puede hacer referencia al contenido de este apartado usando la variable `\${filecomment}`
Tipos	Aparece al inicio de cada nuevo tipo (clase, interfaz,...) creado. Se suele usar la variable `\${tags}` que permite colocar automáticamente los comentarios javadoc pertinentes. Se puede hacer referencia a este apartado con la variable `\${typecomment}`
Campos	Comentarios que aparecen delante de cada nuevo campo
Constructores	Comentarios para nuevos constructores
Métodos	Comentarios para nuevos métodos
Métodos delegados	
Métodos de obtención	Comentarios para métodos get
Métodos de establecimiento	Comentarios para métodos set

formateador

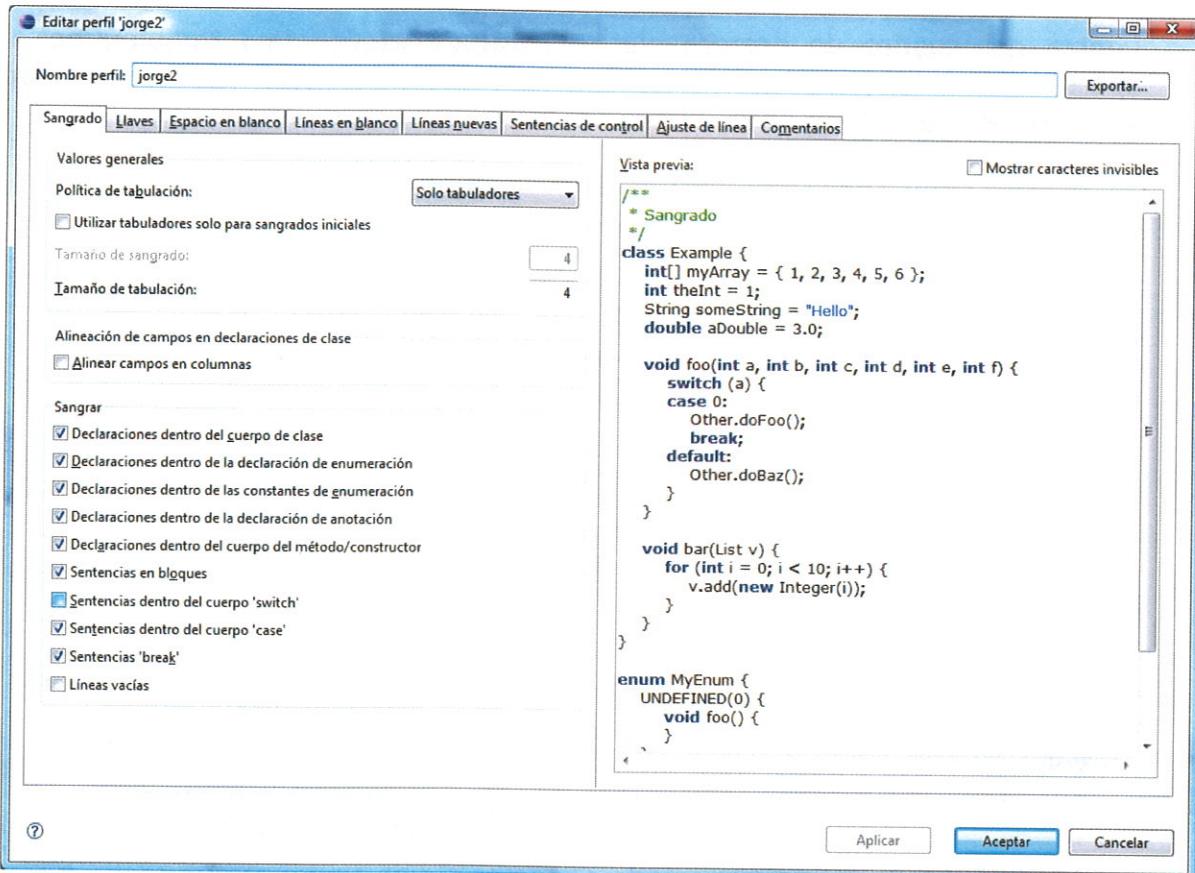


Ilustración 3-18. Creación de un nuevo perfil de formato de código

Este subapartado del estilo de código permite que nuestra forma personal de escribir código (como realizamos las sangrías, como abrimos y cerramos llaves, etc.) la utilice Eclipse. Por ejemplo hay programadores que para el método **main** escribirían:

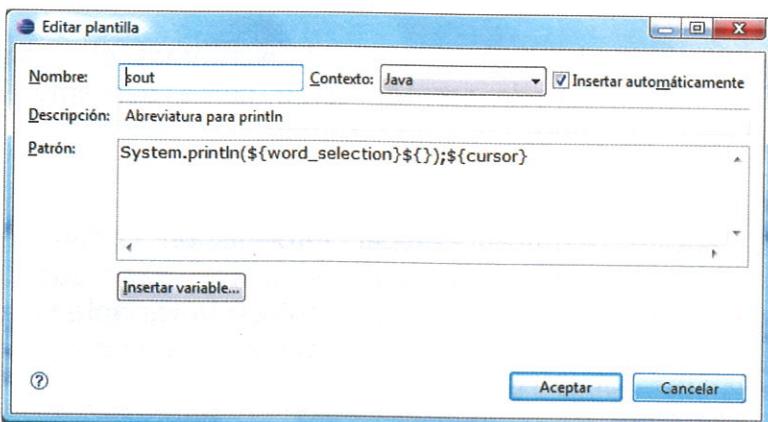
```
public static void main(String args[]){  
    ....  
}
```

Pero otros escriben:

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

Evidentemente es lo mismo y funciona correctamente. Es una cuestión de estilo decidir donde se coloca la apertura. Lo cierto es que el estilo de Eclipse es el prácticamente tomado como estándar por todos los organismos

Por ejemplo si creamos esta plantilla (que por cierto es mi favorita):



Si en el código escribimos directamente la palabra **sout** y después pulsamos **Ctrl+Barra**, Eclipse directamente escribirá **System.out.println()**; como hemos hecho uso de la variable vacía (**\${}**), el cursor se quedará dentro del paréntesis esperando que escribamos en este caso lo que debe salir por pantalla. Al pulsar después el tabulador, el cursor pasa a la siguiente variable que es **\${cursor}** que indica la posición final del cursor, por lo que rápidamente con esa plantilla podremos utilizar la muy utilizada función **println**.

