

- ◆ 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.

Este 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.

(3.6) 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 está obsoleta u obsoleto.
- ◆ **@throws**. Indica las excepciones que pueden lanzarse en ese código.
- ◆ **@param**. Palabra a la que le sigue texto que 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`

All Classes [prueba1](#)

Package [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS
SUMMARY: NESTED | FIELD | [CONSTR](#) | [METHOD](#)

FRAMES NO FRAMES
DETAIL: FIELD | [CONSTR](#) | [METHOD](#)

Class prueba1

java.lang.Object
└ prueba1

```
public class prueba1
extends java.lang.Object
```

Esto es un comentario para probar el javadoc este texto aparecerá en el archivo HTML generado. **Realizado en agosto 2003**

Constructor Summary

[prueba1\(\)](#)

Method Summary

static void [main](#)(java.lang.String[] args)
Este método contiene el código ejecutable de la clase

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

prueba1

public [prueba1\(\)](#)

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,

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 sepamos 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 la 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;
```

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 **1,345·10³** 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

(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

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 —*¡¡ No !!*

(3.10.3) operadores condicionales

Sirven para comparar valores. Siempre devuelven valores booleanos. Son:
Ta bla 1.4

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
```

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	>	>=	<	<=	

(3.12) lectura y escritura por teclado

(3.12.1) escritura

java.lang

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;
```

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

java.lang.String.charAt(indice desde 0) → ej: texto.charAt(0)
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 π (3,14159265...)

Por otro lado posee numerosos métodos que son:

operador	significado
double ceil(double x)	Redondea x al entero mayor siguiente: <ul style="list-style-type: none">◆ 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: <ul style="list-style-type: none">◆ Math.floor(2.8) vale 2◆ Math.floor(2.4) vale 2◆ Math.floor(-2.8) vale -3
long round(double x) int round(float x)	Redondea x de forma clásica: <ul style="list-style-type: none">◆ Math.round(2.8) vale 3◆ Math.round(2.4) vale 2◆ Math.round(-2.8) vale -3

