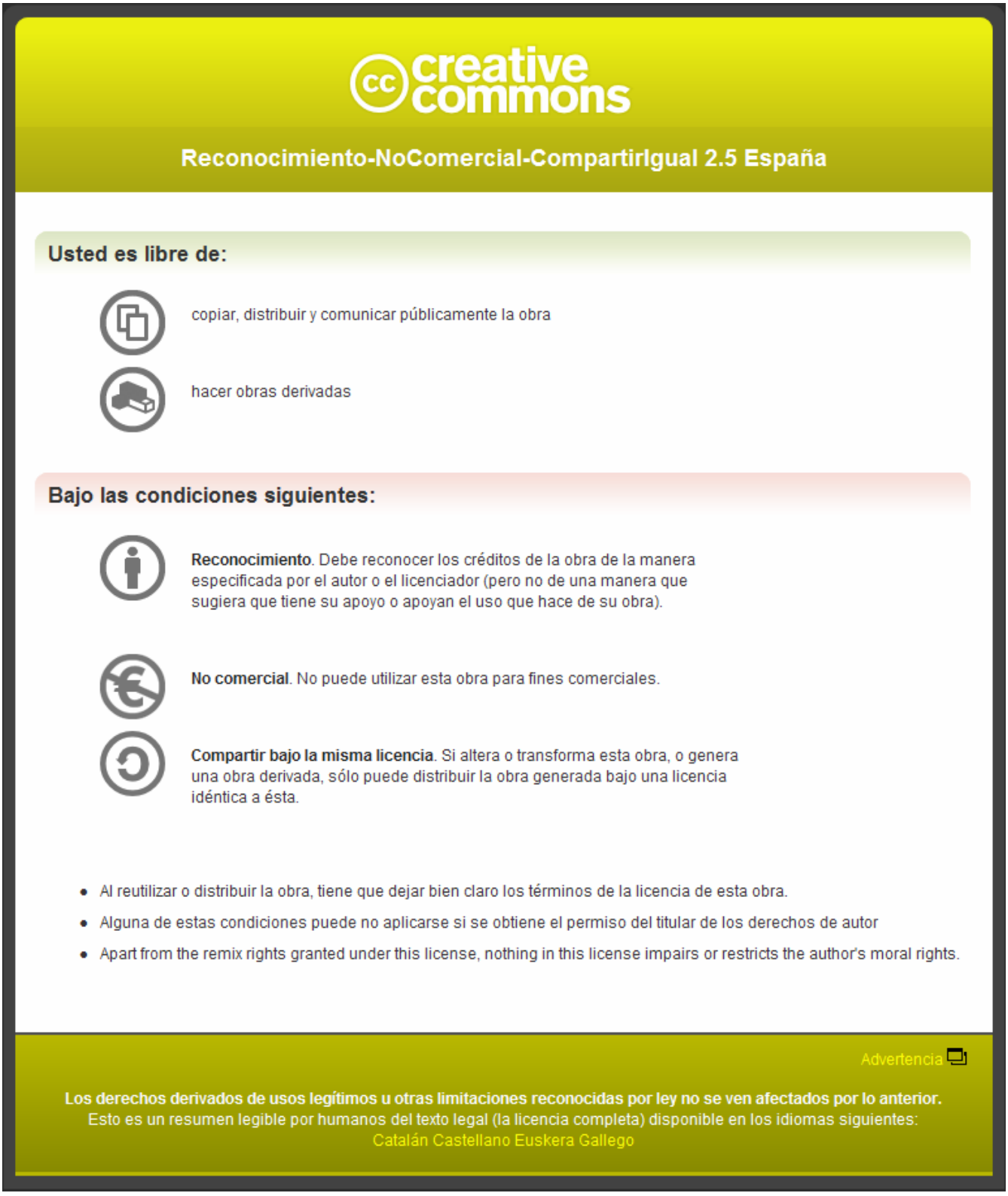
ac

Unidad 3: Programación básica en Lenguaje Java

Fundamentos de Programación. 1º de ASI



Para ver una copia completa de la licencia, acudir a la dirección http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es

(3) programación básica en Java

esquema de la unidad

(3.1) historia de Java

(3.1.1) los antecedentes de Java. influencia de C y C++

Java es un lenguaje de programación que se desarrolló para satisfacer las nuevas necesidades que requería la creación de aplicaciones a finales de los 90.

Desde los primeros lenguajes aparecidos en los años cincuenta, hasta la aparición de Java, la ciencia de la creación de programas ha sufrido numerosas transformaciones. Todas ellas se basan en intentar que los programadores y programadoras consigan trabajar de la forma más eficiente posible.

La búsqueda del lenguaje perfecto es la búsqueda del lenguaje que sea más fácil de aprender y que otorgue más posibilidades a aquellos programadores y programadoras que lo utilicen.

En general ambos conceptos han dado lenguajes muy diversos. Por ejemplo, el lenguaje **Basic** es un lenguaje muy fácil de aprender, pero en cuanto se quieren resolver problemas complicados, resulta ineficaz. Por otro lado el lenguaje **C** es un lenguaje muy poderoso, capaz de crear todo tipo de aplicaciones; pero es bastante más difícil de aprender.

Java intenta cumplir ambas premisas, pero de forma equilibrada: ni es un lenguaje muy fácil de aprender, ni es un lenguaje capaz de realizar todo tipo de aplicaciones. En realidad Java es uno de los muchos lenguajes influenciados por el exitoso **lenguaje C**. Este lenguaje ha sido el favorito de los creadores de aplicaciones (especialmente de sistemas) en los años 60 y 70.

la influencia del lenguaje C

La aparición del lenguaje **Fortran**, supuso la creación del primer lenguaje de alto nivel. Por primera vez el programador podía programar un poco más alejado de la lógica de la máquina, es decir cada vez más lejos del lenguaje de unos y ceros que es el único que los computadores reconocen.

Poco a poco aparecieron cada vez más lenguajes con la pretensión de mejorar la forma de programar (**Lisp, Pascal, Fortran, Cobol**,…). Algunos de ellos siguen vigentes incluso hoy en día. La mayoría se especializaron en diferentes tipos de aplicaciones (Lisp para aplicaciones de ingeniería, Pascal para aprendizaje de la ciencia de la programación, Cobol para aplicaciones de gestión,…).

El caso es que para crear aplicaciones de alto rendimiento y de sistema, los programadores seguía utilizando el lenguaje **Ensamblador**. Por ello a finales de los 60 aparece el lenguaje C.

C aportó a los lenguajes existentes las siguientes ventajas:

- Un lenguaje de nivel medio (más cercano a la forma de pensar del ordenador) que permitía tanto utilizar estructuras de los lenguajes de alto nivel (funciones, bucles avanzados,…) como instrucciones de nivel bajo (punteros)
- Una sintaxis que permite escribir código de forma rápida
- Un lenguaje potente capaz de crear todo tipo de aplicaciones
- Un lenguaje capaz de utilizar todo tipo de estructuras estáticas y dinámicas y de manejar todos los recursos de la máquina.

Sin embargo C también tiene sus problemas. Uno de los principales es que cuando la aplicación crece, el código es muy difícil de manejar. Las técnicas de programación estructurada y programación modular, que en C pueden ser aplicadas, paliaban algo el problema. Pero fue la **programación orientada a objetos (POO** u **OOP)** la que mejoró notablemente el situación.

No obstante C sigue siendo uno de los lenguajes más utilizados y académicamente sigue utilizándose por su versatilidad, que permite aprender todas las características de la programación clásica. De hecho a un buen programador de lenguaje C no le debería ser difícil aprender a programar en otros lenguajes (una vez que conozca las bases de la programación orientada a objetos).

la influencia de la programación orientada a objetos

La **POO** permite fabricar programas de forma más parecida al pensamiento humano. De hecho simplifica el problema dividiéndolo en objetos y permitiendo centrarse en cada objeto, para de esa forma eliminar la complejidad. Cada objeto se programa de forma autónoma y esa es la principal virtud.

Al aparecer la programación orientada a objetos (en los años setenta), aparecieron varios lenguajes orientados a objetos y también se realizaron versiones orientadas a objetos (o semiorientadas a objetos) de lenguajes clásicos.

Una de las más famosas adaptaciones fue la que capacitó al lenguaje C a utilizar objetos. A ese lenguaje se le llamó **C++** indicando con esa simbología que era un incremento del lenguaje C (en el lenguaje C, como en Java, los símbolos **++** significan incrementar). Las ventajas que añadió C++ a C fueron:

- Añadir soporte para objetos (POO)
- Librerías de clases de objetos (como **MFC**[1] por ejemplo) que facilitaban el uso de código ya creado para las nuevas aplicaciones.
- Todo lo bueno del C (incluso compatibilidad con este lenguaje)

[1] ***Microsoft Foundation Classes***, librería creada por Microsoft para facilitar la creación de programas para el sistema Windows.

C++ pasó a ser el lenguaje de programación más popular a principios de los 90 y sigue siendo un lenguaje muy utilizado. Muchas personas le consideran el lenguaje de programación más potente.

Otras adaptaciones famosas de lenguajes clásicos a lenguajes orientados a objetos, fueron:

- El paso de **Pascal** a **Turbo Pascal** y posteriormente a **Delphi**.
- El paso de **Basic** a **QuickBasic** y después a **Visual Basic**.

A pesar de las evidentes ventajas del lenguaje C++. Tiene sus serios inconvenientes.

- Su complejidad
- El hecho de ser un **lenguaje híbrido**, es decir que permite programar de forma no orientada a objetos, lo que provoca malas prácticas de programador.
- Los punteros, que requieren un especial cuidado por parte de la programadora o programador, ya que son los responsables de los errores más peligrosos y difíciles de detectar.
- El que sea un lenguaje apto para crear programas dañinos como virus y programas espías.
- No es un lenguaje apto para transmitirse en redes de ordenadores; especialmente en Internet (porque al ser compilado requiere cargar todo el código para ser compilado).

La llegada de Internet propició la creación de lenguajes más aptos para su uso en esta red de redes.

(3.1.2) la llegada de Java

En 1991, la empresa **Sun Microsystems** crea el lenguaje **Oak (**de la mano del llamado **proyecto Green**)**.** Mediante este lenguaje se pretendía crear un sistema de televisión interactiva. Este lenguaje sólo se llegó a utilizar de forma interna. en la empresa. Su propósito era crear un lenguaje independiente de la plataforma para uso en dispositivos electrónicos.

Se intentaba con este lenguaje paliar uno de los problemas fundamentales del C++; que consiste en que al compilar se produce un fichero ejecutable cuyo código sólo vale para la plataforma en la que se realizó la compilación. Sun deseaba un lenguaje para programar pequeños dispositivos electrónicos. La dificultad de estos dispositivos es que cambian continuamente y para que un programa funcione en el siguiente dispositivo aparecido, hay que rescribir el código. Por eso Sun quería crear un lenguaje **independiente del dispositivo.**

En 1995 Oak pasa a llamarse **Java**. Java es un importante exportador de café; por eso en EEUU se conoce como Java al café, tomarse una taza de Java

es tomarse una taza de café (aunque no sea precisamente de Java). Parece que los desarrolladores de Java tomaron muchas tazas de ***Java***.[2]

Ese año se da a conocer al público. y adquiere notoriedad rápidamente, casi desde su lanzamiento. Durante estos años se ha mejorado y se le ha revisado. La versión 1.2 modificó tanto Java que se la llamó **Java 2** y también a sus descendientes (Java 1.3 y Java 1.4). Actualmente el número 2 se ha quitado del nombre y la última versión se conoce como **Java v6**.

En general la sintaxis de Java es similar a C y C++. Pero posee estas diferencias:

- No hay punteros (lo que le hace más seguro y fácil de manejar)
- No es híbrido, sino totalmente orientado a objetos (aunque muchos programadores tienen reservas respecto a esta aseveración). Los lenguajes orientados a objetos híbridos permiten crear aplicaciones no orientadas a objetos.
- Muy preparado para ser utilizado en redes TCP/IP y especialmente en Internet
- Implementa excepciones (control de errores) de forma nativa
- Es un lenguaje interpretado (lo que acelera su ejecución remota, aunque provoca que las aplicaciones Java sean más lentas en la ejecución que las aplicaciones escritas en lenguajes compilados como C++).
- Permite múltiples hilos de ejecución, es decir que se ejecuten varias tareas en paralelo.
- Admite firmas digitales
- Tipos de datos y control de sintaxis más rigurosa que los lenguajes C y C++, lo que facilita la gestión de errores
- Es independiente de la plataforma, ejecutable en cualquier sistema con máquina virtual

La última ventaja (quizá la más importante) se consigue ya que el código Java no se compila, sino que se **precompila,** de tal forma que se crea un **código intermedio** que no es directamente ejecutable. No es código máquina. Para ejecutarle hace falta pasarle por un intérprete que va interpretando cada línea. Ese intérprete suele ser la **máquina virtual de Java**. Por lo que cualquier sistema que posea máquina virtual de Java, podrá ejecutar código precompilado en Java. Más adelante se explica este proceso en detalle.

[2] La simbología del café sigue presente en Java. El logotipo oficial de Java es una taza humeante de café. Mucho software desarrollado para Java ha mantenido esa simbología: Visual Café, Kawa *(café en ruso)*,…

(3.1.3) Java y JavaScript

Una de las confusiones actuales la provoca el parecido nombre que tienen estos dos lenguajes. Sin embargo no tienen mucho que ver entre sí. Sun creó Java y la empresa **Netscape** creó JavaScript. Java es un lenguaje completo que permite realizar todo tipo de aplicaciones. JavaScript es un lenguaje que permite incrustar código dentro de las páginas web.

La finalidad de JavaScript es mejorar las páginas web, hacerlas más vistosas y dinámicas. La finalidad de Java es crear aplicaciones de todo tipo (aunque está muy preparado para crear sobre todo aplicaciones en red).

Aunque la sintaxis tiene elementos en común, desde luego no se parece tanto. De hecho Javascript es mucho más fácil que Java.

(3.2) características de Java

(3.2.1) construcción de programas en Java. bytecodes

compilación tradicional

En el mundo de la programación siempre se ha hablado de **lenguajes compilados** y de **lenguajes interpretados**. En el segundo caso, un programa intérprete se encarga de traducir cada línea al código máquina correspondiente. Los lenguajes interpretados a partir de los setenta se han dejado de usar porque no eran los apropiados para conseguir código eficiente.

Por el contrario, los lenguajes compilados producen código máquina analizando todas las líneas de código en conjunto. Los compiladores buscan el mejor código máquina posible. El resultado del proceso de compilación (en realidad de compilación y enlazado) es un **archivo ejecutable**.

Un archivo ejecutable es un programa que se puede lanzar directamente en el sistema operativo; en el caso de Windows o Linux simplemente con hacer doble clic sobre el archivo, se ejecutan sus instrucciones. La ventaja es que los programas ejecutables no necesitan compilarse de nuevo, son programas terminados. El problema es que los sistemas operativos utilizan diferentes tipos de archivos ejecutables: es decir, un archivo ejecutable en Linux no sería compatible con Windows.

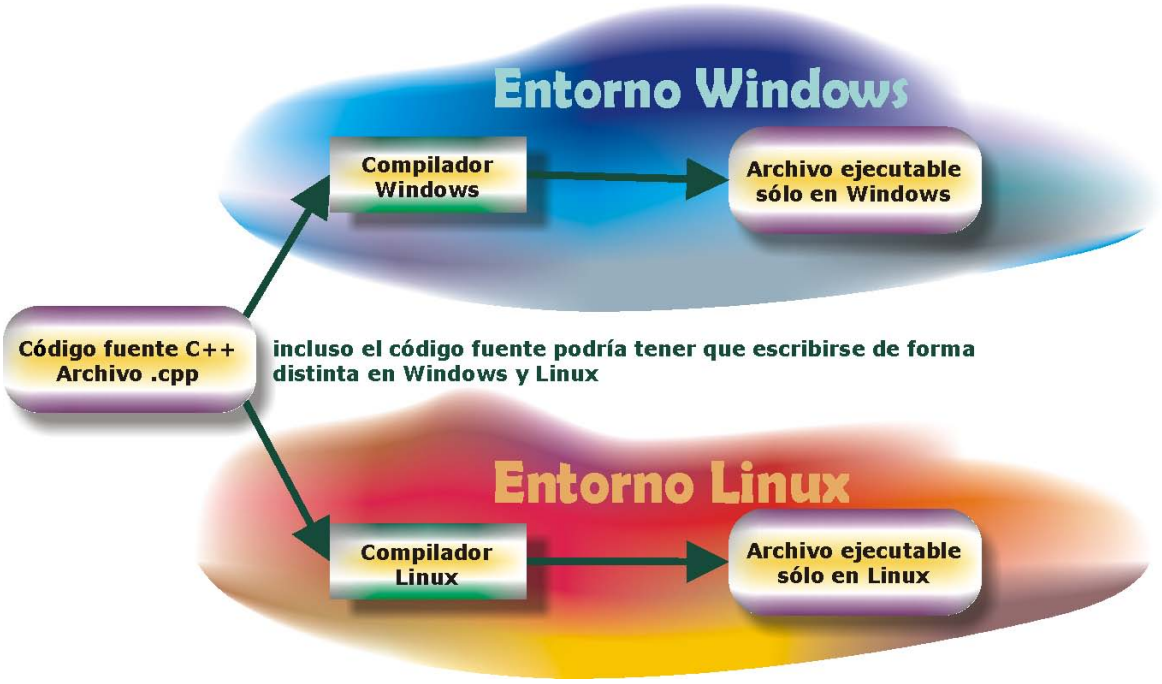


Ilustración 3-1, Proceso de compilación de un programa C++ en Windows y Linux

la "compilación" en Java

En Java el código no se traduce a código ejecutable. En Java el proceso se conoce como **precompilación** y sirve para producir un archivo (de extensión **class**) que contiene código que no es directamente ejecutable (no es código Java). Es un código intermedio llamado **bytecode** (también se le llama **J-code**).

Al no ser ejecutable, el archivo **class** no puede ejecutarse directamente con un doble clic en el sistema. El bytecode tiene que ser **interpretado** (es decir, traducido línea a línea) por una aplicación conocida como la **máquina virtual de Java** (**JVM**). Hoy se conoce como **JRE** (***Java Runtime Environment***, entorno de ejecución de Java).

La gran ventaja es que el entorno de ejecución de Java lo fabrica Sun para todas las plataformas; lo que significa que un archivo class se puede ejecutar en cualquier ordenador o máquina que incorpore el JRE. Sólo hay una pega, si programamos utilizando por ejemplo la versión 1.6 de Java, el ordenador en el que queramos ejecutar el programa deberá incorporar el JRE al menos de la versión 1.6.

El JRE o la máquina virtual de Java son un programas muy pequeños y que se distribuyen gratuitamente para prácticamente todos los sistemas operativos.

A la forma de producir código final de Java se la llama **JIT** (***Just In Time***, justo en el momento) ya que el código ejecutable se produce sólo en el instante de ejecución del programa. Es decir, no hay en ningún momento código ejecutable.

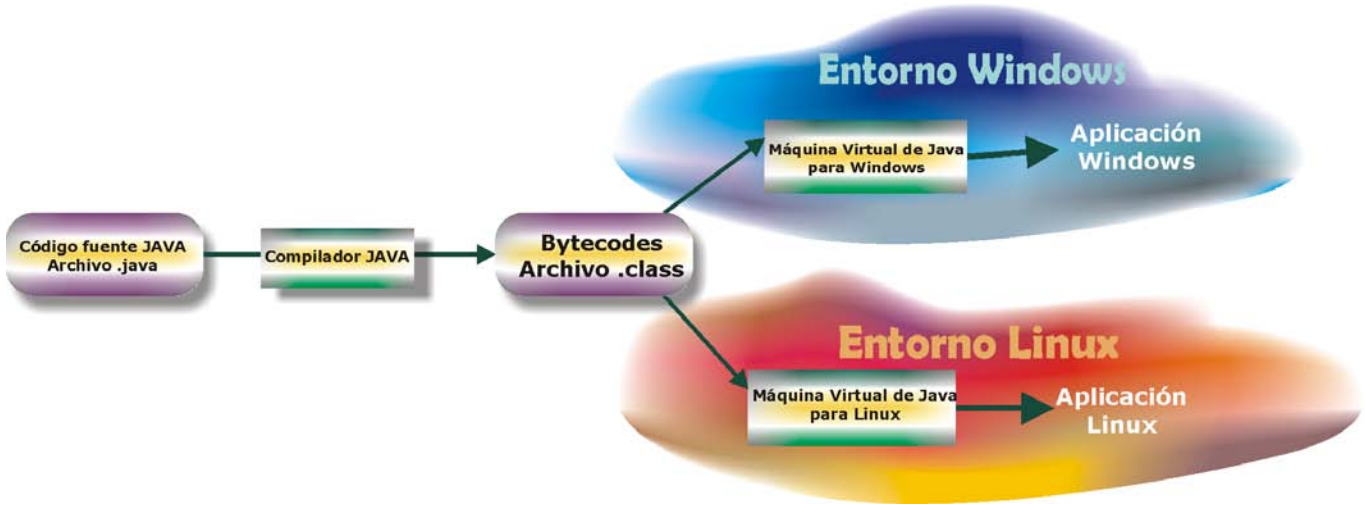


Ilustración 3-2, **Proceso de compilación de un programa Java**

(3.2.2) seguridad

Al interpretar el código, el JRE puede delimitar las operaciones peligrosas, con lo cual la seguridad es fácilmente controlable. Además, Java elimina las instrucciones dependientes de la máquina y los **punteros** que generaban terribles errores en C y la posibilidad de generar programas para atacar sistemas. Tampoco se permite el acceso directo a la memoria y recursos del ordenador.