Con esta misma plantilla podemos hacer uso de la selección de texto (gracias a haber colocado la variable **\${word_selection}**). Si escribimos por ejemplo **"Hola"** y luego lo seleccionamos y pulsamos **Ctrl+Barra**, aparece el cuadro de plantillas, eligiendo la nuestra (**sout**), el texto **"Hola"** aparece dentro de un **println**:

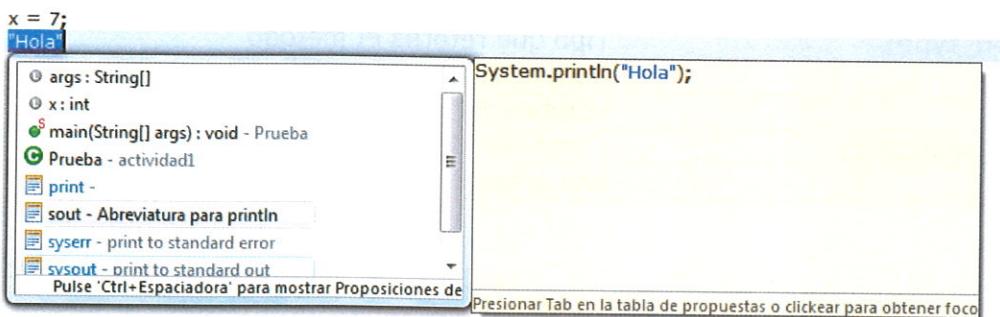


Ilustración 3-16, Cuadro de plantillas en Eclipse según aparece tras escribir "Hola" e invocarla

Un buen uso de las plantillas ahorra muchísimo tiempo de escritura de código y además evita cometer numerosos errores tontos. Las variables son muy importantes, de hecho hay muchas más que las señaladas en el cuadro anterior, su uso y potencia se comprenderá más cuanto más avancemos en nuestros conocimientos en Java.

(I.vii.vii) plantillas

Como se ha comentado anteriormente, se trata de una de las opciones más interesantes de ayuda al escribir código. En el caso de Eclipse las posibilidades de las plantillas son espectaculares.

La lista de las plantillas actuales se puede observar en el menú de preferencias, apartado **Java-Editor-Plantillas (Templates)**.

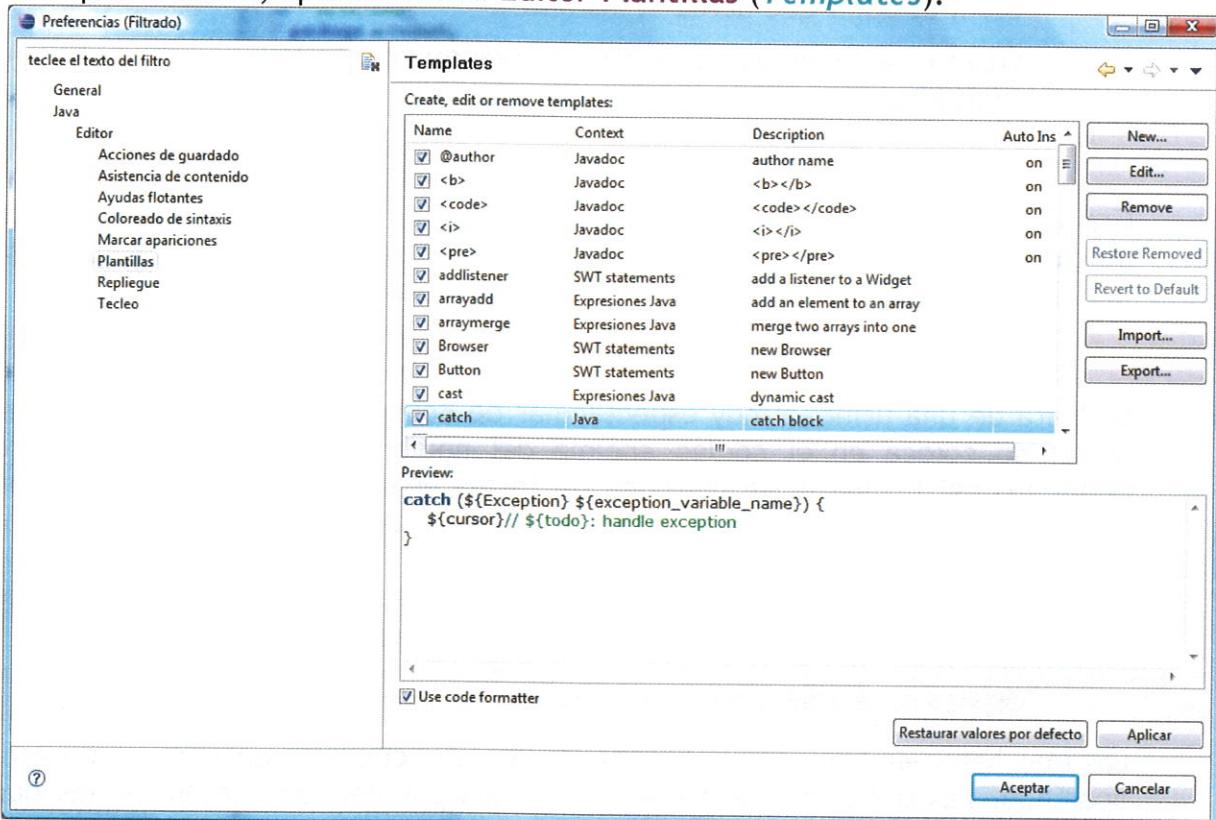


Ilustración 3-15, Cuadro de preferencias en el apartado de plantillas

Desde este cuadro podemos crear nuevas plantillas (botón **New**) , editarlas o borrarlas (**Remove**). Incluso podemos importar y exportar plantillas de una instalación de Eclipse a otra.

crear nuestras propias plantillas

Podemos crear nuestras propias plantillas. Cuanto más perfectas sean, más útiles serán. Los pasos son:

- (1)** Desde el cuadro de plantillas, pulsar el botón **Nueva (New)**.
- (2)** Escribir el nombre de la plantilla. Este texto es importante será el que invoque a la plantilla cuando le escribamos en el código y pulsemos **Ctrl+Barra espaciadora**.
- (3)** Hay que seleccionar la casilla **Insertar automáticamente** si deseamos que en cuanto pulsemos **Ctrl+Barra** se sustituya el nombre escrito por el código (patrón) correspondiente.