(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()*20+10)

Ejercicio 4

Programa que genera aleatoriamente una letra minúscula y determine si es vocal o no lo es mediante la siguiente salida:

La letra letra es vocal: true/false.

Ejercicio 3

Programa que dado un importe en euros, nos indique el menor número de billetes y la cantidad sobrante.

Ej: 232 euros

0 billetes de 500

1 billete de 200

0 billetes de 100

0 billetes de 50

1 billete de 20

1 billete de 10

0 billetes de 5

0 billetes de 2 euros.

(45)

(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).

The screenshot shows the Eclipse Downloads page. At the top, there are links for Home, Users, Members, Committers, Downloads, Resources, Projects, and About Us. A search bar is also present. The main section is titled "Eclipse Downloads" and features a navigation bar with tabs for "Eclipse Packages" (which is selected), "Member Distros", and "Projects". Below this, there's a section for "Ganymede Packages (based on Eclipse 3.4.2) - Compare Packages". It lists several software packages with their names, sizes, descriptions, download counts, and supported operating systems (Windows, Mac OS X, Linux 32bit, Linux 64bit). To the right of the package list, there are banners for "eclipseCON 2009" (with a "3 Days Left for Advance Registration Prices" message) and "D EVELOP Oracle Develop Beijing 10-11 December Moscow 4-5 February Prague 10-11 February Register Now ORACLE". At the bottom right, there's a list titled "Popular projects (Mar 2/09)" which includes PDT, EMFT, Web Tools, MDT, C/C++ Development, EMF, Business Intelligence and Reporting, VE, Mylyn, and Subversive.

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 primera 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.

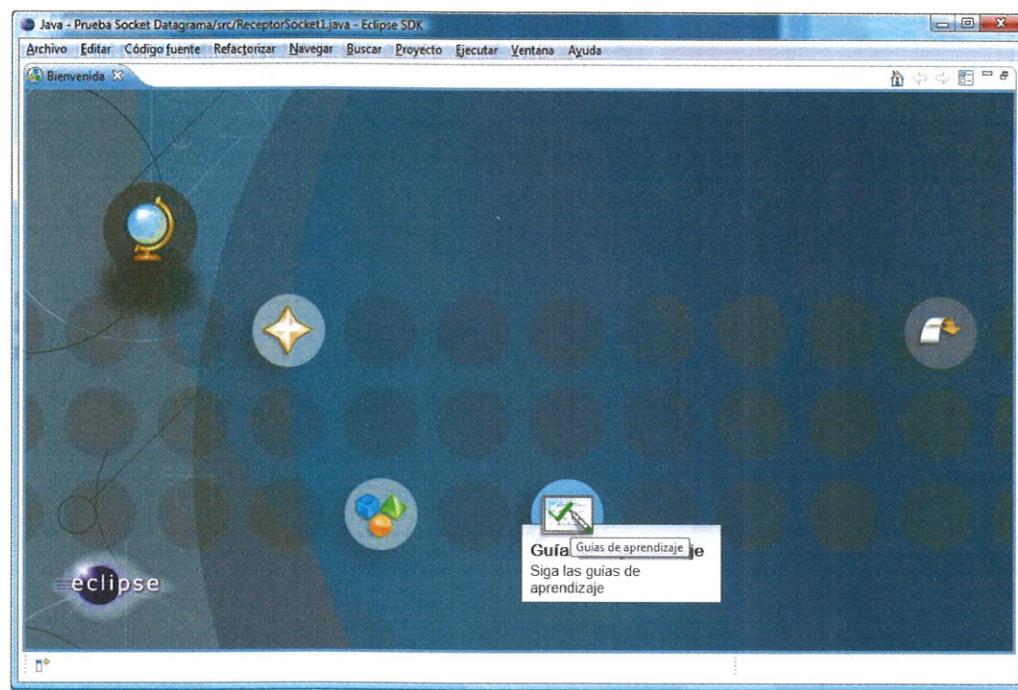


Ilustración 3-10, Aspecto de la bienvenida de Eclipse una vez descomprimido el paquete de Babel para lenguaje español en la carpeta de Eclipse

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.

Las vistas se puede minimizar haciendo clic en el botón . En ese caso aparecerá una miniatura en forma de ícono 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 aglutan 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 ().

En el caso de jugar mucho con los paneles y ventanas de Eclipse, podríamos desear recuperar la perspectiva típica para trabajar 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)-**

(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 

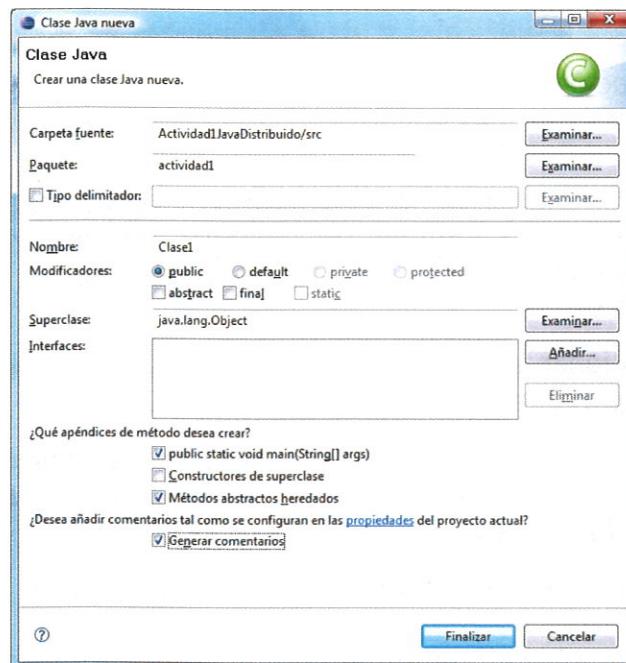


Ilustración 3-13, Cuadro de nueva clase Java en Eclipse

- (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

