



# Creación de aplicaciones con JBuilder®

---



VERSIÓN 8

Borland®  
**JBuilder®**

Borland Software Corporation  
100 Enterprise Way, Scotts Valley, CA 95066-3249  
[www.borland.com](http://www.borland.com)

En el archivo `deploy.html` ubicado en el directorio raíz del producto JBuilder encontrará una lista completa de archivos que se pueden distribuir de acuerdo con la licencia de JBuilder y la limitación de responsabilidad.

Borland Software Corporation puede tener patentes concedidas o en tramitación sobre los temas tratados en este documento. Diríjase al CD del producto o al cuadro de diálogo Acerca de para la lista de patentes. La modificación de este documento no le otorga derechos sobre las licencias de estas patentes.

COPYRIGHT © 1997-2003 Borland Software Corporation. Reservados todos los derechos. Todos los nombres de productos y marcas de Borland son marcas comerciales o registradas de Borland Software Corporation en Estados Unidos y otros países. Las otras marcas pertenecen de sus respectivos propietarios.

Si desea más información acerca de las condiciones de contrato de terceras partes y acerca de la limitación de responsabilidades, consulte las notas de esta versión en su CD de instalación de JBuilder.

Impreso en EE.UU.

JBE0080WW21001bajb 7E10R1002  
0203040506-9 8 7 6 5 4 3 2 1  
PDF

# Índice de materias

<b>Capítulo 1</b>	
<b>Introducción</b>	<b>1-1</b>
Convenciones de la documentación . . . . .	1-5
Asistencia a los desarrolladores . . . . .	1-7
Cómo ponerse en contacto con el servicio técnico de Borland . . . . .	1-7
Recursos en línea . . . . .	1-7
World Wide Web . . . . .	1-8
Grupos de noticias de Borland . . . . .	1-8
Usenet, grupos de noticias . . . . .	1-8
Información sobre errores . . . . .	1-9
<b>Capítulo 2</b>	
<b>Creación y gestión de proyectos</b>	<b>2-1</b>
Creación de proyectos . . . . .	2-2
Creación de proyectos con ayuda del Asistente para proyectos . . . . .	2-2
Selección del nombre del proyecto y la plantilla . . . . .	2-3
Configuración de las vías de acceso del proyecto . . . . .	2-4
Configuración de las opciones generales del proyecto . . . . .	2-5
Creación de proyectos a partir de archivos anteriores . . . . .	2-8
Selección del directorio fuente y el nombre del nuevo proyecto de JBuilder . . . . .	2-8
Presentación de los archivos . . . . .	2-10
Cambio entre archivos . . . . .	2-10
Almacenamiento de proyectos . . . . .	2-11
Apertura de proyectos . . . . .	2-12
Creación de un archivo fuente Java . . . . .	2-13
Gestión de proyectos . . . . .	2-14
Cómo añadir elementos a un proyecto . . . . .	2-14
Cómo añadir carpetas . . . . .	2-15
Cómo añadir archivos y paquetes . . . . .	2-15
Eliminación de partes de un proyecto . . . . .	2-16
Eliminación de partes del proyecto . . . . .	2-16
Apertura de archivos fuera de un proyecto . . . . .	2-17
Cómo cambiar el nombre a proyectos y archivos . . . . .	2-17
Adición de un nuevo directorio de navegación . . . . .	2-18
Definición de las propiedades del proyecto . . . . .	2-20
Configuración del JDK . . . . .	2-21
Modificación del JDK . . . . .	2-22
Depuración con -classic . . . . .	2-22
Configuración del JDK en SE y Enterprise . . . . .	2-23
Configuración de los JDK . . . . .	2-24
Configuración de vías de acceso a bibliotecas necesarias . . . . .	2-25
Trabajo con varios proyectos . . . . .	2-26
Cambio de un proyecto a otro . . . . .	2-26
Guardar varios proyectos . . . . .	2-27
Información adicional acerca de los proyectos	2-28
<b>Capítulo 3</b>	
<b>Los grupos de proyectos</b>	<b>3-1</b>
Creación de grupos de proyectos . . . . .	3-1
Adición y eliminación de proyectos de los grupos de proyectos . . . . .	3-3
Desplazamiento por los grupos de proyectos . . . . .	3-4
Adición de proyectos como bibliotecas necesarias . . . . .	3-5
<b>Capítulo 4</b>	
<b>Gestión de las vías de acceso</b>	<b>4-1</b>
Las bibliotecas . . . . .	4-1
Adición y configuración de bibliotecas . . . . .	4-2
Modificación de bibliotecas . . . . .	4-5
Adición de proyectos como bibliotecas necesarias . . . . .	4-5
Presentación de la lista de bibliotecas . . . . .	4-6
Paquetes . . . . .	4-6
Ubicación del archivo .java = vía de acceso a archivos fuente + vía de acceso a paquetes . . . . .	4-7
Ubicación del archivo .class = vía de salida + vía de acceso a paquetes . . . . .	4-8
Utilización de paquetes en JBuilder . . . . .	4-9
Directrices de nomenclatura de paquetes . . . . .	4-9
Cómo construye JBuilder las vías de acceso . . . . .	4-10
Vía de acceso a archivos fuente . . . . .	4-10
Vía de acceso a archivos generados . . . . .	4-11
Vía de acceso a clases . . . . .	4-11
Vía de búsqueda . . . . .	4-11
Vía de acceso a documentos . . . . .	4-12
Vía de acceso a las copias de seguridad . . . . .	4-12
Directorio de trabajo . . . . .	4-13