(I.vii) modificar las preferencias del editor

Mediante esta posibilidad se puede modificar el comportamiento del editor de Java de Eclipse en todas sus posibilidades. Esto se hace desde el menú **Ventana-Preferencias**. Después para el caso de Java hay que elegir el apartado **Java** y elegir **Editor**. Ahí dispondremos de varios apartados que se comentan a continuación

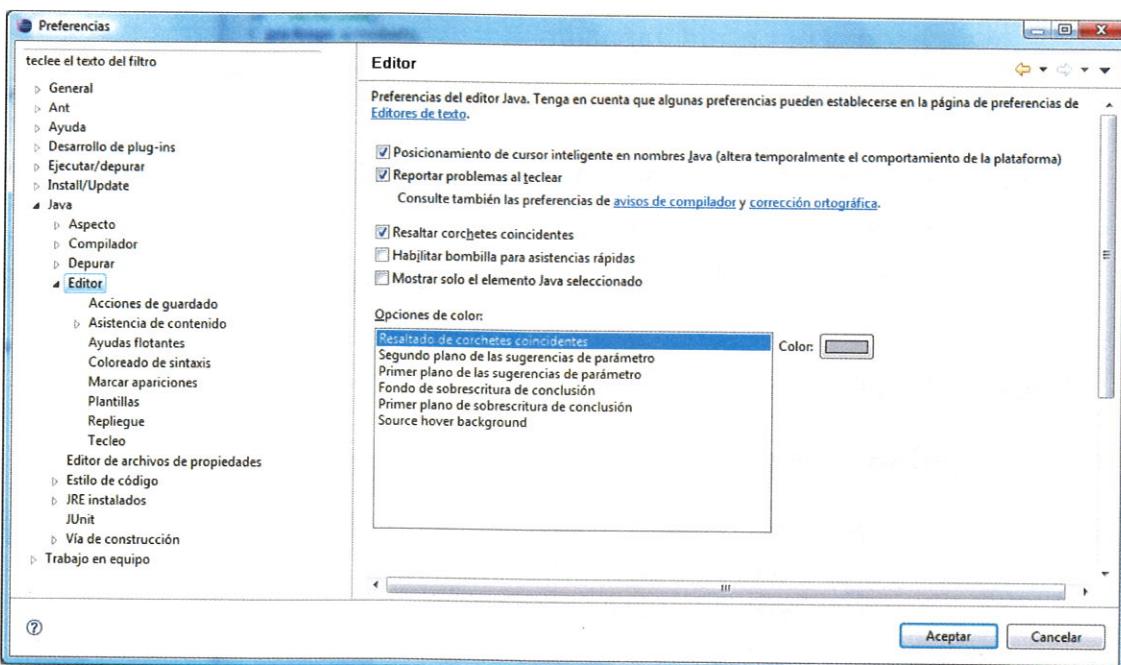


Ilustración 3-14. Cuadro de preferencias de Eclipse mostrando el apartado referente al Editor de Java

(I.vii.i) opciones generales

El apartado Java-Editor del cuadro de preferencias muestra algunas opciones interesantes. Entre ellas de la **reportar errores al teclear** (*Report problems as you type*) que, activada, es la que permite mostrar los subrayados de errores.

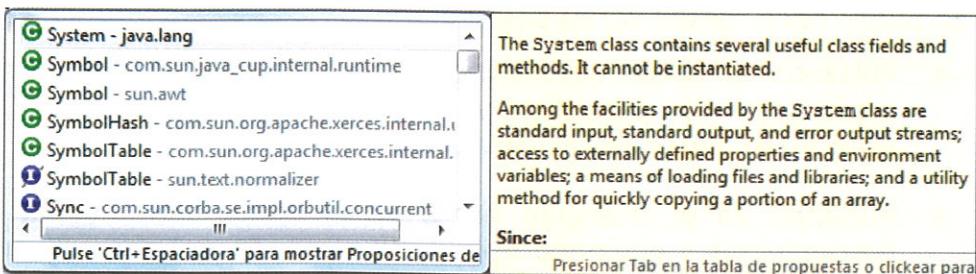
También interesante es la de **resaltar los corchetes coincidentes** (*Highlight matching brackets*) que hace que el editor cuando cerramos un paréntesis, llave o corchete, ilumine de color gris el inicio del mismo para saber qué es lo que estamos cerrando.

(I.vii.ii) realizar acciones al guardar

Es un subapartado del editor que permite que antes de guardar el documento se realicen acciones previas, como por ejemplo, dar formato al código (para que esté bien tabulado). No se usa mucho, pero es interesante.

modificar las preferencias del editor, subapartado plantillas.

La mejor forma de utilizar el asistente de contenido es empezar a escribir el nombre de una función o variable que conozcamos y pulsar **Ctrl+Barra** por ejemplo supongamos que queremos utilizar la función de escritura **System.out.println**, empezaríamos escribiendo por ejemplo **Sy** y después al pulsar **Ctrl+Barra** aparece este menú flotante:



En la pantalla de la izquierda aparecen una lista de elementos que comienzan con **Sy**, puesto que el queremos es **System** podemos elegirle con el ratón o con las teclas (utilizando el tabulador y luego la flecha y finalmente Intro).

Después al escribir el punto aparecen los miembros de **System** entre los que podemos elegir **out** y al escribir el último punto aparece una nueva lista en la que podemos elegir **println**.

En todo momento podemos escribir sin hacer caso al asistente de ese modo el asistente irá afinando más la puntería al sugerir lo que podríamos desear escribir. Incluso podemos ignorarle por completo.

El funcionamiento del asistente se puede modificar como se explica en el apartado **O**

atañen a aspectos de Java que aún no han sido comentados en estos apuntes.