(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 ícono  de la barra de herramientas y eligiendo **Aplicación Java**. Si ya hemos ejecutado esa acción últimamente se puede directamente hacer clic sobre el ícono 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**.

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

(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.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 desuso 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 teclado (*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 muy 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.

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

(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.

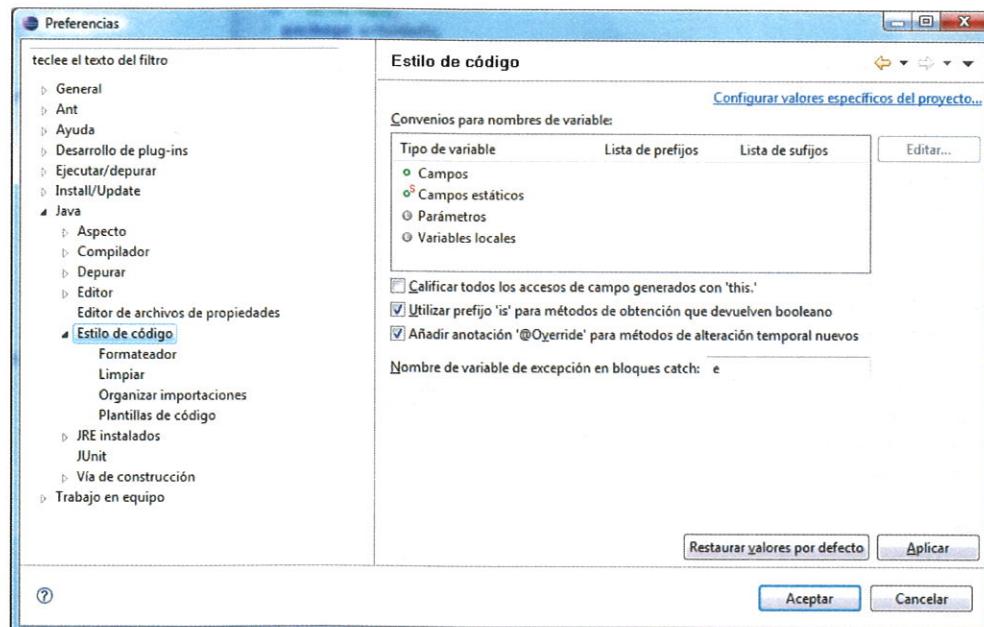


Ilustración 3-17, El cuadro de preferencias en el apartado Estilo de código

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).

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).

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.

(Avanzado) exportar preferencias

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

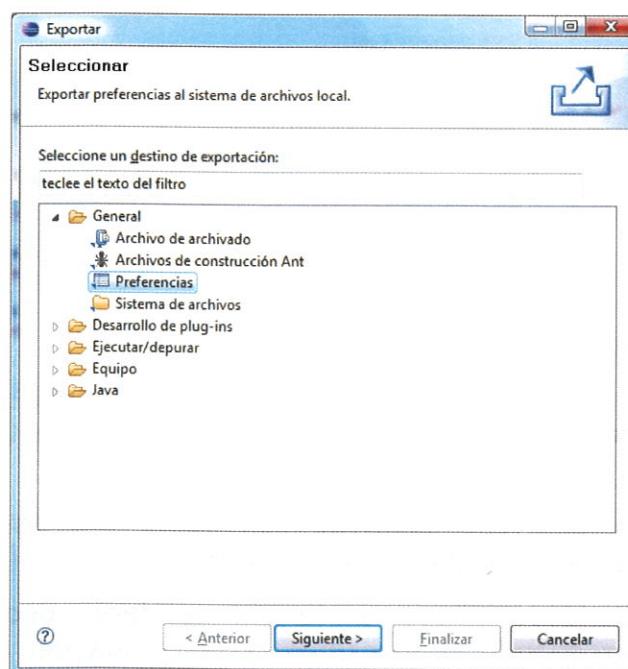
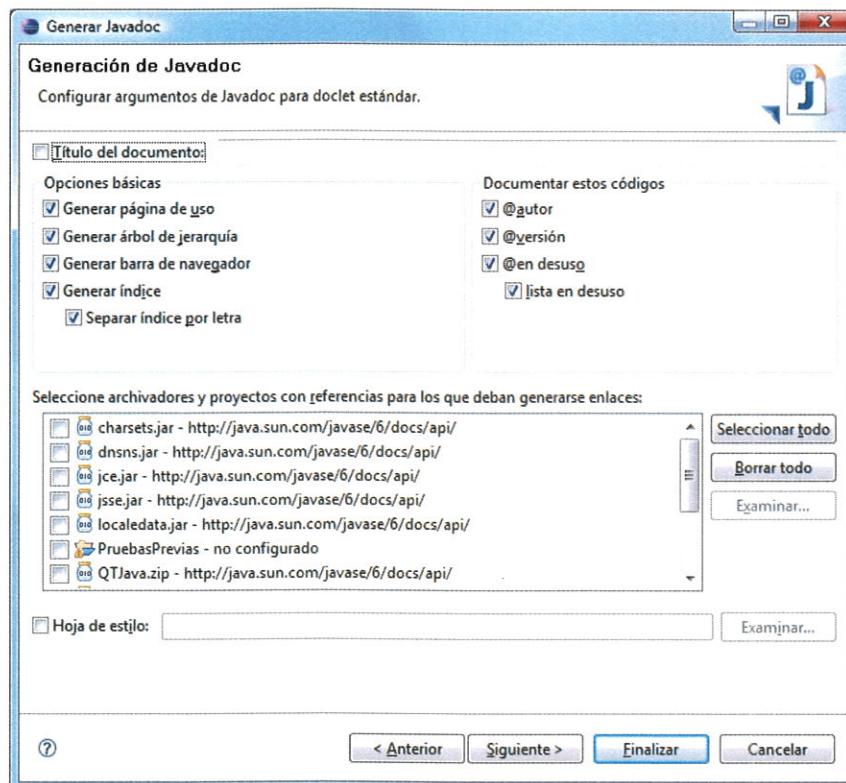


Ilustración 3-20, Cuadro de exportación de preferencias de Eclipse

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.