Localización de los archivos . . . . .	4-13	Generación con archivos Ant externos . . . . .	6-9
Cómo encuentra JBuilder los archivos al profundizar . . . . .	4-14	Adición de archivos de generación Ant a los proyectos . . . . .	6-10
Cómo encuentra JBuilder los archivos al compilar . . . . .	4-14	Adición de archivos Ant con el Asistente para Ant . . . . .	6-10
Cómo encuentra JBuilder los archivos de clase al ejecutar o depurar . . . . .	4-14	Adición manual de archivos Ant . . . . .	6-11
<b>Capítulo 5</b>		<b>Creación y modificación de archivos de generación Ant . . . . .</b>	6-12
<b>Compilación de programas en Java 5-1</b>		Importación de proyectos Ant . . . . .	6-12
Comprobación inteligente de dependencias . . .	5-2	Generación de proyectos en Ant . . . . .	6-13
Compilación de un programa . . . . .	5-3	Definición del JDK . . . . .	6-14
Menús de generación de JBuilder . . . . .	5-4	Generación de proyectos Ant con el comando Ejecutar . . . . .	6-14
Generación de proyectos con el comando Ejecutar . . . . .	5-5	Configuración de las propiedades Ant . . . . .	6-15
Errores de sintaxis y mensajes de error . . .	5-5	Opciones Ant . . . . .	6-17
Problemas de compilación al abrir proyectos . . . . .	5-6	Adición de bibliotecas Ant personalizadas . . . . .	6-17
Comprobación de correspondencia entre paquetes y directorios . . . . .	5-7	Generación de archivos SQLJ . . . . .	6-17
Definición de opciones del compilador . . . . .	5-7	Creación de tareas externas de generación . . . . .	6-19
Definición de un compilador . . . . .	5-9	Asistente para tareas externas de generación . . . . .	6-19
Configuración de las opciones adicionales de compilación y generación . . . . .	5-9	Generación de tareas externas . . . . .	6-20
Configuración de la vía de salida . . . . .	5-10	Configuración de propiedades de las tareas externas de generación . . . . .	6-21
Compilación de proyectos en un grupo de proyectos . . . . .	5-11	Configuración del menú Proyecto . . . . .	6-21
Compilación desde la línea de comandos . . . . .	5-11	Configuración del menú Proyecto para grupos de proyectos . . . . .	6-23
bmj (Make de Borland para Java) . . . . .	5-11	Recopilación automática de paquetes fuente . . . . .	6-25
bcj (Compilador de Borland para Java) . . . .	5-12	Filtrado de paquetes . . . . .	6-27
Creación de proyectos desde la línea de comandos . . . . .	5-12	Exclusión de paquetes . . . . .	6-28
Cambio entre la línea de comandos y el IDE . . . . .	5-12	Inclusión de paquetes . . . . .	6-29
<b>Capítulo 6</b>		Copia selectiva de los recursos . . . . .	6-29
<b>Generación de programas en Java 6-1</b>		Propiedades de recursos individuales . . . . .	6-30
El sistema de generación de JBuilder . . . . .	6-2	Opciones específicas de archivos . . . . .	6-30
Términos del sistema de generación . . . . .	6-2	Opciones específicas del proyecto . . . . .	6-31
Fases de generación . . . . .	6-3	Adición de tipos de archivos no reconocidos como archivos de recursos genéricos . . . . .	6-31
El comando Ejecutar Make . . . . .	6-4	La ficha Recursos del cuadro de diálogo Propiedades de proyecto . . . . .	6-32
El comando Generar de nuevo . . . . .	6-5		
El comando Limpiar . . . . .	6-5		
Generación de grupos de proyectos . . . . .	6-6		
Definición del orden de generación de un grupo de proyectos . . . . .	6-7		
Generación de un grupo de proyectos . . . . .	6-8		
Adición de tipos de generación de grupos de proyectos al menú Proyecto . . . . .	6-8		
<b>Capítulo 7</b>			
<b>Ejecución de programas en Java 7-1</b>			
Ejecución de archivos de programa . . . . .	7-2		
Ejecución de archivos Web . . . . .	7-3		
Ejecución de proyectos . . . . .	7-3		
El comando Ejecutar . . . . .	7-4		
Ejecución de proyectos agrupados . . . . .	7-5		
Ejecución de OpenTools . . . . .	7-6		