- (3)** Una vez finalizado el asistente aparecerá una nueva clase en el explorador de paquetes.

(I.IV.IV) cambiar el nombre a los elementos de Java

cambiar el nombre a una clase

En Java no es tan fácil cambiar el nombre a una clase o programa. La razón tiene que ver con el hecho de que el nombre de la clase y el del archivo tienen que ser el mismo; lo que significa que no podemos cambiar el uno sin cambiar el otro. Además otras clases pueden hacer referencia a aquella que queremos cambiar de nombre.

Todo esto se arregla si cambiamos el nombre desde el explorador de paquetes eligiendo **Refactorizar (Refactor)-Redenominar (Rename)** que aparece si pulsamos el botón secundario del ratón sobre la clase que queremos cambiar de nombre.

cambiar el nombre a variables y otros elementos de Java

La misma opción utilizada en el apartado anterior nos permite simplificar una de las operaciones más engorrosas al escribir programas, cambiar el nombre a una variable.

Lo normal es que una variable aparezca varias veces en el código. Si, por lo que sea, deseamos cambiar el nombre de una variable tendríamos que modificar todas las veces que aparezca la variable.

Más cómo es:

- (1)** Seleccionar el nombre de la variable en cualquier parte del código en que aparezca
- (2)** Pulsar el botón secundario sobre la selección y elegir **Refactorizar-Redenominar** (también se puede pulsar directamente la combinación de teclas **Mayús+Alt+R**)
- (3)** Escribir el nuevo nombre y pulsar la tecla **Intro**

Para cualquier otro elemento al que le tengamos que haber puesto nombre, le podemos cambiar el nombre de la misma manera.

Nota: La operación de *redenombrar* un elemento de Java se puede deshacer desde **Edición-Deshacer (Undo)**

Proyecto Java (*Java Project*), también se puede pulsar en el triangulito negro del botón . Al hacerlo aparece esta imagen:

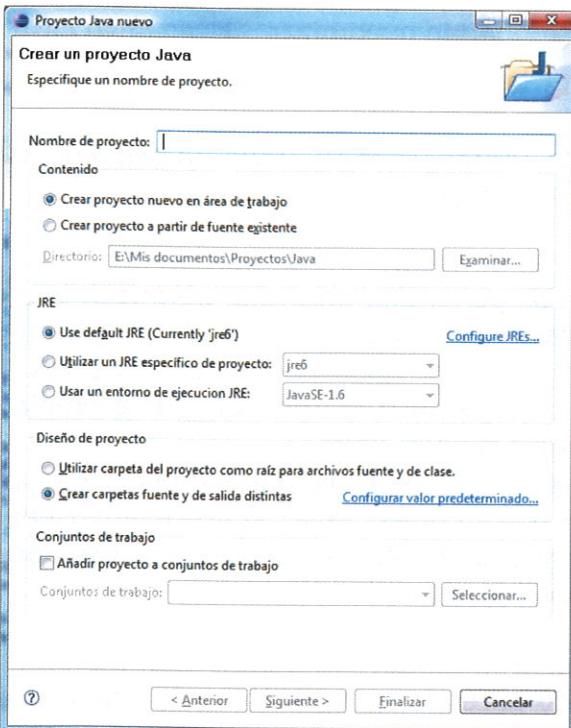


Ilustración 3-12, Cuadro de creación de proyectos nuevos en Eclipse

En este cuadro hay que elegir:

- ◆ **Nombre de proyecto (Project name).** Es un nombre cualquier para el proyecto, sólo sirve para identificarle en la lista de proyectos.
- ◆ **Contenido.** En este apartado se nos permite elegir la carpeta o directorio en la que se guardará el proyecto. Al instalar Eclipse se elige un directorio por defecto (que se puede modificar). En cualquier caso desde este apartado se puede elegir otro directorio o carpeta e incluso señalar un directorio que contenga código fuente.
- ◆ **JRE.** Sirve para elegir la versión de Java con la que se compilará el proyecto.

Una vez creado el proyecto aparece en la vista **Explorador de paquetes (Package Explorer)** y desde ahí podremos examinarle (ahí aparecerán todos los proyectos). Se observará la existencia de la carpeta **src** en la que se almacena el código fuente. Si fuéramos al directorio en el que se encuentra el proyecto desde el sistema operativo, veríamos que dentro de la carpeta del proyecto estará la carpeta **src** y otra carpeta llamada **bin** que sirve para almacenar el código precompilado (los archivos **class**).

(I.iii) aspecto de Eclipse

Una vez cerrada la pantalla de bienvenida Eclipse presenta un aspecto parecido a éste:

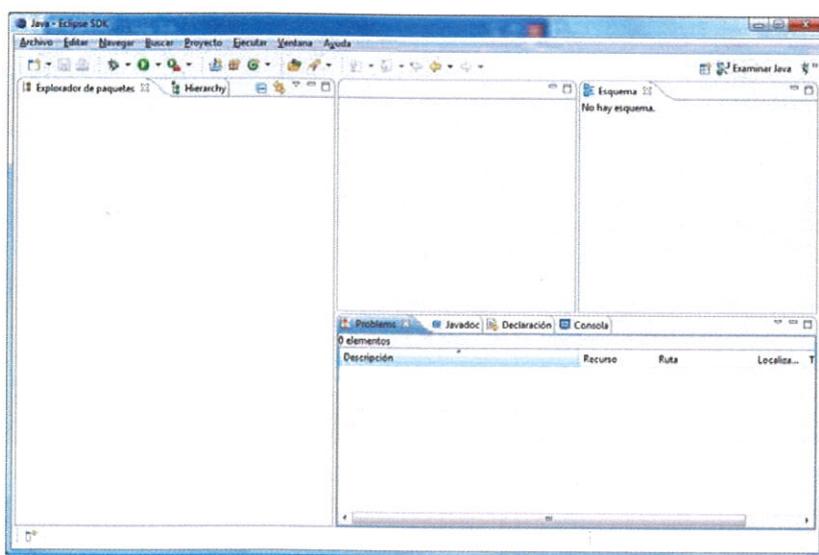


Ilustración 3-11, Aspecto del área de trabajo de Eclipse al iniciar por primera vez

Como se observa dispone de numerosas ventanas y paneles, los cuales se pueden cerrar y modificar el tamaño. Eso hace que de un programador a otro el aspecto de Eclipse pueda cambiar.