La primera línea de seguridad de Java es un **verificador del bytecode** que permite comprobar que el comportamiento del código es correcto y que sigue las reglas del lenguaje Java. Normalmente los compiladores de Java no pueden generar código que se salte las reglas de seguridad de Java. Pero un programador ***malévolo*** podría generar artificialmente bytecode que se salte las reglas. El verificador intenta eliminar esta posibilidad.

Hay un segundo paso que verifica la seguridad del código que es el **verificador de clase** que es el programa que proporciona las clases necesarias al código. Lo que hace es asegurarse que las clases que se cargan son realmente las del sistema original de Java y no clases creadas reemplazadas artificialmente.

Finalmente hay un **administrador de seguridad** que es un programa configurable que permite al usuario indicar niveles de seguridad a su sistema para todos los programas de Java.

Hay también una forma de seguridad relacionada con la confianza. Esto se basa es saber que el código Java procede de un sitio de confianza y no de una fuente no identificada. Se consigue gracias a que en Java se permite añadir firmas digitales al código para verificar la autoría del mismo.

(3.2.3) tipos de aplicaciones Java

applets

Son programas Java pensados para ser colocados dentro de una página web. Pueden ser interpretados por cualquier navegador con capacidades Java. Estos programas se insertan en las páginas usando una etiqueta especial (como también se insertan vídeos, animaciones flash u otros objetos).

Los applets son programas independientes, pero al estar incluidos dentro de una página web las reglas de éstas le afectan. Normalmente un applet sólo puede actuar sobre el navegador.

Hoy día mediante applets se pueden integrar en las páginas web aplicaciones multimedia avanzadas (incluso con imágenes 3D o sonido y vídeo de alta calidad)

aplicaciones de consola

Son programas independientes al igual que los creados con los lenguajes tradicionales.

aplicaciones gráficas

Aquellas que utilizan las clases con capacidades gráficas (como **awt** por ejemplo).

servlets

Son aplicaciones que se ejecutan en un servidor de aplicaciones web y que como resultado de su ejecución resulta una página web.

midlet

Aplicación creada con Java para su ejecución en sistemas de propósito simple o dispositivos móviles. Los juegos Java creados para teléfonos móviles son midlets.

(3.2.4) plataformas

Actualmente hay tres ediciones de Java. Cada una de ellas se corresponde con una plataforma que incluye una serie de funciones, paquetes y elementos del lenguaje (es decir la **API**, ***Application Program Interface***).

Java SE

Java Standard Edition. Antes se la conocía como **J2SE** (el dos se refiere a Java 2). Permite escribir código Java relacionado con la creación de aplicaciones y applets en lenguaje Java común. Es decir, es el Java normal. La última versión del kit de desarrollo de aplicaciones en esta plataforma[3] es la JSE 1.6.14.

[3] En el momento de escribir este manual

Java EE

Java Enterprise Edition. Todavía conocida como **J2EE**. Pensada para la creación de aplicaciones Java empresariales y del lado del servidor. Su última versión es la 1.4

Java ME

Java Mobile Edition. También conocida como **J2ME**. Pensada para la creación de aplicaciones Java para dispositivos móviles.

(3.3) empezar a trabajar con Java

(3.3.1) el kit de desarrollo Java (SDK)

Para escribir en Java hacen falta los programas que realizan el precompilado y la interpretación del código, Hay entornos que permiten la creación de los bytecodes y que incluyen herramientas con capacidad de ejecutar aplicaciones de todo tipo. El más famoso (que además es gratuito) es el **Java Developer Kit (JDK)** de Sun, que se encuentra disponible en la dirección http://java.sun.com/javase/downloads.

Actualmente ya no se le llama así sino que se le llama **SDK** y al descargarlo de Internet hay que elegir la plataforma deseada (SE, EE o ME).

(3.3.2) versiones de Java

Como se ha comentado anteriormente, para poder crear los bytecodes de un programa Java, hace falta el SDK de Sun. Sin embargo, Sun va renovando este kit actualizando el lenguaje. De ahí que se hable de Java 1.1, Java 1.2, etc. Los nombres de los distintos SDK y del lenguaje correspondiente, están reflejados en esta tabla:

Versión del SDK para la versión estándar de Java	Nombre que se le da al kit de desarrollo
1.1	JDK 1.1
1.2	J2SE 1.2
1.3	J2SE 1.3
1.4	J2SE 1.4
1.5	J2SE 1.5
1.6	Java SE 6
1.7	Java SE 7

Desde la versión 1.2 se habla de Java 2. Desde la versión 1.6 se ha abandonado la terminología ***Java 2*** y ahora se habla de ***Java 6*** y ***Java 7*** para las versiones 1.6 y 1.7 del kit de desarrollo.

Cada versión tiene varias revisiones, así la versión 1.6.7 del SDK indica versión 6 de Java, revisión 7.

Java 1.0 (JDK 1.0)

Fue la primera versión de Java y propuso el marco general en el que se desenvuelve Java. está oficialmente obsoleto, pero hay todavía muchos clientes con esta versión.

Java 1.1 (JDK 1.1)

Mejoró la versión anterior incorporando las siguientes mejoras:

- **JDBC**, API de acceso a bases de datos
- **RMI** llamadas a métodos remotos. Es una técnica de comunicación de procesos en red
- **JavaBeans**, componentes independientes reutilizables.
- Internacionalización para crear programas adaptables a todos los idiomas
- Clases internas

Java 2 (J2SE 1.2)

Apareció en Diciembre de 1998 al aparecer el JDK 1.2. Incorporó notables mejoras como por ejemplo:

- **JFC**. ***Java Foundation classes***. El conjunto de clases de todo para crear programas más atractivos de todo tipo. Dentro de este conjunto están:
- **El paquete Swing**. Mejorando notablemente al anterior paquete AWT. Se trata de todo un conjunto de clases que permiten desarrollar fácilmente entornos de ventanas. Es parte de JFC.
- **Enterprise Java beans.** Para la creación de componentes para aplicaciones distribuidas del lado del servidor
- **Java Media.** Conjunto de paquetes para crear paquetes multimedia:
  - **Java 2D**. Paquete (parte de JFC) que permite crear gráficos de alta calidad en los programas de Java.
  - **Java 3D**. Paquete (parte de JFC) que permite crear gráficos tridimensionales.
  - **Java Media Framework**. Paquete marco para crear elementos multimedia
  - **Java Speech**. Para reconocimiento de voz.
  - **Java Sound**. Audio de alta calidad
  - **Java TV**. Televisión interactiva
- **JNDI.** ***Java Naming and Directory Interface***. Servicio general de búsqueda de recursos. Integra los servicios de búsqueda más populares (como LDAP por ejemplo).

- **Java Servlets.** Herramienta para crear aplicaciones de servidor web (y también otros tipos de aplicaciones).
- **Java Cryptography.** Algoritmos para encriptar y desencriptar.
- **Java Help.** Creación de sistemas de ayuda.
- **Jini.** Permite la programación de electrodomésticos.
- **Java card.** Versión de Java dirigida a pequeños dispositivos electrónicos.
- **Java IDL.** Lenguaje de definición de interfaz. Permite crear aplicaciones tipo **CORBA** (plataforma de desarrollo de sistemas distribuidos)
- **Clases para la creación de colecciones**

Java 1.3 (J2SE 1.3)

- Se utiliza la máquina virtual de **Hotspot** (más rápida y segura).
- Se modifica RMI para que trabaje con CORBA
- **JPDA**, ***Java Platform Debugger Architectura***

Java 1.4 (J2SE 1.4)

- Aparecen las aserciones (**assert**)
- Expresiones regulares estilo **Perl**.
- **NIO**. Nuevo interfaz de entrada y salida de datos.
- **JAXP**. API de desarrollo de documentos **XML**.

Java 1.5 (J2SE 1.5)

- Aparecen las plantillas
- Metadatos
- **Autoboxing**, conversión automática de tipos a tipos envolventes.
- **Enumeraciones**
- Argumentos variables (***varargs***)
- Mejora del bucle **for**

Java 1.6 (Java SE 6)

- Combinación con otros lenguajes (**PHP**, **Ruby**, **Perl**,…)
- Últimas especificaciones de **JAX-WS 2.0**, **JAXB 2.0**, **STAX** y **JAXP** para crear servicios web.

(3.3.3) instalación del SDK

Es un requisito previo antes de programar en lenguaje Java.

en Windows

Desde la página de descarga http://java.sun.com/javase/downloads se elige la versión deseada del entorno de desarrollo. Una vez descargado el programa de instalación del SDK, basta con ejecutarle.

Hay que prestar atención al directorio en el que se ha instalado el SDK. La razón es que debemos modificar tres variables del sistema (variables que utiliza Windows para la configuración correcta de comandos). Son:

- **PATH**. Variable que contiene rutas por defecto a los programas que indiquemos. La razón es que por ejemplo el comando **java** debe de estar disponible estemos en la carpeta que estemos. Dicho comando (junto con el resto de comandos del SDK) está en la carpeta **bin** dentro de la carpeta en la que hemos instalado el SDK.

  Ejemplo de contenido de la variable **path**:

```
PATH=C:\WINNT\SYSTEM32;C:\WINNT;C:\WINNT\SYSTEM32\WBEM;C:\Archivos de programa\Microsoft Visual Studio\Common\Tools\WinNT;C:\Archivos de programa\Microsoft Visual Studio\Common\MSDev98\Bin;C:\Archivos de programa\Microsoft Visual Studio\Common\Tools;C:\Archivos de programa\Microsoft Visual Studio\VC98\bin;C:\Archivos de programa\java\jsdk1.6.2\bin
```

- **JAVA_HOME**. Variable utilizada por la mayoría de aplicaciones basadas en Java que contiene la ruta a la carpeta en la que se instaló el SDK.
- **CLASSPATH**. Se explicara con detalle más adelante en este mismo manual. Es una variable similar al PATH que sirve para indicar rutas a las carpetas en las que se almacenarán aplicaciones Java. Dicho de una manera más técnica: contiene las rutas de todos los **filesystems** de Java.

La forma de configurar estas variables en Windows (si al menos tenemos versión de Windows superior o igual al 2000):

(1) Señalar al icono **Mi PC** (o **Equipo** en Windows Vista, Windows 2008 o Windows 7) y elegir Propiedades. Después elegir **Propiedades Avanzadas** y finalmente pulsar en el botón **Variables de entorno**.

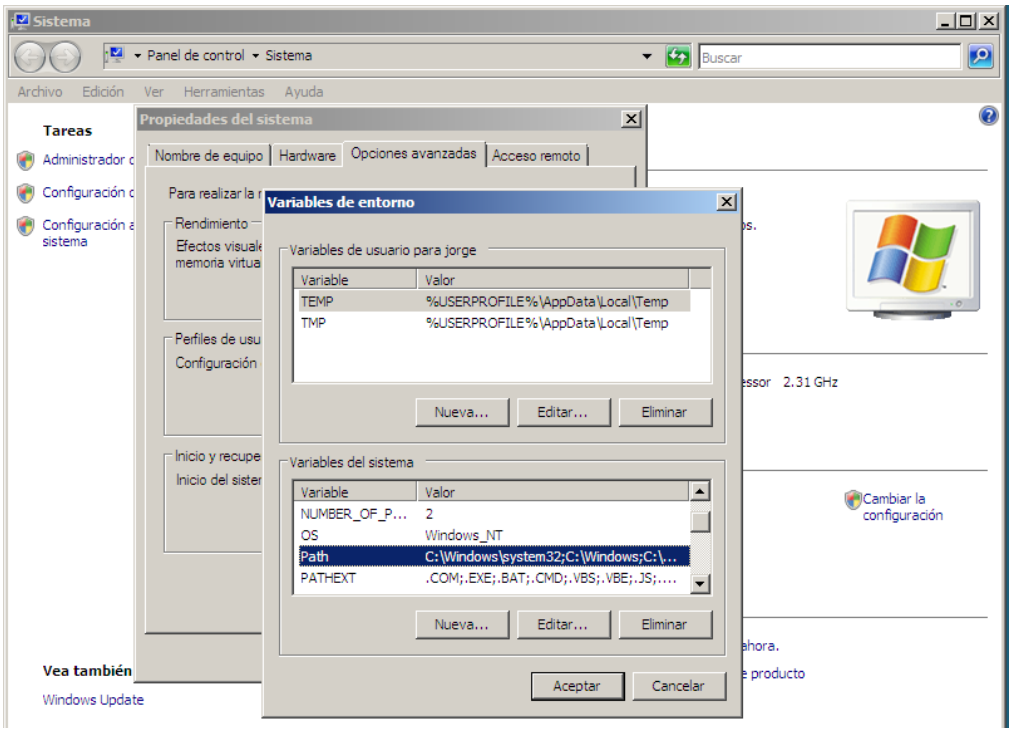


Ilustración 3-3, El cuadro de las variables del Sistema en Windows Server 2008

(2) Dentro de este cuadro, ya estará la variable PATH. Habrá que elegirla y pulsar en modificar. **Sin borrar nada de lo que contiene**, debemos añadir al final del texto el símbolo **;** y después la ruta al directorio bin dentro de la carpeta del SDK (por ejemplo **C:\"Program Files"\Java\jdk1.6.14\bin**).

(3) Tras aceptar el cuadro anterior, podremos pulsar en **Nueva** para añadir la variable JAVA_HOME indicando como valor la ruta al SDK.

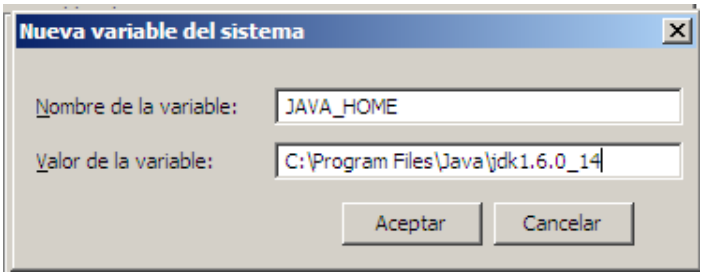


Ilustración 3-4, Ejemplo de configuración de la variable JAVA_HOME en Windows

(4) Hacer el mismo proceso para la variable CLASSPATH

Para comprobar la versión de Java basta con ir al símbolo del sistema Windows y escribir **java -version**

en Linux

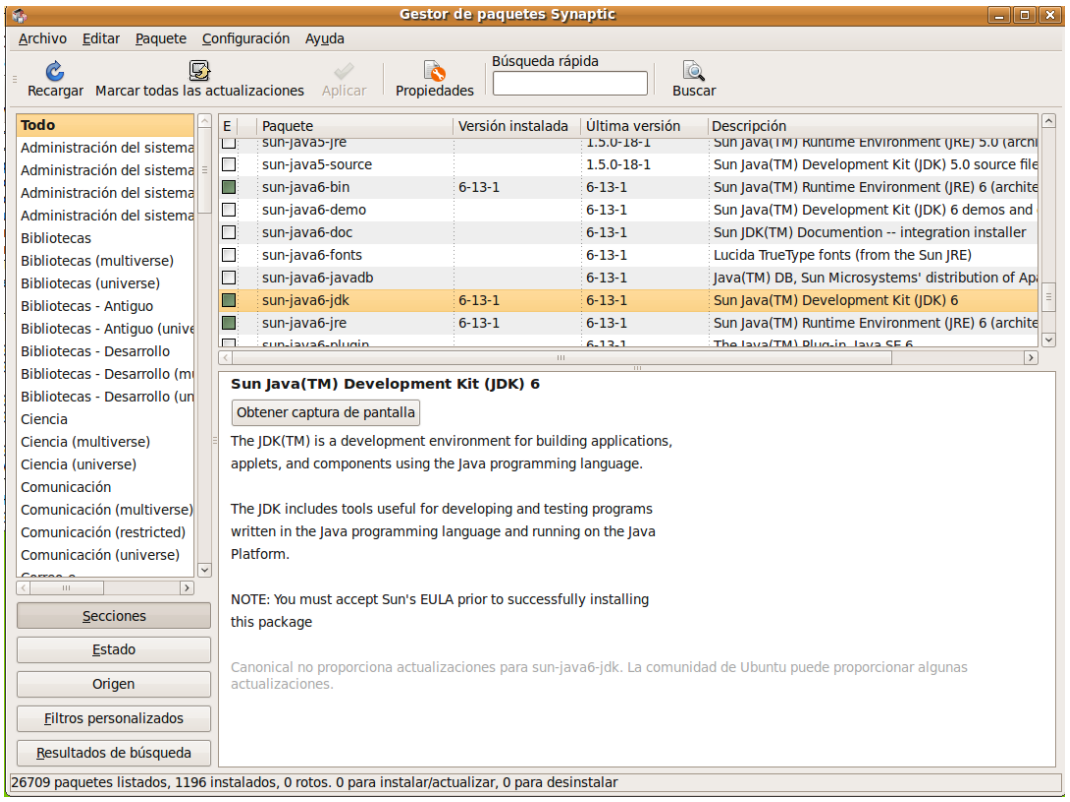


Ilustración 3-5, El gestor de paquetes Synaptic en el sistema Linux Ubuntu, mostrando información sobre los paquetes Java

Casi todas las versiones actuales de Linux incluyen el entorno de ejecución (**JRE**) de Java y la mayoría el entorno de desarrollo (**SDK**). Esto significa que seguramente no haya que instalar nada. Para conocer la versión instalada habría que ejecutar el comando **java -version**.

Si deseamos instalar Java o actualizarlo, hay que instalar el último paquete, una opción es utilizar el gestor **Synaptic** para descargar la última versión del paquete de desarrollo en Java (ver Ilustración 3-5).

También podemos instalar desde la línea de comandos, sería algo como

```
sudo apt-get install sun-java6-jdk
```

Finalmente siempre podemos acudir a la página de descargas de Sun, http://java.sun.com/javase/downloads y descargar la versión deseada. El archivo hay que descomprimirlo (si es un paquete **rpm**) o ejecutarlo (si es simplemente un archivo **bin**). Se obtendrá un directorio con todo el SDK de Java. Ahora bastará con colocarlo en el directorio adecuado.

En todo caso, sea cual sea la forma de instalar el SDK, habrá que modificar tres variables de entorno. Para lo cual lo normal es modificar el fichero **/etc/bash.bashrc** y al final añadir las siguientes entradas:

- **export JAVA_HOME="ruta en la que está el SDK de Java"**
  La ruta podría ser algo como ***/usr/lib/jvm/java6sun***

- **export PATH="$PATH:$JAVA_HOME/bin"**
- **export CLASSPATH="rutas a los directorios en los que se almacenarán programas en Java"**

En el caso de la variable CLASSPATH, es muy conveniente que su ruta incluya el directorio actual (el símbolo .)~. Un ejemplo de uso de CLASSPATH en el archivo bashrc sería:

```
export CLASSPATH="/usr/lib/jvm/java-6-sun:."
```

Gracias a ello podremos colocar programas Java y ejecutarlos sin problemas en la carpeta raíz del SDK y en la carpeta actual en la que nos encontremos.

instalación del código fuente de las bibliotecas

Como otros lenguajes, Java se compone de una serie de bibliotecas. En el caso de Java estas bibliotecas son inmensas y están comprimidas en el archivo src.jar.

Es muy habitual en Java el uso de archivos **JAR**. En realidad son archivos que aglutinan código fuente en java, comprimiéndoles en formato **ZIP**. Para poder acceder al código fuente en sí debemos descomprimir dicho archivo. Lo cual se hace (en cualquier sistema):

(1) Yendo a la carpeta o directorio en el que se encuentra el SDK (es decir yendo a **JAVA_HOME**).

(2) Allí se encuentra el archivo **src.jar**

(3) Descomprimirle con la orden **jar xvf src.jar**

Actualmente el archivo se llama **src.zip**, con lo que se puede descomprimir con cualquier programa capaz de manejar archivos ZIP.

documentación

En **http://java.sun.com/docs** se encuentra la documentación oficial de Java. La más interesante es la documentación del API de Java (clases estándar de Java). Aunque inicialmente es conveniente pasar por **Get Started**. La única pega es que no hay documentación en castellano.

Otra posibilidad (más recomendable incluso), es descargarse la documentación en un archivo **zip**. De esta forma no dependeremos de la conexión a Internet para consultar la documentación oficial. Esto se puede realizar de la página de descarga http://java.sun.com/javase/downloads/index.jsp

(3.3.4) entornos de trabajo

El código en Java se puede escribir en cualquier editor de texto. Y para compilar el código en bytecodes, sólo hace falta descargar la versión del JDK deseada. Sin embargo, la escritura y compilación de programas hecha de esta forma es un poco incomoda. Por ello numerosas empresas fabrican sus propios

entornos de edición, algunos incluyen el compilador y otras utilizan el propio JDK de Sun.

