

JavaTM 2

Curso de programación

Microsoft Windows, Linux, Solaris y otros

4^a EDICIÓN

Entorno de desarrollo

NetBeans+JDK



Programación
orientada a objetos



Elementos del
lenguaje



Clases de uso común



Estructura de un
programa



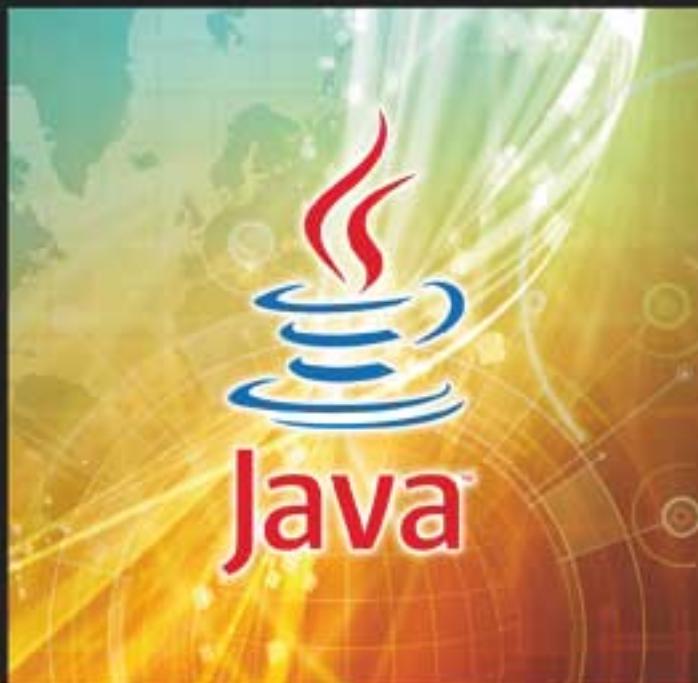
Sentencias de control



Matrices y cadenas



Métodos y
colecciones



Clases, subclases,
paquetes e interfaces

Clases genéricas



Excepciones



Ficheros



Estructuras
dinámicas



Algoritmos



Hilos



Interfaces gráficas
y aplicaciones para
Internet



Fco. Javier Ceballos

Puede descargarse el
CD-ROM con las URL para
obtener el software de
desarrollo y las aplicaciones
contenidas en el libro.



Ra-Ma®

Java™ 2

Curso de Programación

4^a edición

Fco. Javier Ceballos Sierra

Profesor titular de la
Escuela Politécnica Superior
Universidad de Alcalá

<http://www.fjceballos.es>





Java 2: Curso de programación. 4^a edición.

© Fco. Javier Ceballos Sierra

© De la edición: RA-MA 2010

MARCAS COMERCIALES: Las marcas de los productos citados en el contenido de este libro (sean o no marcas registradas) pertenecen a sus respectivos propietarios. RA-MA no está asociada a ningún producto o fabricante mencionado en la obra, los datos y los ejemplos utilizados son ficticios salvo que se indique lo contrario.

RA-MA es una marca comercial registrada.

Se ha puesto el máximo empeño en ofrecer al lector una información completa y precisa. Sin embargo, RA-MA Editorial no asume ninguna responsabilidad derivada de su uso, ni tampoco por cualquier violación de patentes ni otros derechos de terceras partes que pudieran ocurrir. Esta publicación tiene por objeto proporcionar unos conocimientos precisos y acreditados sobre el tema tratado. Su venta no supone para el editor ninguna forma de asistencia legal, administrativa ni de ningún otro tipo. En caso de precisarse asesoría legal u otra forma de ayuda experta, deben buscarse los servicios de un profesional competente.

Reservados todos los derechos de publicación en cualquier idioma.

Según lo dispuesto en el Código Penal vigente ninguna parte de este libro puede ser reproducida, grabada en sistema de almacenamiento o transmitida en forma alguna ni por cualquier procedimiento, ya sea electrónico, mecánico, reprográfico, magnético o cualquier otro, sin autorización previa y por escrito de RA-MA; su contenido está protegido por la Ley vigente que establece penas de prisión y/o multas a quienes intencionadamente, reprodujeren o plagiaren, en todo o en parte, una obra literaria, artística o científica.

Editado por:

RA-MA Editorial

C/ Jarama, 3A, Polígono industrial Igarsa

28860 PARACUELLOS DEL JARAMA, Madrid

Teléfono: 91 658 42 80

Telefax: 91 662 81 39

Correo electrónico: editorial@ra-ma.com

Internet: www.ra-ma.es y www.ra-ma.com

ISBN: 978-84-9964-032-7

Depósito Legal: M-xxxxx-2010

Autoedición: Fco. Javier Ceballos

Filmación e impresión: Closas-Orcoyen, S.L.

Impreso en España

Primera impresión: Noviembre 2010

RESUMEN DEL CONTENIDO

PARTE 1. PROGRAMACIÓN BÁSICA	1
CAPÍTULO 1. FASES EN EL DESARROLLO DE UN PROGRAMA	3
CAPÍTULO 2. INTRODUCCIÓN A JAVA.....	19
CAPÍTULO 3. INTRODUCCIÓN A LA POO.....	33
CAPÍTULO 4. ELEMENTOS DEL LENGUAJE	59
CAPÍTULO 5. ESTRUCTURA DE UN PROGRAMA	85
CAPÍTULO 6. CLASES DE USO COMÚN	109
CAPÍTULO 7. SENTENCIAS DE CONTROL.....	147
CAPÍTULO 8. MATRICES	185
CAPÍTULO 9. MÁS SOBRE MÉTODOS Y COLECCIONES	239
PARTE 2. MECANISMOS DE ABSTRACCIÓN.....	271
CAPÍTULO 10. CLASES Y PAQUETES	273
CAPÍTULO 11. SUBCLASES E INTERFACES	365
CAPÍTULO 12. CLASES GENÉRICAS	427
CAPÍTULO 13. EXCEPCIONES	439

CAPÍTULO 14. FLUJOS	463
PARTE 3. DISEÑO Y PROGRAMACIÓN	535
CAPÍTULO 15. ESTRUCTURAS DINÁMICAS	537
CAPÍTULO 16. ALGORITMOS	623
CAPÍTULO 17. HILOS	669
CAPÍTULO 18. INTERFACES GRÁFICAS Y APLICACIONES PARA INTERNET	731
PARTE 4. APÉNDICES.....	777
A. INSTALACIÓN DEL SOFTWARE	779
B. ENTORNO DE DESARROLLO INTEGRADO PARA JAVA.....	783
C. PLATAFORMAS UNIX/LINUX.....	817
D. FICHEROS JAR.....	819
E. JAVA COMPARADO CON C/C++.....	823
ÍNDICE	827

CONTENIDO

PRÓLOGO.....	XXIII
PARTE 1. PROGRAMACIÓN BÁSICA	1
CAPÍTULO 1. FASES EN EL DESARROLLO DE UN PROGRAMA.....	3
QUÉ ES UN PROGRAMA.....	3
LENGUAJES DE PROGRAMACIÓN.....	4
Compiladores	6
Intérpretes.....	6
¿QUÉ ES JAVA?	7
HISTORIA DE JAVA.....	8
¿POR QUÉ APRENDER JAVA?	9
REALIZACIÓN DE UN PROGRAMA EN JAVA.....	9
Cómo crear un programa.....	11
Interfaz de línea de órdenes.....	11
¿Qué hace este programa?.....	12
Guardar el programa escrito en el disco.....	13
Compilar y ejecutar el programa.....	13
Biblioteca de clases.....	15
Guardar el programa ejecutable en el disco	15
Depurar un programa	16
Entorno de desarrollo integrado.....	16
EJERCICIOS RESUELTOS	16
EJERCICIOS PROPUESTOS.....	18

CAPÍTULO 2. INTRODUCCIÓN A JAVA	19
DECLARACIÓN DE UNA VARIABLE	19
ASIGNAR VALORES	22
AÑADIR COMENTARIOS	23
MOSTRAR DATOS POR LA PANTALLA	24
EXPRESIONES ARITMÉTICAS	25
EXPRESIONES CONDICIONALES	27
ESCRIBIR NUESTROS PROPIOS MÉTODOS.....	29
EJERCICIOS PROPUESTOS.....	31
CAPÍTULO 3. INTRODUCCIÓN A LA POO.....	33
PENSAR EN OBJETOS	34
Clases y objetos.....	34
Mensajes y métodos	35
DISEÑO DE UNA CLASE DE OBJETOS	36
CONSTRUCTORES	43
HERENCIA.....	46
EJERCICIOS RESUELtos	53
EJERCICIOS PROPUESTOS.....	57
CAPÍTULO 4. ELEMENTOS DEL LENGUAJE.....	59
PRESENTACIÓN DE LA SINTAXIS DE JAVA.....	59
CARACTERES DE JAVA	60
Letras, dígitos y otros.....	60
Espacios en blanco	60
Caracteres especiales y signos de puntuación	61
Secuencias de escape.....	61
TIPOS DE DATOS	62
Tipos primitivos	62
Tipos referenciados	64
Tipos enumerados	64
LITERALES	65
Literales enteros	65
Literales reales	66
Literales de un solo carácter	66
Literales de cadenas de caracteres.....	67
IDENTIFICADORES	67
PALABRAS CLAVE.....	68
COMENTARIOS	68

DECLARACIÓN DE CONSTANTES SIMBÓLICAS	69
¿Por qué utilizar constantes?	70
Sentencia import static	70
DECLARACIÓN DE UNA VARIABLE	71
Iniciación de una variable	72
EXPRESIONES NUMÉRICAS	73
CONVERSIÓN ENTRE TIPOS DE DATOS	73
OPERADORES	74
Operadores aritméticos	74
Operadores de relación	75
Operadores lógicos	76
Operadores unitarios	77
Operadores a nivel de bits	77
Operadores de asignación	78
Operador condicional	80
PRIORIDAD Y ORDEN DE EVALUACIÓN	81
EJERCICIOS RESUELtos	82
EJERCICIOS PROPUESTOS	84
 CAPÍTULO 5. ESTRUCTURA DE UN PROGRAMA.....	 85
ESTRUCTURA DE UNA APLICACIÓN JAVA	85
Paquetes y protección de clases	89
Protección de una clase	90
Sentencia import	91
Definiciones y declaraciones	92
Sentencia simple	93
Sentencia compuesta o bloque	94
Métodos	94
Definición de un método	94
Método main	95
Crear objetos de una clase	95
Cómo acceder a los miembros de un objeto	97
Protección de los miembros de una clase	98
Miembro de un objeto o de una clase	99
Referencias a objetos	101
Pasando argumentos a los métodos	104
PROGRAMA JAVA FORMADO POR MÚLTIPLES FICHEROS	105
ACCESIBILIDAD DE VARIABLES	107
EJERCICIOS PROPUESTOS	108

CAPÍTULO 6. CLASES DE USO COMÚN	109
ENTRADA Y SALIDA	110
Flujos de entrada	111
Flujos de salida.....	112
Excepciones	113
Flujos estándar de E/S.....	114
Determinar la clase a la que pertenece un objeto	115
BufferedInputStream.....	116
BufferedReader	117
PrintStream	119
Trabajar con tipos de datos primitivos	121
Clases que encapsulan los tipos primitivos.....	122
Clase Leer	125
¿DÓNDE SE UBICAN LAS CLASES QUE DAN SOPORTE?.....	128
Variable CLASSPATH	128
CARÁCTER FIN DE FICHERO	129
CARACTERES \r\n	131
ALTERNATIVA A LOS FLUJOS DE E/S	133
Entrada de datos	133
Salida con formato	135
Clase Formatter	136
MÉTODOS MATEMÁTICOS	138
EJERCICIOS RESUELtos	140
EJERCICIOS PROPUESTOS.....	144
 CAPÍTULO 7. SENTENCIAS DE CONTROL.....	 147
SENTENCIA if	147
ANIDAMIENTO DE SENTENCIAS if	150
ESTRUCTURA else if.....	152
SENTENCIA switch	154
SENTENCIA while.....	159
Bucles anidados.....	162
SENTENCIA do ... while.....	165
SENTENCIA for	168
SENTENCIA break.....	171
SENTENCIA continue.....	172
ETIQUETAS.....	172
SENTENCIAS try ... catch.....	174
EJERCICIOS RESUELtos	175
EJERCICIOS PROPUESTOS.....	182

CAPÍTULO 8. MATRICES	185
INTRODUCCIÓN A LAS MATRICES	186
MATRICES NUMÉRICAS UNIDIMENSIONALES	187
Declarar una matriz	187
Crear una matriz	188
Iniciar una matriz	189
Acceder a los elementos de una matriz	189
Métodos de una matriz	190
Trabajar con matrices unidimensionales	191
Tamaño de una matriz	192
ArrayList	193
Añadir un elemento	194
Insertar un elemento	194
Modificar un elemento	194
Obtener un elemento	194
Iteradores	194
Tamaño	195
Eliminar elementos	195
Buscar elementos	195
Copiar listas	195
Ejemplo	196
Matrices asociativas	198
Map	200
CADENAS DE CARACTERES	203
Leer y escribir una cadena de caracteres	204
Clase String	206
String(String valor)	207
String toString()	207
String concat(String str)	208
int compareTo(String otroString)	208
int length()	209
String toLowerCase()	209
String toUpperCase()	210
String trim()	210
boolean startsWith(String prefijo)	210
boolean endsWith(String sufijo)	210
String substring(int IndiceInicial, int IndiceFinal)	210
char charAt(int índice)	211
int indexOf(int car)	211
int indexOf(String str)	211
String replace(char car, char nuevoCar)	211
static String valueOf(tipo dato)	211

char[] toCharArray()	211
byte[] getBytes()	212
Clase StringBuffer.....	212
StringBuffer([arg])	212
int length()	213
int capacity()	213
StringBuffer append(tipo x)	213
StringBuffer insert(int índice, tipo x)	213
StringBuffer delete(int p1, int p2)	213
StringBuffer replace(int p1, int p2, String str)	214
StringBuffer reverse()	214
String substring(int IndiceInicial, int IndiceFinal)	214
char charAt(int índice)	214
void setCharAt(int índice, char car)	215
String toString()	215
Clase StringTokenizer.....	215
Conversión de cadenas de caracteres a datos numéricos	216
MATRICES DE REFERENCIAS A OBJETOS.....	217
Matrices numéricas multidimensionales	218
Matrices de cadenas de caracteres.....	222
SENTENCIA for para colecciones.....	227
EJERCICIOS RESUELTOS	228
EJERCICIOS PROPUESTOS.....	235
CAPÍTULO 9. MÁS SOBRE MÉTODOS Y COLECCIONES	239
PASAR UNA MATRIZ COMO ARGUMENTO A UN MÉTODO	239
DATOS RETORNADOS POR UN MÉTODO	241
REFERENCIA A UN TIPO PRIMITIVO	243
ARGUMENTOS EN LA LÍNEA DE ÓRDENES	246
MÉTODOS RECURSIVOS.....	248
MÉTODOS SOBRECARGADOS	250
NÚMERO VARIABLE DE PARÁMETROS	252
LA CLASE Object	254
boolean equals(Object <i>obj</i>)	254
String toString()	255
void finalize()	256
LA CLASE Arrays.....	256
binarySearch.....	256
equals	257
fill.....	258
sort.....	258

COLECCIONES	259
MÁS SOBRE REFERENCIAS Y OBJETOS String	260
EJERCICIOS RESUELtos	263
EJERCICIOS PROPUESTOS.....	268
PARTE 2. MECANISMOS DE ABSTRACCIÓN.....	271
CAPÍTULO 10. CLASES Y PAQUETES	273
DEFINICIÓN DE UNA CLASE	273
Atributos	275
Métodos de una clase	276
Control de acceso a los miembros de la clase	277
Acceso predeterminado.....	278
Acceso público.....	279
Acceso privado.....	279
Acceso protegido	279
IMPLEMENTACIÓN DE UNA CLASE	280
MÉTODOS SOBRECARGADOS	283
NÚMERO VARIABLE DE PARÁMETROS	285
IMPLEMENTACIÓN DE UNA APLICACIÓN.....	286
CONTROL DE ACCESO A UNA CLASE.....	287
REFERENCIA this	288
VARIABLES, MÉTODOS Y CLASES FINALES	289
INICIACIÓN DE UN OBJETO.....	290
Constructor.....	292
Sobrecarga del constructor.....	295
Llamar a un constructor	296
Asignación de objetos	297
Constructor copia	298
DESTRUCCIÓN DE OBJETOS.....	299
Destructor.....	300
Ejecutar el recolector de basura	302
REFERENCIAS COMO MIEMBROS DE UNA CLASE	302
REDEFINIR MÉTODOS HEREDADOS DE Object.....	310
Método equals	310
Método clone.....	312
MIEMBROS STATIC DE UNA CLASE	314
Atributos static	314
Acceder a los atributos static.....	316
Métodos static	317