(I.iii.i) las perspectivas de Eclipse

Se llama perspectiva a un conjunto de paneles y su disposición en la pantalla. Como Eclipse sirve para realizar numerosas tareas, podemos elegir entre varias perspectivas. Inicialmente la perspectiva de trabajo es la de Java (que es la que se muestra en este manual).

Más exactamente una perspectiva de Eclipse es un conjunto de **Vistas** y **Editores**.

vistas

Una vista es un componente dentro de Eclipse que sirve para mostrar información. Por ejemplo el **Explorador de paquetes** es una vista que permite examinar los proyectos realizados.

Las vistas se pueden colocar como deseemos. Si arrastramos desde su pestaña a otro sitio, la vista cambiará de posición. Eso permite dejar el espacio de trabajo a nuestro gusto.

Se pueden cerrar vistas simplemente haciendo clic en el signo del nombre de la vista.

Para mostrar una vista que ahora está cerrada se elige **Ventana-Mostrar-Vista** (**Window>Show view**) y luego se hace clic sobre la vista a mostrar.

Después de responder a esta pregunta aparece el entorno de Eclipse en este primer arranque aparece la pantalla de bienvenida (*Welcome*) con sólo cinco iconos que nos permiten acceder a tutoriales, ejemplos, novedades y la ayuda de Eclipse.

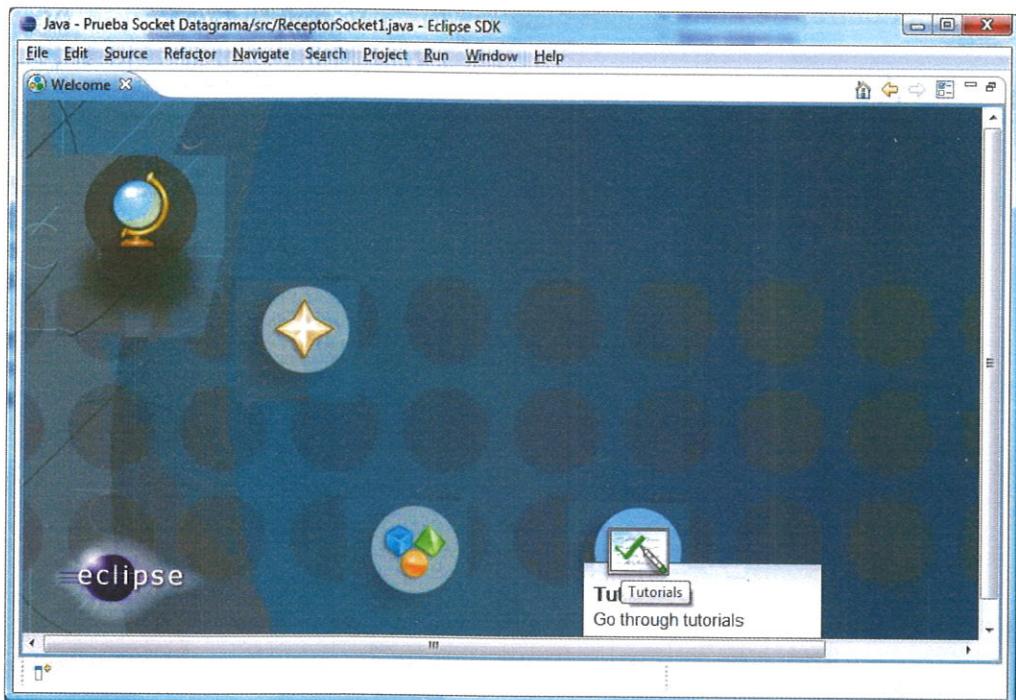


Ilustración 3-9, Aspecto del primer arranque de Eclipse

Eclipse está en lengua inglesa, pero a través del proyecto **Babel** (http://download.eclipse.org/technology/babel/babel_language_packs/) se pueden obtener traducciones a otros idiomas. El problema es que las traducciones suelen ir muy por detrás de las versiones de Eclipse y que están incompletas.

Apéndice (I)

Eclipse

(I.i) entornos de desarrollo integrado (IDE)

Los IDE son software que permiten a los programadores crear aplicaciones de forma cómoda. En este anexo se explica brevemente como empezar a trabajar con Eclipse. Este capítulo no pretende ser una guía completa de Eclipse sino explicar lo mínimo para utilizar cómodamente Eclipse a fin de utilizarle como software básico de creación de aplicaciones en Java.

Eclipse integra editores para escribir código, compilación y ejecución desde el propio entorno, depuración, diseño asistido para crear el interfaz de la aplicación y casi cualquier herramienta interesante para programar en Java (o en otros lenguajes, aunque desde luego el núcleo de desarrollo de Eclipse es Java).

Eclipse es el nombre del entorno de desarrollo integrado más popular de la actualidad para programar en Java.

Eclipse se define a sí misma en la página oficial www.eclipse.org como una *comunidad basada en los modelos de código abierto, que desarrolla proyecto, plataformas y herramientas para crear, diseñar y administrar software cubriendo todo el ciclo de vida de la aplicación. Eclipse es una fundación del sin ánimo de lucro apoyada por empresas y entidades que permiten su desarrollo.*

Efectivamente en la actualidad Eclipse intenta cubrir todo el ciclo de vida de desarrollo de una aplicación. Ya que además de facilitar la escritura de código, añade herramientas de diseño, gestión comunicación con bases de datos y cada vez más y más módulos que se añaden al núcleo del IDE Eclipse.