Definición de las configuraciones para la ejecución . . . . .	7-7
Creación de una configuración de ejecución	7-9
Modificación de una configuración de ejecución . . . . .	7-11
Tipos de generación . . . . .	7-11
Tipos de configuraciones de ejecución . . . . .	7-13
Ejecución de programas desde la línea de comandos . . . . .	7-14
Ejecución del programa distribuido desde la línea de comandos. . . . .	7-14
<b>Capítulo 8</b>	
<b>Depuración de programas en Java</b>	<b>8-1</b>
Tipos de errores . . . . .	8-2
Errores de ejecución . . . . .	8-2
Errores lógicos . . . . .	8-3
Descripción general del proceso de depuración . . . . .	8-3
Creación de una configuración de ejecución	8-4
Compilación del proyecto con información simbólica de depuración. . . . .	8-4
Inicio del Depurador. . . . .	8-6
Inicio del depurador con la opción -classic . . . . .	8-7
Ejecución bajo el control del depurador . . . . .	8-8
Pausar la ejecución del programa . . . . .	8-8
Finalización de una sesión de depuración. . . . .	8-8
Interfaz del depurador . . . . .	8-9
Sesiones de depuración . . . . .	8-10
Vistas del depurador. . . . .	8-10
Vista Salida, entrada y errores de consola . . . . .	8-12
Vista Clases con inspección desactivada	8-13
Vista Puntos de interrupción de datos y código . . . . .	8-14
Vista Hilos, pilas de llamada y datos . . . . .	8-16
Vista Puntos de observación de datos . . . . .	8-20
Vista Clases cargadas y datos estáticos. . . . .	8-23
Vista Monitores de sincronización . . . . .	8-26
Barra de herramientas del depurador . . . . .	8-27
Métodos abreviados para el depurador . . . . .	8-28
ExpressionInsight . . . . .	8-29
Ayuda inmediata. . . . .	8-30
Depuración de fuente distinta de Java. . . . .	8-30
Control de la ejecución del programa . . . . .	8-31
Ejecución e interrupción del programa . . . . .	8-31
Reinicio del programa . . . . .	8-32
El punto de ejecución . . . . .	8-32
Definición del punto de ejecución . . . . .	8-33
Gestión de hilos . . . . .	8-34
Utilización del panel dividido . . . . .	8-34
Presentación del hilo actual. . . . .	8-35
Presentación del marco de pila superior	8-35
Elección de un hilo para inspeccionarlo.	8-35
Interrupción prolongada de un hilo . . . . .	8-35
Detección de conflictos . . . . .	8-36
Desplazamiento a través del código. . . . .	8-37
Inspección de código de llamadas a métodos . . . . .	8-37
Omisión de inspección en las llamadas a métodos . . . . .	8-38
Salida de métodos . . . . .	8-38
Paso inteligente . . . . .	8-39
Ejecución hasta un punto de interrupción .	8-40
Ejecución hasta el final de un método. . . . .	8-40
Ejecución hasta la posición del cursor. . . . .	8-41
Visualización de llamadas a métodos . . . . .	8-41
Localización de una llamada a un método .	8-41
Determinación de las clases que se han de inspeccionar . . . . .	8-42
Inspección de clases cuando el archivo fuente no está disponible . . . . .	8-44
Puntos de interrupción y configuración de Inspección desactivada . . . . .	8-45
Puntos de interrupción. . . . .	8-46
Definición de puntos de interrupción . . . . .	8-46
Definición de puntos de interrupción de línea . . . . .	8-47
Definición de puntos de interrupción por excepción. . . . .	8-49
Definición de puntos de interrupción de clase . . . . .	8-50
Definición de puntos de interrupción de método . . . . .	8-51
Definición de puntos de interrupción de campo . . . . .	8-53
Configuración de un punto de interrupción interprocesal. . . . .	8-53
Definición de las propiedades de los puntos de interrupción . . . . .	8-56
Definición de las acciones de puntos de interrupción . . . . .	8-57
Detención de la ejecución del programa	8-57
Registro de mensajes . . . . .	8-58
Creación de puntos de interrupción condicionales. . . . .	8-59

<b>Capítulo 9</b>	
<b>Depuración remota</b>	<b>9-1</b>
Apertura y depuración de programas en un equipo remoto . . . . .	9-2
Depuración de un programa que se ejecuta en un ordenador remoto . . . . .	9-6
Depuración del código local que se ejecuta en un proceso independiente . . . . .	9-10
Depuración con puntos de interrupción interprocesales . . . . .	9-10
<b>Capítulo 10</b>	
<b>Creación de JavaBeans con BeansExpress</b>	<b>10-1</b>
Definición de JavaBean. . . . .	10-1
¿Por qué desarrollar JavaBeans? . . . . .	10-2
Generación de clases bean . . . . .	10-2
Diseño de la interfaz de usuario de un bean . . . . .	10-4
Adición de propiedades a un bean . . . . .	10-4
Modificación de propiedades . . . . .	10-7
Eliminación de propiedades . . . . .	10-8
Adición de propiedades monitorizables y restringidas. . . . .	10-8
Creación de una clase BeanInfo . . . . .	10-9
Especificación de datos BeanInfo de una propiedad . . . . .	10-10
Utilización del diseñador de BeanInfo . .	10-10
Modificación de clases BeanInfo. . . . .	10-11
Adición de sucesos a un bean . . . . .	10-12
Activación de sucesos. . . . .	10-12
Monitorización de sucesos . . . . .	10-15
Creación de un conjunto de sucesos personalizado . . . . .	10-16
Creación de editores de propiedades . . . . .	10-18
Creación de un editor de lista de cadenas .	10-18
Creación de un editor de listas de etiquetas de cadena . . . . .	10-19
Creación de un editor de listas de etiquetas de entero . . . . .	10-20
Creación de editores de propiedades basados en componentes personalizados. . . . .	10-21
Admisión de serialización . . . . .	10-22
Comprobación de la validez de un JavaBean . . . . .	10-23
Instalación de beans en la paleta de componentes. . . . .	10-24
<b>Capítulo 11</b>	
<b>Presentación de código con UML</b>	<b>11-1</b>
Java y UML . . . . .	11-2
Términos de Java y UML . . . . .	11-2
JBuilder y UML . . . . .	11-3
Diagrama limitado a las dependencias de paquetes . . . . .	11-4
Diagrama combinado de clases . . . . .	11-5
Glosario de los diagramas UML de JBuilder . . . . .	11-7
Iconos de accesibilidad . . . . .	11-9