CLASES ANIDADAS	319
Clases internas.....	320
Clases definidas dentro de un método.....	322
Clases anónimas	323
TIPOS ENUMERADOS	325
MATRICES DE OBJETOS	327
PAQUETES	336
Crear un paquete	337
La clase aplicación pertenece a un paquete.....	339
Compilar y ejecutar la aplicación desde NetBeans	339
Compilar y ejecutar la aplicación desde una consola.....	342
EJERCICIOS RESUELtos	343
EJERCICIOS PROPUESTOS.....	358
 CAPÍTULO 11. SUBCLASES E INTERFACES	365
CLASES Y MÉTODOS ABSTRACTOS.....	366
SUBCLASES Y HERENCIA	367
DEFINIR UNA SUBCLASE	370
Control de acceso a los miembros de las clases	372
Qué miembros hereda una subclase	373
ATRIBUTOS CON EL MISMO NOMBRE.....	378
REDEFINIR MÉTODOS DE LA SUPERCLASE	379
CONSTRUCTORES DE LAS SUBCLASES	382
COPIA DE OBJETOS	385
DESTRUCTORES DE LAS SUBCLASES.....	387
JERARQUÍA DE CLASES	389
REFERENCIAS A OBJETOS DE UNA SUBCLASE.....	398
Conversiones implícitas	398
Conversiones explícitas.....	400
INFORMACIÓN DE TIPOS DURANTE LA EJECUCIÓN	401
POLIMORFISMO.....	401
MÉTODOS EN LÍNEA.....	411
INTERFACES.....	412
Definir una interfaz	412
Un ejemplo: la interfaz IFecha.....	413
Utilizar una interfaz	415
Clase abstracta frente a interfaz	418
Utilizar una interfaz como un tipo	419
Interfaces frente a herencia múltiple	421
Para qué sirve una interfaz	421
Implementar múltiples interfaces	422

EJERCICIOS RESUELTOS	422
EJERCICIOS PROPUESTOS.....	425
CAPÍTULO 12. CLASES GENÉRICAS.....	427
DEFINICIÓN DE CLASES GENÉRICAS	428
Relación entre clases genéricas	432
Tipo comodín	432
MÉTODOS GENÉRICOS	435
EJERCICIOS RESUELTOS	436
EJERCICIOS PROPUESTOS.....	438
CAPÍTULO 13. EXCEPCIONES.....	439
EXCEPCIONES DE JAVA	441
MANEJAR EXCEPCIONES	443
Lanzar una excepción.....	444
Capturar una excepción.....	444
Excepciones derivadas	446
BLOQUE DE FINALIZACIÓN	447
DECLARAR EXCEPCIONES	449
CREAR EXCEPCIONES	451
FLUJO DE EJECUCIÓN.....	453
CUÁNDΟ UTILIZAR EXCEPCIONES Y CUÁNDΟ NO.....	455
EJERCICIOS RESUELTOS	456
EJERCICIOS PROPUESTOS.....	461
CAPÍTULO 14. FLUJOS.....	463
VISIÓN GENERAL DE LOS FLUJOS DE E/S.....	465
Flujos que no procesan los datos de E/S	466
Flujos que procesan los datos de E/S	468
ABRIENDO FICHEROS PARA ACCESO SECUENCIAL.....	473
Flujos de bytes	474
FileOutputStream.....	474
FileInputStream.....	477
Clase File.....	478
Flujos de caracteres	481
FileWriter.....	482
FileReader.....	483

Flujos de datos	484
DataOutputStream.....	485
DataInputStream.....	486
Un ejemplo de acceso secuencial	487
SERIACIÓN DE OBJETOS	492
Escribir objetos en un fichero	494
Leer objetos desde un fichero	495
Seriar objetos que referencian a objetos.....	497
ABRIENDO FICHEROS PARA ACCESO ALEATORIO	501
La clase RandomAccessFile.....	501
La clase CPersona	504
La clase CListaTfnos.....	505
Constructor CListaTfnos.....	505
Escribir un registro en el fichero.....	507
Añadir un registro al final del fichero	508
Leer un registro del fichero.....	508
Eliminar un registro del fichero	509
¿Hay registros marcados para eliminar?	510
Buscar un registro en el fichero	510
Un ejemplo de acceso aleatorio a un fichero.....	511
Modificar un registro	514
Actualizar el fichero.....	516
ESCRIBIR DATOS EN LA IMPRESORA	517
EJERCICIOS RESUELTOS	518
EJERCICIOS PROPUESTOS.....	532
 PARTE 3. DISEÑO Y PROGRAMACIÓN	535
 CAPÍTULO 15. ESTRUCTURAS DINÁMICAS.....	537
 LISTAS LINEALES	538
Listas lineales simplemente enlazadas	538
Operaciones básicas	541
Inserción de un elemento al comienzo de la lista.....	542
Buscar en una lista un elemento con un valor x	544
Inserción de un elemento en general.....	544
Borrar un elemento de la lista	545
Recorrer una lista	546
Borrar todos los elementos de una lista	546
UNA CLASE PARA LISTAS LINEALES	547
Clase genérica para listas lineales	550
Clase LinkedList<T>	558

LISTAS CIRCULARES	560
Clase CListaCircularSE<T>.....	562
PILAS.....	566
COLAS.....	567
EJEMPLO	569
LISTA DOBLEMENTE ENLAZADA.....	571
Lista circular doblemente enlazada.....	572
Clase CListaCircularDE<T>.....	572
Ejemplo	578
ÁRBOLES.....	579
Árboles binarios	580
Formas de recorrer un árbol binario.....	582
ÁRBOLES BINARIOS DE BÚSQUEDA.....	584
Clase CArbolBinB<T>	585
Buscar un nodo en el árbol.....	588
Insertar un nodo en el árbol.....	589
Borrar un nodo del árbol	590
Utilización de la clase CArbolBinB<T>	593
ÁRBOLES BINARIOS PERFECTAMENTE EQUILIBRADOS.....	596
Clase CArbolBinE<T>.....	597
Utilización de la clase CArbolBinE<T>	602
CLASES APORTADAS POR LA BIBLIOTECA JAVA	604
EJERCICIOS RESUELTOS	605
EJERCICIOS PROPUESTOS.....	619
 CAPÍTULO 16. ALGORITMOS	623
RECURSIVIDAD	623
ORDENACIÓN DE DATOS.....	629
Método de la burbuja	630
Método de inserción.....	633
Método quicksort	634
Comparación de los métodos expuestos.....	638
BÚSQUEDA DE DATOS	638
Búsqueda secuencial	638
Búsqueda binaria.....	639
Búsqueda de cadenas	640
ORDENACIÓN DE FICHEROS EN DISCO.....	643
Ordenación de ficheros. Acceso secuencial	644
Ordenación de ficheros. Acceso aleatorio.....	651
ALGORITMOS HASH.....	654
Matrices hash	655

Método hash abierto	656
Método hash con desbordamiento.....	657
Eliminación de elementos	658
Clase CHashAbierto.....	658
Un ejemplo de una matriz hash	662
EJERCICIOS RESUELtos	665
EJERCICIOS PROPUESTOS.....	667
 CAPÍTULO 17. HILOS	 669
CONCEPTO DE PROCESO	669
HILOS	674
Estados de un hilo	676
Cuándo se debe crear un hilo	677
PROGRAMAR CON HILOS	677
Crear un hilo.....	677
Hilo derivado de Thread	679
Hilo asociado con una clase	681
Demonios	684
Finalizar un hilo	685
Controlar un hilo	687
Preparado	688
Bloqueado	688
Dormido	689
Esperando.....	689
SINCRONIZACIÓN DE HILOS	690
Secciones críticas	691
Exclusión mutua.....	695
Monitor reentrant	698
Utilizar wait y notify	699
¿Por qué los métodos almacenar y obtener utilizan un bucle?.....	705
Interbloqueo	706
GRUPO DE HILOS	707
Grupo predefinido	707
Grupo explícito	709
PLANIFICACIÓN DE HILOS	709
¿Qué ocurre con los hilos que tengan igual prioridad?	710
Asignar prioridades a los hilos	711
TUBERÍAS	714
ESPERA ACTIVA Y PASIVA.....	719
EJERCICIOS RESUELtos	719
EJERCICIOS PROPUESTOS.....	728

CAPÍTULO 18. INTERFACES GRÁFICAS Y APLICACIONES PARA INTERNET	731
INTERFACES GRÁFICAS	732
Estructura de una aplicación	732
Compilar y ejecutar la aplicación.....	736
DISEÑO DE LA INTERFAZ GRÁFICA.....	736
Crear un componente Swing	737
Componentes Swing más comunes.....	737
Contenedores.....	738
Administradores de diseño	739
Añadir los componentes al contenedor	740
Asignar un administrador de diseño.....	741
Añadir una etiqueta y editar sus propiedades.....	741
Añadir un botón de pulsación y editar sus propiedades.....	742
MANEJO DE EVENTOS	743
Asignar manejadores de eventos a un objeto	744
Adaptadores	746
Responder a los eventos	748
ACCEDER A LA WEB	749
PÁGINAS WEB.....	751
PÁGINAS WEB DINÁMICAS	752
APPLETS	753
Crear un applet	754
Un ejemplo simple	757
Restricciones de seguridad con los applets	761
Instalación de un applet en un servidor.....	762
SERVLETS	763
Estructura de un servlet.....	763
Software necesario para ejecutar un servlet	768
Desplegar un servlet en el servidor	769
Invocando al servlet desde una página HTML	770
Descriptor de despliegue	772
Ejecutar un servlet.....	773
EJERCICIOS RESUELTO.....	773
EJERCICIOS PROPUESTOS.....	776
PARTE 4. APÉNDICES.....	777
INSTALACIÓN DEL SOFTWARE.....	779
ENTORNO DE DESARROLLO INTEGRADO PARA JAVA	783

PLATAFORMAS UNIX/LINUX	817
FICHEROS JAR	819
JAVA COMPARADO CON C/C++	823
ÍNDICE	827

PRÓLOGO

Java, junto con C#, es actualmente el lenguaje de programación más popular en Internet. Pero, además, está disponible para el desarrollo de programas de uso general. Conocer esta faceta del lenguaje Java, sin olvidar que tiene un alcance completo sobre la *Web*, es la idea fundamental de esta obra.

Hace pocos años, quizás “Java” nos traía a la mente una taza de café. ¿Por qué una taza de café? Seguramente por las muchas que se tomaron sus creadores. De hecho la taza de café ha pasado a ser su logotipo. Hoy en día, cualquiera que haya tenido contacto con una página *Web* tiene otro concepto. Sabe que Java es un lenguaje de programación introducido por *Sun Microsystems* cuyas características lo sitúan entre los productos ideales para desarrollar programas para la *Web*.

Cuando Java se introdujo de forma importante, allá por 1995, fue cuando su uso en el diseño de páginas *Web* revolucionó la naturaleza de éstas. ¿Recuerda? Todo el mundo hablaba de *applets*, esos pequeños programas que se ejecutan en el contexto de una página *Web* en cualquier ordenador, introduciendo animación y efectos especiales. Y quizás, esta idea esté enmascarando que “Java” no sólo es eso. Java está también disponible para desarrollar aplicaciones de uso general; esto es, como muchos lenguajes, permite trabajar con todo tipo de datos, crear estructuras dinámicas, trabajar con ficheros, atacar a bases de datos, diseñar interfaces gráficas de usuario, etc. Más aún, Java es un lenguaje simple y potente de propio derecho. Java está orientado a objetos. Su sintaxis incita al programador a generar programas modulares y fácilmente mantenibles.

Por lo tanto, Java le permite crear programas para su uso personal, para su grupo de trabajo, para una empresa, aplicaciones distribuidas a través de Internet, aplicaciones de bases de datos y otras muchas que usted puede imaginar.

Actualmente en el mercado hay multitud de herramientas de programación Java como *Eclipse* o *NetBeans* de Sun. Pero la mejor forma de ver el alcance de Java es desarrollando directamente a través del kit de desarrollo de Java (JDK). Se trata de un paquete que se puede obtener de la red compuesto por un conjunto de herramientas (programas y bibliotecas) para editar, compilar, ejecutar y depurar programas Java.

La primera edición de este libro fue escrita con Java 2 SDK versión 1.3, la segunda con la 1.4, la tercera con la 5.0 y esta cuarta ha sido revisada con la 6.0. Casi en su totalidad está dedicado al aprendizaje del lenguaje Java, de la programación orientada a objetos y al desarrollo de aplicaciones. Un capítulo final le introducirá también en otros conceptos como *swing*, *applets* y *servlets*. Esta materia puede agruparse en los siguientes apartados:

- *Programación básica.*
- *Mecanismos de abstracción.*
- *Diseño y programación.*

Para quién es este libro

Este libro está pensado para aquellas personas que quieran aprender o afianzar sus conocimientos en lo que a la programación básica se refiere: tipos, sentencias, matrices, métodos, ficheros, etc., para a continuación aprender programación orientada a objetos (POO) en detalle: clases, clases derivadas, interfaces, espacios de nombres, excepciones, flujos, etc.; después, utilizando la POO, el libro añade otros temas como estructuras dinámicas de datos, algoritmos de uso común, hilos (programación concurrente), etc. Éste sí que es un libro de programación con Java en toda su extensión. Finalmente, hace una introducción a las interfaces gráficas y a las aplicaciones para Internet.

Evidentemente, todo lo aprendido tiene continuación. Quizás, como siguiente paso, desee aprender a desarrollar aplicaciones que muestren una interfaz gráfica a base de ventanas, o a desarrollar aplicaciones que accedan a bases de datos, o bien a desarrollar aplicaciones para Internet a base de páginas Web. Pues todo esto es lo que se expone ampliamente en mi otro libro *Java 2 – Interfaces gráficas y aplicaciones para Internet*. Puede ver más detalles de cada uno de los libros en mi Web: www.fjceballos.es.

Cómo está organizado el libro

La primera parte, capítulos 1 a 9, está pensada para que en poco tiempo pueda convertirse en programador de aplicaciones Java. Y para esto, ¿qué necesita? Pues

simplemente leer ordenadamente los capítulos del libro, resolviendo cada uno de los ejemplos que en ellos se detallan.

La segunda parte, capítulos 10 a 14, abarca en profundidad la programación orientada a objetos. En la primera parte el autor ha tratado de desarrollar aplicaciones sencillas, para introducirle más bien en el lenguaje y en el manejo de la biblioteca de clases de Java que en el diseño de clases de objetos. No obstante, sí ha tenido que quedar claro que un programa orientado a objetos sólo se compone de objetos. Es hora pues de entrar con detalle en la programación orientada a objetos la cual tiene un elemento básico: la *clase*.

Pero si el autor finalizara el libro con las dos partes anteriores, privaría al lector de saber que aún Java proporciona mucho más. Por eso hay una tercera parte, capítulos 15 a 18, que se dedica a estudiar las estructuras dinámicas de datos, los algoritmos de uso común, los hilos, y a introducirle en el diseño de interfaces gráficas de usuario, en el trabajo con *applets* y en aplicaciones para Internet.

Todo ello se ha documentado con abundantes problemas resueltos. Cuando complete todas las partes, todavía no sabrá todo lo que es posible hacer con Java, pero sí habrá dado un paso importante.

Qué se necesita para utilizar este libro

Este libro ha sido escrito utilizando el paquete *J2SE Development Kit 6.0 (JDK 6.0)* y el entorno de desarrollo *NetBeans* de *Sun Microsystems* (adquirida por *Oracle* en 2009) que incluye todo lo necesario para escribir, construir, verificar y ejecutar aplicaciones Java. Por lo tanto, basta con que instale en su máquina ambos paquetes.

Por eso el autor considera importante que antes de continuar, eche una ojeada a los apéndices. En ellos se expone cómo utilizar el entorno de desarrollo integrado para Java *NetBeans*, la documentación sobre el JDK, se indica cómo proceder para trabajar sobre la plataforma Linux, las principales diferencias de este lenguaje con respecto a C/C++ (Java se crea a partir de C/C++, por lo que este apéndice aporta información valiosa para los lectores que tienen conocimientos sobre este lenguaje), ayuda para la instalación del software necesario para implementar las aplicaciones expuestas en el libro, etc.

Sobre los ejemplos del libro

El código fuente de todos los ejemplos del libro podrá descargarse, según se indica en los apéndices, de la Web www.ra-ma.es desde la página Web correspondiente al libro.

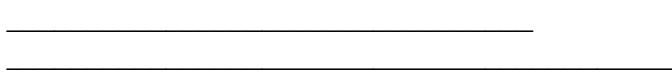
Agradecimientos

He recibido ayuda de algunas personas durante la preparación de este libro, y por ello estoy francamente agradecido; pero en especial, quiero expresar mi agradecimiento a mi colega **Oscar García Población** por sus buenas recomendaciones y aportaciones en la primera versión, base de las siguientes; y a mis otros colegas porque cuando a ellos he recurrido, de una forma u otra me han ayudado. También, expresar mi agradecimiento a *Sun Microsystems* por poner a mi disposición en particular y de todos los lectores en general, los productos que la creación y el estudio de esta obra requiere.

Francisco Javier Ceballos Sierra
<http://www.fjceballos.es/>

P A R T E

1

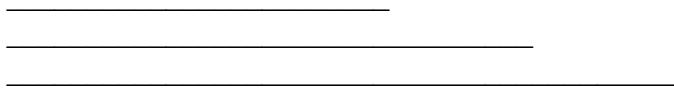


Programación básica

- Fases en el desarrollo de un programa
- Introducción a Java
- Introducción a la POO
- Elementos del lenguaje
- Estructura de un programa
- Clases de uso común
- Sentencias de control
- Matrices
- Más sobre métodos y colecciones

P A R T E

2



Mecanismos de abstracción

- Clases y paquetes
- Subclases e interfaces
- Clases genéricas
- Excepciones
- Flujos

P A R T E

3

Diseño y programación

- Estructuras dinámicas
- Algoritmos
- Hilos
- Interfaces gráficas y aplicaciones para Internet

P A R T E

4

Apéndices

- Instalación del software
- Entorno de desarrollo integrado para Java
- Plataformas Unix/Linux
- Ficheros JAR
- Java comparado con C/C++
- Índice

APÉNDICE A

© F.J.Ceballos/RA-MA

INSTALACIÓN DEL SOFTWARE

En los capítulos de este libro, vamos a necesitar utilizar distintos paquetes de software para poder implementar y probar las aplicaciones que en ellos se explican; por ejemplo, el entorno de desarrollo *NetBeans* o el servidor de aplicaciones *Tomcat*. Este apéndice le indicará de forma breve cómo instalar estos paquetes y cómo ponerlos en marcha.