El proyecto Eclipse fue iniciado por **IBM** en noviembre de 2001 con la idea de sustituir a **Visual Age** que era su anterior producto de desarrollo de aplicaciones Java. En 2004 se independizó La fundación del eclipse fue creada en enero de 2004 pasó a ser desarrollado de manera independiente por la **Fundación Eclipse**, que está apoyada por empresas como **IBM, Adobe, Borland, Oracle, Intel, Motorola, SAP...**

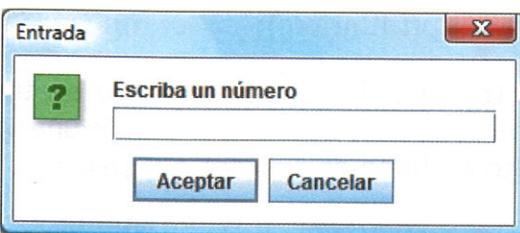
operador	significado
<code>double rint(double x)</code>	Idéntico al anterior, sólo que éste método da como resultado un número double mientras que round da como resultado un entero tipo int
<code>double random()</code>	Número aleatorio decimal situado entre el 0 y el 1
<code>tiponúmero abs(tiponúmero x)</code>	Devuelve el valor absoluto de x .
<code>tiponúmero min(tiponúmero x, tiponúmero y)</code>	Devuelve el menor valor de x o y
<code>tiponúmero max(tiponúmero x, tiponúmero y)</code>	Devuelve el mayor valor de x o y
<code>double sqrt(double x)</code>	Calcula la raíz cuadrada de x
<code>double pow(double x, double y)</code>	Calcula x^y
<code>double exp(double x)</code>	Calcula e^x
<code>double log(double x)</code>	Calcula el logaritmo neperiano de x
<code>double acos(double x)</code>	Calcula el arco coseno de x
<code>double asin(double x)</code>	Calcula el arco seno de x
<code>double atan(double x)</code>	Calcula el arco tangente de x
<code>double sin(double x)</code>	Calcula el seno de x
<code>double cos(double x)</code>	Calcula el coseno de x
<code>double tan(double x)</code>	Calcula la tangente de x
<code>double toDegrees(double anguloEnRadianes)</code>	Convierte de radianes a grados
<code>double toRadians(double anguloEnGrados)</code>	Convierte de grados a radianes
<code>double signum(double n)</code>	Devuelve el valor del signo del número n . Si n vale cero, la función devuelve cero; si es positivo devulve 1.0 y si es negativo -1.0 Esta función apareció en la versión 1.5 de Java.
<code>double hypot(double x, double y)</code>	Suponiendo que x e y son los dos catetos de un triángulo rectángulo, la función devuelve la hipotenusa correspondiente según el teorema de Pitágoras. Disponible desde la versión 1.5
<code>double nextAfter(double valor, double dir)</code>	Devuelve el siguiente número representable desde el valor indicado hacia la dirección que indique el valor del parámetro dir . Por ejemplo <code>Math.nextAfter(34.7, 90)</code> devolvería 34.7000000001 Función añadida en la versión Java 1.6

Esta es una clase pensada para manejar cuadros de diálogo de tipo **Swing** (véase tema ¡Error! No se encuentra el origen de la referencia. ¡Error! No se encuentra el origen de la referencia.). Uno de estos cuadros permite introducir datos y que una variable les almacene. El problema es que todos los datos les devuelve en forma de **String** (texto), lo que implica almacenarlos en una variable de ese tipo y luego convertirlos al tipo apropiado (**int**, **double**, **char**,...). Ejemplo:

```
String texto=JOptionPane.showInputDialog("Escriba un número entero");
int n=Integer.parseInt(texto);
```

Evidentemente este código no es fácil. Para explicar vamos línea a línea:

- ◆ En la primera línea, la variable **texto** almacena lo que el usuario escriba en el cuadro de mensaje. **JOptionPane.showInputDialog** es la función que permite sacar el cuadro de mensaje. "**Escriba un número entero**" es el texto que presenta el mensaje. Es decir el resultado de esta instrucción es:



- ◆ En cuanto el usuario o usuaria escriba el número y lo acepten (botón **Aceptar**), lo escrito se almacena en la variable **texto**
- ◆ En la segunda línea, la función **Integer.parseInt** sirve para convertir el número escrito a forma de entero **int**. Sin esta conversión no podemos manipular lo escrito como número, ya que se considera un texto (un **String**).
- ◆ Si el usuario cancela el cuadro o bien no escribe un número entero, ocurre un error que provocará que el programa finalice la ejecución. Arreglar estos errores será algo que veremos más adelante

Aunque el código es un tanto críptico a estas alturas, merece la pena aprenderle ya que permite más posibilidades de hacer programas para practicar. De otro modo no tendremos manera de leer datos por teclado y eso supone una tarea importante para nuestros programas.

Hay que señalar que hay más funciones de conversión, se pueden apreciar en esta tabla:

Algunas funciones de conversión

Función	Convierte a
Integer.parseInt	int
Short.parseShort	short
Byte.parseByte	byte
Long.parseLong	long

nivel	operador
7	<code>==</code>
8	<code>&</code>
9	<code>^</code>
10	<code> </code>
11	<code>&&</code>
12	<code> </code>
13	<code>?:</code>
14	<code>=</code>
	<code>+ =, - =, * =, ...</code>

En la tabla anterior los operadores con mayor precedencia está en la parte superior, los de menor precedencia en la parte inferior. De izquierda a derecha la precedencia es la misma. Es decir, tiene la misma precedencia el operador de suma que el de resta.

Esto último provoca conflictos, por ejemplo en:

```
resultado = 9 / 3 * 3;
```

El resultado podría ser uno ó nueve. En este caso el resultado es nueve, porque la división y el producto tienen la misma precedencia; por ello el compilador de Java realiza primero la operación que este más a la izquierda, que en este caso es la división.

Una vez más los paréntesis podrían evitar estos conflictos.

(3.11) constantes

Una constante es una variable de sólo lectura. Dicho de otro modo más correcto, es un valor que no puede variar (por lo tanto no es una *variable*).

La forma de declarar constantes es la misma que la de crear variables, sólo que hay que anteponer la palabra **final** que es la que indica que estamos declarando una constante y por tanto no podremos variar su valor inicial:

```
final double PI=3.141591;  
PI=4; //Error, no podemos cambiar el valor de PI
```

Como medida aconsejable (aunque no es obligatoria, sí altamente recomendable), los nombres de las constantes deberían ir en mayúsculas.

Ejercicio 2

Programa que calcule en cm/cm² la longitud
y la superficie de una circunferencia/círculo
de radio 2m.