Algunas ventajas que ofrecen son:

- Facilidades para escribir código: coloreado de las palabras clave, autocorrección al escribir, abreviaturas,…
- Facilidades de depuración, para probar el programa
- Facilidad de configuración del sistema. Elección concreta del directorio del SDK, manipulación de la variable CLASSPATH, etc.
- Facilidades para organizar los archivos de código.
- Facilidad para exportar e importar proyectos

Los IDE más utilizados en la actualidad son:

- **Eclipse**. Es un entorno completo de código abierto que posee numerosas extensiones (incluido un módulo para JEE y otro para programar en C++) y posibilidades. Hoy en día es el más utilizado y recomendable. Posee una comunidad muy dinámica y desde luego es uno de los más potentes. Disponible en www.eclipse.org.
- **NetBeans**. Entorno gratuito de código abierto para la generación de código en diversos lenguajes (especialmente pensado para Java). Contiene prácticamente todo lo que se suele pedir a un IDE, editor avanzado de código, depurador, diversos lenguajes, extensiones de todo tipo (CORBA, Servlets,...). Incluye además un servidor de aplicaciones **Tomcat** para probar aplicaciones de servidor. Se descarga en www.netbeans.org.
- **JBuilder**. Entorno completo creado por la empresa Borland (famosa por su lenguaje **Delphi**) para la creación de todo tipo de aplicaciones Java, incluidas aplicaciones para móviles. Es un entorno muy potente.
- **JDeveloper**. De Oracle. Entorno completo para la construcción de aplicaciones Java y XML. Uno de los más potentes y completos (ideal para programadores de Oracle).
- **IntelliJ Idea**. Entorno comercial de programación bastante fácil de utilizar pero a la vez con características similares al resto. Es menos pesado que los anteriores y muy bueno con el código.
- **Microsoft Visual J++ y Visual J#.** Ofrece un compilador recomendable para los conocedores de los entornos de desarrollo de **Microsoft** (como **Visual Basic** por ejemplo), de hecho son lenguajes integrados en el **Visual Studio** y **Visual Studio.Net** respectivamente. Pero el Java de Microsoft no es estándar. El único uso que tiene es para aquellos programadores de Java que necesitan programar para la plataforma .Net de Microsoft y que no desean aprender los lenguajes propios de esta plataforma como **C#** o **Visual Basic** por ejemplo.

En este documento hay información sobre el manejo de **Eclipse** y de **Netbeans** para crear aplicaciones en Java (véanse los apéndices 1 y 2).

(3.4) escritura de programas Java

(3.4.1) codificación del texto

Todos el código fuente Java se escriben en documentos de texto con extensión **.java**. Por lo tanto para escribirlo basta cualquier editor de texto como el **bloc de notas** de Windows o los editores **vi** o **gedit** de Linux.

Al ser un lenguaje multiplataforma y especialmente pensado para su integración en redes, la codificación de texto utiliza el estándar **Unicode**; lo que implica que los programadores y programadoras hispanohablantes podemos utilizar sin problemas símbolos de nuestra lengua como la eñe o las vocales con tildes o diéresis a la hora de poner nombre a nuestras variables.

La codificación Unicode[4] usa normalmente 16 bits (2 bytes por carácter) e incluye la mayoría de los códigos alfabéticos del mundo.

(3.4.2) notas previas

Los archivos con código fuente en Java deben guardarse con la extensión .java. Como se ha comentado cualquier editor de texto basta para crearle. Algunos detalles importantes son:

- En java (como en C) hay diferencia entre mayúsculas y minúsculas.
- Cada línea de código debe terminar con ;
- Una instrucción puede abarcar más de una línea. Además se pueden dejar espacios y tabuladores a la izquierda e incluso en el interior de la instrucción para separar elementos de la misma.
- Los comentarios; si son de una línea debe comenzar con **//** y si ocupan más de una línea deben comenzar con **/*** y terminar con ***/**

```
/* Comentario
de varias líneas */
//Comentario de una línea
```

También se pueden incluir comentarios **javadoc** (se explican en detalle más adelante)

[4] Para más información acudir a **http://www.unicode.org**

- A veces se marcan bloques de código, es decir código agrupado. Cada bloque comienza con **{** y termina con **}**

```
{
   ...código dentro del bloque
}
código fuera del bloque
```

(3.4.3) el primer programa en Java

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

Este código sirve para escribir ***¡Mi primer programa!*** en la pantalla. Para empezar a entender el código:

- La primera línea (***public class PrimerPrograma***) declara el nombre de la **clase** del código. Más adelante se explicará que significa el concepto de clase; por ahora entenderemos que el nombre de la clase es el nombre del programa.
- La línea ***public static void main(String args[])***, sirve para indicar el inicio del método **main**. Este método contiene las instrucciones que se ejecutarán cuando el programa arranque. Es decir lo que está tras las llaves del ***main***, es el programa en sí.
- La instrucción ***System.out.println*** sirve para escribir en pantalla. Como lo que escribimos es un texto, se encierra entre comillas.

Además, el archivo tiene que llamarse obligatoriamente **PrimerPrograma.java** ya que el nombre del programa (en realidad el nombre de la clase) y el del archivo deben coincidir.

Por último, aunque no es obligatorio, es más que aconsejable que el nombre del programa comience con una letra mayúscula y le sigan letras en minúsculas. Si consta de varias palabras no pueden utilizarse espacios en blanco, por lo que se suelen juntar las palabras poniendo cada inicial de la palabra en mayúsculas.

Éste tipo de reglas no obligatorias sino aconsejables (como por ejemplo el hecho de que las instrucciones interiores a un bloque dejen espacio a su izquierda) producen un código más legible y sobre todo hace que todos los programadores del planeta adoptemos la misma forma de escribir, simplificando el entendimiento del código.

(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.

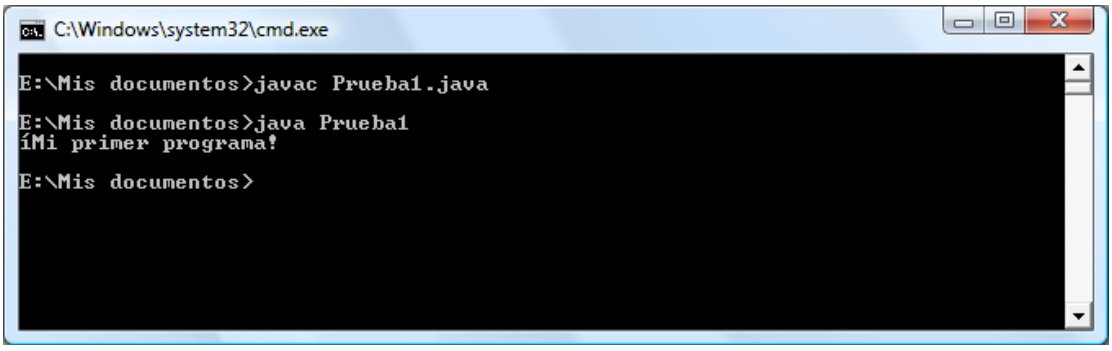


Ilustración 3-6, Ejemplo de compilación y ejecución en la línea de comandos

(3.6) javadoc

Javadoc es una herramienta muy interesante del kit de desarrollo de Java para generar automáticamente documentación Java. genera documentación para paquetes completos o para archivos java. Su sintaxis básica es:

```
javadoc archivo.java o paquete
```

El funcionamiento es el siguiente. Los comentarios que comienzan con los códigos **/**** se llaman comentarios de documento y serán utilizados por los programas de generación de documentación javadoc. Los comentarios javadoc comienzan con el símbolo **/**** y terminan con ***/**

Cada línea ***javadoc*** se suele iniciar con el símbolo de asterisco para mejorar su legibilidad. Dentro se puede incluir cualquier texto; incluso se pueden utilizar códigos **HTML** para que al generar la documentación se tenga en cuenta el código HTML indicado.

En el código javadoc se pueden usar **etiquetas** especiales, las cuales comienzan con el símbolo @. Pueden ser:

- **@author.** Tras esa palabra se indica el autor del documento.
- **@version.** Tras lo cual sigue el número de versión de la aplicación
- **@see.** Tras esta palabra se indica una referencia a otro código Java relacionado con éste.
- **@since.** Indica desde cuándo esta disponible este código
- **@deprecated.** Palabra a la que no sigue ningún otro texto en la línea y que indica que esta clase o método esta obsoleta u obsoleto.
- **@throws.** Indica las excepciones que pueden lanzarse en ese código.
- **@param.** Palabra a la que le sigue texto qué describe a los parámetros que requiere el código para su utilización (el código en este caso es un método de clase). Cada parámetro se coloca en una etiqueta @param distinta, por lo que puede haber varios @param para el mismo método.
- **@return.** Tras esta palabra se describe los valores que devuelve el código (el código en este caso es un método de clase)

El código javadoc hay que colocarle en tres sitios distintos dentro del código java de la aplicación:

- **Al principio del código de la clase** (antes de cualquier código Java). En esta zona se colocan comentarios generales sobre la clase o interfaz que se crea mediante el código Java. Dentro de estos comentarios se pueden utilizar las etiquetas: **@author**, **@version**, **@see**, **@since** y **@deprecated**

- **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.

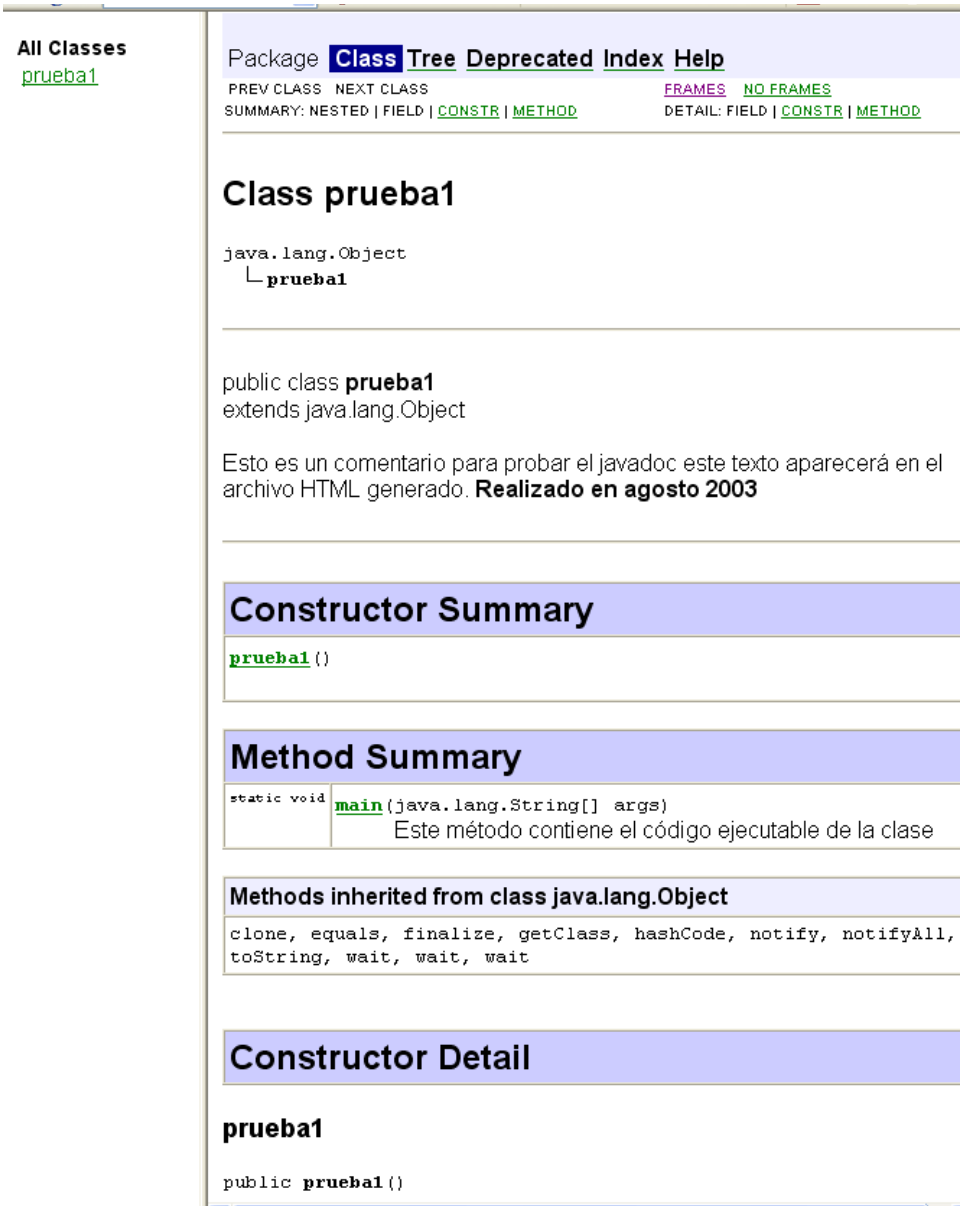


Ilustración 3-7, Página de documentación de un programa Java

(3.7) import

En cualquier lenguaje de programación existen librerías que contienen código ya escrito que nos facilita la creación de programas. En el caso de Java no se llaman librerías, sino paquetes. Los paquetes son una especie de carpetas que contienen clases ya preparadas y más paquetes.

Cuando se instala el kit de desarrollo de Java, además de los programas necesarios para compilar y ejecutar código Java, se incluyen miles de clases dentro de cientos de paquetes ya listos que facilitan la generación de programas. Algunos paquetes sirven para utilizar funciones matemáticas,

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 quisiéramos 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**

que simbolice letras (valen símbolos como la ***ñ***, ***á***,... no permitidos en la mayoría de lenguajes por no ser parte del ASCII estándar)

(3.8.2) declaración de variables

Antes de poder utilizar una variable, ésta se debe declarar. Lo cual se debe hacer de esta forma:

```
tipo nombrevariable;
```

Donde ***tipo*** es el tipo de datos que almacenará la variable (texto, números enteros,...) y ***nombrevariable*** es el nombre con el que se conocerá la variable. Ejemplos:

```
int días; // días es un número entero, sin decimales
boolean decisión; //decisión sólo puede ser verdadera o falsa
```

También se puede hacer que la variable tome un valor inicial al declarar:

```
int días=365;
```

Y se puede declarar más de una variable a la vez del mismo tipo en la misma línea si las separamos con comas

```
int días=365, año=23, semanas;
```

Al declarar una variable se puede incluso utilizar una expresión:

```
int a=13, b=18;
int c=a+b; //es válido, c vale 31
```

Java es un lenguaje muy estricto al utilizar tipos de datos. Variables de datos distintos son incompatibles. Algunos autores hablan de ***lenguaje fuertemente tipado*** o incluso ***lenguaje muy tipificado***. Se debe a una traducción muy directa del inglés ***strongly typed*** referida a los lenguajes que, como Java, son muy rígidos en el uso de tipos.

El caso contrario sería el lenguaje C en el que jamás se comprueban de manera estricta los tipos de datos.

Parte de le seguridad y robustez de las que hace gala Java se deben a esta característica.

(3.8.3) asignación

En Java para asignar valores a una variable, basta con utilizar el signo **=**. Ya se ha visto en el apartado anterior que al declarar se puede asignar un valor:

```
int x=7;
```

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.

```
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;
```

```
byte b=i;//error de compilación, posible pérdida de precisión
```

La solución es hacer un **cast**. Esta operación permite convertir valores de un tipo a otro. Se usa así:

```
int i=12;
byte b=(byte) i;        //El (cast) evita el error
```

Hay que tener en cuenta en estos castings que si el valor asignado sobrepasa el rango del elemento, el valor convertido no tendrá ningún sentido ya que no puede almacenar todos los bits necesarios para representar ese número:

```
int i=1200;
byte b=(byte) i;        //El valor de b no tiene sentido
```

(3.9.2) números en coma flotante

Los decimales se almacenan en los tipos **float** y **double**. Se les llama de coma flotante por como son almacenados por el ordenador. Los decimales no son almacenados de forma exacta por eso siempre hay un posible error. En los decimales de coma flotante se habla, por tanto de precisión. Es mucho más preciso el tipo **double** que el tipo **float.**

Para asignar valores literales a una variable de coma flotante, hay que tener en cuenta que el separador decimal es el punto y no la coma. Es decir para asignar el valor ***2,75*** a la variable x se haría:

```
x=2.75;
```

A un valor literal (como 1.5 por ejemplo), se le puede indicar con una **f** al final del número que es float (1.5**f** por ejemplo) o una **d** para indicar que es **double.** Si no se indica nada, un número literal siempre se entiende que es double, por lo que al usar tipos float hay que convertir los literales.

Los valores decimales se pueden representar en notación decimal: ***1.345E+3*** significaría $1{,}345 \cdot 10^3$ o lo que es lo mismo ***1345***.

Lógicamente no podemos asignar valores decimales a tipos de datos enteros:

```
int x=9.5; //error
```

Sí podremos mediante un cast:

```
int x=(int) 9.5;
```

pero perderemos los decimales (en el ejemplo, ***x*** vale ***9***). El caso contrario sin embargo sí se puede hacer:

```
int x=9;
double y=x; //correcto
```

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:

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*

(3.9.5) conversión entre tipos (casting)

Ya se ha comentado anteriormente la necesidad de uso del operador de casting para poder realizar asignaciones entre tipos distintos. Como resumen general del uso de casting véanse estos ejemplos:

```
int a;byte b=12;
a=b;
```

El código anterior es correcto porque un dato ***byte*** es más pequeño que uno ***int*** y Java le convertirá de forma implícita. Lo mismo pasa de ***int*** a ***double*** por ejemplo. Sin embargo en:

```
int a=1;
byte b;
b=a;
```

El compilador devolverá error aunque el número 1 sea válido para un dato byte. Para ello hay que hacer un ***casting***. Eso significa poner el tipo deseado entre paréntesis delante de la expresión.

```
int a=1;
byte b;
b= (byte) a; //correcto
```

En el siguiente ejemplo:

```
byte n1=100, n2=100, n3;
n3= n1 * n2 /100;
```

Aunque el resultado es 100, y ese resultado es válido para un tipo byte; lo que ocurrirá en realidad es un error. Eso es debido a que la multiplicación ***100 * 100*** da como resultado ***10000***, es decir un número de tipo ***int***. Aunque luego se divide entre 100, no se vuelve a convertir a byte; ya que **ante cualquier operación el tipo resultante siempre se corresponde con el tipo más grande que intervenga en la operación**. Lo correcto sería:

```
n3 = (byte) (n1 * n2 / 100);
```

(3.9.6) ámbito de las variables

Toda variable tiene un **ámbito**. Esto es la parte del código en la que una variable se puede utilizar. De hecho las variables tienen un ciclo de vida:

(1) En la declaración se reserva el espacio necesario para que se puedan comenzar a utilizar (digamos que se avisa de su futura existencia)

(2) Se la asigna su primer valor (la variable ***nace***)

(3) Se la utiliza en diversas sentencias

(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:

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.

Es más incluso:

```
double resultado;
int i1=7,i2=2;
resultado=i1/i2; //Resultado valdrá 3
resultado=(double)i1/i2; //Resultado valdrá 3.5
```

El operador del módulo (%) sirve para calcular el resto de una división entera. Ejemplo:

```
int resultado, i1=14, i2=5;

resultado = i1 % i2; //El resultado será 4
```

El módulo sólo se puede utilizar con tipos enteros

(3.10.3) operadores condicionales

Sirven para comparar valores. Siempre devuelven valores booleanos. Son:

operador	significado
<	Menor
>	Mayor
>=	Mayor o igual
<=	Menor o igual
==	Igual
!=	Distinto
!	No lógico (NOT)
&&	"Y" lógico (AND)
||	"O" lógico (OR)

Los operadores lógicos (**AND**, **OR** y **NOT**), sirven para evaluar condiciones complejas. NOT sirve para negar una condición. Ejemplo:

```
boolean mayorDeEdad, menorDeEdad;
int edad = 21;
mayorDeEdad = edad >= 18; //mayorDeEdad será true
menorDeEdad = !mayorDeEdad; //menorDeEdad será false
```

El operador **&&** (**AND**) sirve para evaluar dos expresiones de modo que si ambas son ciertas, el resultado será **true** sino el resultado será **false**. Ejemplo:

```
boolean carnetConducir=true;
int edad=20;
boolean puedeConducir= (edad>=18) && carnetConducir;
//Si la edad es de al menos 18 años y carnetConducir es
//true, puedeConducir es true
```

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
>>	Desplazamiento a la derecha
<<	Desplazamiento a la izquierda
>>>	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.

Ejemplo:

```
int x=5, y=5, z;
z=x++; //z vale 5, x vale 6
z=++y; //z vale 6, y vale 6
```

(3.10.6) operador ?