Presentación de los diagramas UML . . . . .	11-11	Presentación de los cambios antes del perfeccionamiento . . . . .	12-11
El visualizador UML de JBuilder . . . . .	11-11	Ejecución del perfeccionamiento . . . . .	12-14
Presentación de diagramas de paquetes . .	11-12	Optimizar importaciones . . . . .	12-15
Presentación de diagramas de clases . . . .	11-12	Optimización de las importaciones . .	12-18
Presentación de las clases internas . . . . .	11-13	Perfeccionamiento por cambio de nombre de paquetes . . . . .	12-18
Presentación del código fuente . . . . .	11-14	Perfeccionamiento por cambio de nombre de clases . . . . .	12-19
Visualización de Javadoc . . . . .	11-14	Perfeccionamiento por desplazamiento de clases . . . . .	12-20
Uso del menú contextual . . . . .	11-15	Perfeccionamiento por cambio de nombre de métodos . . . . .	12-21
Desplazamiento de la vista . . . . .	11-15	Perfeccionamiento por cambio de nombre de variables locales . . . . .	12-22
Vista completa . . . . .	11-15	Perfeccionamiento por cambio de nombre de campos . . . . .	12-23
Vista parcial . . . . .	11-16	Perfeccionamiento por cambio de nombre de propiedades . . . . .	12-24
Actualización de la vista . . . . .	11-16	Cambio de parámetros de métodos . . . .	12-24
Desplazamiento por diagramas . . . . .	11-16	Extracción de métodos . . . . .	12-26
UML y el panel de estructura . . . . .	11-17	Introducción de variables . . . . .	12-27
Diagramas de paquetes . . . . .	11-17	Perfeccionamiento con sentencia try / catch . . . . .	12-28
Diagramas de clases . . . . .	11-18	Cómo deshacer un perfeccionamiento . . . .	12-28
Personalización de diagramas UML . . . . .	11-18	Almacenamiento de perfeccionamientos .	12-29
Definición de las propiedades del proyecto . . . . .	11-18		
Filtrado de paquetes y clases . . . . .	11-19		
Inclusión de referencias de bibliotecas de proyecto . . . . .	11-20		
Inclusión de referencias del código generado . . . . .	11-20		
Configuración de las opciones del IDE . . .	11-21		
Creación de imágenes de diagramas UML . .	11-21		
Impresión de diagramas UML . . . . .	11-22		
Perfeccionamiento y Buscar referencias . .	11-22		
<b>Capítulo 12</b>		<b>Capítulo 13</b>	
<b>Perfeccionamiento de símbolos de código</b>	<b>12-1</b>	<b>Test de módulos</b>	<b>13-1</b>
Tipos de perfeccionamiento . . . . .	12-1	JUnit . . . . .	13-1
Optimizar importaciones . . . . .	12-2	Cactus . . . . .	13-2
Perfeccionamiento por cambio de nombre .	12-2	Funciones de test de módulos de JBuilder . .	13-2
Perfeccionamiento por desplazamiento . .	12-4	Detección de tests . . . . .	13-3
Cambiar parámetros . . . . .	12-4	Recopilador de tests de JUnit . . . . .	13-3
Extraer método . . . . .	12-4	Creación de tests y conjuntos de tests JUnit .	13-5
Introducir variables . . . . .	12-5	El Asistente para tests . . . . .	13-6
Insertar en sentencia try / catch . . . . .	12-5	Adición de código a los tests . . . . .	13-6
Herramientas de perfeccionamiento de		El Asistente para conjuntos de tests . . . .	13-7
JBuilder . . . . .	12-5	El Asistente para clientes de prueba EJB .	13-8
Configuración para la detección y el perfeccionamiento de referencias . . . . .	12-6	Montajes para tests predefinidos . . . . .	13-8
Obtención de información sobre un símbolo antes del perfeccionamiento . .	12-8	Montaje JDBC . . . . .	13-9
Búsqueda de definiciones de símbolos .	12-8	Montaje JNDI . . . . .	13-10
Búsqueda de referencias a símbolos .	12-9	Montaje para comparación . . . . .	13-10

Creación de un test Cactus para los Enterprise JavaBean . . . . .	13-13
Ejecución de test Cactus . . . . .	13-14
Ejecución de tests . . . . .	13-15
JBTestRunner . . . . .	13-15
Jerarquía del test . . . . .	13-16
Fallos del test . . . . .	13-17
Resultados del test . . . . .	13-17
JUnit TextUI. . . . .	13-17
JUnit SwingUI . . . . .	13-17
Configuraciones de ejecución . . . . .	13-17
Definición del filtro de seguimiento de la pila de tests . . . . .	13-18
Depuración de tests . . . . .	13-19
<b>Capítulo 14</b>	
<b>Creación de Javadoc a partir de archivos fuente de la API</b>	<b>14-1</b>
Adición de comentarios Javadoc a los archivos fuente API . . . . .	14-2
Colocación de los comentarios Javadoc . . . . .	14-3
Etiquetas Javadoc. . . . .	14-5
Creación automática de etiquetas Javadoc . . . . .	14-7
Etiquetas @todo Javadoc . . . . .	14-8
Conflictos en comentarios Javadoc . . . . .	14-8
Generación del nodo de documentación . . . . .	14-9
Selección del formato de la documentación . . . . .	14-9
Selección de opciones para generar documentación . . . . .	14-11
Selección de paquetes que se han de documentar . . . . .	14-13
Especificación de opciones de línea de comandos para el doclet . . . . .	14-14
Creación de archivos de salida . . . . .	14-18
Creación de archivos adicionales. . . . .	14-20
Archivos de paquetes . . . . .	14-20
Archivos de comentarios de aspectos generales . . . . .	14-21
Visualización de Javadoc . . . . .	14-22
Apariencia de Javadoc en JBuilder . . . . .	14-24
Mantenimiento de Javadoc . . . . .	14-25
Cambio de propiedades para el nodo de documentación . . . . .	14-25
Cambio de las propiedades del nodo . . . . .	14-26
Modificación de propiedades Javadoc . . . . .	14-26
Modificación de propiedades doclet . . . . .	14-27
Creación de un archivo recopilatorio de documentación . . . . .	14-28
Creación de un doclet personalizado . . . . .	14-30
<b>Capítulo 15</b>	
<b>Distribución de programas en Java</b>	<b>15-1</b>
Distribución de archivos recopilatorios de Java (JAR) . . . . .	15-2
Conceptos básicos acerca del archivo descriptor. . . . .	15-3
Estrategias de distribución . . . . .	15-5
Utilización de la herramienta de recopilación de Java del JDK . . . . .	15-6
Ejecución de programas desde un archivo JAR . . . . .	15-6
Presentación del contenido de un archivo recopilatorio . . . . .	15-7
Actualización del contenido de un archivo JAR . . . . .	15-7
Aspectos relativos a la distribución. . . . .	15-8
¿Está todo lo necesario en la vía de acceso a clases? . . . . .	15-8
¿El programa depende de funciones JDK 1.1 o Java 2 (JDK 1.2 y superior)? . . . . .	15-9
¿Tiene el usuario bibliotecas Java instaladas en su ordenador? . . . . .	15-9
¿Se trata de un applet o de una aplicación? . . . . .	15-10
Tiempo de descarga de archivos. . . . .	15-11
Distribución rápida . . . . .	15-11
Aplicaciones . . . . .	15-11
Applets . . . . .	15-12
JavaBeans. . . . .	15-14
Sugerencias de distribución . . . . .	15-15
Configuración del entorno de trabajo . . . . .	15-15
Distribución en Internet . . . . .	15-15
Distribución de aplicaciones distribuidas	15-16
Redistribución de las clases que se suministran con JBuilder . . . . .	15-16
Información adicional de distribución . . . . .	15-17
Distribución con el Creador de recopilatorios . . . . .	15-18
El Creador de recopilatorios y los recursos . . . . .	15-18
Selección de tipos de recopilatorios . . . . .	15-19
Definición del archivo que se ha de crear. . . . .	15-20
Selección de los archivos descriptores de la distribución . . . . .	15-21
Especificar las partes del proyecto que se van a recopilar . . . . .	15-23
Definición del contenido para un recopilatorio RAR (Adaptador de recursos) . . . . .	15-24