El operador **||** (OR) sirve también para evaluar dos expresiones. El resultado será **true** si al menos uno de las expresiones es **true**. Ejemplo:

boolean nieva =**true**, llueve=**false**, graniza=**false**;
boolean malTiempo= nieva **||** llueve **||** graniza;

(3.10.4) operadores de BIT

Manipulan los bits de los números. Son:

operador	significado
&	AND
	OR
~	NOT
^	XOR (valor 0 cuando son iguales)
>>	Desplazamiento a la derecha (entra valor de bit de siguiente)
<<	Desplazamiento a la izquierda (entran ceros)
>>>	Desplazamiento derecha con relleno de ceros
<<<	Desplazamiento izquierda con relleno de ceros

(3.10.5) operadores de asignación

Permiten asignar valores a una variable. El fundamental es “**=**”. Pero sin embargo se pueden usar expresiones más complejas como:

x += 3;

En el ejemplo anterior lo que se hace es sumar **3** a la **x** (es lo mismo **x+=3**, que **x=x+3**). Eso se puede hacer también con todos estos operadores:

+= **-=** ***=** **/=**
&= **|=** **^=** **%=**
>>= **<<=**

También se pueden concatenar asignaciones (aunque no es muy recomendable):

x1 = x2 = x3 = 5; //todas valen 5

Otros operadores de asignación son “**++**” (incremento) y “**--**” (decremento). Ejemplo:

x++; //esto es x=x+1;
x--; //esto es x=x-1;

Pero hay dos formas de utilizar el incremento y el decremento. Se puede usar por ejemplo **x++ o ++x**

La diferencia estriba en el modo en el que se comporta la asignación.

- (4) Cuando finaliza el bloque en el que fue declarada, la variable **muere**. Es decir, se libera el espacio que ocupa esa variable en memoria. No se la podrá volver a utilizar.

Una vez que la variable ha sido eliminada, no se puede utilizar. Dicho de otro modo, no se puede utilizar una variable más allá del bloque en el que ha sido definida. Ejemplo:

```
{  
    int x=9;  
}  
int y=x; //error, ya no existe x
```

(3.10) operadores

(3.10.1) introducción

Los datos se manipulan muchas veces utilizando operaciones con ellos. Los datos se suman, se restan, ... y a veces se realizan operaciones más complejas.

(3.10.2) operadores aritméticos

Son:

Tabla 1.3

operador	significado
+	Suma
-	Resta
*	Producto
/	División
%	Módulo (resto)

Hay que tener en cuenta que el resultado de estos operadores varía notablemente si usamos enteros o si usamos números de coma flotante.

Por ejemplo:

```
double resultado1, d1=14, d2=5;  
int resultado2, i1=14, i2=5;  
  
resultado1= d1 / d2;  
resultado2= i1 / i2;
```

resultado1 valdrá 2.8 mientras que **resultado2** valdrá 2.

la razón es que los tipos de coma flotante son más grandes que los enteros, por lo que no hay problema de pérdida de valores.

Al declarar números del tipo que sean, si no se indican valores iniciales, Java asigna el valor cero.

(3.9.3) booleanos

Los valores booleanos (o lógicos) sirven para indicar si algo es verdadero (**true**) o falso (**false**).

boolean b=true;

Si al declarar un valor booleano no se le da un valor inicial, se toma como valor inicial el valor **false**. Por otro lado, a diferencia del lenguaje C, **no se pueden en Java asignar números a una variable booleana** (en C, el valor **false** se asocia al número 0, y cualquier valor distinto de cero se asocia a **true**).

Tampoco tiene sentido intentar asignar valores de otros tipos de datos a variables booleanas mediante casting:

boolean b=(boolean) 9; //no tiene sentido

(3.9.4) caracteres

Los valores de tipo carácter sirven para almacenar símbolos de escritura (en Java se puede almacenar cualquier código Unicode). Los valores Unicode son los que Java utiliza para los caracteres. Ejemplo:

```
char letra;  
letra='C'; //Los caracteres van entre comillas  
letra=67; //El código Unicode de la C es el 67. Esta línea  
//hace lo mismo que la anterior
```

También hay una serie de caracteres especiales que van precedidos por el símbolo \, son estos:

figura caracteres especiales

carácter	significado
\b	Retroceso
\t	Tabulador
\n	Nueva línea
\f	Alimentación de página
\r	Retorno de carro
\"	Dobles comillas
\'	Comillas simples
\\\	Barra inclinada (backslash)
\udddd	Las cuatro letras d, son en realidad números en hexadecimal. Representa el carácter Unicode cuyo código es representado por las dddd

(34)

Ejercicio 1

- Programa que intercambia entre si el contenido de dos variables enteras.

Pero la asignación se puede utilizar en cualquier momento (tras haber declarado la variable):

```
int x;  
x=7;  
x=x*2;
```

Como se ve en la última línea anterior, la expresión para dar el valor a la variable puede ser tan compleja como queramos.

A diferencia de lenguajes como C, en Java siempre se asigna una valor inicial a las variables en cuanto se declaran. En el caso de los números es el cero.

Si no... solo si son miembros de una clase
`int x; //x ya vale cero`

(3.9) tipos de datos primitivos

Tipo de variable	Bytes que ocupa	Rango de valores
boolean	2	true, false
byte	1	-128 a 127
short	2	-32.768 a 32.767
int	4	-2.147.483.648 a 2.147.483.649
long	8	$-9 \cdot 10^{18}$ a $9 \cdot 10^{18}$
double	8	$-1,79 \cdot 10^{308}$ a $1,79 \cdot 10^{308}$
float	4	$-3,4 \cdot 10^{-38}$ a $3,4 \cdot 10^{-38}$
char	2	Caracteres (en Unicode)

(3.9.1) enteros

Los tipos **byte**, **short**, **int** y **long** sirven para almacenar datos enteros. Los enteros son números sin decimales. Se pueden asignar enteros normales o enteros octales y hexadecimales. Los octales se indican anteponiendo un cero al número, los hexadecimales anteponiendo **0x**.

```
int numero=16; //16 decimal  
numero=020; //20 octal=16 decimal  
numero=0x14; //10 hexadecimal=16 decimal
```

Normalmente un número literal se entiende que es de tipo **int** salvo si al final se le coloca la letra **L**; se entenderá entonces que es de tipo **long**.