Este operador (conocido como **if** de una línea) permite ejecutar una instrucción u otra según el valor de la expresión. Sintaxis:

```
expresionlogica?valorSiVerdadero:valorSiFalso;
```

Ejemplo:

```
paga=(edad>18)?6000:3000;
```

En este caso si la variable edad es mayor de 18, la paga será de 6000, sino será de 3000. Se evalúa una condición y según es cierta o no se devuelve un valor u otro. Nótese que esta función ha de devolver un valor y no una expresión correcta. Es decir, no funcionaría:

```
(edad>18)? paga=6000: paga=3000; //ERROR!!!!
```

(3.10.7) precedencia

A veces hay expresiones con operadores que resultan confusas. Por ejemplo en:

```
resultado = 8 + 4 / 2;
```

Es difícil saber el resultado. ¿Cuál es? ¿seis o diez? La respuesta es 10 y la razón es que el operador de división siempre precede en el orden de ejecución al de la suma. Es decir, siempre se ejecuta antes la división que la suma. Siempre se pueden usar paréntesis para forzar el orden deseado:

```
resultado = (8 + 4) / 2;
```

Ahora no hay duda, el resultado es seis. No obstante el orden de precedencia de los operadores Java es:

nivel	operador			
1	()	[]	.	
2	++	--	~	!
3	*	/	%	
4	+	-		
5	>>	>>>	<<	<<<
6	>	>=	<	<=

nivel	operador			
7	==	!=		
8	&			
9	^			
10	|			
11	&&			
12	||			
13	?:			
14	=	+=, -=, *=,...		

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.

(3.12) lectura y escritura por teclado

(3.12.1) escritura

Ya hemos visto que hay una función para escribir que es ***System.out.println***. Dicha función puede recibir como parámetro cualquier tipo básico de datos: es decir puede recibir un texto literal (siempre entre comillas), pero también puede escribir expresiones enteras, booleanas, decimales y caracteres simples.

Ejemplo:

```
int a=5, b=9;
double c=5.5;

System.out.println("Este es un texto literal");
System.out.println(a+b); //Escribe 14
System.out.println(c*c); //Escribe 30.25
System.out.println(a<c);//Escribe true
```

Esta función tras escribir añade un salto de línea, de modo que lo siguiente que se escriba saldrá en otra línea. Existe una variante de esta función que no inserta el salto de línea es ***System.out.print***:

```
System.out.print("todo en la ");
System.out.print("misma línea ");
```

Si deseamos que el mismo ***println*** o ***print*** escriba varios valores en la misma instrucción, podemos usar el operador de encadenar textos, ejemplo:

```
int a=5, b=9;
System.out.println("La suma es "+(a+b));
```

Es necesario usar paréntesis ya que se utiliza en la expresión el operador **+** con dos significados. Este operador concatena cuando al menos hay un texto a la izquierda o derecha del operador; y suma cuando tenemos dos números (del tipo que sea).

(3.12.2) lectura

La lectura en Java es mucho más complicada. Leer de la consola de salida requiere manejar muchos conocimientos que pertenecen a temas más avanzados que el actual.

Una forma no tan complicada (aunque desde luego no es tan sencilla como la escritura) es utilizar la clase **JOptionPane**. Dicha clase pertenece al paquete **javax.swing**, por lo que para utilizarla sin tener que escribir el nombre completo conviene usar la instrucción:

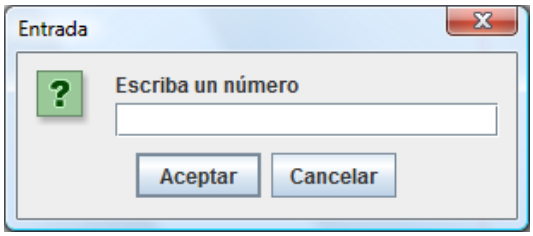
```
import javax.swing.JOptionPane;
```

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 almacenarles 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 tara importante para nuestros programas.

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

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

Float.parseFloat	**float**
Double.parseDouble	**double**
Boolean.parseBoolean	**boolean**

Hay que tener cuidado con las mayúsculas, son obligatorias donde aparezcan.

(3.13) la clase Math

Se echan de menos operadores matemáticos más potentes en Java. Por ello se ha incluido una clase especial llamada **Math** dentro del paquete **java.lang.** Para poder utilizar esta clase, se debe incluir esta instrucción (aunque normalmente no es necesario porque todo el paquete *java.lang* ya estará incluido en nuestro código):

```
import java.lang.Math;
```

Esta clase posee métodos muy interesantes para realizar cálculos matemáticos complejos. Por ejemplo:

```
double x= Math.pow(3,3); //x es 3³, es decir 27
```

Math posee dos constantes, que son:

constante	significado
double **E**	El número **e** (2, 7182818245…)
double **PI**	El número **∏** (3,14159265…)

Por otro lado posee numerosos métodos que son:

operador	significado
double **ceil(**double *x***)**	Redondea *x* al entero mayor siguiente: ♦ Math.ceil(2.8) vale 3 ♦ Math.ceil(2.4) vale 3 ♦ Math.ceil(-2.8) vale -2
double **floor(**double x**)**	Redondea *x* al entero menor siguiente: ♦ Math.floor(2.8) vale 2 ♦ Math. floor (2.4) vale 2 ♦ Math. floor (-2.8) vale -3
int **round(**double *x***)**	Redondea *x* de forma clásica: ♦ Math.round(2.8) vale 3 ♦ Math. round (2.4) vale 2 ♦ Math. round (-2.8) vale -3

operador	significado
double **rint(**double *x***)**	Idéntico al anterior, sólo que éste método da como resultado un número **double** mientras que **round** da como resultado un entero tipo **int**
double **random()**	Número aleatorio decimal situado entre el ***0*** y el ***1***
tiponúmero **abs(** tiponúmero *x***)**	Devuelve el valor absoluto de ***x.***
tiponúmero **min(** tiponúmero *x*, tiponúmero *y***)**	Devuelve el menor valor de ***x*** o ***y***
tiponúmero **max(** tiponúmero *x*, tiponúmero *y***)**	Devuelve el mayor valor de ***x*** o ***y***
double **sqrt(**double x**)**	Calcula la raíz cuadrada de ***x***
double **pow(**double x**,** double y**)**	Calcula x^y
double **exp(**double x**)**	Calcula e^x
double **log(**double x**)**	Calcula el logaritmo neperiano de ***x***
double **acos(**double x**)**	Calcula el arco coseno de ***x***
double **asin(**double x**)**	Calcula el arco seno de ***x***
double **atan(**double x**)**	Calcula el arco tangente de ***x***
double **sin(**double x**)**	Calcula el seno de ***x***
double **cos(**double x**)**	Calcula el coseno de ***x***
double **tan(**double x**)**	Calcula la tangente de ***x***
double **toDegrees(** double anguloEnRadianes**)**	Convierte de radianes a grados
double **toRadians(** double anguloEnGrados**)**	Convierte de grados a radianes
double **signum(**double n**)**	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 apareción en la versión 1.5 de Java.
double **hypot(** double x, double y**)**	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
double **nextAfter(** double valor, double dir**)**	Devuelve el siguiente número representable desde el valor indicado hacia la dirección que indique el valor del parámetro ***dir***. Por ejemplo ***Math.nextAfter(34.7, 90)*** devolvería 34.7000000001 Función añadida en la versión Java 1.6

(3.13.2) números aleatorios

Una de las aplicaciones más interesantes de **Math** es la posibilidad de crear números aleatorios. Para ello se utiliza el método **random** que devuelve un número **double** entre cero y uno.

Para conseguir un número decimal por ejemplo entre cero y diez bastaría utilizar la expresión:

```
Math.random()*10
```

Si el número queremos que se encuentre entre uno y diez, sería:

```
Math.random()*9+1
```

Y si queremos que sea un número entero entre 1 y 10, la expresión correcta es:

```
(int) Math.floor(Math.random()*10+1)
```

Entre 10 y 30 sería:

```
(int) Math.floor(Math.random()*21+10)
```

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**...

(I.ii) descarga de Eclipse

La descarga se realiza desde la página de la Fundación Eclipse, www.eclipse.org en el apartado **downloads**, se puede elegir descargar el entorno para **Java EE** o para programar en **Java Standard** (hay disponibles muchas otras descargas y en cualquier caso se pueden ampliar las funcionalidades de Eclipse descargando nuevos paquetes).

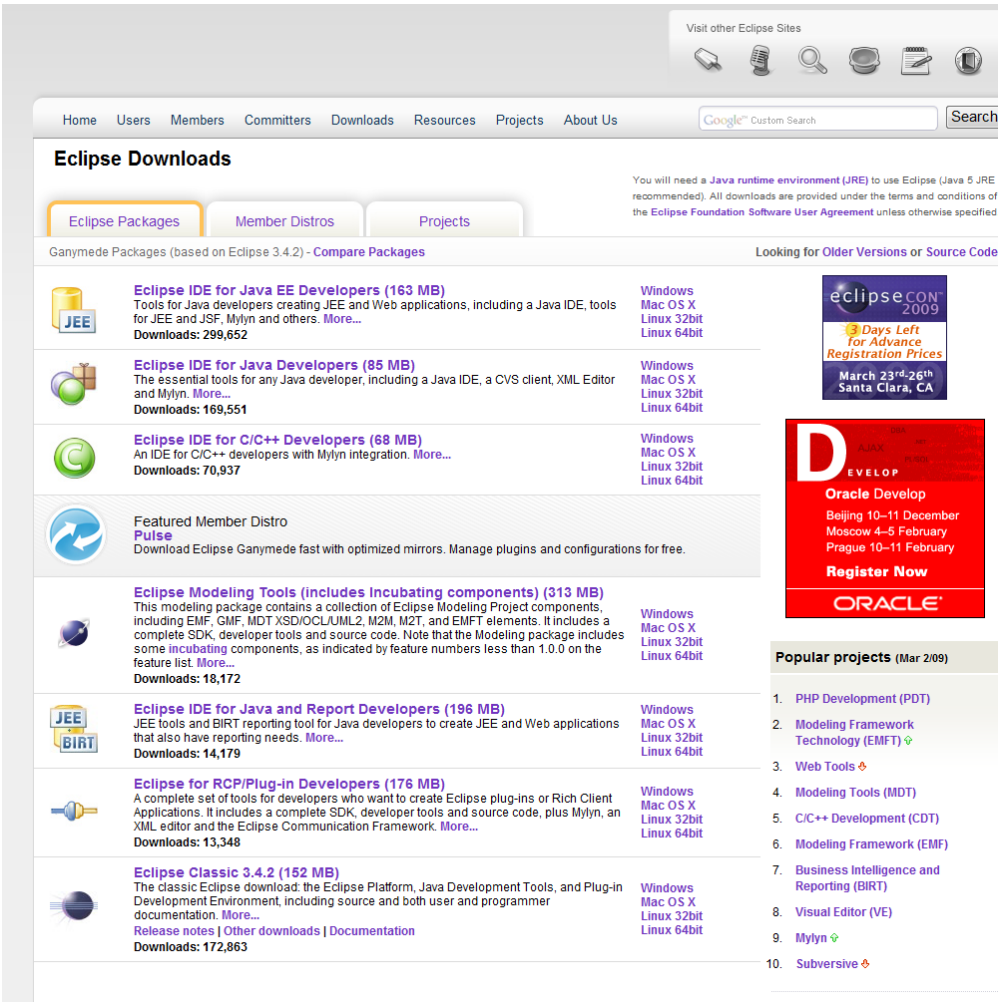


Ilustración 3-8, Pantalla de la página de descargas de Eclipse

Tras la descarga se obtiene un archivo comprimido que habrá que descomprimir. Eclipse no requiere instalación ni siquiera en Windows con lo que basta con colocarle en el sitio que deseemos y después ejecutar el archivo ejecutable **Eclipse** para hacer funcionar el entorno.

Al arrancar por primer vez Eclipse, nos pregunta por la carpeta o directorio en la que se almacenarán los proyectos, conviene elegir una carpeta distinta respecto a la que propone Eclipse para tener más controlada la ubicación de los archivos.

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.

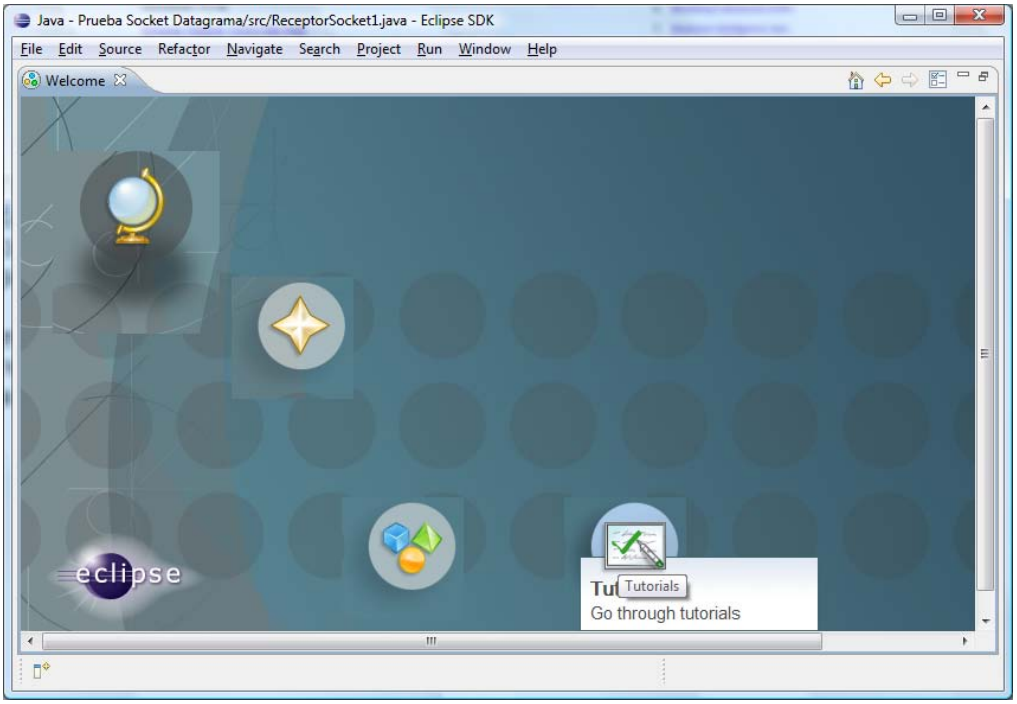


Ilustración 3-9, **Aspecto del primer arranque de Eclipse**

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.

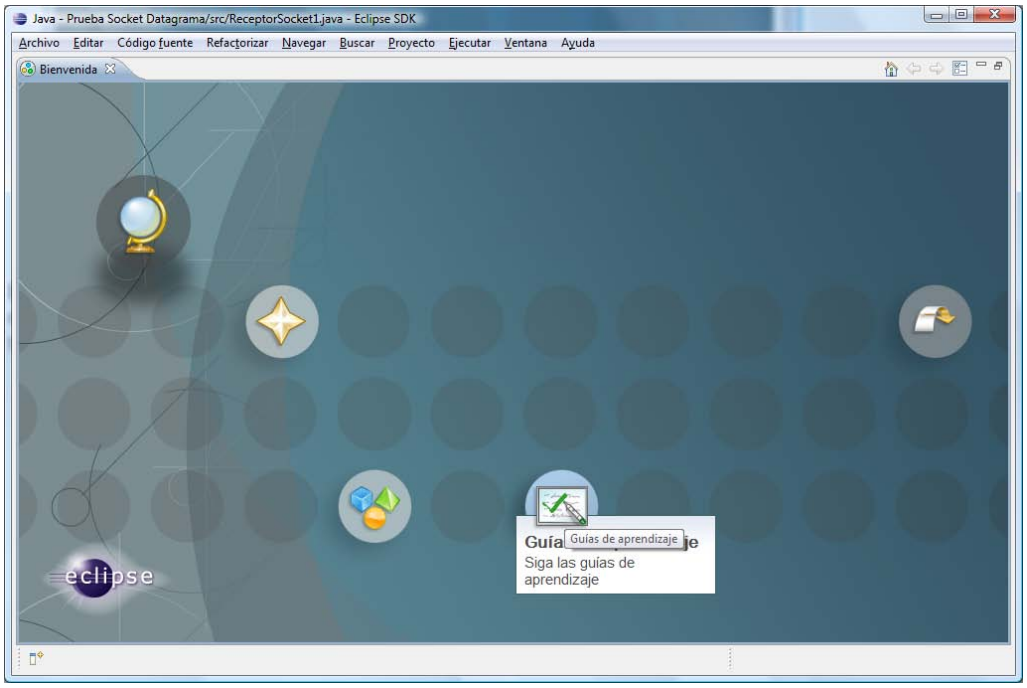


Ilustración 3-10, **Aspecto de la bienvenida de Eclipse una vez descomprimido el paquete de Babel para lenguaje español en la carpeta de Eclipse**

En este manual se utiliza Eclipse en español y también se indica la traducción al inglés.

Nota: Parece ser que existen problemas de incompatibilidad en Windows entre Eclipse y la versión de la máquina virtual de Java de 64 bit. Por eso en Windows si se desea utilizar Eclipse hay que seguir instalando la versión del SDK de 32 bits. O bien lanzar Eclipse con una máquina de 32 bits aunque nuestros programas Java les probemos en un SDK de 64 bits, ya que en Eclipse se pueden especificar SDK diferentes, como se ha visto antes.

(I.iii) aspecto de Eclipse

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

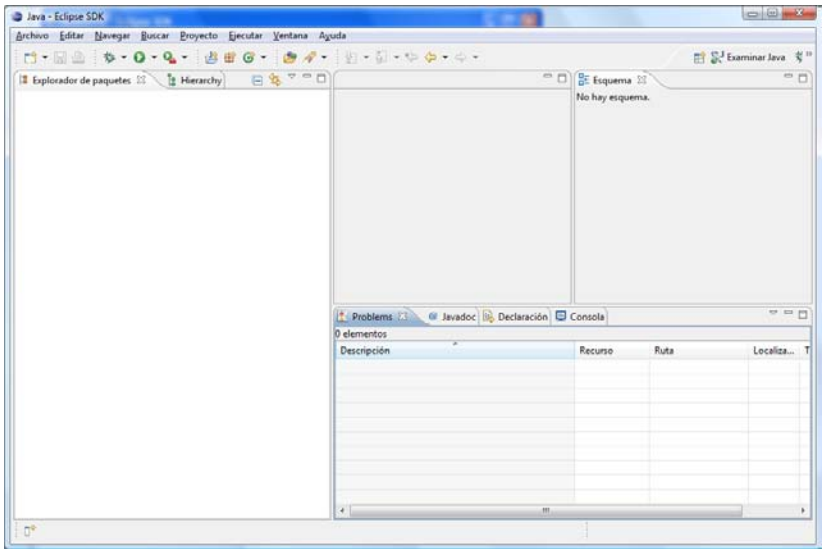


Ilustración 3-11, Aspecto del área de trabajo de Eclipse al iniciar por primera vez

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.

Las vistas se puede minimizar haciendo clic en el botón . En ese caso aparecerá una miniatura en forma de icono referido a la vista minimizada. Después el botón , permite recuperar el estado anterior de la vista.

Las vistas se pueden maximizar , en cuyo caso ocupan la pantalla entera. Nuevamente el botón de restaurar ( ), permite colocar la vista a su estado normal.

Pulsando el botón derecho sobre la pestaña con el nombre de la vista, disponemos de otras opciones interesantes

editores

También son componentes, pero que permiten la escritura y la navegación. El editor de Java es el componente fundamental para escribir código en Java. Normalmente un editor aparece cuando hacemos doble clic en Eclipse sobre un elemento que permite su edición (por ejemplo al hacer doble clic en un archivo de código fuente Java).

Los editores son distintos dependiendo del elemento que editan. Por ejemplo si se edita código Java, aparecerá coloreado de manera especial las palabras reservadas de Java; mientras que si se abre un archivo de texto, todo el texto aparece igual.

cambiar la perspectiva

Como se indicó antes las perspectivas aglutinan una serie de editores y vistas en una determinada posición en la pantalla.

Mediante el menú **Ventana** (*Window*)-**Abrir perspectiva** (*Open perspective*) se puede cambiar la perspectiva actual. Incluso se pueden almacenar perspectivas creadas por nosotros mismos (mediante **Ventana** (*Window*)-**Guardar perspectiva** (*Save perspective as*)).

También podemos cambiar de perspectiva utilizando los botones para esa tarea que están arriba y a la derecha de Eclipse (Examinar Java).

En el caso de jugar mucho con los paneles y ventanas de Eclipse, podríamos desear recuperar la perspectiva típica para trabaja con Java la cual se abre con menú **Ventana** (*Window*)-**Abrir perspectiva** (*Open perspective*)-**Java.**

(I.iv) crear proyectos en Eclipse

En casi todos los IDE del mercado se llama proyecto al conjunto de archivos de código y recursos que permiten generar una aplicación. Considerando que cualquier programa es una aplicación, para poder crear programas necesitamos crear un proyecto.