J2SE 6.0

Java 2 Platform Standard Edition Development Kit 6.0 (abreviadamente, *J2SE Development Kit 6.0* o *JDK 6.0*) proporciona la base para desarrollar y distribuir aplicaciones que se podrán ejecutar en un servidor o en un ordenador personal con distintos sistemas operativos. Actualiza las versiones anteriores e incluye nuevas características (desarrollo más fácil, más rápido y a menor coste y ofrece mayor funcionalidad para servicios Web, soporte de lenguajes dinámicos, diagnósticos, aplicaciones de escritorio, bases de datos, etc.), pero conservando la compatibilidad con la versión anterior y la estabilidad.

Instalación

Para instalar el paquete de desarrollo J2SE 6.0, siga los pasos indicados a continuación:

1. Descargue de Internet, de la dirección URL indicada en el CD que acompaña al libro, el fichero *jdk-6u21-windows-i586.exe*, o la versión que en ese momento esté disponible.
2. Siga los pasos indicados durante la instalación.

3. Si a continuación desea instalar la documentación, descargue de Internet, de la dirección URL indicada en el CD que acompaña al libro, el fichero *jdk-6u21-docs.zip*, o la versión que en ese momento esté disponible. Puede instalarla en *jdk1.6.0\docs*.

NetBeans 6.x

NetBeans 6.x es un entorno de desarrollo integrado para desarrollar y distribuir aplicaciones multicapa distribuidas. Incluye un entorno gráfico de desarrollo de Java, una utilidad para desarrollar aplicaciones Web, otra para desarrollar aplicaciones para dispositivos móviles y un servidor de aplicaciones (*Apache Tomcat 6.x*) y una serie de herramientas que facilitan el desarrollo y la distribución de las aplicaciones.

Instalación

Para instalar el entorno de desarrollo *NetBeans*, siga los pasos indicados a continuación:

1. Descargue de Internet, de la dirección URL indicada en el CD que acompaña al libro, el fichero *netbeans-6.9-m1-java-windows.exe*, o la versión que en ese momento esté disponible.
2. Realice una instalación personalizada, ya que *Tomcat* no se instala por defecto, y a continuación siga los pasos indicados durante la instalación.
3. Para obtener ayuda acerca de la biblioteca de J2SE 6.0, sólo si instaló dicha ayuda, es preciso que dicho entorno tenga conocimiento de la ruta de acceso a la misma. Para ello, ejecute la orden *Plataformas Java* del menú *Herramientas* y asegúrese de que en la lista de rutas mostrada en *Javadoc* hay una que hace referencia a la carpeta donde se encuentra la ayuda mencionada; si no es así, haga clic en el botón *Agregar archivo ZIP/carpeta* para añadirla.

CONTENEDOR DE SERVLET/JSP TOMCAT 6.x

Tomcat 6.0 es un servidor de aplicaciones que implementa las tecnologías *Java Servlet 2.5* y *JavaServer Pages 2.1*; esto es, funciona como un contenedor de *servlets* y páginas *JSP*.

Un servidor de aplicaciones, a diferencia de un servidor Web, como es *Apache*, incluye un contenedor Web que permite servir páginas dinámicas (un servidor Web sólo sirve páginas HTML estáticas, recursos CGI, páginas PHP y

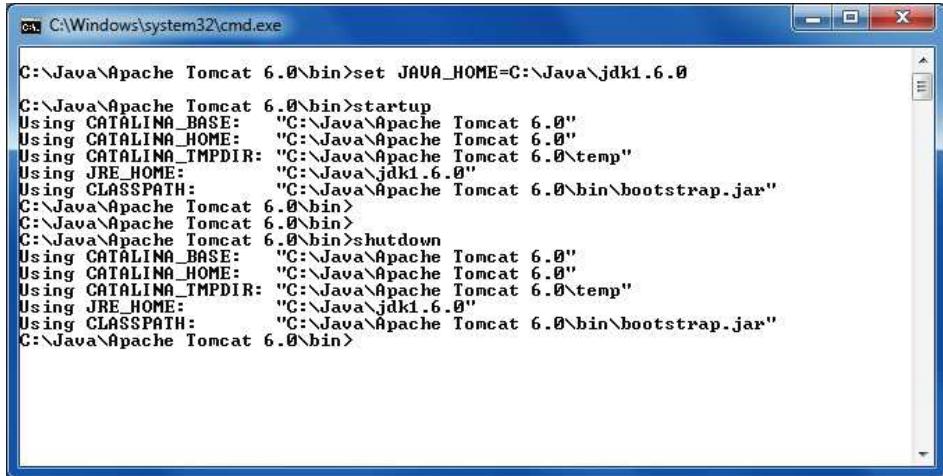
accesos SSL). Evidentemente, si se trata de servir páginas estáticas, es más eficiente un servidor Web, pero también podemos utilizar para este cometido un servidor de aplicaciones, es menos eficiente, pero es más seguro, o bien podríamos utilizar ambos conectados entre sí, de ahí que *Apache Tomcat* sea un servidor Web con soporte para *servlets* y páginas *JSP*.

Instalación

Cuando instaló *NetBeans*, también se realizó la instalación de *Apache Tomcat* justo a continuación.

Iniciar y parar el servidor Apache Tomcat

Suponiendo que la instalación del servidor *Apache Tomcat* es la realizada cuando instaló *NetBeans*, antes de poder utilizar este servidor deberá iniciarla (*startup*) y cuando ya no vaya a utilizarlo deberá pararlo (*shutdown*). La forma de realizar estas operaciones se indica en la figura mostrada a continuación:



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The command entered is 'C:\Java\Apache Tomcat 6.0\bin>set JAVA_HOME=C:\Java\jdk1.6.0'. This is followed by the output of the 'startup' command, which includes environment variable assignments for CATALINA_BASE, CATALINA_HOME, CATALINA_TMPDIR, JRE_HOME, and CLASSPATH, all pointing to the Tomcat 6.0 directory. Finally, the 'shutdown' command is run, which performs similar assignments but with slightly different paths for the temporary directory.

```
C:\Java\Apache Tomcat 6.0\bin>set JAVA_HOME=C:\Java\jdk1.6.0
C:\Java\Apache Tomcat 6.0\bin>startup
Using CATALINA_BASE:      "C:\Java\Apache Tomcat 6.0"
Using CATALINA_HOME:      "C:\Java\Apache Tomcat 6.0"
Using CATALINA_TMPDIR:    "C:\Java\Apache Tomcat 6.0\temp"
Using JRE_HOME:           "C:\Java\jdk1.6.0"
Using CLASSPATH:          "C:\Java\Apache Tomcat 6.0\bin\bootstrap.jar"
C:\Java\Apache Tomcat 6.0\bin>
C:\Java\Apache Tomcat 6.0\bin>shutdown
Using CATALINA_BASE:      "C:\Java\Apache Tomcat 6.0"
Using CATALINA_HOME:      "C:\Java\Apache Tomcat 6.0"
Using CATALINA_TMPDIR:    "C:\Java\Apache Tomcat 6.0\temp"
Using JRE_HOME:           "C:\Java\jdk1.6.0"
Using CLASSPATH:          "C:\Java\Apache Tomcat 6.0\bin\bootstrap.jar"
C:\Java\Apache Tomcat 6.0\bin>
```


APÉNDICE B

© F.J.Ceballos/RA-MA

ENTORNO DE DESARROLLO INTEGRADO PARA JAVA

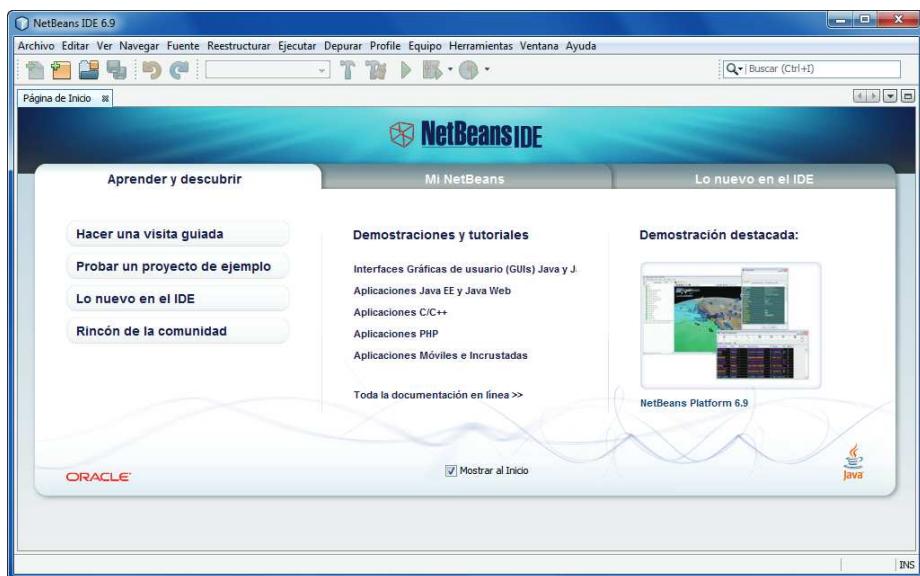
Evidentemente, para poder escribir programas se necesita un entorno de desarrollo Java. *Sun Microsystems*, propietario de Java, proporciona uno de forma gratuita, **J2SE Development Kit 6.0 (JDK 6.0)** para Microsoft Windows, en todas sus versiones, y para Linux. En el apéndice A se explica cómo obtenerlo e instalarlo.

Opcionalmente, puede instalar un entorno de desarrollo integrado (EDI) que le facilite las tareas de creación de la interfaz gráfica de usuario, edición del código, compilación, ejecución y depuración, como por ejemplo: *NetBeans* de *Sun Microsystems*. Para instalarlo, véase el apéndice A.

Asegúrese de que las variables de entorno *PATH* y *CLASSPATH* están perfectamente establecidas (en el caso de instalar *NetBeans* esta operación se realizará automáticamente).

DISEÑO DE UNA APLICACIÓN DE CONSOLA

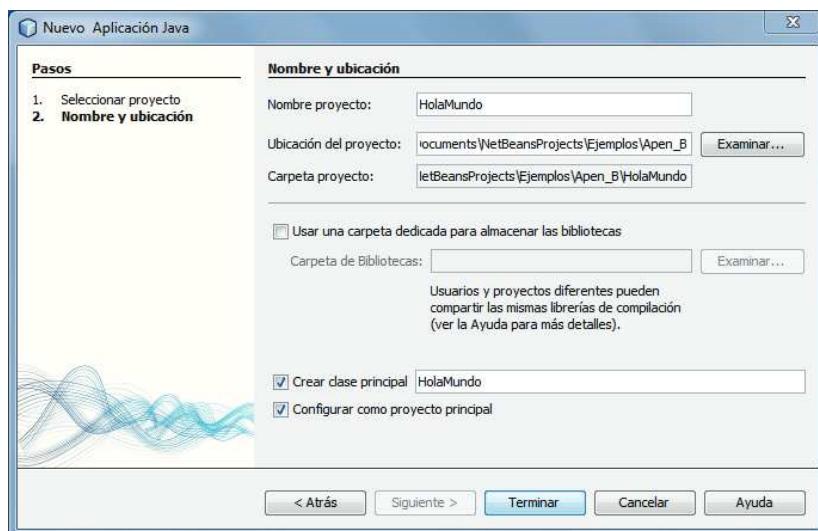
Cuando se utiliza un entorno de desarrollo integrado (EDI), lo primero que hay que hacer una vez instalado es asegurarse de que las rutas donde se localizan las herramientas, las bibliotecas, la documentación y los ficheros fuente hayan sido establecidas; algunos EDI sólo requieren la ruta donde se instaló el compilador. Este proceso normalmente se ejecuta automáticamente durante el proceso de instalación de dicho entorno. Si no es así, el entorno proporcionará algún menú con las órdenes apropiadas para realizar dicho proceso. Por ejemplo, en el entorno de desarrollo integrado *NetBeans* que se presenta a continuación, esas rutas a las que nos referimos quedan establecidas durante la instalación del mismo.



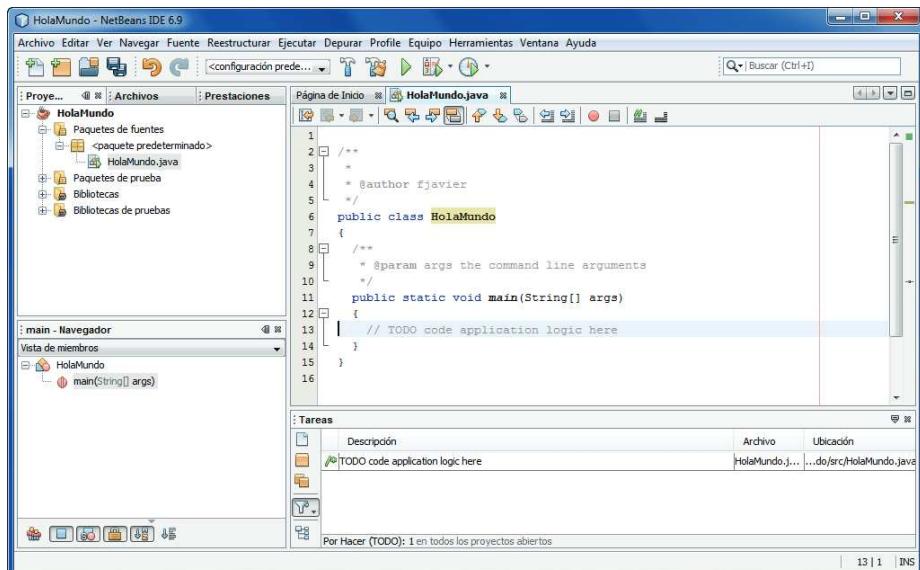
Para personalizar el EDI, ejecute la orden *Options* del menú *Herramientas*.

Para editar y ejecutar la aplicación realizada en el capítulo 1 “¡¡¡Hola mundo!!!” utilizando este EDI, los pasos a seguir se indican a continuación:

1. Suponiendo que ya se está visualizando el entorno de desarrollo, ejecute la orden *Archivo > Proyecto Nuevo*. Se muestra la ventana *Proyecto Nuevo*.
2. Seleccione *Java* en la lista *Categorías* y en la lista *Proyectos* seleccione *Java Application* (Aplicación Java). Después haga clic en el botón *Siguiente*. Se muestra la ventana *Nueva Aplicación Java*.
3. Escriba el nombre del proyecto; en nuestro caso será *HolaMundo* y, a continuación, seleccione la carpeta donde quiere guardarlo.
4. Asegúrese de que las casillas *Configurar como proyecto principal* y *Crear clase principal* están marcadas.
5. Observe la caja de texto correspondiente al nombre de la clase principal; muestra *holamundo.Main*. Esto significa que la clase principal se llama *Main* y que pertenece al paquete *holamundo*. Asumiremos el paquete por omisión, por lo que en esta caja escribiremos solamente el nombre de la clase; en nuestro caso se llamará *HolaMundo*.



6. Para finalizar haga clic en el botón *Terminar*. El resultado será el siguiente:



El EDI crea la carpeta *Ejemplos\Apen_B\HolaMundo* en la que guardará el proyecto compuesto en este caso por un solo fichero, *HolaMundo.java*, que almacena el código correspondiente a la clase *HolaMundo*.

En la ventana mostrada en la figura anterior distinguimos otras tres ventanas, algunas de ellas, con varios paneles. La que está en la parte superior derecha está mostrando el panel de edición para el código fuente de nuestra aplicación y tiene

oculto el panel de inicio. La que está en la parte superior izquierda muestra el panel de proyectos; éste lista el nombre del proyecto y el nombre de los ficheros que componen el proyecto. Observe el fichero *HolaMundo.java*; contiene el código de las acciones que tiene que llevar a cabo nuestra aplicación. También distinguimos un elemento *Bibliotecas* que hace referencia a las bibliotecas que pueden ser necesarias para compilar la aplicación. Finalmente, la ventana que hay debajo de la de proyectos permite navegar por el código del proyecto. Puede visualizar otras ventanas desde el menú *Ventana*; por ejemplo, la ventana *Salida*, que será utilizada para mostrar los resultados de la compilación y de la ejecución.

Una vez creado el esqueleto de la aplicación, editamos el código de la misma. En nuestro caso, simplemente hay que completar el método *main* como se indica a continuación:

```
public static void main(String[] args)
{
    System.out.println("Hola mundo!!!");
}
```

El paso siguiente es construir el fichero ejecutable (fichero *HolaMundo.class*). Para ello, ejecute la orden *Ejecutar > Generar Proyecto Principal*, o bien pulse la tecla *F11*. Si la compilación es correcta, puede pasar a ejecutar la aplicación ejecutando la orden *Ejecutar > Ejecutar Proyecto Principal*, o bien pulsando la tecla *F6*; observe el resultado en la ventana *Salida*.