Determinar las dependencias entre bibliotecas . . . . .	15-25
Configurar las opciones del archivo descriptor del recopilatorio . . . . .	15-27
Seleccionar un método para indicar la clase principal de la aplicación . . . . .	15-28
Determinar los archivos ejecutables que se van a crear . . . . .	15-29
Ejecución de ejecutables . . . . .	15-30
Definir las opciones de la configuración de ejecución. . . . .	15-31
Creación de ejecutables con el Creador de ejecutables nativos . . . . .	15-32
Generación de archivos recopilatorios. . . . .	15-34
Los nodos de recopilatorios. . . . .	15-35
Presentación del archivo recopilatorio y el archivo descriptor . . . . .	15-35
Modificación de las propiedades de los nodos de recopilatorios . . . . .	15-36
Eliminación, borrado y asignación de nombres a recopilatorios. . . . .	15-36
<b>Capítulo 16</b>	
<b>Internacionalización de programas con JBuilder</b>	<b>16-1</b>
Términos y definiciones de internacionalización . . . . .	16-1
Funciones de internacionalización de JBuilder . . . . .	16-3
Aplicación de ejemplo multilingüe . . . . .	16-3
Eliminación de cadenas no modificables (hard-coded) incluidas en el código . . . . .	16-5
Utilización del Asistente para extracción de recursos . . . . .	16-6
Utilización del cuadro de diálogo Propiedad localizable . . . . .	16-8
Funciones de internacionalización de dbSwing . . . . .	16-9
Utilización de componentes que identifican el país . . . . .	16-10
Los componentes de JBuilder muestran todos los caracteres Unicode . . . . .	16-11
Funciones de internacionalización del diseñador de interfaz de usuario . . . . .	16-11
Unicode en el Depurador IDE . . . . .	16-13
Elección de una codificación nativa para el compilador . . . . .	16-13
Definición de la opción de codificación . . . . .	16-14
Codificaciones nativas admitidas. . . . .	16-14
Adición y redefinición de codificaciones . . . . .	16-15
Otros asuntos relacionados con las codificaciones nativas . . . . .	16-15
Formato Unicode de 16 bits . . . . .	16-16
Utilización de Unicode por medio de ASCII y '\u'. . . . .	16-16
JBuilder en el mundo . . . . .	16-17
Asistencia internacional en línea . . . . .	16-17
<b>Capítulo 17</b>	
<b>Tutorial: Tutorial de compilación, ejecución y depuración</b>	<b>17-1</b>
Paso 1: Abrir el proyecto de ejemplo . . . . .	17-2
Paso 2: Solucionar errores de sintaxis. . . . .	17-3
Paso 3: Solucionar errores de compilación . . . . .	17-4
Paso 4: Ejecución del programa . . . . .	17-7
Almacenamiento de archivos y ejecución del programa . . . . .	17-9
Paso 5: Corregir el método subtractValues(). . . . .	17-10
Almacenamiento de archivos y ejecución del programa . . . . .	17-16
Paso 6: Corregir el método divideValues(). . . . .	17-17
Almacenamiento de archivos y ejecución del programa . . . . .	17-20
Paso 7: Corregir el método oddEven() . . . . .	17-20
Paso 8: Buscar excepciones producidas durante la ejecución. . . . .	17-24
<b>Capítulo 18</b>	
<b>Tutorial: Generación con archivos Ant</b>	<b>18-1</b>
Paso 1: Crear un proyecto y una aplicación . . . . .	18-2
Paso 2: Crear el archivo de generación Ant . . . . .	18-2
Paso 3: Ejecutar tipos de generación individuales . . . . .	18-4
Paso 4: Ejecutar el objetivo por defecto. . . . .	18-4
Paso 5: Tratamiento de errores con Ant . . . . .	18-5
Paso 6: Añadir un tipo de generación al menú Proyecto. . . . .	18-6
Paso 7: Configuración de las propiedades Ant . . . . .	18-7
Paso 8: Añadir tareas Ant personalizadas a su proyecto. . . . .	18-9
<b>Capítulo 19</b>	
<b>Tutorial: Depuración remota</b>	<b>19-1</b>
Paso 1: Abrir el proyecto de ejemplo . . . . .	19-2
Paso 2: Definir las configuraciones de ejecución y depuración . . . . .	19-3