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 **Glassfish** 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.

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

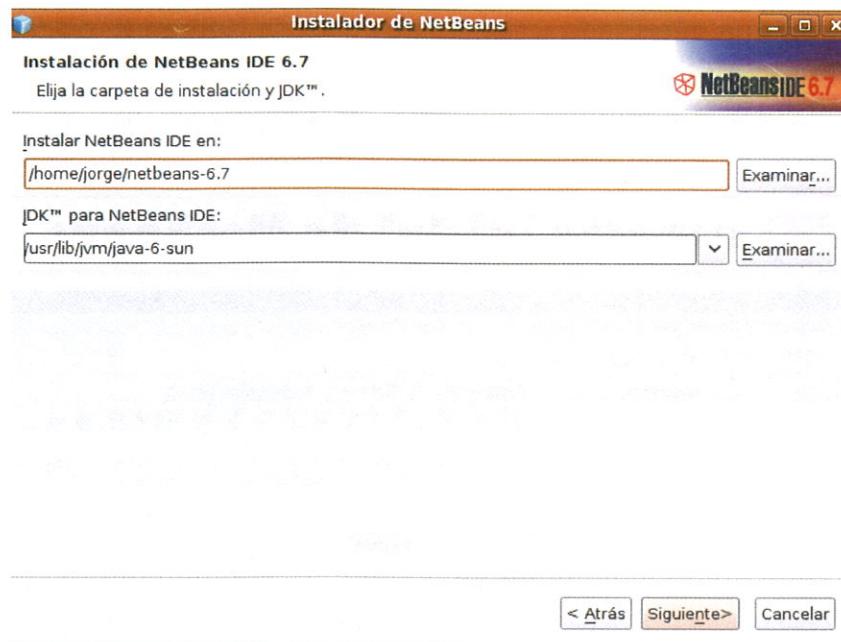


Ilustración 3-24, Selección de directorios durante la instalación de Netbeans

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

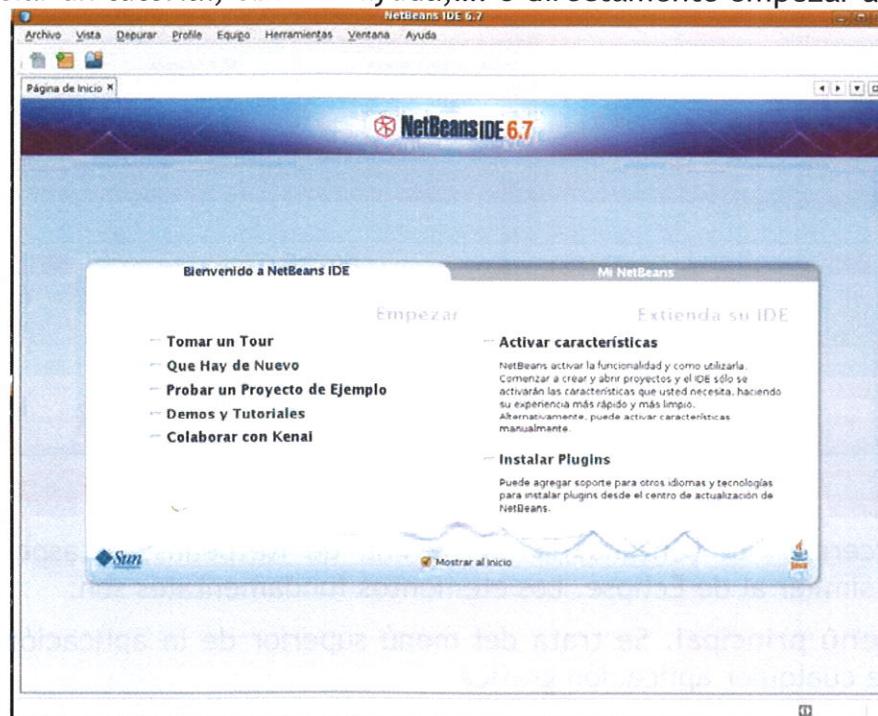


Ilustración 3-25, Pantalla inicial de Netbeans

◆ **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 ícono. 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 ícono 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.

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 desechar estructura

(II.iv.ii) La ventana del proyecto

El aspecto de Netbeans tras crear un proyecto es este:

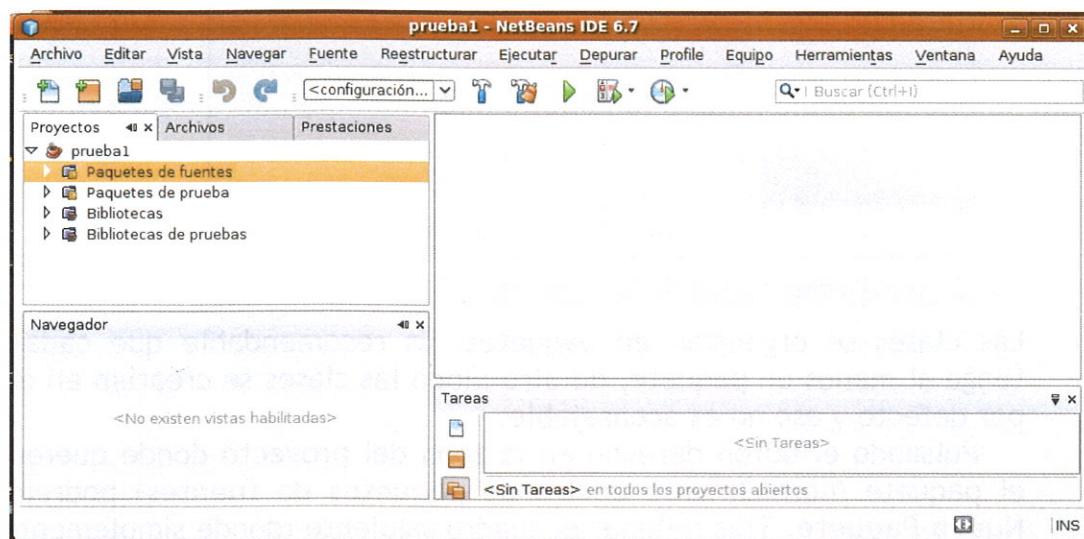


Ilustración 3-29, Aspecto de Netbeans con un proyecto vacío

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.

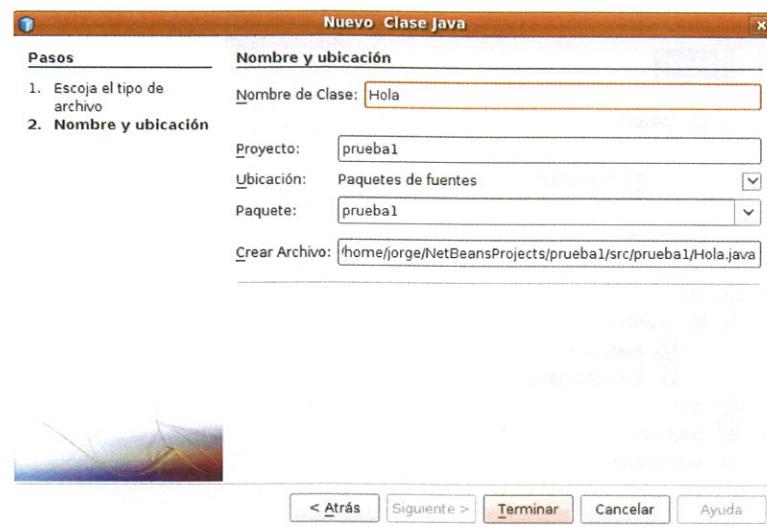


Ilustración 3-31, Cuadro de creación de nuevas clases

(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.

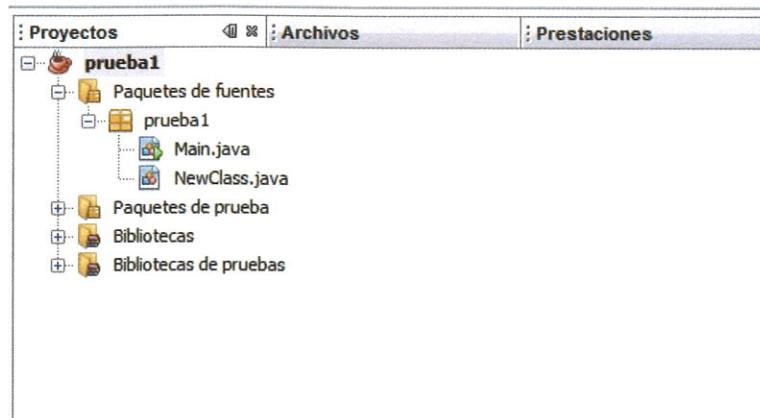


Ilustración 3-32, Ventana de proyectos

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:

(II.iv.vii) borrar un proyecto