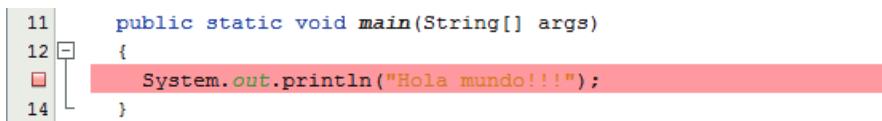
Cuando la aplicación necesite más de un fichero, el proceso es igual de sencillo. Añadir otro fichero a una aplicación, por ejemplo un nuevo fichero que almacene una nueva clase, supone hacer clic con el botón secundario del ratón sobre el nombre del proyecto, elegir la orden *Nuevo* y seleccionar del menú contextual que se visualiza el tipo de elemento que se desea añadir.

DEPURAR UNA APLICACIÓN CON NETBEANS

¿Por qué se depura una aplicación? Porque los resultados que estamos obteniendo con la misma no son correctos y no sabemos por qué. El proceso de depuración consiste en ejecutar la aplicación paso a paso, indistintamente por sentencias o por métodos, con el fin de observar el flujo seguido durante su ejecución, así como los resultados intermedios que se van sucediendo, con la finalidad de detectar las anomalías que producen un resultado final erróneo.

Por ejemplo, para depurar una aplicación utilizando el depurador del entorno de desarrollo *NetBeans*, debe establecer un punto de parada inicial. Para ello, haga clic con el botón secundario del ratón sobre la sentencia a partir de la cual quiere ejecutar el código de su aplicación paso a paso y ejecute la orden *Ocultar/Mostrar*

línea de punto de interrupción del menú contextual que se visualiza, o haga clic en la zona sombreada a su izquierda:



```

11
12 ┌─
13 └─ System.out.println("Hola mundo!!!!");
14

```

Después, ejecute la orden *Depurar > Depurar Proyecto Principal*, o bien pulse la tecla *Ctrl+F5* para iniciar la depuración. Continúe la ejecución paso a paso utilizando las órdenes del menú *Depurar* o los botones correspondientes de la barra de herramientas *Depurar* (para saber el significado de cada botón, ponga el puntero del ratón sobre cada uno de ellos).



De forma resumida, las órdenes disponibles para depurar una aplicación son las siguientes:

- *Depurar Proyecto Principal* o *Ctrl+F5*. Inicia la ejecución de la aplicación en modo depuración hasta encontrar un punto de parada o hasta el final si no hay puntos de parada.
- *Ocultar/Mostrar línea de punto de interrupción* o *Ctrl+F8*. Pone o quita un punto de parada en la línea sobre la que está el punto de inserción.
- *Finalizar sesión del depurador* o *Mayús+F5*. Detiene el proceso de depuración.
- *Paso a paso* o *F7*. Ejecuta la aplicación paso a paso. Si la línea a ejecutar coincide con una llamada a un método definido por el usuario, dicho método también se ejecuta paso a paso.
- *Continuar ejecución* o *F8*. Ejecuta la aplicación paso a paso. Si la línea a ejecutar coincide con una llamada a un método definido por el usuario, dicho método no se ejecuta paso a paso, sino de una sola vez.
- *Ejecutar y salir* o *Ctrl+F7*. Cuando un método definido por el usuario ha sido invocado para ejecutarse paso a paso, utilizando esta orden se puede finalizar su ejecución en un solo paso.
- *Ejecutar hasta el cursor* o *F4*. Ejecuta el código que hay entre la última línea ejecutada y la línea donde se encuentra el punto de inserción.

Para ver los valores intermedios que van tomando las variables ponga el cursor sobre ellas, o bien utilice las ventanas *Watches*, *Local Variables*, etc., del fondo del EDI. Para añadir o quitar ventanas ejecute la orden *Window > Debuggin*.

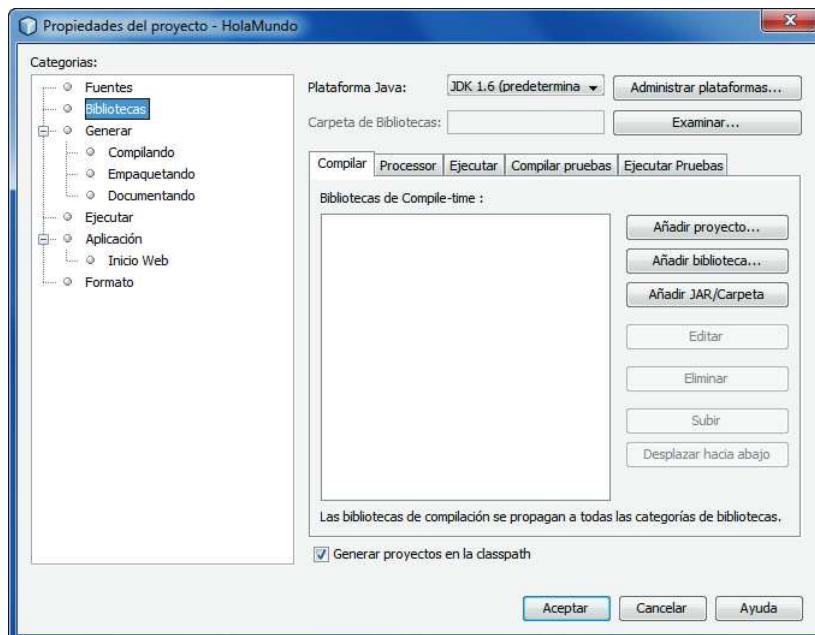


VARIABLE CLASSPATH

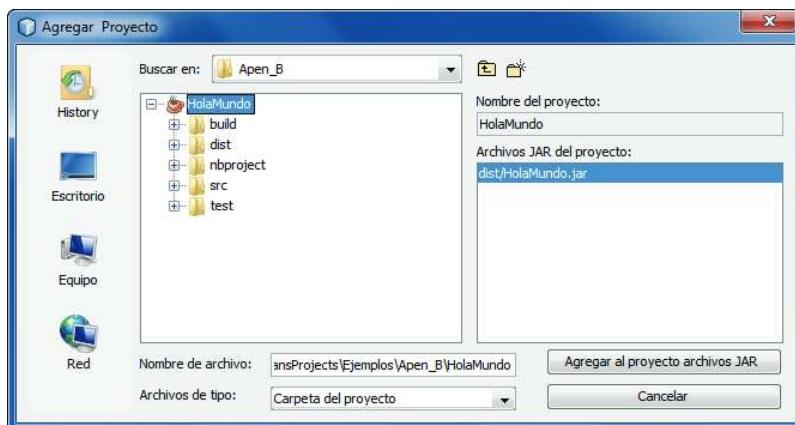
La opción **-classpath** del compilador debe incluir las rutas de todas las carpetas donde se deben buscar las clases necesarias para compilar una aplicación. Algunas de estas clases podrían, incluso, encontrarse empaquetadas en un fichero *.jar*.

Cuando necesite especificar estas rutas:

- Diríjase al panel del proyecto, haga clic con el botón secundario del ratón sobre el nombre del mismo y seleccione la orden *Propiedades* del menú contextual que se visualiza. Se muestra el diálogo siguiente:



2. Seleccione el nodo *Bibliotecas*. Haga clic en la pestaña *Compilar* y después en el botón *Añadir proyecto*.
3. Seleccione la carpeta correspondiente al proyecto y, en el diálogo que se visualiza, observe la lista *Archivos JAR del proyecto*; muestra los ficheros JAR que pueden ser añadidos al proyecto. Observe que se muestra también el fichero JAR correspondiente a nuestra aplicación. Este fichero se crea una vez que hayamos compilado y ejecutado el proyecto.



4. Una vez seleccionado el fichero que desea añadir, haga clic en el botón *Agregar al proyecto archivos JAR* y, finalmente, cierre el diálogo. El fichero JAR seleccionado será añadido.

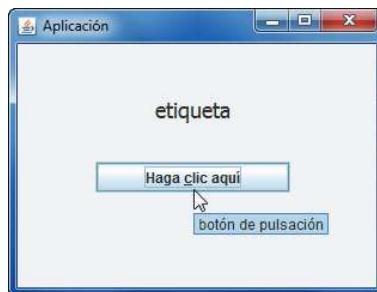
Desde la línea de órdenes esta variable se especificaría análogamente a como indica el ejemplo siguiente:

```
set classpath=%classpath%;;c:\Java\ejemplos;c:\lib\milib.jar
```

DISEÑO DE UNA APLICACIÓN CON INTERFAZ GRÁFICA

Para implementar y ejecutar una aplicación que muestre una interfaz gráfica como la de la figura mostrada a continuación, utilizando el entorno de desarrollo integrado *NetBeans*, los pasos a seguir son los siguientes:

1. Suponiendo que ya está visualizado el entorno de desarrollo, ejecute la orden *Archivo > Proyecto Nuevo*. Se muestra la ventana *Proyecto Nuevo*.



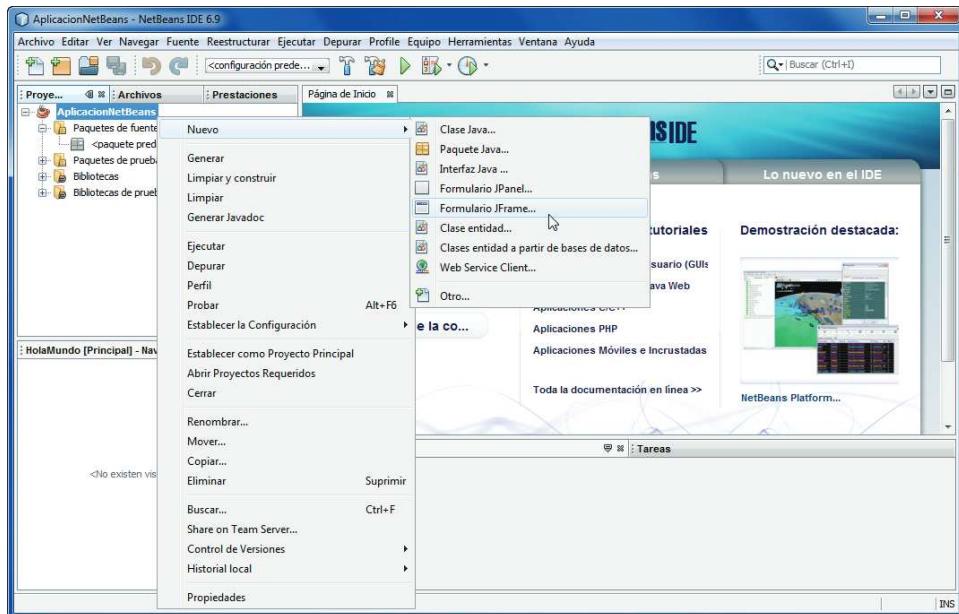
2. Seleccione *Java* en la lista *Categorías*, y en la lista *Proyectos* seleccione *Java Application* (Aplicación Java). Después, haga clic en el botón *Siguiente*. Se muestra la ventana *Nueva Aplicación Java*.
3. Escriba el nombre del proyecto; en nuestro caso será *AplicacionNetBeans*; y, a continuación, seleccione la carpeta donde quiere guardarla.
4. No marque la opción *Crear clase principal*. De esta forma se creará un proyecto vacío. Si marca esta opción el nombre de la clase será el último especificado y los anteriores, separados por puntos, darán lugar al nombre del paquete al que pertenecerá la clase.



5. Para finalizar, haga clic en el botón *Terminar*. El EDI crea la carpeta *Ejemplos\Apen_B\AplicacionNetBeans* en la que guardará el proyecto.

Una vez creado un proyecto vacío, el paso siguiente consistirá en añadir una nueva clase derivada de **JFrame** que nos sirva como contenedor de la interfaz gráfica (en el capítulo 18 se explican los contenedores).

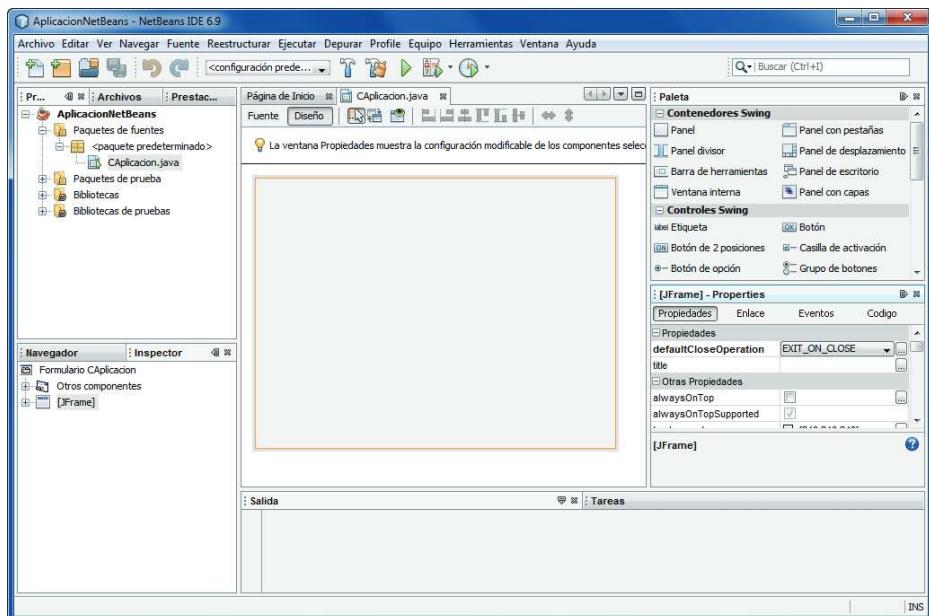
Para añadir una ventana marco (objeto **JFrame**) al proyecto, haga clic sobre el nombre del mismo, utilizando el botón secundario del ratón, y seleccione del menú contextual que se visualiza la orden *Nuevo > Formulario JFrame*.



Se visualizará una ventana en la que puede elegir el nombre para la nueva clase de objetos que vamos a añadir. En nuestro caso, elegiremos como nombre, por ejemplo, *CAplicacion*.



Para continuar, haga clic en el botón *Terminar*. Se visualizará la ventana mostrada a continuación, en la que observamos que la clase *Caplicacion* almacenada en el fichero *Caplicacion.java* es la que da lugar a la ventana marco, también llamada formulario, que se visualiza en el centro de la pantalla (objeto **JFrame**).



Observe también que encima del formulario hay una barra de herramientas con una serie de botones. Los dos primeros (*Fuente* y *Diseño*) le permitirán alternar entre el panel de edición (el que muestra el código fuente de la clase *Caplicacion*) y el panel de diseño (el que se está mostrando).

También, a la derecha del formulario, se observan otras dos ventanas: una muestra varias paletas de herramientas (la que se está mostrando es la paleta de componentes *Swing*; si no se muestra ejecute *Ventana > Paleta*) y la otra está mostrando el panel de propiedades con las propiedades del formulario. La primera le muestra los contenedores y controles que puede seleccionar y colocar sobre el formulario; observe que, además de la paleta de componentes *Swing*, hay otras como *AWT* (kit de herramientas de ventanas abstractas) y *Beans* (componentes reutilizables); y la segunda le permite mostrar y editar las propiedades del componente seleccionado (tamaño, color, fuente...), los manejadores de eventos (botón *Eventos*), etc.

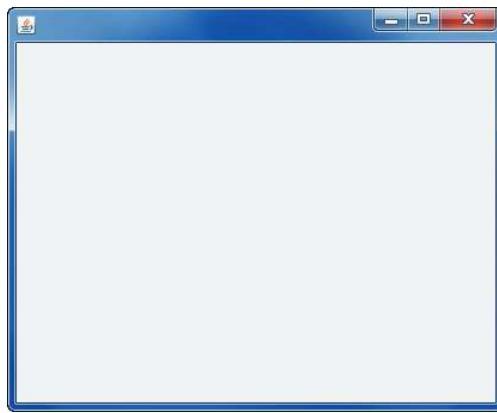
A la izquierda del formulario, debajo del panel del proyecto, hay otra ventana con dos paneles: el supervisor (*Inspector*) y el navegador (*Navegador*). El supervisor permite ver y seleccionar los componentes que forman la interfaz gráfica y

el navegador permite navegar por los componentes software de la aplicación (clases, métodos, etc.).

En el panel de edición de código (clic en el botón *Fuente*) se puede observar que se ha generado una clase *CAplicacion*, con un constructor público y una serie de métodos. En los pasos siguientes añadiremos los componentes al formulario, utilizando el panel de diseño, y el código necesario para que la aplicación realice lo deseado.

Ejecutar la aplicación

Si ahora compilamos y ejecutamos esta aplicación, para lo cual tendremos que elegir la orden *Ejecutar Proyecto Principal (F6)* del menú *Ejecutar*, o bien hacer clic en el botón correspondiente de la barra de herramientas del EDI, aparecerá sobre la pantalla la ventana de la figura mostrada a continuación y podremos actuar sobre cualquiera de sus controles (minimizar, maximizar, mover, ajustar el tamaño, etc.).



Ésta es la parte que *NetBeans* realiza por nosotros y para nosotros; pruébelo. Para finalizar, haga clic en el botón para cerrar la ventana.

Observe en el panel de propiedades que el valor de la propiedad **defaultCloseOperation** es *EXIT_ON_CLOSE*. Esto indica que al hacer clic en el botón la ventana se cerrará y la aplicación finalizará invocando al método *exit*.