Paso 3: Definición de puntos de interrupción.	.19-7
Paso 4: Compilar el servidor y almacenar archivos de clase al equipo remoto.	.19-9
Paso 5: Iniciar el registro de RMI y el servidor en el equipo remoto	.19-10
Paso 6: Iniciar el proceso servidor y el cliente en modo depuración.	.19-12

**Capítulo 20**

<b>Tutorial: Visualización de código con el visualizador UML</b>	<b>20-1</b>
Paso 1: Compilación del ejemplo.	.20-2
Paso 2: Visualización de un diagrama de paquete UML.	.20-3
Paso 3: Visualización de un diagrama de clase UML.	.20-6
Paso 4: Añadir referencias de bibliotecas	.20-10
Paso 5: Filtrado de diagramas UML	.20-13

**Capítulo 21**

<b>Tutorial: Creación y ejecución de tests y conjuntos de tests</b>	<b>21-1</b>
Paso 1: Apertura de proyectos	.21-2
Paso 2: Creación de montajes para tests	.21-2
Paso 3: Implementación de un método de test que lanza una excepción esperada	.21-3
Visualización de la salida de fallo del test	.21-4
Corrección del test para que pase	.21-5
Paso 4: Creación de un segundo método de test	.21-5
Paso 5: Creación de un conjunto de tests	.21-6
Paso 6: Ejecución de tests	.21-7

**Capítulo 22**

<b>Tutorial: Utilización de montajes para tests</b>	<b>22-1</b>
Paso 1: Creación de proyectos	.22-2
Paso 2: Creación de un módulo de datos	.22-2
Paso 3: Creación de un montaje de comparación	.22-3
Paso 4: Creación de un montaje JDBC	.22-4
Paso 5: Modificación del montaje JDBC para ejecutar scripts SQL	.22-6
Paso 6: Creación de un test utilizando montajes para tests	.22-7
Paso 7: Implementación del test	.22-8
Paso 8: Adición de una biblioteca necesaria	.22-9
Paso 9: Ejecución del test	.22-9

**Apéndice A**

<b>Creación de archivos de configuración para ejecutables nativos</b>	<b>A-1</b>
Inicio de la MV	.A-3
Requisitos del archivo de configuración	.A-3
Tipo de archivo y ubicación	.A-3
Líneas en blanco y comentarios	.A-3
Convenciones de las vías de acceso	.A-3
Directivas	.A-4
javapath	.A-4
mainclass	.A-4
addpath	.A-5
addjars	.A-5
addbootpath	.A-5
addbootjars	.A-6
addskippath	.A-6
vmparam	.A-6
include	.A-6
includedir	.A-7
copyenv	.A-7
exportenv	.A-7
addparam	.A-8
clearparams	.A-8
restartcode	.A-8
Funciones optativas de inicio en un solo paso	.A-8

**Apéndice B**

<b>Las herramientas de línea de comandos</b>	<b>B-1</b>
Definición de la vía de acceso a clases para herramientas de línea de comandos	.B-2
Opción -classpath	.B-2
Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos	.B-2
UNIX: variable de entorno CLASSPATH	.B-3
Windows: variable de entorno CLASSPATH	.B-3
Interfaz de la línea de comandos de JBuilder	.B-4
Acceso a una lista de opciones	.B-5
Sintaxis	.B-5
Opciones	.B-5
El compilador de Borland para Java (bcj)	.B-7
Sintaxis	.B-7
Descripción	.B-7
Opciones	.B-8
Opciones de compilación cruzada	.B-11
Opciones de MV	.B-12

Make de Borland para Java (bmj) . . . . .	B-12	
Sintaxis . . . . .	B-12	Especificadores para las clases raíz . . . . . B-18
Descripción . . . . .	B-12	Opciones de MV . . . . . B-19
Opciones. . . . .	B-13	
Opciones de compilación cruzada . . . . .	B-17	

## Índice

I-1

# Tablas

1.1	Convenciones tipográficas y de símbolos.	1-5
1.2	Convenciones de las plataformas . . . . .	1-6
4.1	Colores de las listas de bibliotecas . . . . .	4-6
6.1	Términos del sistema de generación. . . . .	6-2
6.2	Fases del sistema de generación . . . . .	6-3
6.3	Iconos del filtrado de paquetes. . . . .	6-29
8.1	Comandos de menú para iniciar el depurador . . . . .	8-6
8.2	Vistas del depurador . . . . .	8-11
8.3	Iconos de la vista Consola . . . . .	8-12
8.4	Menú contextual de la vista Consola . .	8-12
8.5	Iconos en la vista Clases con inspección desactivada. . . . .	8-13
8.6	Menú contextual con clase o paquete seleccionado en la vista Clases con inspección desactivada . . . . .	8-13
8.8	Iconos de la vista Puntos de interrupción por datos y código. . . . .	8-14
8.7	Menú contextual sin selección en la vista Clases con inspección desactivada. . . . .	8-14
8.9	Menú contextual con punto de interrupción seleccionado en la vista Puntos de interrupción de datos y código. . . . .	8-15
8.10	Menú contextual sin selección en la vista Puntos de interrupción de datos y código. . . . .	8-15
8.11	Iconos en la vista Hilos, pilas de llamadas y datos . . . . .	8-17
8.12	Menú contextual con selección en la vista Hilos, pilas de llamadas y datos . . . . .	8-17
8.14	Iconos en la vista Puntos de observación de datos. . . . .	8-20
8.13	Menú contextual sin selección en la vista Hilos, pilas de llamadas y datos . . . . .	8-20
8.15	Menú contextual con punto de observación seleccionado en la vista Puntos de observación de datos . . . . .	8-21
8.16	Menú contextual sin selección en la vista Puntos de observación de datos. . . . .	8-23
8.17	Iconos en la vista Clases cargadas y datos estáticos . . . . .	8-24
8.18	Menú contextual con selección de la vista Clases cargadas y datos estáticos . . . . .	8-24
8.19	Menú contextual sin selección de la vista Clases cargadas y datos estáticos . . . . .	8-26
8.20	Iconos en la vista Monitores de sincronización. . . . .	8-26
8.21	Menú contextual de la vista Monitores de sincronización . . . . .	8-27
8.22	Botones de la barra de herramientas . . . . .	8-27
8.23	Métodos abreviados para el depurador . . . . .	8-28
8.24	Características del depurador . . . . .	8-63
8.25	Tipos de variables de puntos de observación en ámbito . . . . .	8-68
11.1	Términos de Java y UML . . . . .	11-2
11.2	Definiciones de los diagramas UML . . . . .	11-7
11.3	Iconos de accesibilidad de UML . . . . .	11-10
12.1	Perfeccionamiento y símbolos de código . . . . .	12-3
12.2	Detalles de Buscar referencias. . . . .	12-10
12.3	Detalles de perfeccionamiento . . . . .	12-13
14.1	Etiquetas Javadoc . . . . .	14-5
14.2	Opciones que no se configuran en el asistente . . . . .	14-16
19.1	Fichas del cuadro de diálogo para definir las configuraciones de depuración y ejecución servidor y cliente . . . . .	19-3
19.2	Argumentos de línea de comandos para RMI y el depurador. . . . .	19-11
19.1	Mensajes de error de RMI cliente/servidor. . . . .	19-14