Si deseamos quitar un proyecto podemos: pulsar el botón secundario del ratón sobre el ícono 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.

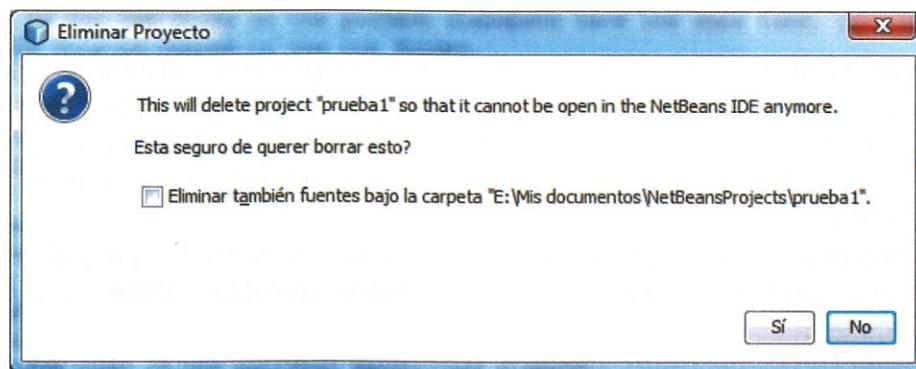


Ilustración 3-35, cuadro de confirmación de borrado de un proyecto

(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.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

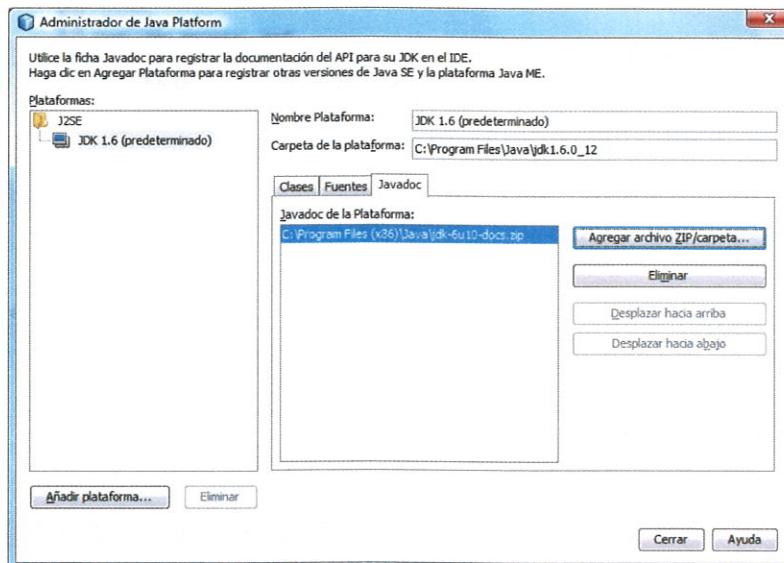


Ilustración 3-36, Seleccionando la documentación de Java

(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**.

(II.vii.ii) ayudas al escribir código

Como la mayoría de IDEs actuales, Netbeans posee numerosas mejoras para escribir 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 conocamos 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 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**

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.

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package prueba1;

/**
 *
 * @author jorge
 */
public class NewClass {
    static int prueba() {
        return 1;
    }
}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package prueba1;

/**
 *
 * @author jorge
 */
public class Main {
    /**
     * Starts args the Command Line arguments
     */
    public static void main(String[] args) {
        System.out.println(NewClass.prueba());
    }
}
```