Propiedades del formulario

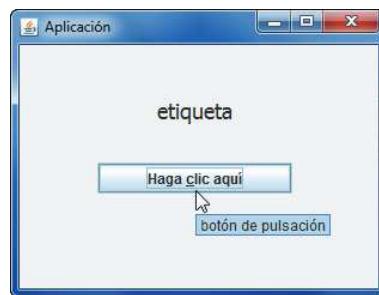
Supongamos ahora que deseamos añadir un título a la ventana y fijar su tamaño inicial. Una forma de hacer esto es establecer el título de la ventana a través del

panel de propiedades del formulario; esto es, estando en el panel de diseño, seleccionamos el formulario, nos dirigimos al panel de propiedades y asignamos a la propiedad **title** el valor “Aplicación”, y para establecer el tamaño del formulario, nos dirigimos a la plantilla de diseño, hacemos doble clic sobre el borde cuando el cursor del ratón toma la forma que permite redimensionar la misma y, en el diálogo que se visualiza, escribimos las dimensiones de la ventana; esta acción asigna a la propiedad **preferredSize** del formulario los valores introducidos, con lo que también podríamos haber procedido modificando directamente el valor de esta propiedad.

Observe también que el constructor invoca a *initComponents* para ejecutar las operaciones de iniciación requeridas para los componentes de la aplicación y que este método ejecuta el método **pack** para ajustar el tamaño de la ventana al tamaño preferido, propiedad **preferredSize**, o al mínimo que permita visualizar todos sus componentes.

Añadir los componentes al contenedor

Nuestro objetivo es diseñar una aplicación que muestre una ventana principal con un botón y una etiqueta:

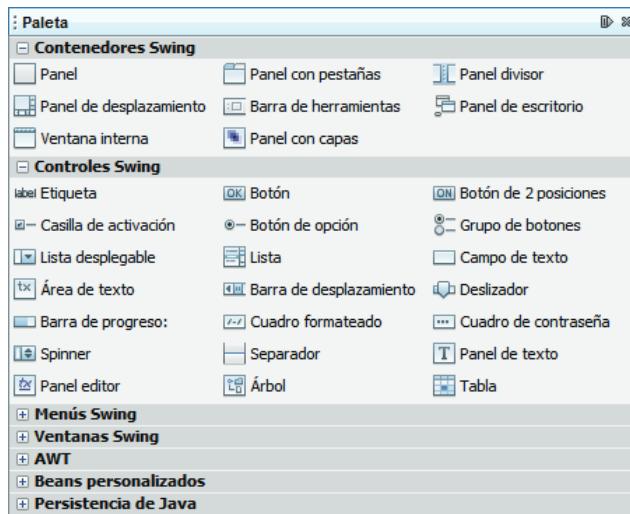


Cuando el usuario haga clic en el botón, la etiqueta mostrará el mensaje “¡¡¡Hola mundo!!!”. La figura anterior muestra el aspecto de esta ventana. En ella se puede observar que el botón puede activarse, además de con un clic del ratón, utilizando las teclas *Alt+c*, y que tiene asociada una breve descripción.

Los componentes, tales como cajas de texto, botones, etiquetas, marcos, listas o temporizadores, son objetos gráficos que permiten introducir o extraer datos. El contenedor más los componentes dan lugar al formulario que hace de interfaz o medio de comunicación con el usuario.

Para añadir un componente a un contenedor, primero visualizaremos el panel de diseño. Para ello, haga clic en el botón *Design*. Después nos dirigiremos a la

paleta de componentes para seleccionar el deseado. La figura siguiente muestra la paleta de componentes *Swing* proporcionada por *NetBeans*:



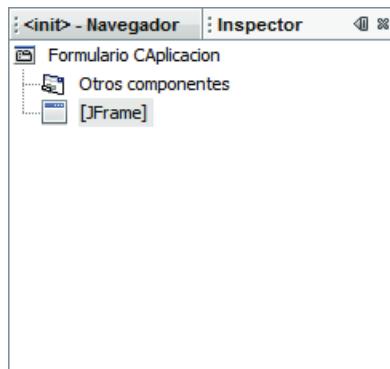
Cada elemento de la paleta crea un único componente. Para saber de qué componente se trata, mueva el ratón encima del componente y espere a que se muestre la descripción asociada. Como podrá observar, estos componentes están clasificados en los siguientes grupos:

- *Swing*. Grupo de componentes de interfaz gráfica de usuario independientes de la plataforma por estar escritos en Java. Estos componentes ofrecen más y mejores funciones que los *AWT*.
- *AWT*. Grupo de componentes de interfaz gráfica de usuario (IGU) que se han implementado utilizando versiones dependientes de la plataforma, sustituido en gran medida por el grupo de componentes *Swing*.
- *Beans*. Componentes software reutilizables en cualquier programa. Mientras que los componentes anteriores son intrínsecos a Java, estos otros existen como ficheros independientes con extensión *.jar*. Se trata de programas Java que nosotros mismos podemos crear con una funcionalidad específica, con la intención de insertarlos posteriormente en otros programas.

Dibujar los componentes

Añadir uno o más de estos componentes a un contenedor implica dos operaciones: seleccionar un administrador de diseño para dicho contenedor y dibujar sobre él los componentes requeridos.

Por ejemplo, volviendo a la aplicación que estamos desarrollando, diríjase al supervisor de componentes (*Inspector*) y observe que el contenedor del formulario (*JFrame*) no tiene asociado un administrador de diseño.



Por omisión, los nuevos formularios creados utilizan un esquema de diseño libre (*FreeDesign*) que permite distribuir los componentes libremente usando unas líneas guía que automáticamente sugieren la alineación y el espaciado óptimo para los mismos, todo esto sin requerir un administrador de diseño de los definidos en Java. Como el diseño libre emplea un modelo de distribución dinámico, siempre que se redimensione el formulario o se modifiquen posiciones la IGU se adaptará para acomodar los cambios sin cambiar las relaciones entre los componentes.

Asignar un administrador de diseño

Un administrador de diseño determina el tamaño y la posición de los componentes dentro de un contenedor. *Swing* proporciona varios administradores de diseño: **FlowLayout**, **GridBagLayout**, **BorderLayout**, **CardLayout**, **GridLayout**, **BoxLayout**, etc.

Para asignar a un contenedor un administrador de diseño específico basta con que:

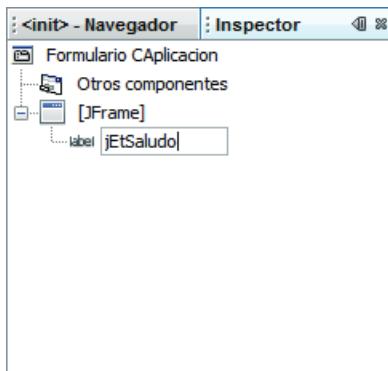
1. Haga clic con el botón secundario del ratón sobre el formulario y seleccione *Activar gestor de distribución* en el menú contextual.
2. Y, a continuación, haga clic en el administrador de diseño deseado.

Nosotros seguiremos utilizando *Diseño libre* por la automatización que aporta a los formularios para recolocar los componentes cuando dichos formularios cambian de tamaño. Otra opción sería utilizar el componente *Diseño nulo*.

Añadir una etiqueta y editar sus propiedades

Para añadir una etiqueta en el contenedor **JFrame**, siga estos pasos:

1. Seleccione en la paleta *Controles Swing*.
2. Haga clic en el componente *Etiqueta (JLabel)* y después diríjase al formulario y haga clic en cualquier parte del contenedor. Ajuste su tamaño y su posición.
3. Cambie su nombre para que sea *jEtSaludo*. ¿Dónde podríamos hacerlo? En el supervisor de componentes (*Inspector*):



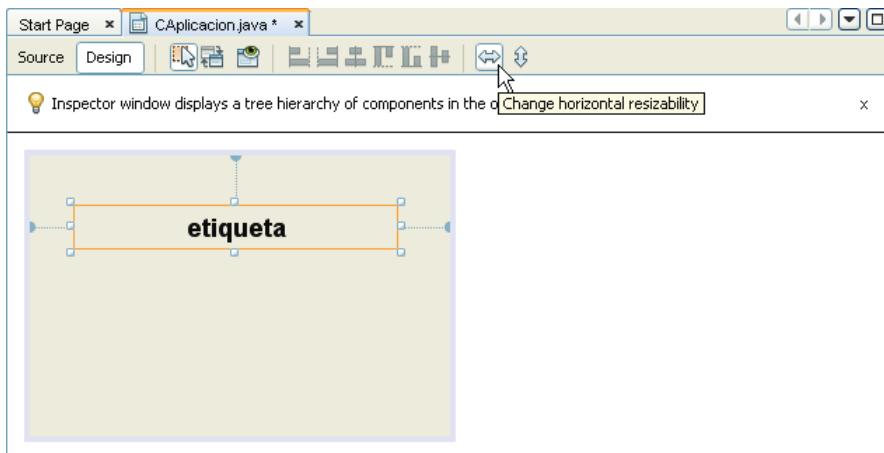
4. Haga que el texto de la etiqueta aparezca centrado, en negrita y de tamaño 18. Para ello, diríjase al formulario y haga clic sobre la etiqueta, o bien diríjase al supervisor de componentes y seleccione el componente *jEtSaludo* para visualizar sus propiedades en la ventana de propiedades.
5. Cambie el valor de la propiedad *horizontalAlignment* a *CENTER*. Observará en el formulario que ahora la etiqueta muestra el texto centrado.
6. Cambie el valor de la propiedad *font*. Para ello, una vez seleccionada la propiedad, haga clic en el botón que hay a la derecha del valor que tiene asignado actualmente para visualizar el diálogo que le permitirá elegir el tipo de fuente, así como su estilo y tamaño.
7. Cambie el valor de la propiedad *text* a “etiqueta”.

El trabajo realizado ha generado en la clase *CAplicacion* un código que ya fue expuesto en el capítulo 18.

Redimensionamiento automático

Un control puede ser anclado arriba y a la izquierda, abajo y a la derecha, etc., seleccionándolo, haciendo clic sobre él con el botón secundario del ratón y eligiendo *Anclaje > Izquierda* (o *Anclaje > Parte inferior*, etc.) del menú contextual. Así

mismo, puede ser redimensionado horizontalmente y/o verticalmente cuando se redimensione el formulario, haciendo clic sobre él con el botón secundario del ratón y eligiendo *Cambio de tamaño automático > Horizontal* (o *Cambio de tamaño automático > Vertical*) del menú contextual; también puede utilizar los botones equivalentes de la barra de herramientas.



Añadir un botón y editar sus propiedades

Para añadir un botón, puede repetir los pasos descritos en el apartado anterior, o bien realizar los siguientes:

1. Seleccione en la paleta *Controles Swing*.
2. Haga clic en el componente *JButton* y después diríjase al formulario y haga clic en cualquier parte del panel. Ajuste su tamaño y su posición.
3. Cambie su nombre para que sea *jBtSaludo*.
4. Modifique su propiedad **text** para que muestre el título “Haga clic aquí”.
5. Modifique su propiedad **toolTipText** para que muestre el mensaje “botón de pulsación”.
6. Modifique su propiedad **mnemonic** para asociarle la tecla de acceso *c*.

Asignar manejadores de eventos a un objeto

Considere el botón *jBtSaludo* que anteriormente añadimos a la interfaz gráfica de *CAplicacion*. Cada vez que el usuario haga clic sobre este botón, se generará un evento de acción que podrá ser recogido por un manejador de este tipo de eventos, si es que tiene uno asociado. La respuesta será mostrar en la etiqueta *jEtSaludo* el

mensaje “¡¡¡Hola mundo!!!”. Para ello, tiene que existir una conexión entre el botón y la etiqueta, lo que se traducirá en la ejecución de un método que asigne esa cadena de caracteres a la etiqueta como respuesta al evento clic.

Para facilitar la realización del tipo de conexiones al que nos hemos referido, *NetBeans* proporciona un asistente para conexiones, el cual permite añadir el código que un componente tiene que ejecutar para responder al mensaje que otro componente le ha enviado, y todo ello con muy poca intervención del desarrollador. Para utilizar este asistente, debe realizar los pasos indicados a continuación:

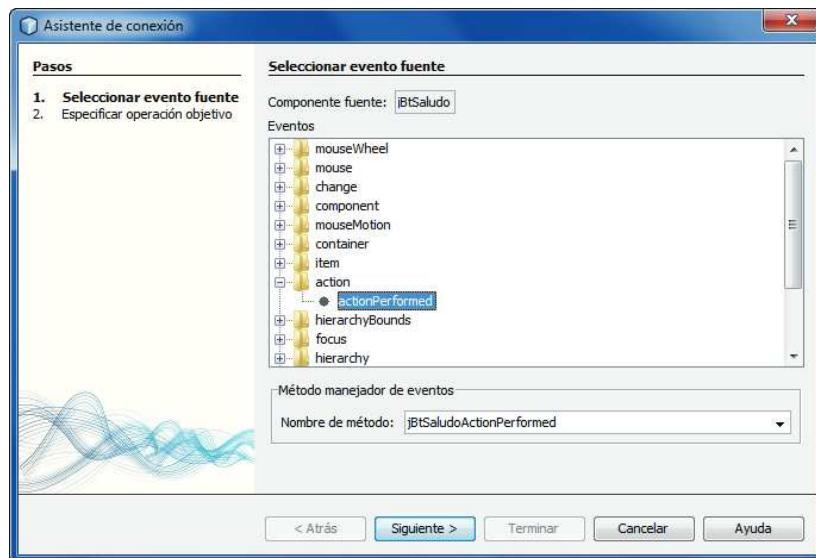
1. Diríjase a la ventana de diseño. Observe los botones que hay en su parte superior.
2. Cambie al modo de conexión haciendo clic en el botón *Modo Conexión*:



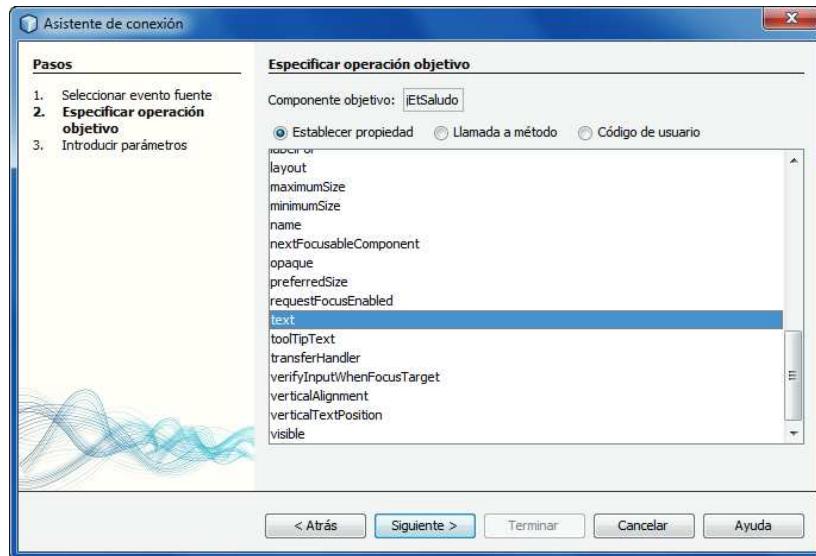
3. A continuación, haga clic primero en el componente sobre el que ocurrirá el evento (en el ejemplo, sobre el botón) y después en el componente sobre el que se realizará la operación (en el ejemplo, sobre la etiqueta). Puede hacer las operaciones indicadas directamente sobre los componentes del formulario, o bien sobre el supervisor de componentes.

Cuando haya ejecutado este punto, *NetBeans* abre el asistente que le permitirá realizar la conexión en dos o tres pasos.

4. Seleccione el evento del componente origen (*Seleccionar evento fuente*). En este primer paso, el diálogo que se muestra permitirá seleccionar el evento que se desea manejar del componente seleccionado y el nombre del método que responderá a ese evento.



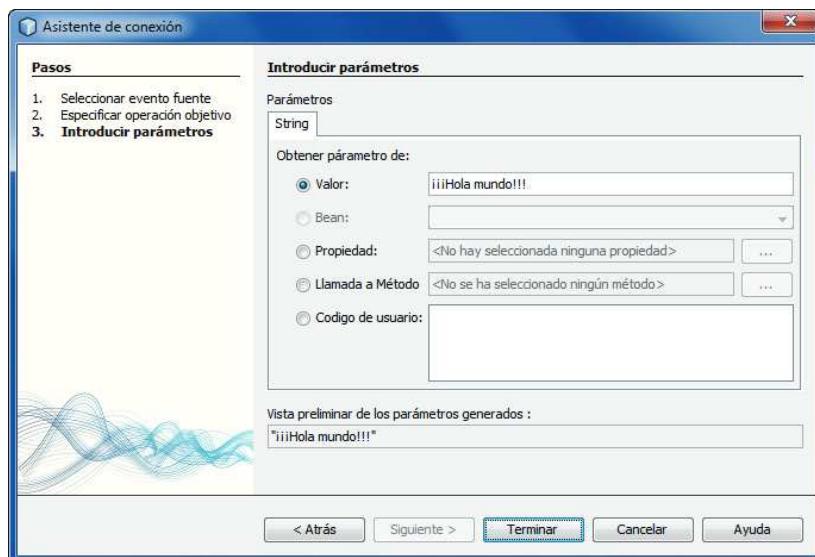
- Haga clic en el botón *Siguiente >* para especificar la operación en el destino (*Especificar operación objetivo*). En este segundo paso, el diálogo que se muestra permite especificar la operación que tiene que realizarse en el componente seleccionado como destino de la misma (*Establecer propiedad > text*):



- Para especificar esta operación, disponemos de varias opciones para establecer el valor: directamente, por medio de un *bean*, de una propiedad o de una llamada a un método, o escribir código.