# Figuras

6.1	Ficha Recursos del cuadro de diálogo Propiedades de proyecto . . . . .	6-30
7.1	Mensajes de error en el Visualizador de aplicaciones . . . . .	7-5
8.1	Interfaz del depurador . . . . .	8-9
8.2	Barra de herramientas del depurador .	8-27
8.3	Ventana ExpressionInsight . . . . .	8-29
8.4	Ventana Ayuda inmediata . . . . .	8-30
8.5	El punto de ejecución . . . . .	8-33
8.6	Panel dividido de la vista Hilos, pilas de llamadas y datos . . . . .	8-35
8.7	Vista Monitores de sincronización . .	8-37
8.8	Archivo fuente stub . . . . .	8-45
8.9	Cuadro de diálogo Detenido en clase con inspección desactivada . . . . .	8-45
8.10	Vista Puntos de interrupción de datos y código . . . . .	8-46
8.11	Acciones de punto de interrupción . .	8-57
8.12	Mensaje de la barra de estado de punto de interrupción . . . . .	8-58
8.13	Puntos de interrupción condicionales .	8-59
8.14	Vista Clases cargadas y datos estáticos.	8-64
8.15	Vista Hilos, pilas de llamadas y datos .	8-64
8.16	Vista Puntos de observación de datos .	8-67
8.17	Evaluación de expresiones en el cuadro de diálogo Evaluar/Modificar . . . . .	8-70
8.18	Evaluación de métodos en el cuadro de diálogo Evaluar/Modificar . . . . .	8-71
8.19	Ficha Depurar del cuadro de diálogo Propiedades de configuración de ejecución . . . . .	8-74
11.1	Diagrama de paquetes . . . . .	11-4
11.2	Diagrama de clases . . . . .	11-5
11.3	Diagrama de clases con las propiedades por separado . . . . .	11-6
11.4	Diagrama de clases sin las propiedades por separado . . . . .	11-6
11.5	Iconos de accesibilidad de JBuilder . .	11-10
11.6	Visualizador UML . . . . .	11-12
11.7	Presentación de las clases internas . .	11-13
11.8	Panel de estructura de los diagramas UML . . . . .	11-17
12.1	Referencias a clases en la pestaña Buscar Resultados . . . . .	12-10
12.2	Referencias a métodos en la pestaña Buscar Resultados . . . . .	12-11
12.3	Referencias a campos y variables locales en la pestaña Resultado de la búsqueda . . . . .	12-11
12.4	Cuadro de diálogo Cambiar nombre a la clase . . . . .	12-12
12.5	La pestaña Perfeccionamiento antes de la acción . . . . .	12-12
12.6	La pestaña Perfeccionamiento después de la acción . . . . .	12-13
12.7	El archivo fuente y la pestaña Perfeccionamiento después de la acción . . . . .	12-14
14.1	Carpeta Por Hacer del panel de estructura . . . . .	14-8
14.2	Conflictos Javadoc en el panel de estructura . . . . .	14-9
14.3	Paso de selección de doclet . . . . .	14-10
14.4	Paso de selección de opciones de generación y del proyecto . . . . .	14-11
14.5	Paso de selección de paquetes y nivel de visibilidad . . . . .	14-13
14.6	Paso de especificación de opciones doclet de línea de comandos . . . . .	14-14
14.7	Nodo de documentación en el panel del proyecto . . . . .	14-18
14.8	Nodos de documentación ampliados .	14-22
14.9	Salida de archivos de índices desde el Doclet estándar . . . . .	14-23
14.10	Salidas de archivos de índices desde el Doclet JDK 1.1 . . . . .	14-23
14.11	Información Javadoc sobre la marcha .	14-25

# Tutoriales

Tutorial de compilación, ejecución y depuración . . . . .	17-1
Generación con archivos Ant . . . . .	18-1
Depuración remota . . . . .	19-1
Visualización de código con el visualizador UML . . . . .	20-1
Creación y ejecución de tests y de conjuntos de tests . . . . .	21-1
Utilización de montajes para tests . . . . .	22-1

## Introducción

*Creación de aplicaciones con JBuilder* explica cómo utilizar el IDE de JBuilder para gestionar proyectos y para compilar, ejecutar y depurar programas en Java. También explica cómo utilizar BeansExpress para crear JavaBeans. También describe técnicas avanzadas como la distribución de aplicaciones y la internacionalización para diferentes versiones localizadas, la presentación del código, el perfeccionamiento (refactoring) y la comprobación de módulos.