(I.iv.i) crear proyectos básicos de Java

Para empezar a practicar Java al menos necesitaremos un primer proyecto. Para crear un proyecto Java hay que elegir **Archivo** (*File*)-**Nuevo** (*New*)-

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

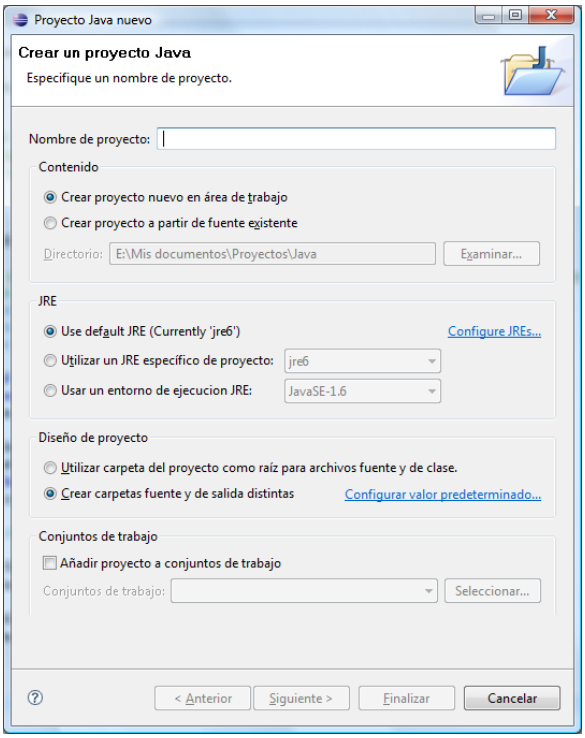


***Ilustración 3-12*, Cuadro de creación de proyectos nuevos en Eclipse**

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.iv.ii) modificar recursos del proyecto

Sin entrar mucho en detalles (hay que recordar que este es un manual de Java y no de Eclipse) desde el explorador de paquetes se puede cambiar el nombre, borrar, mover,... a cualquier recurso del proyecto (incluido el propio proyecto).

(I.iv.iii) crear programas Java (clases)

Se ha comentado en el capítulo anterior que, por ahora, un programa Java y una clase será lo mismo. Por eso no es así siempre. Desde el punto de vista de Java un programa ejecutable es una clase que tiene método **main**.

Un proyecto puede estar compuesto de numerosas clases. Para crear una clase hay que:

(1) Elegir donde queremos crear la clase (normalmente en la carpeta **src** de un proyecto) y ahí pulsar el botón derecho y elegir **Nuevo-clase** (también se puede desde el botón 

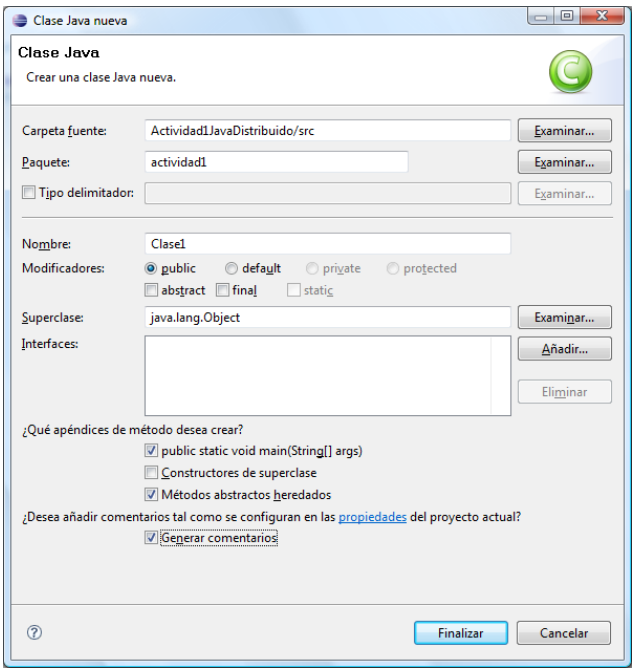


Ilustración 3-13, **Cuadro de nueva clase Java en Eclipse**

(2) En el cuadro que aparece tendremos que poner nombre a la clase y marcar la casilla con el texto ***public static void main(String[] args)*** que es la que permite generar automáticamente el método **main**. también es aconsejable marcar la casilla **Generar comentarios** (***Generate comments***) para que aparezcan los comentarios de usuario en el código (estos comentarios iniciales se pueden modificar).

El resto de elementos del cuadro (como el nombre del paquete, los modificadores, la superclases,...) se conocerán más adelante ya que

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 *redenominar* un elemento de Java se puede deshacer desde **Edición-Deshacer** (*Undo*)

(I.v) ejecución de programas Java

Sólo se pueden ejecutar los programas que dispongan de método **main**. Gracias a esta posibilidad podemos probar nuestros programas sin abandonar el IDE Eclipse.

Se puede hacer desde varios sitios:

- Pulsando el botón secundario sobre el programa a ejecutar en el explorador de paquetes y eligiendo **Ejecutar como** (*Run as*)**-Aplicación Java** (*Application Java*)
- Pulsando en el triangulo negro en el icono  de la barra de herramientas y eligiendo **Aplicación Java**. Si ya hemos ejecutado esa acción últimamente se puede directamente hacer clic sobre el icono del triángulo blanco.
- Pulsando las teclas **Alt+Mayús+X** y luego pulsando **J**.

SI la aplicación escribe algo por pantalla se podrá examinar en la vista **Consola** normalmente situada en la parte inferior de la pantalla de Eclipse.

(I.vi) ayudas al escribir código

(I.vi.i) esquema flotante (*quick outline*)

El esquema flotante es una ayuda visual que nos permite facilitar la navegación sobre el código que escribimos. Está disponible en el menú **Navegar** (*Navigate*)**-Esquema flotante** (*Quick outline*) o pulsando la combinación de teclas **Ctrl+o**.

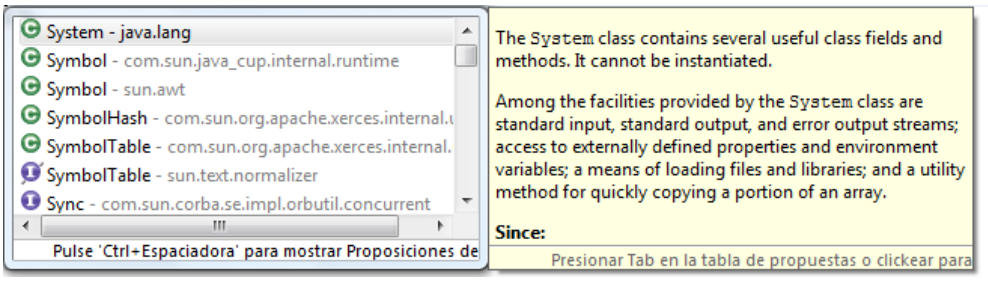
Es muy interesante cuando tenemos clases grandes con muchos elementos.

(I.vi.ii) asistente de contenido

Sin duda es **una de las mejores ayudas** al escribir código. Se hace pesado escribir el nombre de ciertos elementos del lenguaje y también recordar exactamente el nombre que le hemos puesto a variables o el nombre de métodos ya existentes en Java.

Para ello disponemos de una combinación *mágica* de teclas: **Ctrl+Barra espaciadora.**

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 flechas 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 ***0***

modificar las preferencias del editor.

(I.vi.iii) plantillas de código

Las plantillas de código (**code templates**) también aparecen cuando pulsamos **Ctrl+Barra espaciadora**, solo que las plantillas lo que hacen es sustituir un texto por otro que puede constar incluso por varias líneas.

La más conocida sin duda de Eclipse es la plantilla para utilizar fácil el método ***System.out.println***; para utilizarla se escribe la palabra ***sysout*** (en minúsculas) y después al pulsar la barra espaciadora se reemplazara por ***System.out.println***. Al igual que ***sysout*** podremos utilizar otras palabras clave almacenadas como plantillas, e incluso (como veremos más adelante (crear nuestras propias plantillas).

Para crear o modificar plantillas ver el apartado 0

modificar las preferencias del editor, subapartado plantillas.

(I.vi.iv) dar formato al código

Se trata de una opción que hace que el código se presente con los tabuladores y sangrados correcto. Por ejemplo si el código es:

```
public static void main(String[] args) {
            int x=3;
        System.out.print("Este código no está");
                System.out.print(" muy bien ");
  System.out.print("tabulado");
  x=7;
}
```

Podemos formatearlo correctamente realizando alguna de estas acciones:

- Pulsando **Ctrl+Mayús+F**
- Pulsando el botón secundario y eligiendo **Código Fuente** (***Source***)-**Formatear** (***Format***)

El resultado con el código anterior será:

```
public static void main(String[] args) {
        int x = 3;
        System.out.print("Este código no está");
        System.out.print(" muy bien ");
        System.out.print("tabulado");
        x = 7;
}
```

(I.vi.v) errores

Otra de las ventajas grandes de Eclipse es que los errores aparecen en el código a medida que escribimos al estilo de los procesadores de texto modernos como **Word** por ejemplo. Eso nos permite corregir errores de forma más eficiente.

Si un proyecto tiene algún archivo de clase con errores, el propio proyecto en el explorador de paquetes aparece marcado con una equis roja. De ese modo sabremos que algún error existe en él.

Si en lugar de error tenemos una advertencia (un ***warning***), entonces aparece un triángulo amarillo. Las líneas con advertencia aparecen subrayadas de amarillo en lugar de rojo.

(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

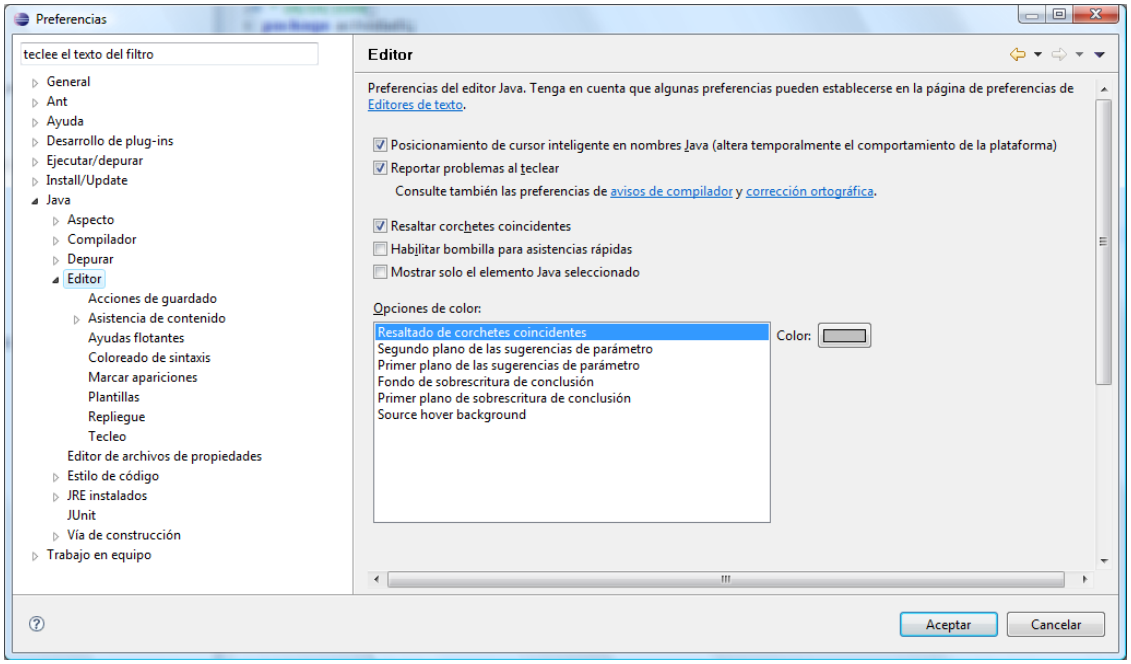


Ilustración 3-14, **Cuadro de preferencias de Eclipse mostrando el apartado referente al Editor de Java**

(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.

(I.vii.iii) asistencia de contenido

En este apartado se modifica el funcionamiento del asistente de contenido. Algunas posibilidades interesantes que se permiten en el cuadro:

- Modificar el tiempo de respuesta del asistente (normalmente está puesto a 200 milisegundos, se puede modificar incluso para que sea cero, es decir que actúe al instante).
- Ocultar de la ayuda del asistentes referencias prohibidas, en deshuso o limitadas.

(I.vii.iv) coloreado de la sintaxis

Gracias a este apartado podemos determinar cómo aparecerá el color y estilo de cada uno de los elementos del lenguaje. Es un apartado muy interesante ya que, bien utilizado, permite aumentar la legibilidad del código.

(I.vii.v) marcar apariciones

Se trata de una utilidad muy espectacular de Eclipse que hace que cuando el cursor se coloca en el código encima del nombre de una variable u otro elemento del lenguaje, se marquen de forma especial (normalmente con un sombreado gris claro) todas las apariciones de dicha variable o elemento, facilitando la detección y depurado de errores.

Sin duda en este apartado, salvo que trabajemos en una máquina poco potente, lo interesante es marcar todas las apariciones.

(I.vii.vi) apartado tecleo (*typing*)

Las más habituales ayudas al escribir código en Eclipse se encuentran en este apartado. Las opciones que vienen marcadas por defecto suelen ser las que mejor funciones. Entre las ayudas que activa o puede activar este apartado están:

- El cierre automático de llaves, comillas, corchetes, paréntesis, etc. Es muy interesante ya que así es mucho más difícil cometer el fallo más habitual al escribir código que es: no cerrar alguno de estos elementos
- Insertar automáticamente puntos y comas, y llaves. No es muy recomendable activar esta opción ya que Eclipse no la domina my bien (algo lógico porque la posición de estos elementos es muy variable).
- Usar la tecla tabulador para espaciar adecuadamente las líneas. Sin duda una de las opciones más útiles.
- Al pegar actualizar sangrados, permite que al copiar código desde otra parte, automáticamente se formatee de manera adecuada.

(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***).

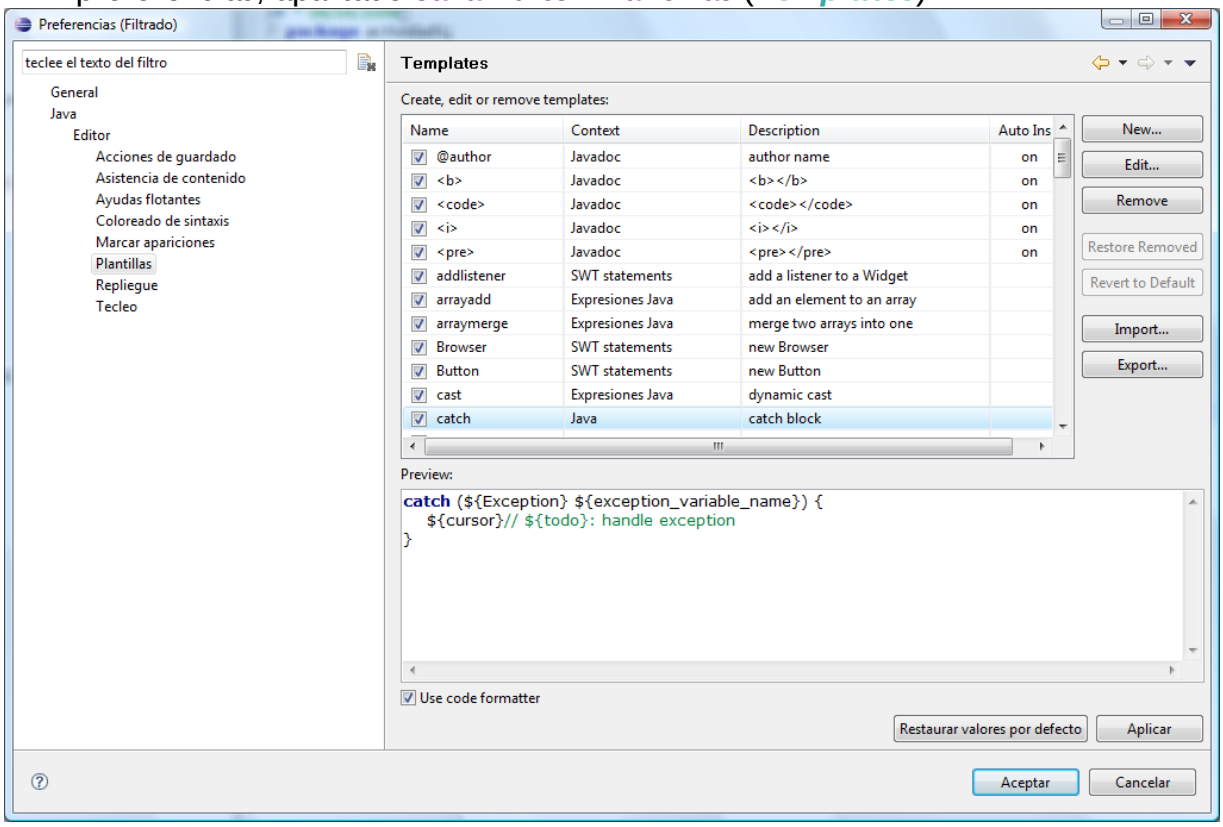


Ilustración 3-15, **Cuadro de preferencias en el apartado de plantillas**

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.

(4) Opcionalmente podemos escribir una descripción (es interesante hacerlo)

(5) En la parte relativa al patrón (***pattern***) escribimos el código de la plantilla. Este código puede ser tan largo como deseemos y puede incluir variables si pulsamos **Insertar variable**.

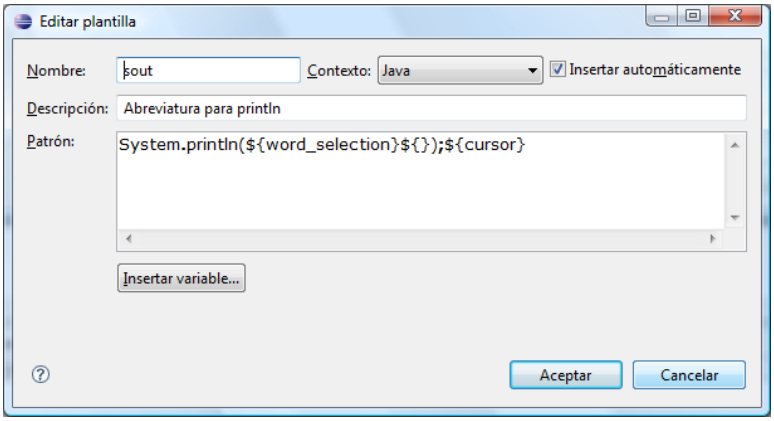
variables en las plantillas

Las variables de plantilla permiten colocar información variable según el contexto del código. Por ejemplo la variable **${author}** hace que cuando la plantilla se ejecute, en la posición en la que se colocó la variable se escriba el nombre del autor. Este nombre varía en cada instalación de Eclipse.

Las variables más interesantes son:

Variable	Significado
${}	Variable vacía, permite rellenar el espacio con lo que deseemos
${cursor}	Indica la posición en la que quedaría el cursor tras insertar el código de la plantilla
${date}	Coloca la fecha actual
${dollar}	Coloca el símbolo dólar **$**
${enclosing_method}	Nombre del método
${enclosing_method_arguments}	Argumentos del método
${enclosing_package}	Nombre del paquete contenedor
${enclosing_project}	Nombre del proyecto
${enclosing_type}	Nombre del tipo de datos
${file}	Nombre del archivo
${line_selection}	Contenido de las líneas seleccionadas
${primary_type_name}	Nombre primario del tipo de la compilación
${return_type}	Tipo que retorna el método
${time}	Hora actual
${user}	Nombre del usuario
${word_selection}	Contenido de la selección actual.
${year}	Año actual

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 esdta plantilla podremos utiliza 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***:

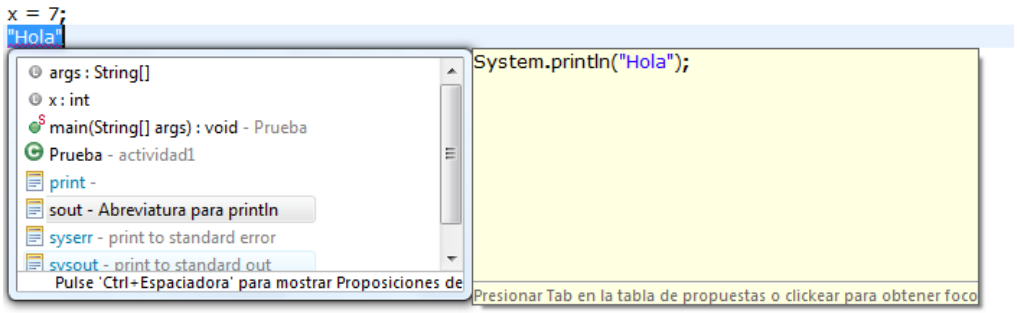


Ilustración 3-16, **Cuadro de plantillas en Eclipse según aparece tras escribir "Hola" e invocarle**

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.viii) estilo del código

Dentro del cuadro de preferencias (**Ventana-Preferencias**) también en el apartado Java disponemos de un apartado llamado **editor de estilo**. Este apartado es muy interesante.