- Si elige establecer el valor de una propiedad, le serán mostradas todas las propiedades del componente destino de la conexión. Seleccione una y en el siguiente paso establezca su valor.
- Si elige llamar a un método, le serán mostrados los posibles métodos a los que puede invocar. Seleccione uno y especifique los parámetros en el siguiente paso.
- Si elige escribir código, se añadirá al programa el método que debe responder al evento seleccionado, pero sin código (en este caso no hay un tercer paso).

En nuestro ejemplo elegiremos establecer el valor directamente, aunque también podría valer la opción de escribir código.



El resultado será que a *CAplicacion* se añade el código siguiente:

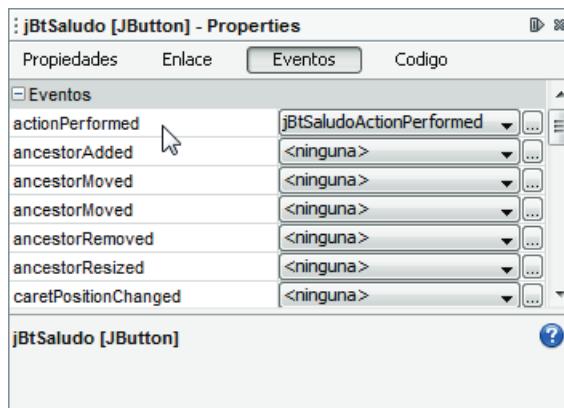
```
jBtSaludo.addActionListener(new java.awt.event.ActionListener()
{
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        jBtSaludoActionPerformed(evt);
    }
});

// ...

private void jBtSaludoActionPerformed(java.awt.event.ActionEvent evt)
```

```
jEtSaludo.setText("iiiHola mundo!!!");  
}
```

Este manejador también podemos añadirlo seleccionando el botón *jBtSaludo*, el panel *Eventos* en la ventana de propiedades, el evento *actionPerformed* en este panel al que, finalmente, asignaremos el nombre del método que debe responder a tal evento:



Eliminar un método añadido por el asistente

Si quisiéramos eliminar el método *jBtSaludoActionPerformed* añadido anteriormente, seleccionaríamos el botón *JBtSaludo*, el panel *Eventos* en la ventana de propiedades, el nombre del método asignado al evento *actionPerformed*, pulsaríamos la tecla *Del*, para borrar el valor actual, y finalizaríamos esta operación pulsando la tecla *Entrar*.

Añadir otro código

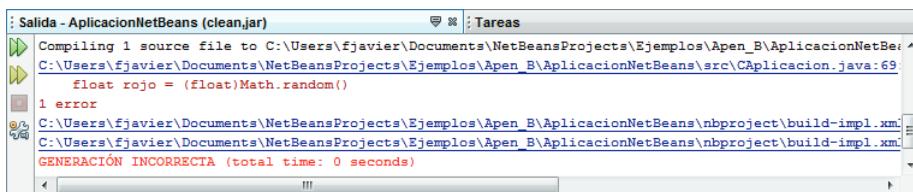
Para finalizar la aplicación que estamos escribiendo, diríjase a la ventana de edición y complete el método *jBtSaludoActionPerformed* como se indica a continuación con el fin de cambiar de forma aleatoria el color del texto:

```
private void jBtSaludoActionPerformed(java.awt.event.ActionEvent evt)  
{  
    float rojo = (float)Math.random();  
    float verde = (float)Math.random();  
    float azul = (float)Math.random();  
    jEtSaludo.setForeground(new java.awt.Color(rojo, verde, azul));  
    jEtSaludo.setText("iiiHola mundo!!!");  
}
```

Compilar la aplicación

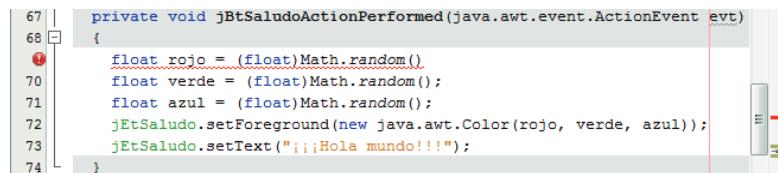
Una vez finalizada la aplicación, puede compilarla y ejecutarla. Para compilar la aplicación, ejecute la orden *Generar proyecto principal (F11)* del menú *Ejecutar*.

Si la construcción del fichero *.class* resulta satisfactoria, verá en la ventana de salida del compilador el mensaje “*GENERACIÓN CORRECTA*”. Si hay problemas con la construcción, verá los mensajes de error correspondientes en la misma ventana. Por ejemplo, suponga que en la primera línea de código del método *jBtSaludoActionPerformed* olvidó el punto y coma final. Cuando ejecute la orden *Build*, le será mostrada una ventana como la siguiente:



La ventana de la figura anterior indica que el compilador ha detectado que falta un punto y coma. Para ir a la línea de código donde el compilador ha detectado el error y corregirlo, puede hacer clic sobre el mensaje de error. Una vez que obtenga una compilación sin errores, puede ejecutar la aplicación.

No obstante, este error también se hace visible antes de la compilación en la ventana de edición. Observe la línea subrayada en la figura siguiente; es la que contiene el error. Si pone el punto de inserción sobre ella, le será mostrado un mensaje corto indicando el error cometido:

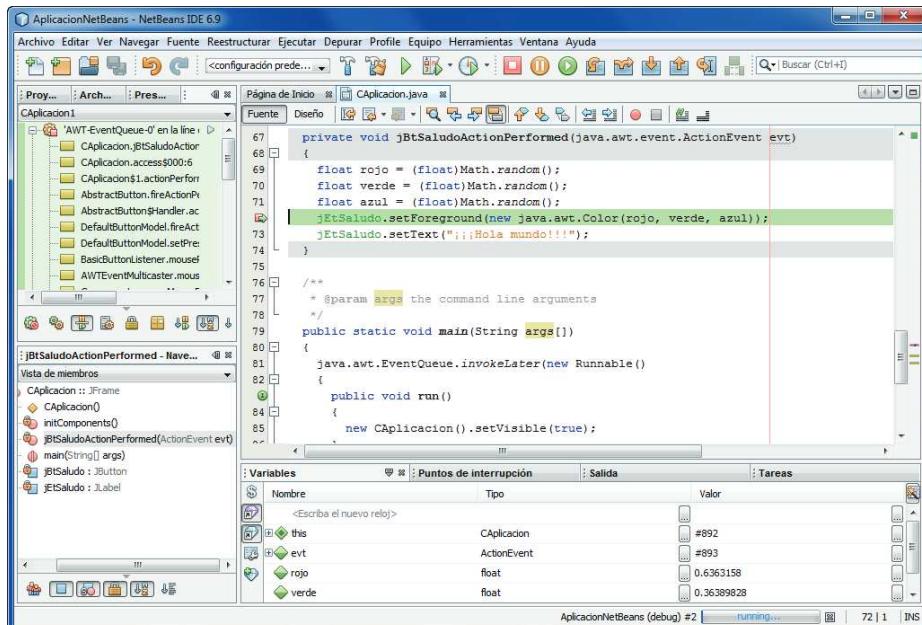


Para ejecutar la aplicación, seleccione la orden *Ejecutar proyecto principal* del menú *Depurar*. Si no hay errores de ejecución, NetBeans mostrará los resultados.

No obstante, cuando la solución obtenida no sea satisfactoria y no seamos capaces de localizar dónde se está produciendo el error (imaginemos una expresión *a/(2*b)* en la que olvidamos poner los paréntesis) podemos utilizar el depurador para ejecutar el programa paso a paso y poder investigar los resultados parciales que se van produciendo a medida que avanza la ejecución.

Depurar la aplicación

Apoyándonos en la aplicación que acabamos de diseñar, enumeramos una serie de pasos que le enseñarán cómo trabajar con el depurador:



- Diríjase a la ventana de edición y haga clic con el botón izquierdo del ratón a la izquierda de la llamada al método `setForeground` que hay en el método `jBtSaludoActionPerformed`, sobre la columna que muestra el número de línea. Ha establecido un punto de parada; esto es, cuando se ejecute la aplicación y haga clic en el botón `jBtSaludo` según indica el punto siguiente, la ejecución será detenida en ese punto. Para quitar el punto de parada, proceda de igual forma.

También puede establecer un punto de parada haciendo clic con el botón secundario del ratón sobre la línea de código y ejecutar la orden *Ocultar/Mostrar línea de interrupción* (*Ctrl+F8*) del menú contextual que se visualiza.

- Ejecute la orden *Depurar proyecto principal* (*Ctrl+F5*) del menú *Depurar*. La ejecución del programa se inicia y cuando se detenga en el punto de parada anteriormente añadido, podrá revisar el espacio de trabajo de depuración compuesto por los paneles *Variables*, *Puntos de interrupción*, etc., que puede observar en la figura anterior.

La depuración puede también iniciarse ejecutando la orden *Paso a paso* (*F7*) del menú *Depurar*. Esta forma de proceder no requiere poner un punto de parada inicial.

En cualquier instante, puede detener la depuración ejecutando la orden *Finalizar sesión del depurador* (*Mayúsculas+F5*) del menú *Depurar*.

3. Puede continuar la ejecución paso a paso por sentencias pulsando la tecla *F7* (*Paso a paso*), paso a paso por métodos pulsando la tecla *F8* (*Continuar ejecución*), hasta la posición del cursor pulsando la tecla *F4* (*Ejecutar hasta el cursor*), hasta que el método actual finalice y el control pase al método que lo invocó pulsando las teclas *Ctrl+F7* (*Ejecutar y salir*), o continuar la ejecución hasta el siguiente punto de parada o hasta el final de la aplicación pulsando la tecla *F5*.

Paso a paso por sentencias significa ejecutar cada método del programa paso a paso. Si no quiere que los métodos invocados por el método actualmente en ejecución se ejecuten sentencia a sentencia, sino de una sola vez, utilice la tecla *F8* en vez de *F7*.

4. Cuando la ejecución está detenida, puede inspeccionar los valores que van tomando las variables del programa. El panel *Elementos observados* (menú *Ventana > Depuración*), entre otros como *Variables*, del espacio de trabajo de depuración será el encargado de presentarnos los valores deseados. Para añadir una variable que aparece en el código fuente a esta lista, haga clic sobre la variable utilizando el botón secundario del ratón, ejecute la orden *Nuevo elemento observado* del menú contextual que se visualiza y pulse el botón *Aceptar* de la ventana que se muestra. Para eliminar una variable del panel de inspección, haga clic sobre ella utilizando el botón secundario del ratón y ejecute la orden *Eliminar* del menú contextual que se visualiza.

Así mismo, en el panel *Variables* puede ver la lista de variables locales, en el panel *Pila de llamadas* la pila de llamadas a los métodos y en el panel *Puntos de interrupción* los puntos de parada.

También, colocando el punto de inserción sobre una variable en una línea de código ya ejecutada, le será mostrada una descripción con el valor de dicha variable.

Como alternativa a las órdenes mencionadas del menú *Depurar* en los puntos anteriores, puede utilizar los botones correspondientes de la barra de herramientas o las teclas de acceso directo.

AÑADIR OTROS FORMULARIOS A LA APLICACIÓN

Normalmente, una aplicación que muestra una interfaz gráfica al usuario despliega una ventana principal y a partir de ella pueden mostrarse otras ventanas de diálogo de alguno de los grupos siguientes:

- *Ventanas de diálogo predefinidas.* Son ventanas de diálogo creadas por medio de los métodos proporcionados por la clase **JOptionPane** de la biblioteca JFC; por ejemplo, **showMessageDialog**.
- *Ventanas de diálogo personalizadas.* Son ventanas de diálogo hechas a medida, para lo cual la biblioteca JFC proporciona la clase **JDialog**.
- *Ventanas de diálogo estándar.* Son ventanas muy comunes; por ejemplo, la ventana de diálogo *Abrir* o *Guardar* proporcionada por la clase **JFileChooser**, o el diálogo *Color* proporcionado por la clase **JColorChooser**.

Para añadir uno de estos diálogos, por ejemplo un **JDialog**, puede optar por alguna de las siguientes opciones:

1. Añada un objeto **JDialog** desde la paleta *Ventanas Swing*. Una vez añadido, diríjase al supervisor de componentes, haga doble clic sobre el nuevo diálogo añadido y complételo de acuerdo a sus necesidades.
2. Añada al proyecto una nueva clase derivada de la clase *Formulario JPanel* (para ello, haga clic con el botón secundario del ratón sobre el nombre del proyecto) y modifique la definición de la misma para que se derive de **JDialog** en lugar de derivarse de **JPanel**. Después, desde cualquier otra parte de su aplicación, podrá crear objetos de la nueva clase añadida.

Si opta por la opción 1, el objeto **JDialog** quedará integrado en su aplicación como cualquier otro control que haya añadido a su ventana principal. En cambio, si opta por la opción 2, se añadirá una nueva clase derivada de **JDialog**.

PROYECTOS

Un proyecto permite agrupar los ficheros requeridos para producir una aplicación o un *applet*. Esto presenta ventajas como poder compilar todo el proyecto sin tener que especificar los ficheros que incluye, especificar la clase principal del proyecto, ver bajo la pestaña *Archivos* del explorador todos los ficheros que componen el proyecto, configurar el entorno de desarrollo integrado para cada proyecto, etc. De esto se deduce que para un determinado proyecto podemos configurar un escenario particular que será guardado cuando se finalice la sesión, lo

que permitirá recuperarlo automáticamente la próxima vez que se cargue ese proyecto.

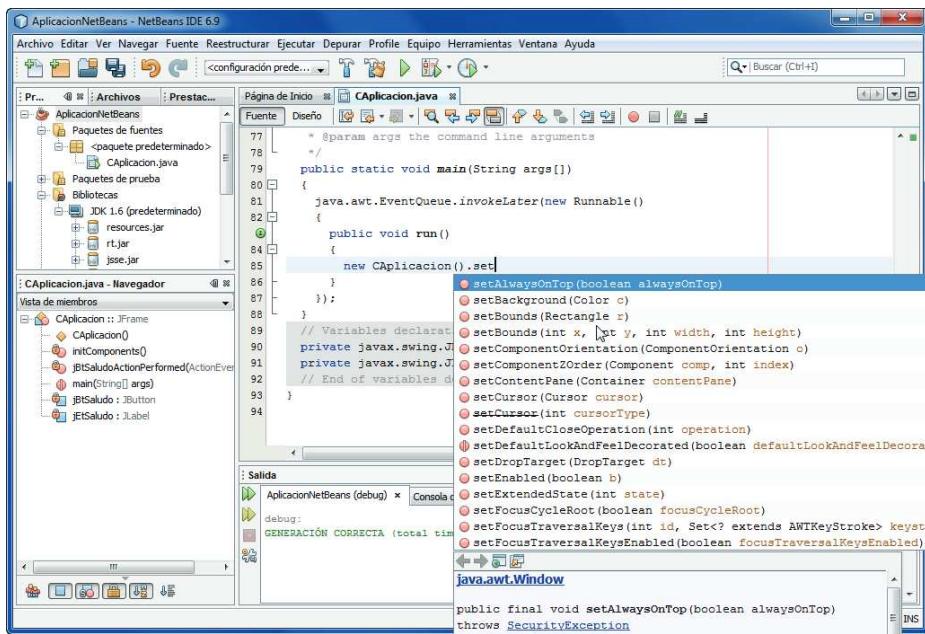


Puede tener cargados varios proyectos simultáneamente y activar en cada instante aquél sobre el que desea trabajar. Para ello, tiene que hacer clic sobre el nombre del proyecto que desea sea el actual y ejecutar la orden *Establecer como proyecto principal* del menú contextual que se visualiza.

De forma predeterminada la codificación de los proyectos *NetBeans* es UTF-8. Si utiliza variables o constantes que incluyen vocales acentuadas y otras letras especiales del idioma español, debe especificar que se utilice el juego de caracteres de ISO-8859-1, conocido también como Latín 1. Para ello, haga clic con el botón secundario del ratón sobre el nombre del proyecto y especifique este código en *Propiedades > Fuentes > Codificación*.

COMPLETAR EL CÓDIGO MIENTRAS SE ESCRIBE

NetBeans proporciona la característica de completar el código mientras lo escribe:

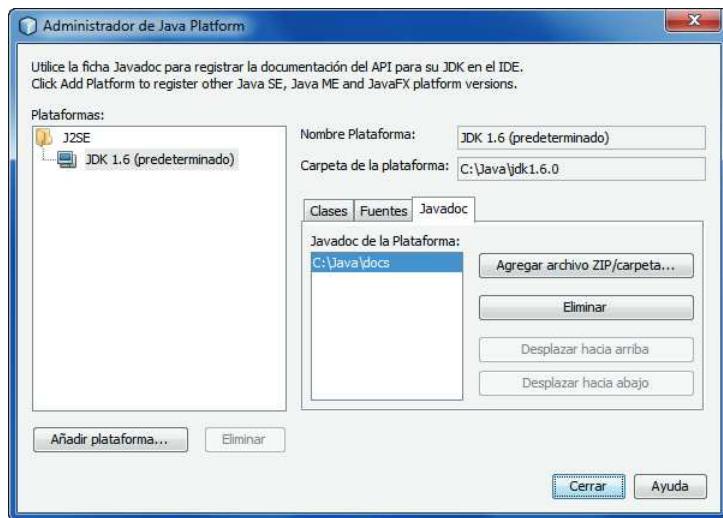


Por ejemplo, según puede verse en la figura anterior, cuando haya escrito *CAplicacion()*. aparecerá una lista de los posibles métodos que pueden ser invocados; seleccione el adecuado y pulse la tecla *Entrar*.