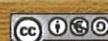
Ilustración 3-40, La herramienta diff comparando archivos

Índice de ilustraciones

Ilustración 3-1, Proceso de compilación de un programa C++ en Windows y Linux.....	13
Ilustración 3-2, Proceso de compilación de un programa Java	14
Ilustración 3-3, El cuadro de las variables del Sistema en Windows Server 2008	20
Ilustración 3-4, Ejemplo de configuración de la variable JAVA_HOME en Windows	20
Ilustración 3-5, El gestor de paquetes Synaptic en el sistema Linux Ubuntu,.....	21
Ilustración 3-6, Ejemplo de compilación y ejecución en la línea de comandos.....	26
Ilustración 3-7, Página de documentación de un programa Java	29
Ilustración 3-8, Pantalla de la página de descargas de Eclipse.....	47
Ilustración 3-9, Aspecto del primer arranque de Eclipse	48
Ilustración 3-10, Aspecto de la bienvenida de Eclipse	49
Ilustración 3-11, Aspecto del área de trabajo de Eclipse al iniciar por primera vez	50
Ilustración 3-12, Cuadro de creación de proyectos nuevos en Eclipse.....	52
Ilustración 3-13, Cuadro de nueva clase Java en Eclipse.....	53
Ilustración 3-14, Cuadro de preferencias de Eclipse referente al Editor de Java.....	60
Ilustración 3-15, Cuadro de preferencias en el apartado de plantillas.....	62
Ilustración 3-16, Cuadro de plantillas en Eclipse	64
Ilustración 3-17, El cuadro de preferencias en el apartado <i>Estilo de código</i>	65
Ilustración 3-18, Creación de un nuevo perfil de formato de código	66
Ilustración 3-19, Cuadro de preferencias en el apartado Plantillas de código.....	68
Ilustración 3-20, Cuadro de exportación de preferencias de Eclipse.....	69
Ilustración 3-21, cuadro de generación de Javadoc en Eclipse	70
Ilustración 3-22, Pantalla inicial de instalación de Netbeans.....	74
Ilustración 3-23, Selección de los componentes Netbeans a instalar	74
Ilustración 3-24, Selección de directorios durante la instalación de Netbeans.....	75
Ilustración 3-25, Pantalla inicial de Netbeans.....	75
Ilustración 3-26, Aspecto habitual de Netbeans.....	76
Ilustración 3-27, Selección del tipo de proyecto en Netbeans	78
Ilustración 3-28, Selección de las opciones del proyecto.....	78
Ilustración 3-29, Aspecto de Netbeans con un proyecto vacío	79
Ilustración 3-30, Cuadro de creación de nuevos paquetes.....	80
Ilustración 3-31, Cuadro de creación de nuevas clases	81
Ilustración 3-32, Ventana de proyectos.....	81
Ilustración 3-33, Ventana de archivos del proyecto.....	82
Ilustración 3-34, Cuadro de propiedades abierto por las opciones de ejecución.....	82
Ilustración 3-35, cuadro de confirmación de borrado de un proyecto.....	83
Ilustración 3-36, Seleccionando la documentación de Java	85
Ilustración 3-37, La ventana de código mostrando dos archivos a la vez	86
Ilustración 3-38, El cuadro de opciones del editor en el apartado de formato del código.....	88
Ilustración 3-39, El cuadro de administración de plantillas	90
Ilustración 3-40, La herramienta diff comparando archivos	91
Ilustración 3-41, Cuadro de exportación de opciones de Netbeans	92

Unidad 4: Programación estructurada en lenguaje Java

Fundamentos de Programación. 1º de ASI



Esta obra está bajo una licencia de Creative Commons.

Autor: Jorge Sánchez Asenjo (año 2009) <http://www.jorgesanchez.net>
e-mail:info@jorgesanchez.net

Esta obra está bajo una licencia de Reconocimiento-NoComercial-CompartirIgual de Creative Commons

Para ver una copia de esta licencia, visite:

<http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es>

o envíe una carta a:

Creative Commons, 559 Nathan Abbot



Reconocimiento-NoComercial-CompartirIgual 2.5 España

Usted es libre de:



copiar, distribuir y comunicar públicamente la obra

hacer obras derivadas

Bajo las condiciones siguientes:



Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).



No comercial. No puede utilizar esta obra para fines comerciales.



Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Apart from the remix rights granted under this license, nothing in this license impairs or restricts the author's moral rights.

Advertencia

Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.
Esto es un resumen legible por humanos del texto legal (la licencia completa) disponible en los idiomas siguientes:
Catalán Castellano Euskera Gallego

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