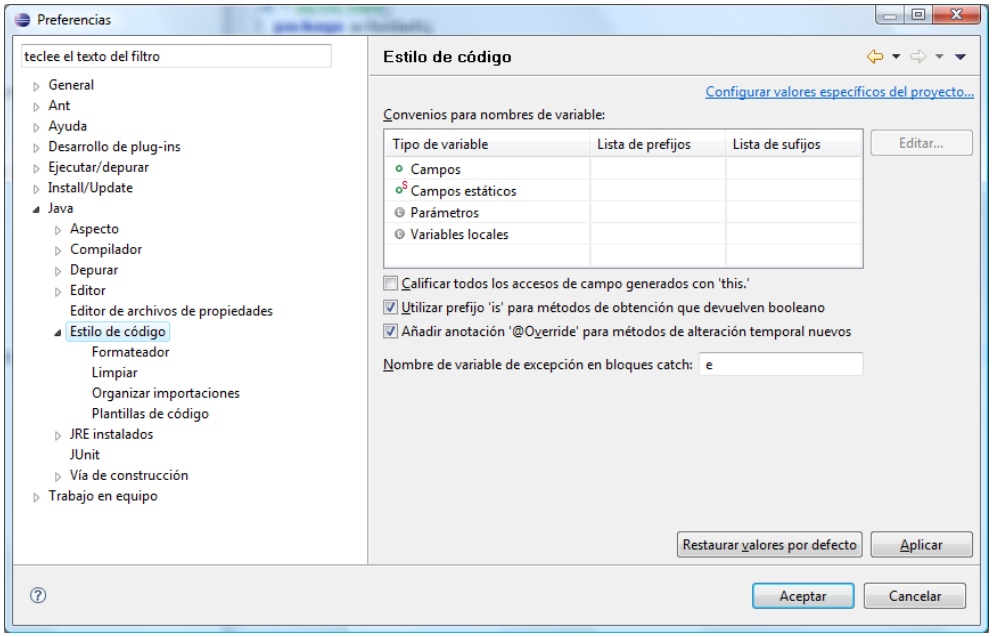


Ilustración 3-17, **El cuadro de preferencias en el apartado *Estilo de código***

Inicialmente en la raíz del apartado podemos elegir opciones que serán mejor entendidas cuando tengamos más conocimientos sobre Java (y que desde luego son muy interesantes).

formateador

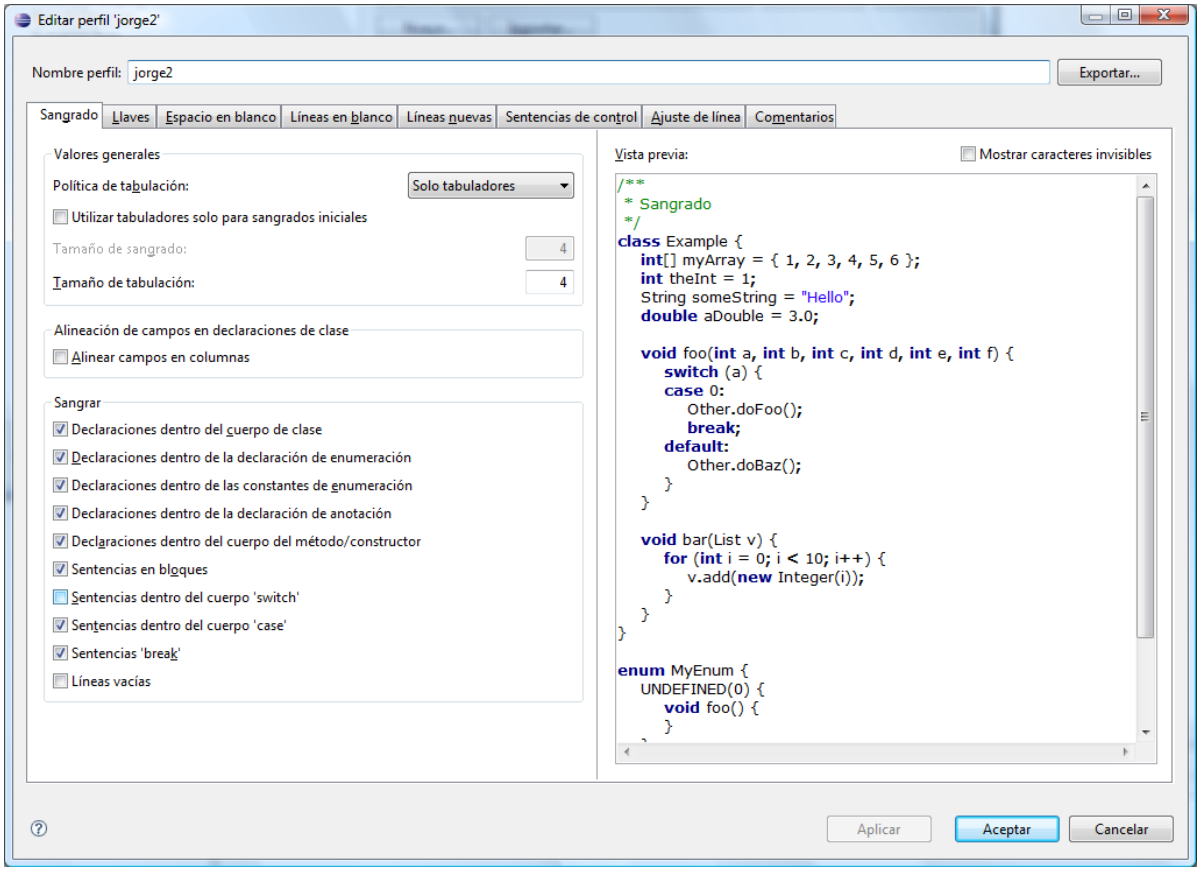


Ilustración 3-18, **Creación de un nuevo perfil de formato de código**

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

relacionados con Java. Pero si deseamos modificar esta forma por defecto, desde el apartado del formateador basta con:

(1) Pulsar el botón **Nuevo** si queremos crear un nuevo perfil o **Editar** si queremos modificar uno existente

(2) Lógicamente si hemos pulsado **Nuevo**, hay que dar nombre al perfil e indicar en quién se basará este perfil inicialmente.

(3) Modificar los parámetros según deseemos.

En Eclipse es espectacular la cantidad de parámetros que podemos configurar, con lo que nuestra forma particular de escribir código podrá ser perfectamente determinada en el cuadro.

Podemos también (como siempre) exportar e importar configuraciones, lo que facilita por ejemplo que una empresa que haya realizado un protocolo concreto de escritura de código pueda fácilmente colocar este protocolo en todas las máquinas de sus programadoras y programadores.

(I.vii.ix) limpiar

El apartado dedicado a limpiar código, permite que Eclipse arregle aspectos diversos en el código, por ejemplo etiquetas **Javadoc** que nos hayamos saltado. Nuevamente podemos editar o crear un perfil para determinar el funcionamiento de este apartado.

La limpieza de código se produce si pulsamos el botón secundario del ratón sobre el código y elegimos **Código fuente-Limpiar**.

organizar importaciones

Eclipse tiene la capacidad de colocar las instrucciones **import** necesarias para que nuestro código funcione correctamente. El orden de estos **import** y otros aspectos se configuran en este apartado. Por ejemplo en mi caso suelo elegir cinco como número máximo de importaciones para utilizar un asterisco en el **import**. Es decir que cuando haya colocado cinco clases procedentes por ejemplo del paquete **java.util**, Eclipse colocará un **import java.util.*** en lugar de los cinco **import** correspondientes.

plantillas de código

Permite configurar el código que Eclipse escribe por defecto cuando creamos una nueva clase. Es un apartado algo extenso, pero como ocurría con las plantillas del editor, muy interesante. Bien utilizado ahorra mucho tiempo de trabajo.

El cuadro se divide en dos apartados: comentarios (referido a los comentarios que escribe Eclipse automáticamente) y código (referido al código que Eclipse escribe automáticamente).

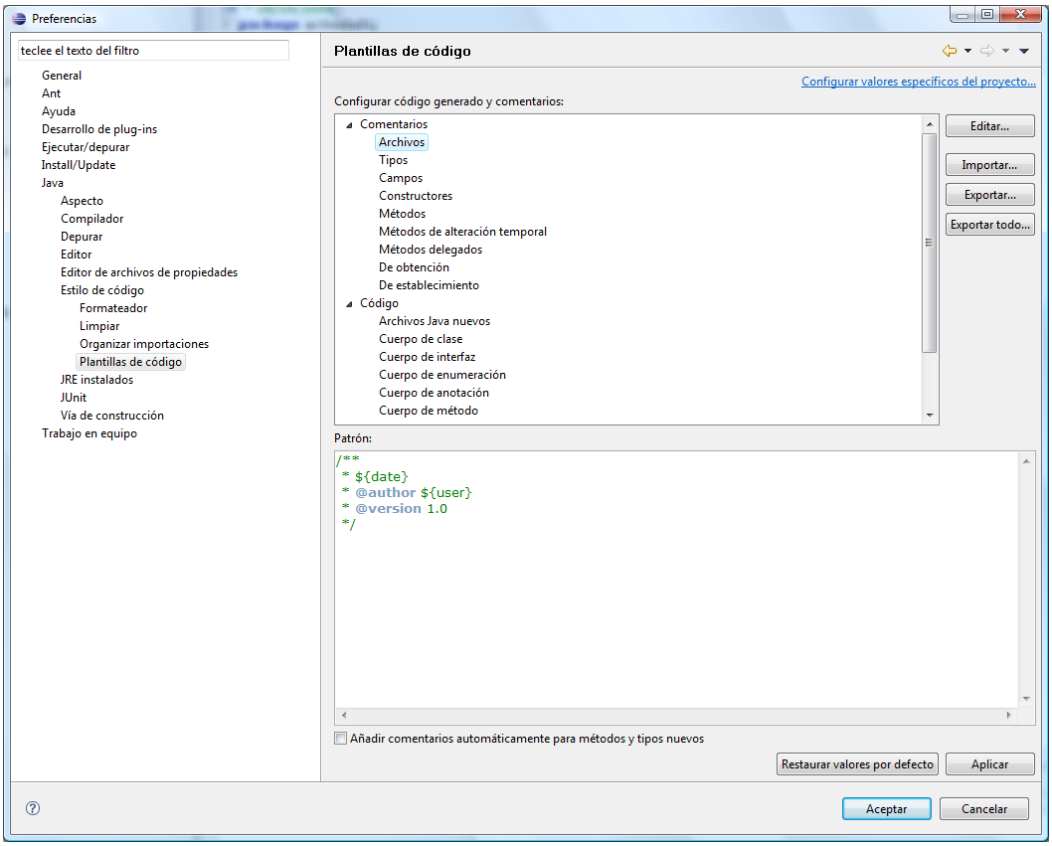


Ilustración 3-19, Cuadro de preferencias en el apartado Plantillas de código

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

código

Establece el código que Eclipse coloca por defecto en diversos apartados. El más interesante es el que se refiere a los archivos Java nuevos, que suele hacer referencia a otros apartados (especialmente a los de los comentarios indicados anteriormente).

La interesante casilla **añadir comentarios automáticamente** (muy recomendable) permite que Eclipse coloque los comentarios Javadoc pertinentes cada vez que se crea un nuevo método o campo; esto permite evitar despistes con la documentación del código.

(I.vii.x) exportar preferencias

Cuando se tienen múltiples instalaciones de Eclipse, es muy útil exportar las preferencias, para que sean cargadas desde otra instalación. Para hacer lo hay que ir a **Archivo-Exportar** y en el cuadro que aparece, elegir el grupo **General** y después **Preferencias**:

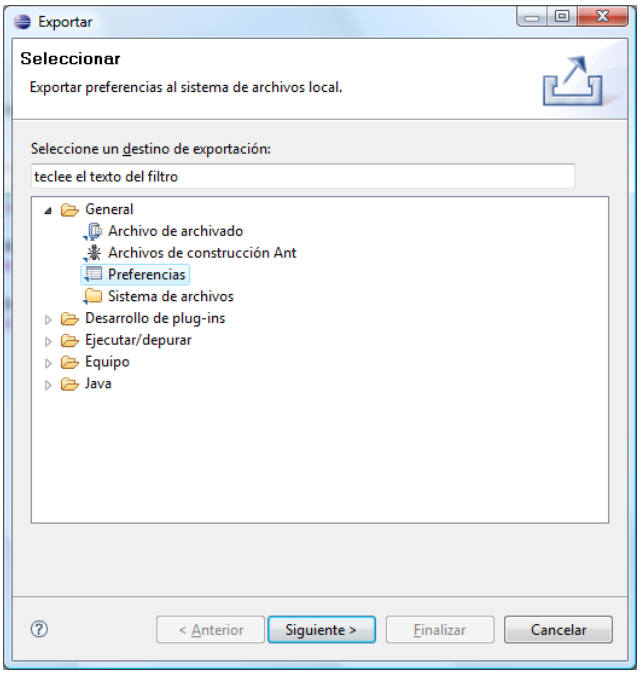


Ilustración 3-20, Cuadro de exportación de preferencias de Eclipse

En el cuadro siguiente se elige lo que se desea exportar y también el archivo destinatario. No obstante en muchas instalaciones de Eclipse, esta opción no es capaz de exportar las preferencias relativas al editor de código.

(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 rdesea realizar Javadoc y elegir **Proyecto-Generar javadoc**.

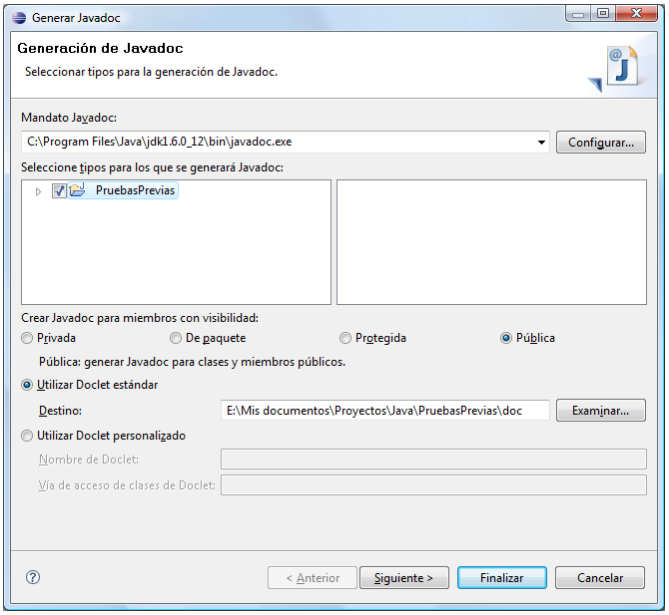
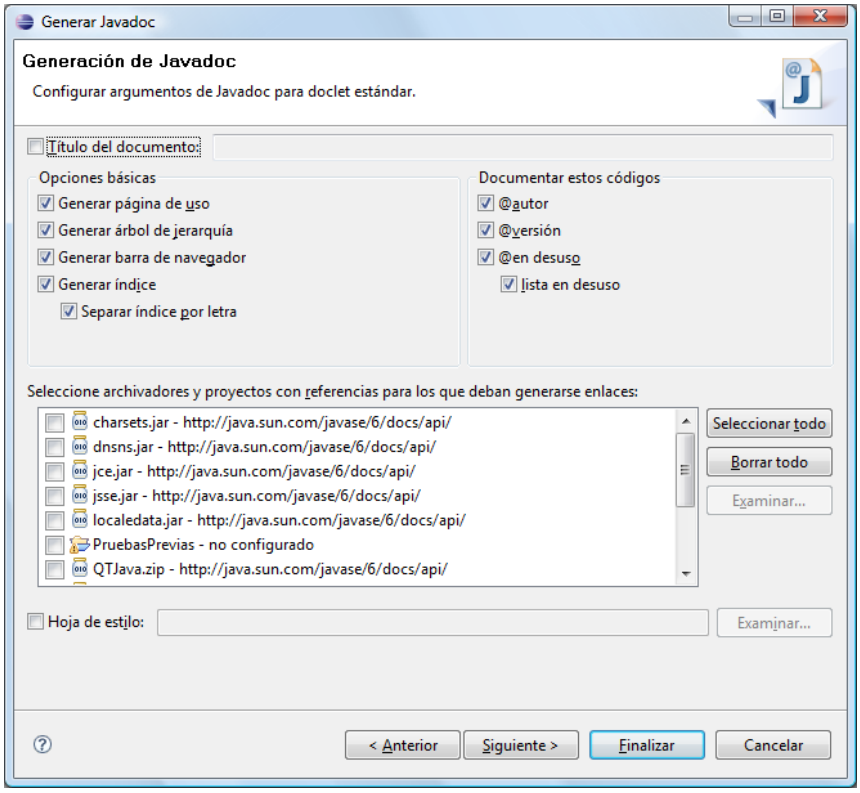


Ilustración 3-21, **cuadro de generación de Javadoc en Eclipse**

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).

Pulsando el botón siguiente disponemos de estas opciones:



Mediante las que se especifica aún más la documentación que realizaremos

Apéndice (II) Netbeans

(II.i) introducción

Inicialmente fue un proyecto realizado por estudiantes de la República Checa. Se convirtió en el primer IDE creado en lenguaje Java. Más tarde se formó una compañía que en el año 1999 fue comprada por **Sun Microsystems** (la propia creadora del lenguaje Java).

Ese mismo año **Sun** compró la empresa Forte que también creaba software IDE; por ello el producto se llamó **Forte for Java**. Sin embargo este nombre duró poco, Sun decidió que el producto sería libre y de código abierto y nació Netbeans como IDE de código abierto para crear aplicaciones Java.

Aunque Sun mantuvo otros IDE por pago; Netbeans alcanzó popularidad. Lleva ya varios pugnando con Eclipse por convertirse en la plataforma más importante para crear aplicaciones en Java.

Actualmente es un producto en el que participan decenas de empresas capitaneadas por Sun, sigue siendo software libre y abierto y es capaz de:

- Escribir código en C, C++, Ruby, Groovy, Javascript, CSS y PHP además de Java
- Permitir crear aplicaciones **J2EE** gracias a que incorpora servidores de aplicaciones Java (actualmente **Glashfish** y **Tomcat**)
- Crear aplicaciones **Swing** de forma sencilla, al estilo del Visual Studio de Microsoft.
- Crear aplicaciones **JME** para dispositivos móviles.

La última versión hasta ahora es la 6.7.

(II.ii) instalación

Primero tenemos que descargar la versión deseada de Netbeans de la página www.netbeans.org. La versión **total** contiene todas las posibilidades de Netbeans. Además disponemos de versiones para Windows, Linux y MacOS; también podemos elegir idioma español.