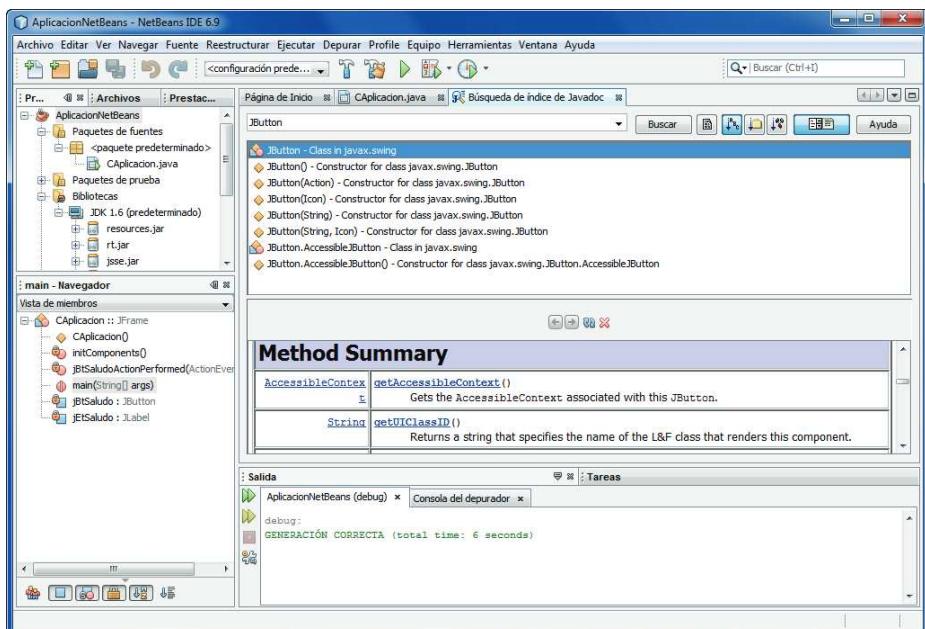
Este tipo de ayuda sólo estará disponible si el valor de la casilla de verificación *Auto Popup Completion Window* tiene el valor *true*. Para verlo, ejecute la orden *Opciones* del menú *Herramientas* y haga clic en el botón *Editor > Finalización de código*.

OBTENER AYUDA

La orden *Búsqueda de índice de Javadoc* del menú *Ayuda (Mayúsculas+F1)* permite obtener ayuda acerca de múltiples temas. Pero si quiere obtener ayuda acerca de la biblioteca de clases de Java (suponiendo que la ha instalado), es preciso que dicho entorno tenga conocimiento de la ruta de acceso a la misma. Para ello, ejecute la orden *Plataformas Java* del menú *Herramientas* y asegúrese de que en la lista de rutas mostrada en el panel *Javadoc* hay una que hace referencia a la carpeta donde se encuentra la ayuda mencionada; si no es así, añádala.

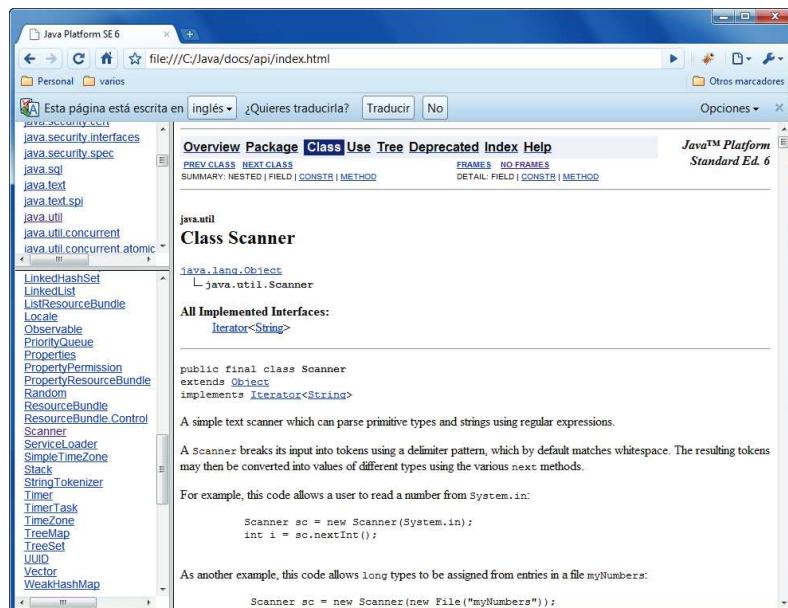
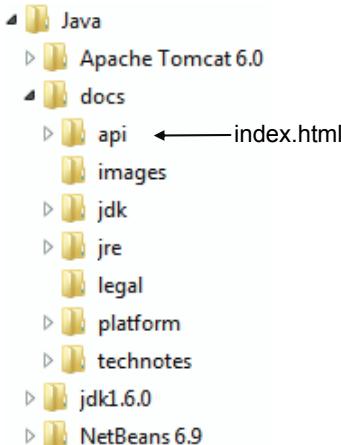


Después del proceso anterior, cuando pulse las teclas *Mayúsculas+F1*, se visualizará el panel que se muestra en la figura siguiente, donde podrá solicitar ayuda acerca de la clase que quiera. También puede dirigirse al editor de código, colocar el cursor sobre el nombre de la clase, método, etc., de la cual quiere ayuda y pulsar las teclas *Mayúsculas+F1*.



La forma más directa de visualizar la ventana de ayuda a partir de la cual podremos navegar por la documentación de la API de Java es haciendo doble clic en

el fichero *index.html* localizado en la carpeta *api* de *docs*. Si observa esa ventana, verá que se trata del explorador, puesto que estamos tratando con documentos *html*, y que su área de trabajo se divide en tres partes:



- En la esquina superior izquierda de este área se muestra la lista de los paquetes que componen la biblioteca de Java. Seleccione el paquete que le interese. Por ejemplo, **java.util** si quiere consultar algo acerca de la entrada/salida por consola.

- Cuando haya seleccionado un paquete, las clases que lo componen aparecen en el área que está justamente debajo. Seleccione la clase que le interese. Por ejemplo, **Scanner** si quiere consultar los atributos y métodos de entrada/salida que proporciona esta clase.
- En el resto del área de trabajo de la ventana, la parte más grande, se mostrará de forma resumida todo lo relativo a la clase que ha seleccionado.

Así mismo, puede obtener ayuda acerca del desarrollo de aplicaciones y del EDI pulsando la tecla *F1*, o bien ejecutando la orden *Ayuda Contenidos* del menú *Ayuda*.

CREAR UN APPLET

Para ejecutar un *applet* en una aplicación Web, hay que crearlo separadamente como un proyecto de tipo *Class Library* (biblioteca de clases) y empaquetar después el fichero JAR del *applet* en la aplicación Web.

Para crear un applet desde el entorno de desarrollo *NetBeans* siga los pasos indicados a continuación:

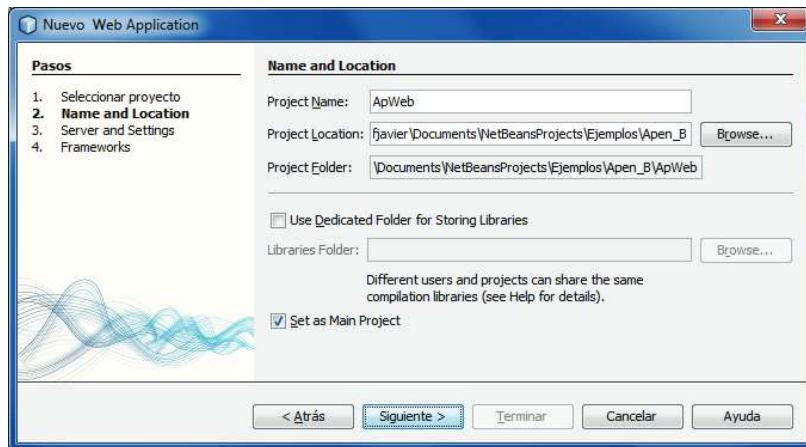
1. Seleccione *Archivo > Proyecto Nuevo > Java > Java Class Library*.
2. Escriba el nombre del proyecto y seleccione la carpeta donde desea almacenarlo.
3. Una vez creado el proyecto, haga clic sobre el nombre del mismo utilizando el botón secundario del ratón y seleccione *Nuevo > Otro... > Java > JApplet*.
4. Escriba el nombre del *applet* y seleccione la carpeta donde desea almacenarlo.
5. Si es necesario, añada una página *html* para desplegarlo. Para ello, haga clic sobre el nombre del proyecto utilizando el botón secundario del ratón y seleccione *Nuevo > Otro... > Otro > Archivo HTML*.

APLICACIÓN JSP-SERVLET

En el capítulo 18 explicamos cómo desarrollar aplicaciones para Internet utilizando *servlets*. *NetBeans* también permite este tipo de desarrollos.

Como ejemplo de desarrollo de una aplicación para Internet con *NetBeans*, vamos a reproducir la misma aplicación que desarrollamos en el capítulo 18 que permitía a un alumno concertar una tutoría (*cap18\Servlets*).

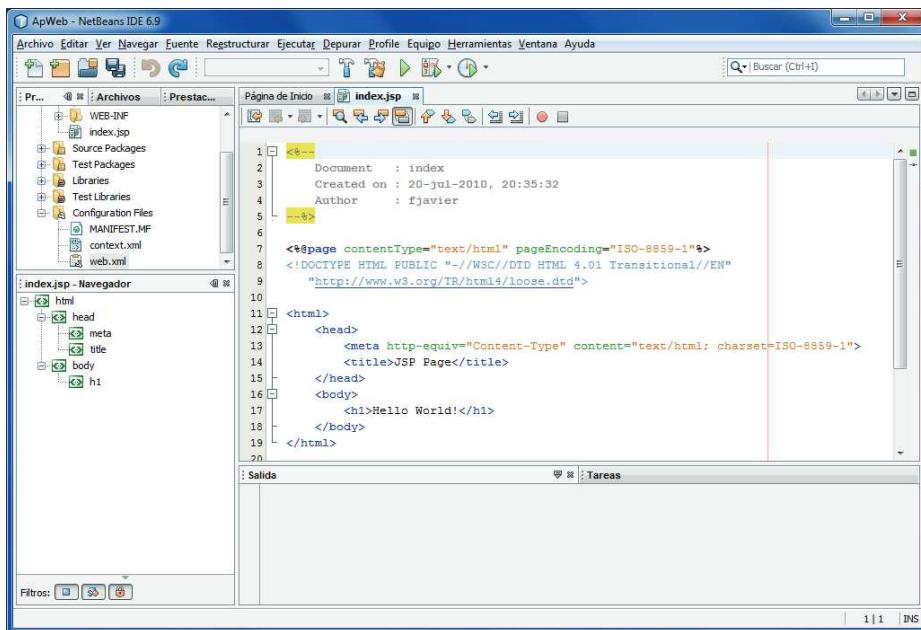
Inicie una nueva aplicación: *Archivo > Proyecto Nuevo > Java Web > Web Application*.



Haga clic en *Siguiente*. Se visualizará la ventana siguiente:



Haga clic en *Siguiente* y después en *Terminar*.



Ya tiene creado el módulo Web. Escriba entre las etiquetas `<html>` y `</html>` del fichero `index.jsp` el contenido del fichero `tutorias.html`. Así, este fichero pasa a llamarse ahora `index.jsp`.

Cuando edite el fichero `index.jsp`, asegúrese de que el URL especificado por el atributo **action** de **form** es:

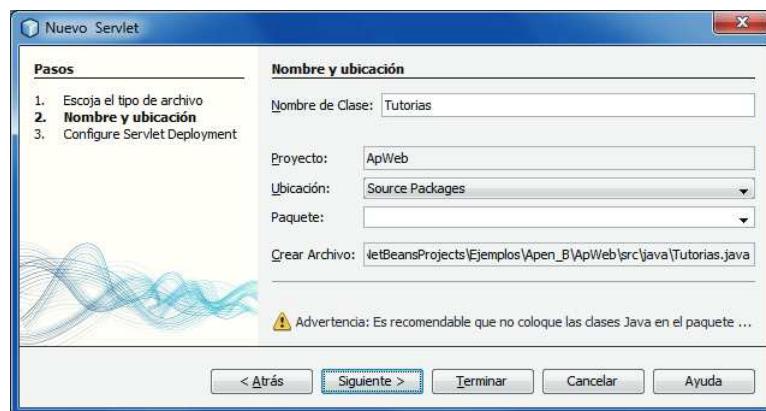
```
<form action=" http://localhost:8084/ApWeb/Tutorias"
```

Si utiliza variables o constantes que incluyen vocales acentuadas y otras letras especiales del idioma español, debe especificar que se utilice el juego de caracteres de ISO-8859-1 conocido también como Latín 1:

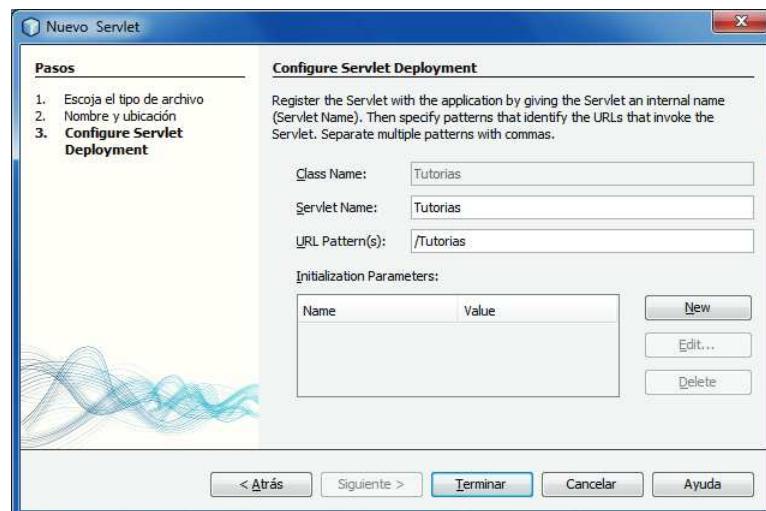
```
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
```

Para añadir a la aplicación un *servlet* o cualquier otro elemento, los pasos siempre son los mismos: clic con el botón secundario del ratón sobre la carpeta donde se va a guardar el elemento, y seleccionar *Nuevo > elemento a añadir*.

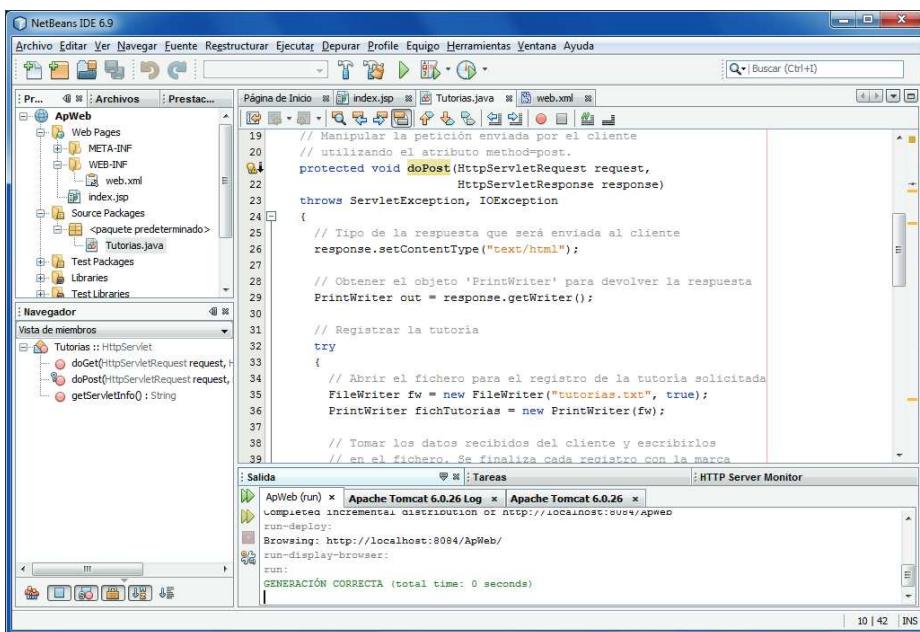
Así, para añadir el *servlet* *Tutorias*, haga clic con el botón secundario del ratón sobre el nombre *ApWeb* del proyecto, y seleccione *Nuevo > Servlet*. En la ventana que se visualiza, introduzca el nombre del fichero *Tutorias*.



Pulse el botón *Siguiente*.



Pulse el botón *Terminar* y escriba el código del fichero.



Finalmente, edite el fichero *web.xml* almacenado en la carpeta *WEB-INF* y eche una ojeada a las distintas secciones del mismo.

Compile la aplicación y ejecútela.