No se acepta en general asignar variables de distinto tipo. Sí se pueden asignar valores de variables enteras a variables enteras de un tipo superior (por ejemplo asignar un valor **int** a una variable **long**). Pero al revés no se puede:

```
int i=12;
```

funciones de lectura y escritura, comunicación en red, programación de gráficos,...

Por ejemplo la clase **System** está dentro del paquete **java.lang** (paquete básico) y posee el método **out.println** que necesitamos para escribir fácilmente por pantalla.

Si quisieramos utilizar la clase **Date** que sirve para manipular fechas, necesitaríamos incluir una instrucción que permita incorporar el código de Date cuando compilemos nuestro trabajo. Para eso sirve la instrucción **import**. La sintaxis de esta instrucción es:

```
import paquete.subpaquete.subsubapquete....clase
```

Esta instrucción se coloca arriba del todo en el código. Para la clase **Date** sería:

```
import java.util.Date;
```

Lo que significa, importar en el código la clase **Date** que se encuentra dentro del paquete **util** que, a su vez, está dentro del gran paquete llamado **java**.

También se puede utilizar el asterisco en esta forma:

```
import java.util.*;
```

Esto significa que se va a incluir en el código todas las clases que están dentro del paquete **util** de **java**.

En realidad no es obligatorio incluir la palabra **import** para utilizar una clase dentro de un paquete, pero sin ella deberemos escribir el nombre completo de la clase. Es decir no podríamos utilizar el nombre **Date**, sino **java.util.Date**.

(3.8) variables

(3.8.1) introducción

Las variables son contenedores que sirven para almacenar los datos que utiliza un programa. Dicho más sencillamente, son nombres que asociamos a determinados datos. La realidad es que cada variable ocupa un espacio en la memoria RAM del ordenador para almacenar el dato al que se refiere. Es decir cuando utilizamos el nombre de la variable realmente estamos haciendo referencia a un dato que está en memoria.

Las variables tienen un nombre (un **identificador**) que sólo puede contener letras, números (pero no puede empezar el nombre con un número) y el carácter de subrayado. El nombre puede contener cualquier carácter **Unicode**

- ◆ **Delante de cada método.** Los métodos describen las cosas que puede realizar una clase. Delante de cada método los comentarios javadoc se usan para describir al método en concreto. Además de los comentarios, en esta zona se pueden incluir las etiquetas: `@see`, `@param`, `@exception`, `@return`, `@since` y `@deprecated`
- ◆ **Delante de cada atributo.** Se describe para qué sirve cada atributo en cada clase. Puede poseer las etiquetas: `@since` y `@deprecated`

Ejemplo:

```
/** Esto es un comentario para probar el javadoc
 * este texto aparecerá en el archivo HTML generado.
 * <strong>Realizado en agosto 2003</strong>
 *
 * @author Jorge Sánchez
 * @version 1.0
 */
public class prueba1 {
//Este comentario no aparecerá en el javadoc

    /** Este método contiene el código ejecutable de la clase
     *
     * @param args Lista de argumentos de la línea de comandos
     * @return void
     */

    public static void main(String args[]){
        System.out.println("¡Mi segundo programa! ");
    }
}
```

Tras ejecutar la aplicación javadoc, aparece como resultado la página web de la página siguiente.

(3.5) ejecución de programas Java

(3.5.1) proceso de compilación desde la línea de comandos

La compilación del código java se realiza mediante el programa **javac** incluido en el software de desarrollo de Java (el **SDK**). La forma de compilar es (desde la línea de comandos):

```
javac archivo.java
```

El resultado de esto es un archivo con el mismo nombre que el archivo java pero con la extensión **class**. Esto ya es el archivo con el código en forma de **bytecode**. Es decir con el código precompilado.

Si el programa es ejecutable (sólo lo son si contienen el método **main**), el código se puede interpretar usando el programa **java** del kit de desarrollo (que se encuentra en el mismo sitio que **javac**). Sintaxis:

```
java archivoClass
```

Estos comandos hay que escribirlos desde la línea de comandos de en la carpeta en la que se encuentre el programa. Pero antes hay que asegurarse de que los programas del kit de desarrollo son accesibles desde cualquier carpeta del sistema. Para ello hay que haber configurado adecuadamente las variables del sistema (véase apartado (3.3.3) instalación del SDK).

En el caso de Linux hay que ser especialmente cuidadoso con la variable de sistema **CLASSPATH** y asegurarnos de que contiene en sus rutas raíces de Java el símbolo **.** (punto) que representa a la carpeta actual. De otro modo la ejecución del comando **java** podría fallar.

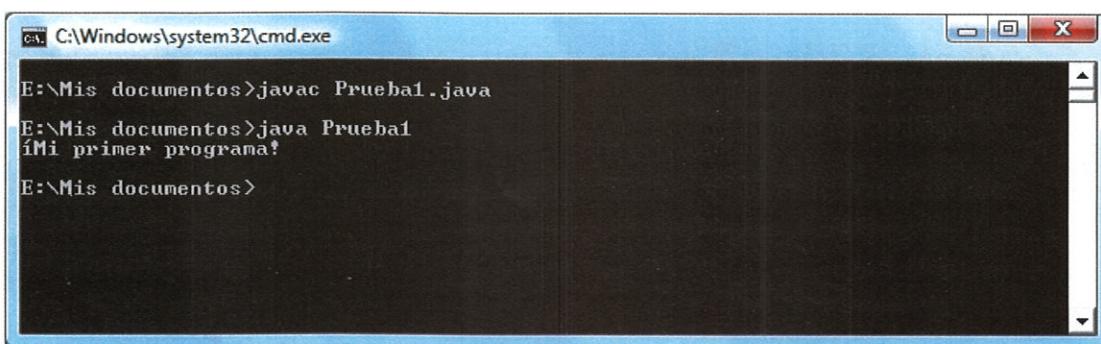


Ilustración 3-6, Ejemplo de compilación y ejecución en la línea de comandos