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

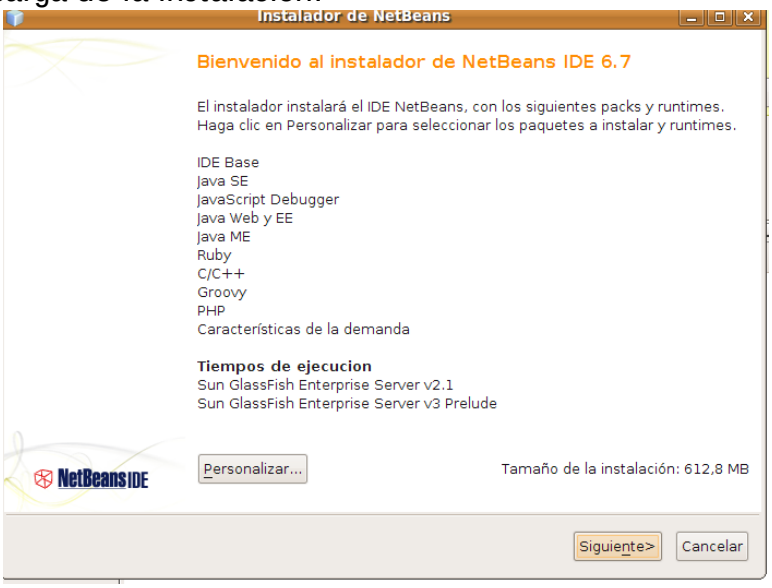


Ilustración 3-22, **Pantalla inicial de instalación de Netbeans**

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

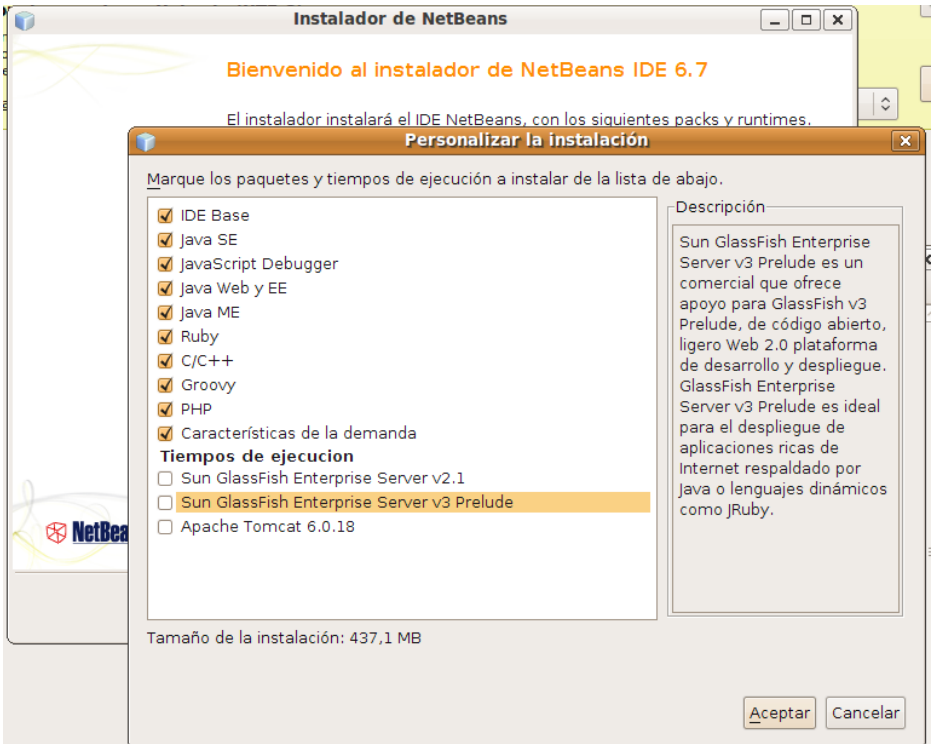


Ilustración 3-23, **Selección de los componentes Netbeans a instalar**

Finalmente seleccionamos el directorio en el que se instalará Netbeans y el directorio que contiene el SDK que se utilizará por defecto:

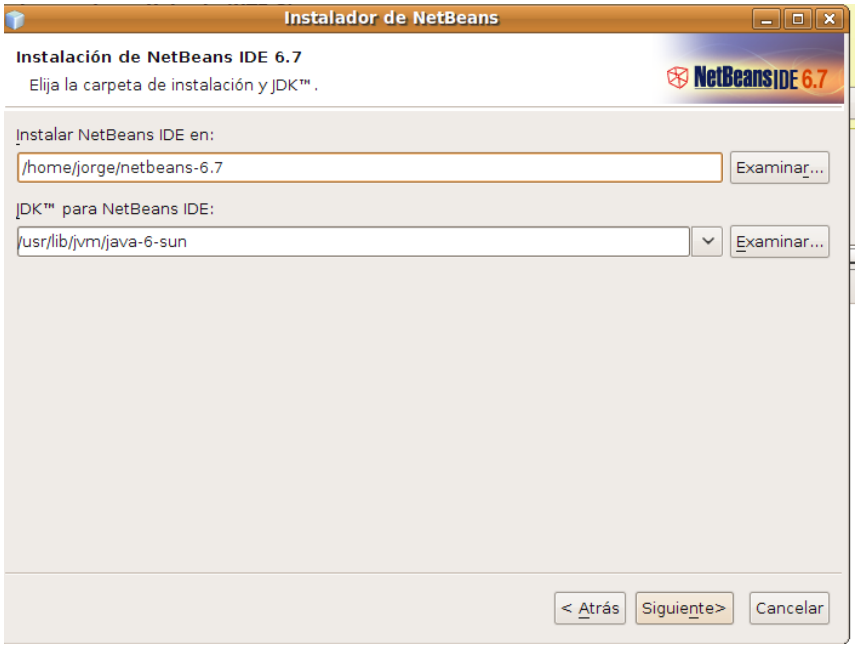


Ilustración 3-24, **Selección de directorios durante la instalación de Netbeans**

La instalación finaliza y disponemos de Netbeans. Una vez lanzado podremos iniciar un tutorial, obtener ayuda,…. o directamente empezar a trabajar.

Ilustración 3-25, **Pantalla inicial de Netbeans**

(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

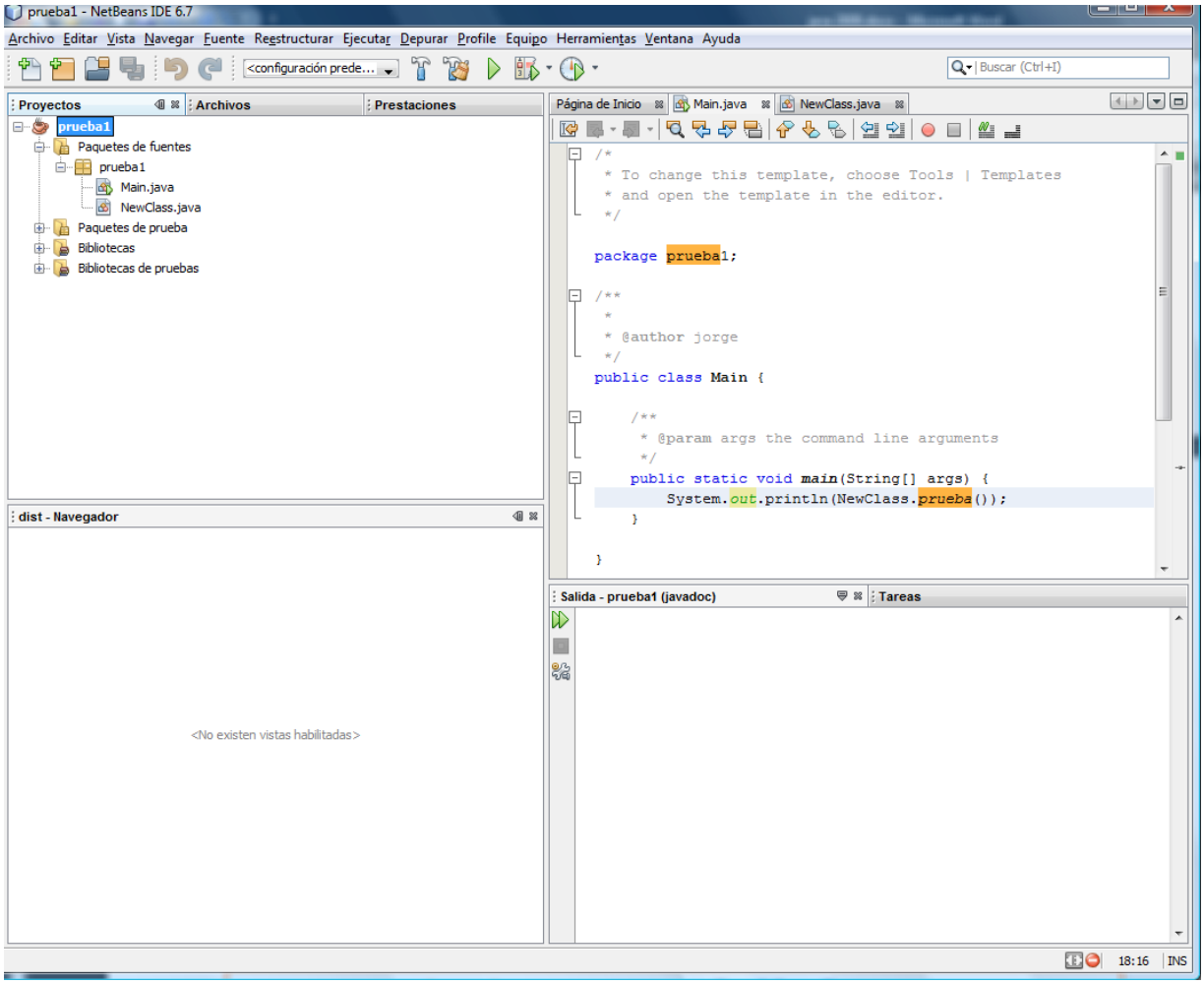


Ilustración 3-26, **Aspecto habitual de Netbeans**

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.

- **Paneles**. Divididos en cuatro zonas.
  - En el extremo superior izquierdo se encuentra la ventana de **Proyectos** (junto con el de **Archivos** y **Prestaciones**) que sirve para gestionar los elementos del proyecto en el que estamos
  - En la zona inferior izquierda está el **Navegador** con el que podemos recorrer y examinar los elementos del archivo (o clase) en el que estamos trabajado.
  - En la zona superior derecha, el panel más grande dedicado a mostrar y editar el código del archivo actual
  - En la parte inferior derecha el panel de salida que muestra los resultados de la compilación y ejecución del proyecto actual y otras informaciones interesantes.

(II.iii.i) cerrar y abrir paneles

Si deseamos quitar un determinado panel basta con pulsar en el botón con una X situado en la pestaña superior del panel. Para mostrar un panel que no está a la vista, basta con ir al menú **Ventana** y elegirle.

(II.iii.ii) iconizar un panel

Esta operación permite esconder un panel y dejarle en forma de icono. Para ello basta pulsar en el botón que tienen los paneles en forma de triángulo, al lado del botón de cerrar.

Para mostrar de nuevo el panel, bastará con hacer clic en su icono y si deseamos recuperar la forma de trabajar anterior (sin iconizar) habrá que pulsar el botón con un cuadrado (que habrá sustituido al triángulo).

(II.iii.iii) mover paneles

Simplemente arrastrándoles de la pestaña podemos cambiarles de sitio.

(II.iii.iv) mostrar y quitar barras de herramientas

Las barras de herramientas se pueden quitar y poner desde el menú Vista-**Barras de herramientas**.

Desde ese mismo menú, la opción personalizar permite incluso modificar los botones de las barras para que se muestren los botones que nos interesan más.

(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.

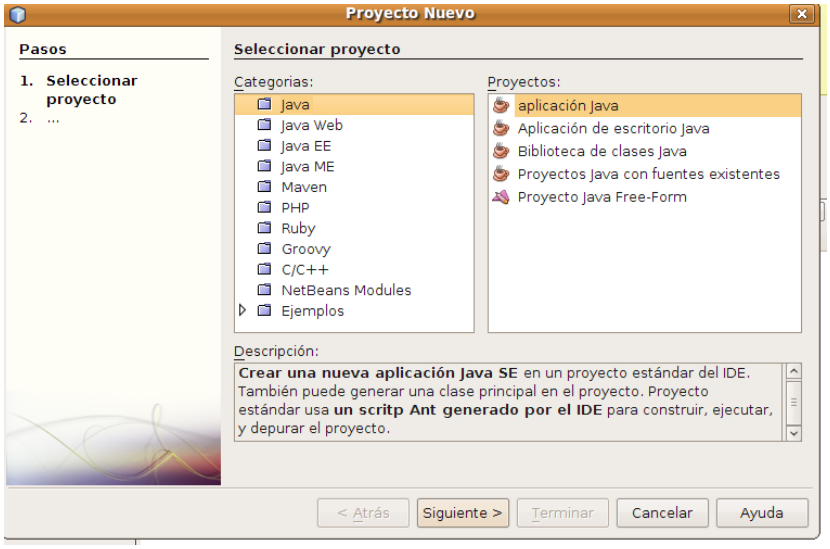


Ilustración 3-27, **Selección del tipo de proyecto en Netbeans**

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

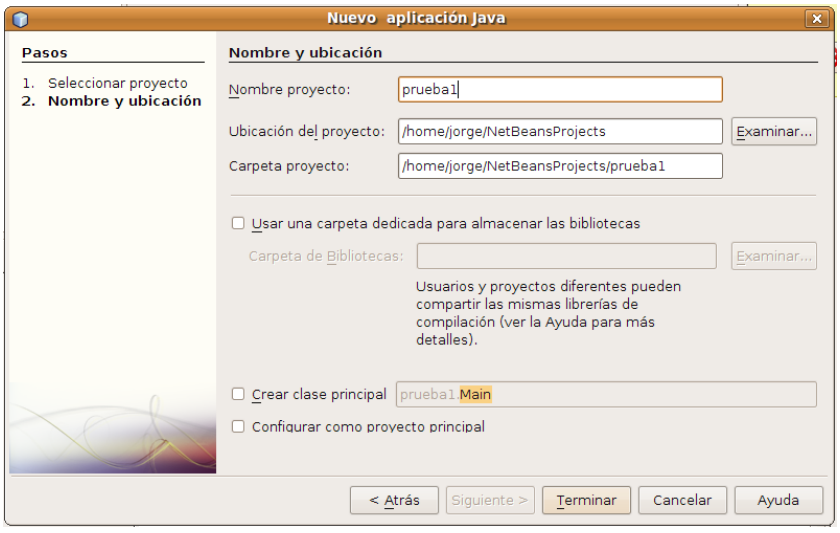


Ilustración 3-28, **Selección de las opciones del proyecto**

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

que se lanza cuando se pida compilar y ejecutar el proyecto. En un proyecto sólo debe haber una clase con el método main, si la elegimos al crear el proyecto, se crea directamente (es lo recomendable).

También por defecto, Netbeans crea un paquete con el mismo nombre que el del proyecto. Aunque no es mala política, podemos desear estructu

(II.iv.ii) la ventana del proyecto

El aspecto de Netbeans tras crear un proyecto es este:

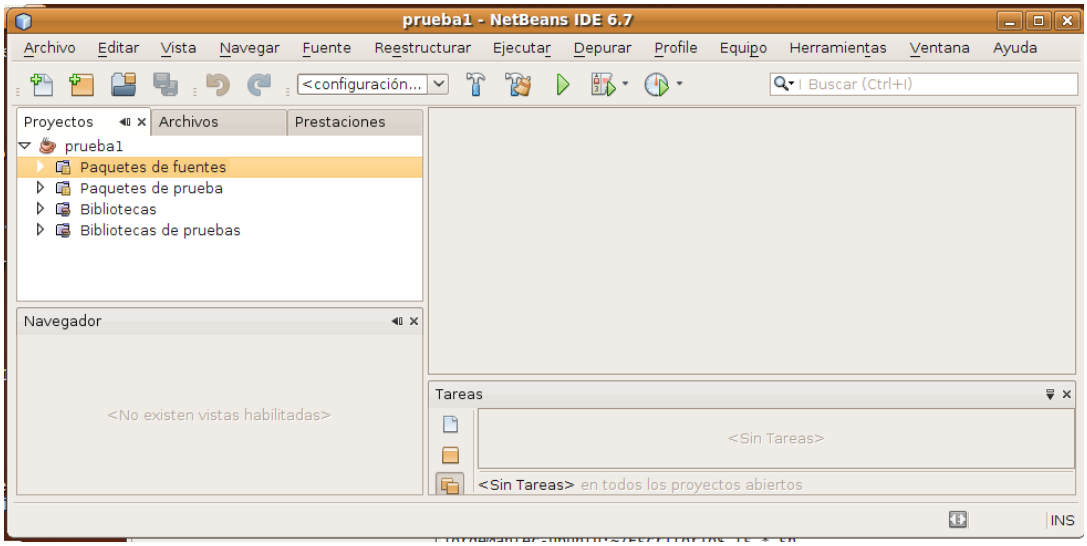


Ilustración 3-29, **Aspecto de Netbeans con un proyecto vacío**

En el lado izquierdo se seleccionan las carpetas fundamentales de los proyectos de Netbeans. Los **paquetes de fuentes** contienen el código de las clases del proyecto. Ahí se crea el código del proyecto, disponemos de **paquetes de pruebas** para nuestras pruebas y podemos instalar también **bibliotecas de pruebas**. Las bibliotecas son clases directamente invocables desde nuestro proyecto.

(II.iv.iii) crear paquetes

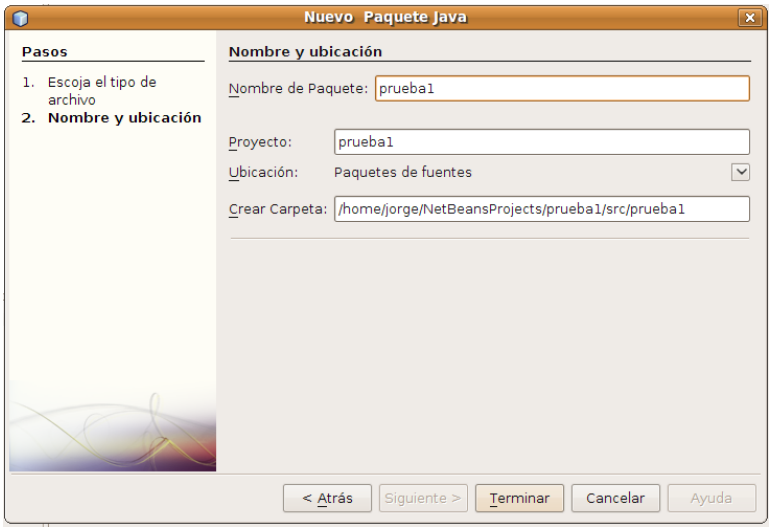


Ilustración 3-30, **Cuadro de creación de nuevos paquetes**

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 rellenar 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.

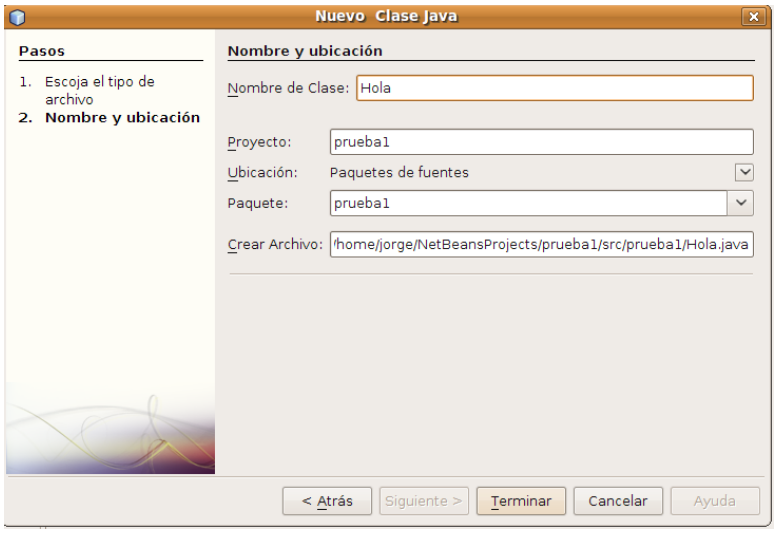


Ilustración 3-31, Cuadro de creación de nuevas clases

(II.iv.v) vistas del proyecto

En Netbeans sobre todo hay dos formas de examinar el proyecto. La primera es a través de la pestaña **Proyectos**. En ella aparece la organización lógica del proyecto.

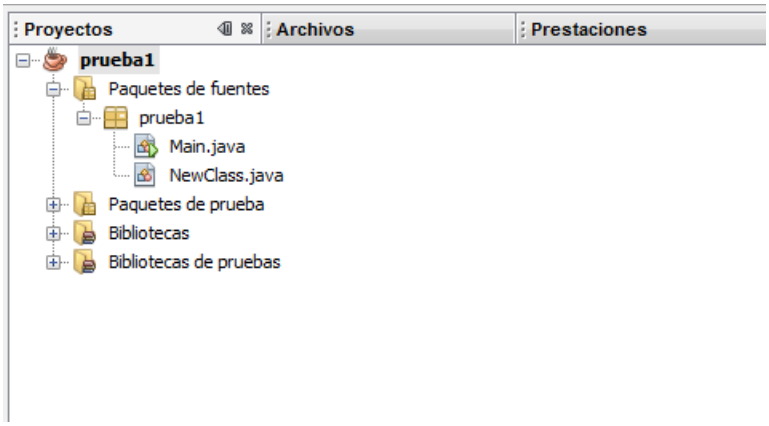


Ilustración 3-32, Ventana de proyectos

Sin embargo la vista **Archivos** nos enseña la realidad de los archivos del proyecto. Nos enseña la organización de los archivos del proyecto desde el punto de vista del sistema:

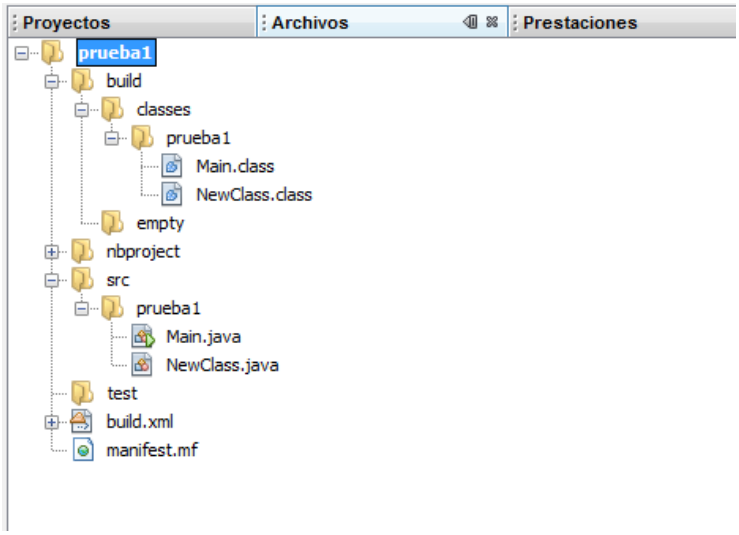


***Ilustración 3-33*, Ventana de archivos del proyecto**

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

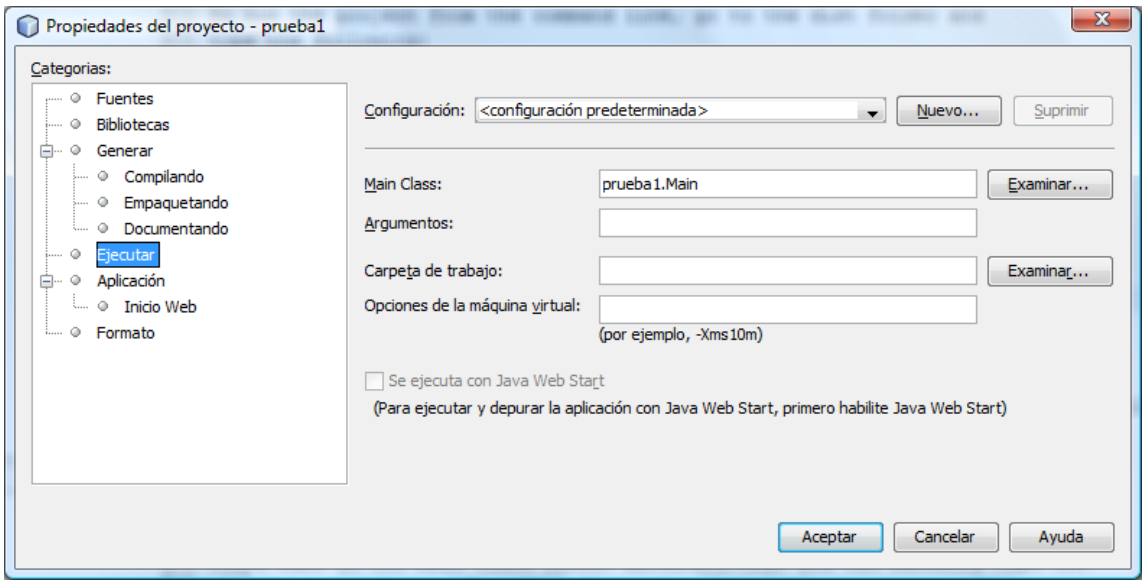


***Ilustración 3-34*, Cuadro de propiedades abierto por las opciones de ejecución**

(II.iv.vii) borrar un proyecto

Si deseamos quitar un proyecto podemos: pulsar el botón secundario del ratón sobre el icono de proyectos (en la ventana de **Proyectos**) y elegir **Suprimir**; o podemos seleccionar el proyecto y pulsar la tecla **Suprimir**.