APÉNDICE C

© F.J.Ceballos/RA-MA

PLATAFORMAS UNIX/LINUX

Aunque este libro se ha escrito sobre una plataforma Windows, todos los ejercicios han sido probados también en una plataforma Linux; concretamente en un ordenador personal con el sistema operativo Linux. Quiere esto decir, que si su caso es éste último, recurra a la dirección de Internet <http://www.oracle.com> de la web de *Oracle*, para obtener el *JDK* y el EDI *NetBeans* que debe instalar. Por ejemplo, puede descargar de la web mencionada el paquete *NetBeans+J2SEDK (JDK 6 Update 21 with NetBeans 6.9.1 o una versión más actualizada)* y seguir el libro normalmente.

INSTALACIÓN DE J2SEDK MÁS NETBEANS

Para instalar el entorno de desarrollo integrado *NetBeans* más el *J2SEDK* desde el fichero *jdk-6u21-nb-6_9_1-linux-m1.sh* (o desde el *.sh* que usted haya descargado), realice los siguientes pasos:

1. Cambie a *root* (administrador): `$ su`
2. Sitúese en el directorio donde copió el fichero *.sh*.
3. Ejecute la orden: `# sh jdk-1_5_0_nb-5_0-1linux-m1.sh`
4. La orden anterior inicia el asistente de instalación.
5. Siga los pasos indicados por el asistente de instalación.
6. Salga de la sesión de *root*: `# exit`

La figura siguiente muestra un ejemplo de instalación, la realizada para probar los ejemplos de este libro:



Si a continuación desea instalar la documentación, descomprima (*unzip*) el fichero *jdk-6u21-docs.zip* e instálelo en la carpeta que usted deseé.

Para arrancar *NetBeans* haga clic sobre la orden que se ha añadido al menú correspondiente.

Para compilar y ejecutar un programa desde la línea de órdenes de una consola (un terminal) debe definir las variables de entorno *JDK_HOME* y *PATH*:

```
$ export JDK_HOME=/usr/local/jdk1.6.0_21
$ export PATH=$PATH:$JDK_HOME/bin
```

En el capítulo 6 escribirá la clase *Leer* que después, si lo desea, podrá utilizar en el resto de los capítulos. Para ello, lo más práctico es crear una carpeta *jdk1.6.0_21\misClases* y copiar el fichero *Leer.class* en ella y asegurarse antes de realizar cualquier operación de compilación que la variable *CLASSPATH* tiene asignada la ruta de esta carpeta.

APÉNDICE D

© F.J.Ceballos/RA-MA

FICHEROS JAR

Los ficheros Java (JAR – *Archive Java*) son ficheros comprimidos con el formato ZIP. Típicamente, un fichero JAR contendrá los ficheros de clases y de recursos de *applets* o de aplicaciones. La utilización de este tipo de ficheros proporciona, entre otras, las siguientes ventajas:

- *Seguridad.* Ya que se pueden firmar digitalmente el contenido de un fichero JAR.
- *Disminución del tiempo de descarga.* Si un *applet* está empaquetado en un fichero JAR, los ficheros de clases y recursos asociados pueden ser descargados por el navegador en una sola transacción HTTP sin necesidad de abrir una nueva conexión para cada fichero.
- *Ahorro de espacio.*
- *Portabilidad.* Ya que el mecanismo para manejar los ficheros JAR es un estándar en Java.

HERRAMIENTA JAR DE JAVA

Para trabajar con ficheros JAR, Java proporciona la herramienta *jar*. La sintaxis para invocarla y las opciones que puede utilizar para realizar distintas operaciones (crear, extraer, mostrar, etc.) se pueden ver ejecutando la orden:

`jar`

Las operaciones básicas que se pueden realizar se resumen en la tabla siguiente:

<i>Operación</i>	<i>Sintaxis</i>
Crear un fichero JAR	<code>jar cf f.jar fe1 [fe2]...</code>
Ver el contenido de un fichero JAR	<code>jar tf f.jar</code>
Extraer el contenido de un fichero JAR	<code>jar xf f.jar</code>
Extraer los ficheros especificados	<code>jar xf f.jar fx1 [fx2]...</code>
Ejecutar un <i>applet</i> empaquetado en un fichero JAR	<code><applet code=ClaseApplet.class archive="FichApplet.jar" width=ancho height=alto> </applet></code>

f Representa un fichero JAR.

fe Representa un fichero a empaquetar en un fichero JAR (nombre completo).

fx Representa un fichero empaquetado (nombre completo).

ap Nombre del fichero JAR que empaqueta la aplicación.

c Esta opción indica que se quiere crear un fichero JAR.

f Esta opción indica que a continuación se especifica el nombre del fichero JAR. Si se omite, se asume la salida estándar cuando se trate de almacenar datos en un fichero de salida, o la entrada estándar cuando se trate de tomar datos de un fichero de entrada.

t Esta opción indica que se desea ver la tabla de contenido del fichero JAR.

x Esta opción especifica que se trata de una operación de extracción.

v Mostrar mensajes en la salida estándar acerca de las operaciones realizadas.

EJECUTAR UNA APLICACIÓN EMPAQUETADA EN UN JAR

Según lo expuesto en el apartado anterior, utilizando la herramienta *jar* podemos empaquetar todos los ficheros *.class* que componen una aplicación en un fichero JAR y después ejecutarla. Por ejemplo:

```
jar cf MiAplicacion.jar *.class
```

Cuando se crea un fichero JAR, también se crea un manifiesto por omisión (para observarlo, desempaque el fichero):

META-INF\MANIFEST.MF

Para que el fichero JAR pueda ser ejecutado el manifiesto debe contener una línea que especifique el nombre de la clase principal:

Main-Class: *nombre-clase-principal*

Por lo tanto, tendrá que editar este fichero y añadir esta línea.

Finalmente, para ejecutar la aplicación empaquetada en el fichero JAR, ejecute la orden siguiente:

```
java -jar MiAplicacion.jar
```

Como ejemplo, volvamos a la clase *Caplicacion* que escribimos en el capítulo 1. Observe la primera línea de código; especifica el paquete al que pertenece dicha clase:

```
package Cap01.EstructuraAp;
```

Según esto, *Caplicacion* es una clase ubicada en ...|*Cap01\EstructuraAp*, donde los tres puntos (...) sustituyen al sistema de ficheros al que pertenece *Cap01*; por ejemplo, *c:\java\ejemplos*.

Según lo expuesto, para empaquetar una aplicación ya compilada y ejecutarla desde la línea de órdenes, tendremos que realizar los pasos siguientes:

1. Especificar la ruta donde se localiza el compilador *java*. Por ejemplo:

```
set path=%path%;c:\Java\jdk1.6.0\bin
```

2. Empaquetar la aplicación; por ejemplo en un fichero *EstructuraAp.jar*. Para ello, escribiríamos:

```
cd c:\java\ejemplos
c:\java\ejemplos>jar -cvf Cap01\EstructuraAp\EstructuraAp.jar
Cap01\EstructuraAp\*.class
```

3. Extraer del fichero *EstructuraAp.jar* el manifiesto *MANIFEST.MF*, añadirle la siguiente línea para especificar cuál es la clase principal, se trata de un fichero de texto, y volverlo a empaquetar:

```
Main-Class: Cap01.EstructuraAp.CAplicacion
```

4. Para ejecutar *EstructuraAp.jar* hay que cambiar a la carpeta donde está y ejecutar la orden indicada a continuación:

```
cd c:\java\ejemplos\Cap01\EstructuraAp
```

```
c:\java\ejemplos\Cap01\EstructuraAp>java -jar EstructuraAp.jar
```

El entorno de desarrollo *NetBeans* crea automáticamente en la carpeta *dist* el fichero JAR que empaqueta la aplicación actualmente en desarrollo.

APÉNDICE E

© F.J.Ceballos/RA-MA

JAVA COMPARADO CON C/C++

En el capítulo 1 se dijo que Java está fundamentado en C++. Quiere esto decir que mucha de la sintaxis y diseño orientado a objetos se tomó de este lenguaje. Este apéndice describe las principales diferencias entre Java y C/C++.

TIPOS DE DATOS

Los tipos de datos de Java fueron expuestos en el capítulo 4. Los tipos de datos primitivos de Java tienen tamaños y comportamiento idénticos en cualquier plataforma (Windows, Unix, Solaris, etc.). Esto no ocurre en C/C++, puesto que el tamaño de algunos de sus tipos depende de la plataforma.

En Java no hay tipos de datos sin signo como ocurría en C/C++, excepto el tipo **char**; éste tiene un tamaño de 16 bits y representa un carácter según el código Unicode.

Java aporta el tipo **boolean** que no tiene C pero sí C++ estándar (aquí se denomina **bool**). Las variables de este tipo pueden tomar un valor **true** o **false**; estos valores no se consideran enteros. Esto marca claras diferencias a la hora de escribir una condición cuando se utilizan sentencias de control. Por ejemplo, en C podríamos escribir:

```
int a = 10;  
  
while (a--)  
    printf("%d ", a);  
printf("\n");
```

Pero una condición en una sentencia Java requiere siempre un resultado **boolean**. Por lo tanto tendremos que escribir:

```
int a = 10;
while (a-- > 0)
    System.out.print(a + " ");
System.out.println();
```

Las palabras clave **struct**, **union** y **typedef** del lenguaje C/C++ no se contemplan en Java, ya que los tipos compuestos en Java se obtienen a través de las clases. La palabra clave **enum** se incluyó en el JDK 5.0.

En Java, la conversión implícita entre tipos diferentes de datos está mucho más controlada que en C/C++; sólo ocurrirá cuando no haya pérdida de datos. El resto de las conversiones deben ser explícitas.

Java no soporta la notación funcional de C++ para realizar conversiones explícitas; por ejemplo:

```
fDato = float(dDato);
```

Sólo admite la notación *cast*:

```
fDato = (float)dDato;
```

OPERADORES

Java no contempla el operador coma (,) de C/C++, permite utilizar + para enlazar cadenas e introduce el operador >>> (desplazamiento a la derecha sin signo).

En una operación lógica puede utilizarse tanto el operador **&&** como **&**, pero con un ligero matiz; si se utiliza **&&** (no **&**) y el primer operando es **false**, el segundo operando no es evaluado. Con respecto a **||** y **|** el razonamiento es similar (vea el capítulo 4). Los operadores **&** y **|** también se pueden utilizar en operaciones a nivel de bits.

Las reglas de prioridad y asociatividad de los operadores son iguales que en C/C++. En cambio sí existe un comportamiento diferente con respecto a **new**. En Java, una expresión como:

```
new miClase().método;
```

crea primero el objeto, para poder después acceder al método. Fíjese que hay una llamada a un método que no se puede efectuar sin **new**. La expresión anterior sería equivalente a:

```
(new miClase()).método;
```

A diferencia de C++, Java no soporta la sobrecarga de operadores.

MATRICES

En el capítulo 8 vimos que en Java las matrices son objetos de una clase genérica “matriz”. Cada elemento de una matriz unidimensional es de un tipo primitivo, o bien una referencia a un objeto; y cada elemento de una matriz multidimensional es, a su vez, una referencia a otra matriz (a un objeto).

Los límites de las matrices son estrictamente obligatorios; intentar leer más allá de estos límites supone un error de compilación o de ejecución. Todo ello, marca una clara diferencia con respecto a C/C++.

CADENAS

A diferencia de C, en Java las cadenas de caracteres son objetos. Por lo tanto, los métodos que las manipulan pueden tratarlas como una única entidad. Esto es, no son matrices de caracteres finalizadas de forma predeterminada con el carácter ‘\0’ como ocurre en C/C++, aunque C++ estándar ya incorpora la clase **string**.

PUNTEROS

Java no tiene un tipo de puntero explícito. En lugar de punteros, utiliza referencias. En realidad, los punteros y las referencias son lo mismo, excepto que con las referencias no se puede hacer aritmética de direcciones.

ARGUMENTOS

En Java, a diferencia de C/C++, los objetos son todos pasados por referencia y los argumentos de tipos primitivos son pasados siempre por valor.

El argumento *args[0]* del método **main** en Java es el valor del primer argumento pasado en la línea de órdenes; en C/C++ es el nombre del programa.

ADMINISTRACIÓN DE MEMORIA

La administración de memoria en Java es automática. La memoria se asigna automáticamente cuando se crea un objeto y un recolector de basura se encarga de liberarla cuando ya no exista ninguna referencia a ese objeto.

OTROS

Para definir constantes, Java utiliza la palabra reservada **final** en lugar de **const** que utiliza C/C++.

Java no permite declarar una variable **static** dentro de un procedimiento. En su lugar utiliza atributos **static**, llamados atributos de la clase (no del objeto).

A diferencia de C/C++, Java permite iniciar los atributos de una clase en el momento de su declaración.

En Java no existe la sentencia **goto**.

En Java un método siempre pertenece a una clase. Al trabajar con objetos, no existe la idea de función externa e independiente como ocurre en C/C++.

A diferencia de C/C++, Java no hace distinción entre métodos **inline** y no **inline**. En Java, la decisión de expandir un método es exclusiva del compilador.

Java no admite plantillas (**template**) ni herencia múltiple como C++. No obstante, el trabajo con plantillas en C++ puede resolverse en Java utilizando la clase **Object**, o bien utilizando clases genéricas, según se vio en el capítulo 12.

Las clases en Java no proporcionan un constructor copia como C++ y aunque lo implementamos no es llamado automáticamente en ningún caso.

Java no tiene un preprocesador igual que C/C++.

Del mismo autor

- Curso de programación con **PASCAL**
ISBN: 978-84-86381-36-3
224 págs.
- Curso de programación **GW BASIC/BASICA**
ISBN: 978-84-86381-87-5
320 págs.
- Manual para **TURBO BASIC**
Guía del programador
ISBN: 978-84-86381-43-1
444 págs.
- Manual para **Quick C 2**
Guía del programador
ISBN: 978-84-86381-65-3
540 págs.
- Manual para **Quick BASIC 4.5**
Guía del programador
ISBN: 978-84-86381-74-5
496 págs.
- Curso de programación **Microsoft COBOL**
ISBN: 978-84-7897-001-8
480 págs.
- Enciclopedia del lenguaje **C**
ISBN: 978-84-7897-053-7
888 págs.
- Curso de programación **QBASIC y MS-DOS 5**
ISBN: 978-84-7897-059-9
384 págs.
- Curso de programación **RM/COBOL-85**
ISBN: 978-84-7897-070-4
396 págs.
- El abecé de **MS-DOS 6**
ISBN: 978-84-7897-114-5
224 págs.
- Microsoft **Visual C ++** (ver. 1.5x de 16 bits)
Aplicaciones para Windows
ISBN: 978-84-7897-180-0
846 págs. + 2 disquetes
- Microsoft **Visual C ++**
Aplicaciones para Win32 (2ª edición)
ISBN: 978-84-7897-561-7
792 págs. + disquete
- Microsoft **Visual C ++**
Programación avanzada en Win32
ISBN: 978-84-7897-344-6
888 págs. + CD-ROM
- **Visual Basic 6**
Curso de programación (2ª edición)
ISBN: 978-84-7897-357-6
528 págs. + disquete
- Enciclopedia de Microsoft **Visual Basic 6**
ISBN: 978-84-7897-386-6
1072 págs. + CD-ROM
- El lenguaje de programación **Java**
ISBN: 978-84-7897-485-6
320 págs. + CD-ROM
- El lenguaje de programación **C#**
ISBN: 978-84-7897-500-6
320 págs. + CD-ROM

Del mismo autor

- El lenguaje de programación
Visual Basic.NET
ISBN: 978-84-7897-525-9
464 págs. + CD-ROM
- **Java 2.**
Curso de programación (3ª edición)
ISBN: 978-84-7897-686-7
880 págs. + CD-ROM
- **Microsoft C#**
Curso de programación
ISBN: 978-84-7897-737-6
866 págs. + CD-ROM
- **Java 2**
Lenguaje y aplicaciones
ISBN: 978-84-7897-745-1
392 págs. + CD-ROM
- Programación orientada a objetos
con **C ++** (4ª edición)
ISBN: 978-84-7897-761-1
648 págs. + CD-ROM
- **C/C++**
Curso de programación (3ª edición)
ISBN: 978-84-7897-762-8
708 págs. + CD-ROM
- **Microsoft Visual Basic .NET**
Curso de programación
ISBN: 978-84-7897-812-0
832 págs. + CD-ROM
- **Microsoft C#**
Lenguaje y aplicaciones (2ª edición)
ISBN: 978-84-7897-813-7
520 págs. + CD-ROM
- **Java 2. Interfaces gráficas y aplicaciones para Internet** (3ª edición)
ISBN: 978-84-7897-859-5
718 págs. + CD-ROM
- **Aplicaciones .Net multiplataforma**
(Proyecto Mono)
ISBN: 978-84-7897-880-9
212 págs. + CD-ROM
- Enciclopedia del lenguaje
C ++ (2ª edición)
ISBN: 978-84-7897-915-8
902 págs. + CD-ROM
- Enciclopedia de Microsoft
Visual C# (3ª edición)
ISBN: 978-84-7897-986-8
1110 págs. + CD-ROM
- Enciclopedia de Microsoft
Visual Basic (2ª edición)
ISBN: 978-84-7897-987-5
1090 págs. + CD-ROM
- **Microsoft Visual Basic .NET**
Lenguaje y aplicaciones (3ª edición)
ISBN: 978-84-9964-020-4
520 págs. + CD-ROM