En cualquier caso Netbeans nos pregunta si deseamos realmente borrar el proyecto. La casilla **Eliminar también fuentes bajo…** en el caso de marcarse, eliminaría no solo el proyecto de Netbeans, sino también todos los archivos del proyecto quedaría eliminados del disco.

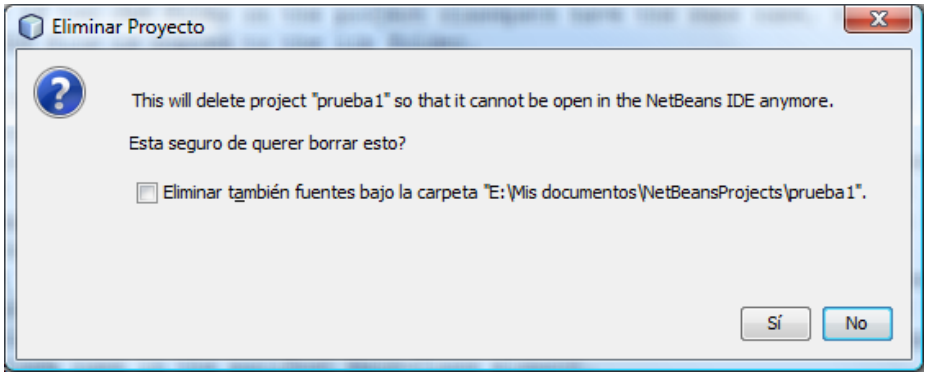


Ilustración 3-35, cuadro de confirmación de borrado de un proyecto

(II.iv.viii) importar un proyecto de Eclipse

Para poder ver y trabajar un proyecto de Eclipse en Netbeans basta con:

(2) Elegir **Archivo-Importar Proyecto-Proyecto Eclipse**

(3) Seleccionar un directorio del que cuelguen proyectos de Eclipse que nos interese utilizar desde Netbeans

(4) En el paso siguiente hay que elegir los proyectos concretos que vamos a importar.

Si modificamos proyectos desde Netbeans y luego queremos utilizarlos en Eclipse, podríamos tener problemas. Por ello es mejor hacer copia primero de los archivo que utilizaremos.

Si no disponemos de la opción de importar proyectos de Eclipse habrá que instalar el módulo que hace esto posible (normalmente está instalado); para ello bastará con ir a **Herramientas-Complementos** y en la pestaña **Complementos disponibles**, marcar ***Eclipse Project Importer***.

(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 ejecutarle. 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** ***nombraArchivoJar***)

(II.vi) javadoc

Netbeans tiene también capacidad para generar toda la documentación **javadoc** de un proyecto.

(II.vi.i) añadir la documentación Java a Netbeans

Para que Netbeans pueda mostrarnos ayuda de tipo javadoc en cada clase estándar que utilicemos en nuestros proyectos, debemos añadir la documentación del SDK con el que estemos compilando nuestros proyectos.

Para hacerlo:

(1) Descargamos la documentación de Java desde la página http://java.sun.com/javase/downloads/index.jsp

(2) Ir al menú **Herramientas-Plataformas Java**

(3) Ir a la pestaña **Javadoc** y pulsar en añadir **archivo ZIP/carpeta**

(4) Elegir el archivo zip en el que se encuentra la documentación

(5) La documentación estará instalada en cuanto aceptemos el cuadro

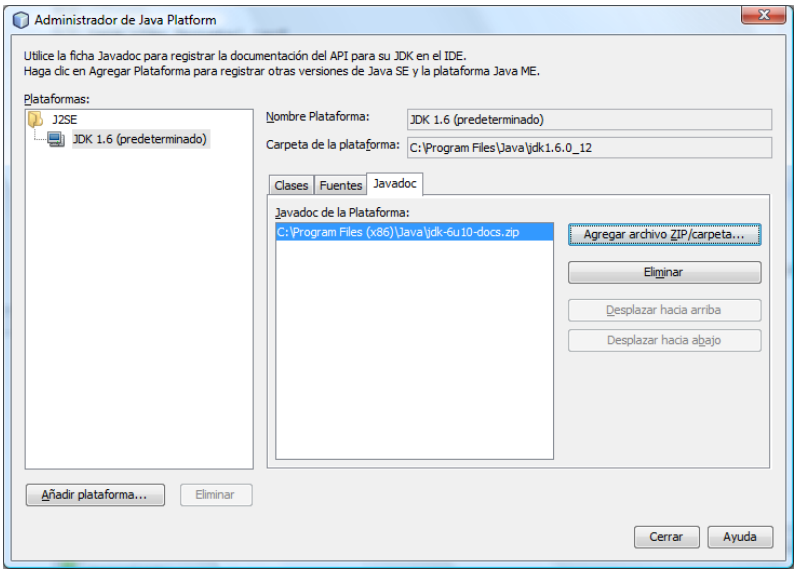


Ilustración 3-36, **Seleccionando la documentación de Java**

(II.vi.ii) generar la documentación Javadoc del proyecto

Para que Netbeans cree la documentación Javadoc de nuestro proyecto basta con:

(6) Seleccionar el proyecto

(7) Pulsar el botón secundario sobre el proyecto y elegir **Generar javadoc** o elegir la misma opción en el menú **Ejecutar**.

(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.

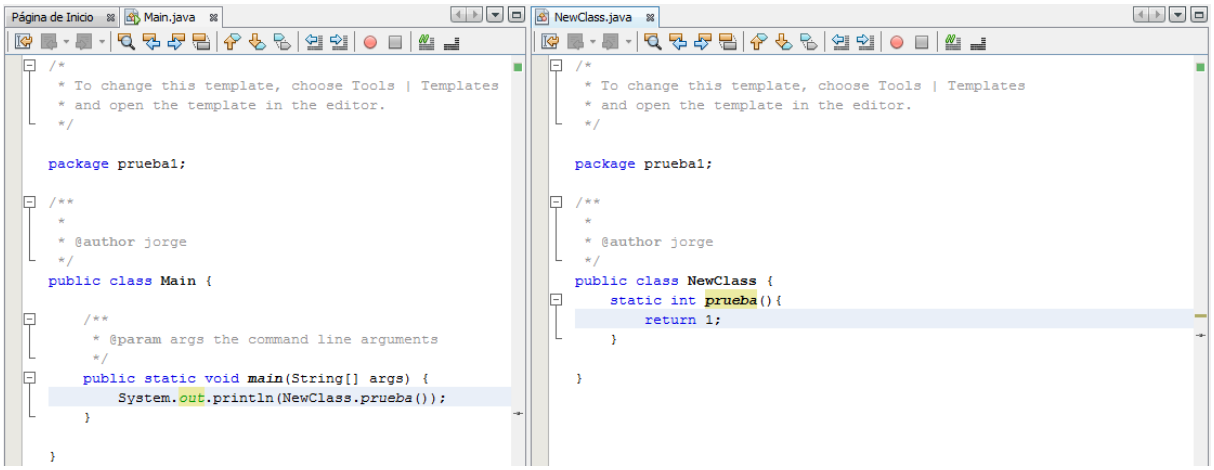


Ilustración 3-37, La ventana de código mostrando dos archivos a la vez

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.vii.ii) ayudas al escribir código

Como la mayoría de IDEs actuales, Netbeans posee numerosas mejoras para escribiré el código. Entre ellas:

- **Autocompletado del código**. Funciona igual que en Eclipse, pulsando **Ctrl+Barra espaciadora**, se nos mostrará ayuda en un cuadro para ayudarnos a completar la expresión Java que estamos escribiendo. Es la opción de ayuda de código más interesante.
- **Plantillas de código**. Abreviaturas que nos permiten escribir de golpe código habitual, ganando mucha rapidez. Para ello se escribe la abreviatura y después se pulsa el **tabulador**. Es posible añadir y modificar las plantillas desde **Herramientas-Opciones-Editor-Plantillas de código**.
- **Marcado de errores y autocorrección**. Se subraya de color rojo el código con errores en cuanto escribimos e incluso se corrigen automáticamente algunos errores. La tecla **Alt+Intro** permite examinar las reparaciones efectuadas.
- **Inserción de código**. Pulsando el botón secundario sobre el código actual y eligiendo **Inserción de código**, se nos facilitara la construcción de algunos elementos del código como constructores, métodos ***get*** y ***set***,... Cuanto más conozcamos de Java, más interesante será esta opción.
- **Diseñador GUI**. Sin duda una de las opciones donde Netbeans aventaja claramente a Eclipse, se trata de un complemento que permite diseñar programas Java que utilicen elementos gráficos, de forma muy sencilla.
- **Refactorizar**. Permite dar de nuevo formato al código para que tenga la apariencia más legible.
- **Accesos rápidos por teclado**.
- **Cerrado automático de llaves, paréntesis, comillas,**...
- **Macros**
- **Administración de cláusulas import**. Mediante las teclas **Ctrl+Mayus+I** Netbeans genera el código de tipo **import** necesario para que no haya errores en nuestro programa.

(II.vii.iii) búsqueda y reemplazo de texto en el código

buscar texto

Se realiza mediante la tecla **Ctrl+F** o bien desde **Edición-Buscar**. En ese momento en la parte inferior de la ventana de código disponemos de un cuadro gris en el que podemos escribir el texto a buscar y también seleccionar las opciones que estimemos convenientes para buscar.

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:

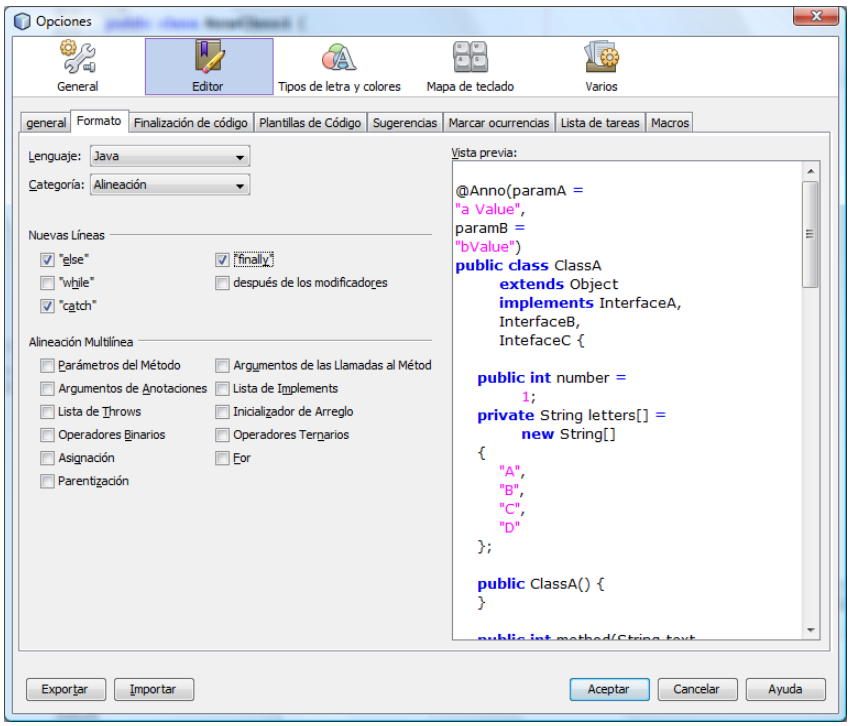


***Ilustración 3-38*, El cuadro de opciones del editor en el apartado de formato del código**

- En el apartado **Formato** se especifica el formato del código. Las opciones generales permiten modificar el tamaño del tabulador, así como la línea del margen derecho (que sirve como referencia para indicar el tamaño máximo recomendado en el que debe finalizar cada línea del código) en el apartado.

  Si elegimos **Java** en el apartado lenguaje podremos seleccionarla forma en la que queremos que se dé formato al código Java. En este apartado hay muchas posibilidades: donde queremos que se cierren las llaves, qué instrucciones tienen que ir al principio de la línea, dónde se dejan líneas y espacios en blanco, etc.

  Todas estas opciones se ejecutan tanto cuando Netbeans autocompleta el código, como cuando elegimos Fuente-Formato para reformatear nuestro código. Por ello es muy importante ajustarlas correctamente, en este sentido el código ejemplo que coloca el programa sirve muy bien de guía.

- En el apartado **Finalizador de código** se eligen las ayudas de completado de texto que deseamos nos realice Netbeans. En general la configuración inicial es la deseable.

- El apartado **Plantillas de código** nos permite crear y modificar las abreviaturas de plantillas de código. Se puede incluso modificar la tecla que produce el reemplazo de la abreviatura por el contenido de la plantilla. Si escribimos la palabra ***fori*** y después pulsamos el tabulador, aparecerán varias líneas correspondientes al bucle **for** con un contador. De esa misma manera podremos crear más abreviaturas e incluso hacer que no sea el tabulador la tecla que desencadena el reemplazado.

Los colores y tipos de letra con los que se mostrará el código están en el apartado **Fuentes y colores** del mismo cuadro de **Opciones**.

Cabe señalar que en este sentido Eclipse posee más opciones de ayuda y formato de código. Pero normalmente tendremos más opciones que las necesitadas con Netbeans.

(II.vii.vi) reestructurar

Son opciones que permiten una modificación del código de modo que sea más legible, que sean más improbables las apariciones de errores y que se realiza de una forma más automática tareas que resultan pesadas. Las posibilidades son:

- **Cambiar de nombre**. Cuando queremos cambiar de nombre a una clase, variable, método,… tenemos que encontrar en el código todas las referencias al nombre antiguo. Para evitar eso, podemos:

  a> Seleccionar el elemento al que queremos cambiar de nombre
  b> Pulsar el botón secundario en la selección y elegir **Reestructurar-cambiar nombre**

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

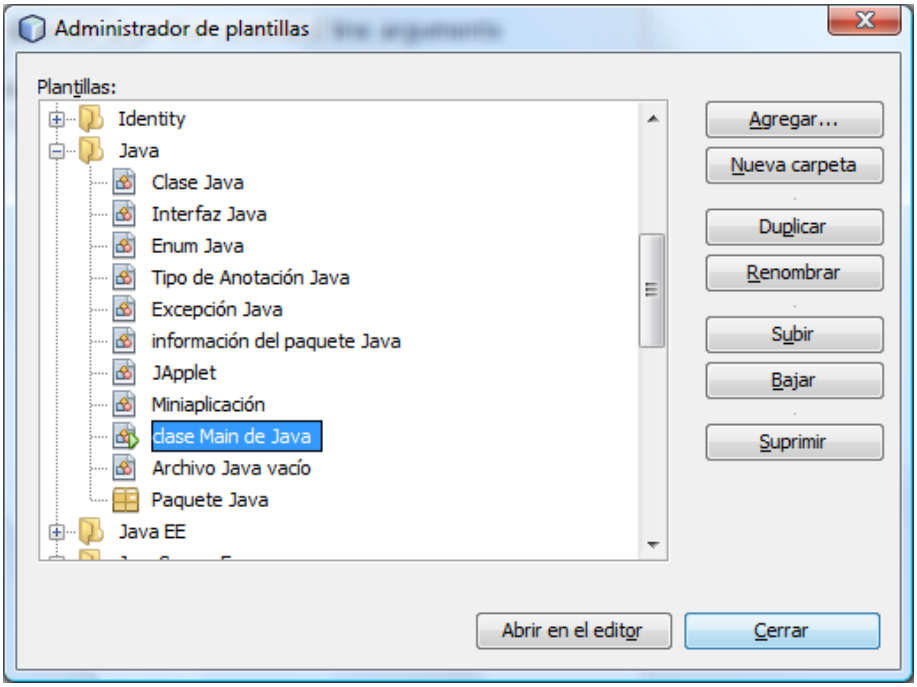


Ilustración 3-39, El cuadro de administración de plantillas

Cuando se crea una nueva clase, aparece código ya creado: código Javadoc, instrucción **package**,… Es posible modificar ese código haciendo lo siguiente:

(2) Yendo a **Herramientas-Plantillas**

(3) Abriendo la carpeta **Java**

(4) Eligiendo la plantilla a modificar y pulsando **Abrir en el editor**

(5) Después basta con realizar los cambios deseados y guardar el documento

Los cambios realizados se aplicarán a todos los documentos basados en la plantilla modificada. Por ejemplo si modificamos la plantilla ***clase Main de Java***, cualquier clase de Java con método ***main*** que hagamos a partir de ahora contendrá el código modificado.

Las plantillas contienen código especiales (comienzan con el símbolo ***$***). Estos códigos facilitan el funcionamiento de la plantilla. Por ejemplo el código ***${name}*** coloca el nombre de la clase en esa posición.

(II.vii.viii) comparar archivos

En el menú **Herramientas** eligiendo la opción **Diff**, podemos comparar el archivo actual con otro. Simplemente se nos pide elegir el segundo archivo y la herramienta marcará las diferencias.

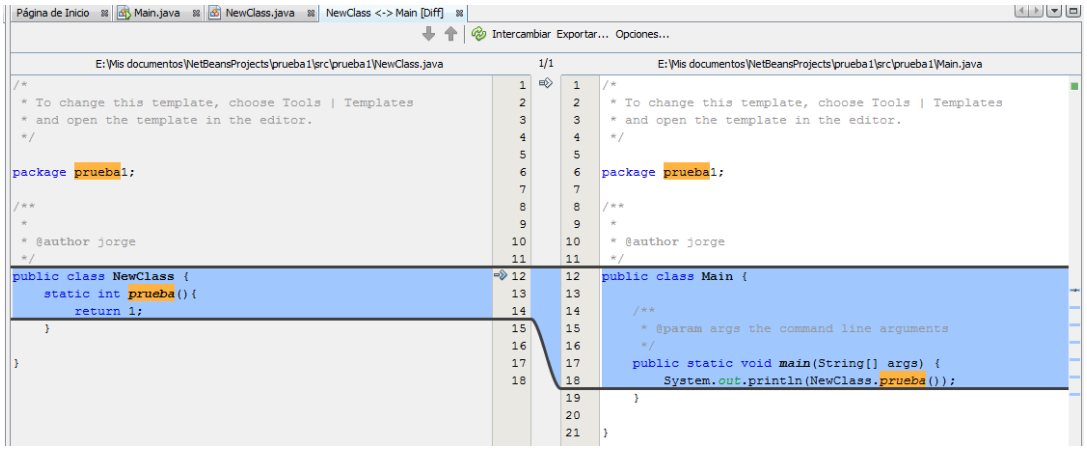


Ilustración 3-40, **La herramienta diff comparando archivos**

(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

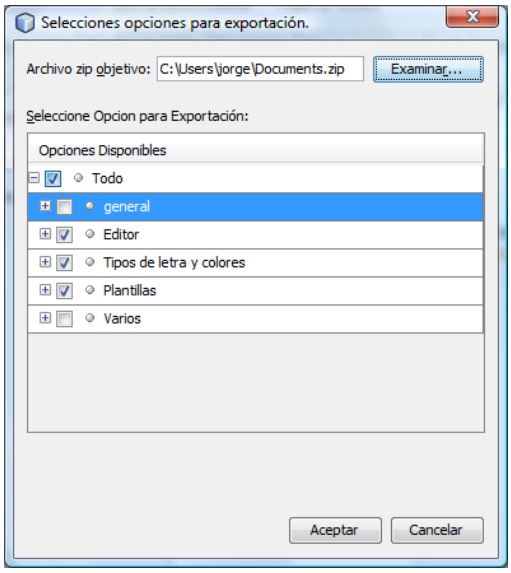


Ilustración 3-41, **Cuadro de exportación de opciones de Netbeans**

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.

índice de ilustraciones