*Creación de aplicaciones con JBuilder* incluye estos capítulos:

- [Capítulo 2, “Creación y gestión de proyectos”](#)

Explica cómo trabajar con los proyectos de JBuilder y cómo establecer las propiedades de los proyectos.

- [Capítulo 3, “Los grupos de proyectos”](#)

Describe cómo colocar proyectos relacionados en grupos de proyectos y cómo utilizarlos.

- [Capítulo 4, “Gestión de las vías de acceso”](#)

Es un capítulo que acompaña a [Capítulo 2, “Creación y gestión de proyectos,”](#) y en el que se describe cómo se utilizan las vías de acceso en JBuilder. Describe cómo se trabaja con bibliotecas y paquetes.

- [Capítulo 5, “Compilación de programas en Java”](#)

Explica cómo compilar el proyecto y cómo definir las opciones del compilador. También explica cómo compilar desde la línea de comandos.

- [Capítulo 6, “Generación de programas en Java”](#)

Explica el proceso de generación de JBuilder. Explica la diferencia entre Ejecutar Make y Generar de nuevo. Describe cómo se generan los archivos Ant externos, cómo se crean los grupos de proyectos y cómo se utilizan las funciones adicionales de JBuilder, cómo la recopilación automática de paquetes fuente, el filtrado de paquetes y la copia de recursos.

- [Capítulo 7, “Ejecución de programas en Java”](#)

Explica cómo utilizar el IDE de JBuilder para ejecutar aplicaciones y applets. También explica cómo gestionar las configuraciones para la ejecución.

- [Capítulo 8, “Depuración de programas en Java”](#)

Explica cómo utilizar el depurador integrado de JBuilder para localizar y solucionar los errores del programa. Describe la totalidad del proceso de depuración de applets y aplicaciones, explica los tipos de errores que pueden producirse y la forma de examinar los valores de las variables de los programas para descubrir los errores.

- [Capítulo 9, “Depuración remota”](#)

Describe la forma de depurar un programa que se ejecuta en un ordenador remoto.

- [Capítulo 10, “Creación de JavaBeans con BeansExpress”](#)

Describe la forma de crear JavaBeans y convertir las clases en JavaBean.

- [Capítulo 11, “Presentación de código con UML”](#)

Describe la forma de utilizar las funciones de presentación de código de JBuilder para examinar el código, desplazarse por él e interpretar su significado.

- [Capítulo 12, “Perfeccionamiento de símbolos de código”](#)

Explica cómo utilizar las funciones de perfeccionamiento de JBuilder.

- [Capítulo 13, “Test de módulos”](#)

Describe las funciones de comprobación de módulos de que dispone JBuilder.

- [Capítulo 14, “Creación de Javadoc a partir de archivos fuente de la API”](#)

Describe la forma de utilizar las funciones relacionadas con Javadoc de JBuilder con el fin de generar archivos de salida HTML a partir de comentarios en código fuente de la API.

- [Capítulo 15, “Distribución de programas en Java”](#)

Proporciona una introducción general de los asuntos relacionados con la distribución y explica la forma de crear archivos recopilatorios de Java con la herramienta **jar**.

- [“Distribución con el Creador de recopilatorios” en la página 15-18](#)

Explica cómo utilizar el Creador de recopilatorios para distribuir su programa Java.

- [“Creación de ejecutables con el Creador de ejecutables nativos” en la página 15-32](#)

Explica cómo utilizar el Creador de ejecutables nativos en la creación de ejecutables nativos para los programas en Java distribuidos.

- [Capítulo 16, “Internacionalización de programas con JBuilder”](#)

Explica la forma de internacionalizar aplicaciones y applets Java con JBuilder.

- Tutoriales:

- [Capítulo 17, “Tutorial: Tutorial de compilación, ejecución y depuración”](#)

Localización y corrección de errores de sintaxis, compilación y lógica.

- [Capítulo 18, “Tutorial: Generación con archivos Ant”](#)

Se utiliza un archivo Ant para crear un proyecto.

- [Capítulo 19, “Tutorial: Depuración remota”](#)

Se utilizan las funciones de depuración remota para vincularse con un programa que ya se está ejecutando en un ordenador remoto y depurarlo por medio de la inspección interprocesal.

- [Capítulo 20, “Tutorial: Visualización de código con el visualizador UML”](#)

Se utilizan las funciones de UML de JBuilder para presentar, analizar y resolver los problemas del código.

- [Capítulo 21, “Tutorial: Creación y ejecución de tests y conjuntos de tests”](#)

Se utilizan las funciones de tests de módulos para crear y ejecutar tests de módulos con JUnit.

- [Capítulo 22, “Tutorial: Utilización de montajes para tests”](#)

Se crean un montaje para JDBC y otro para comparación y se utilizan en un test.

Los siguientes apéndices pertenecen a *Creación de aplicaciones con JBuilder*:

- [Apéndice A, “Creación de archivos de configuración para ejecutables nativos”](#)

En él se aprende a crear archivos de configuración personalizados para iniciar los ejecutables nativos que se crean con el Creador de ejecutables nativos o con el Creador de recopilatorios.

- [Apéndice B, “Las herramientas de línea de comandos”](#)

Explica la forma de utilizar los compiladores de línea de comandos de JBuilder, los argumentos de línea de comandos y las herramientas de JDK. También trata de la configuración de la vía de acceso a clases.

También se pueden encontrar los siguientes temas en la ayuda en línea:

- “Mensajes de advertencia y error”

Esta descripción general presenta los tipos de error y de advertencia que pueden darse durante la compilación, la depuración o la ejecución de aplicaciones de JBuilder. También incluye una lista de errores por número.

- “Mensajes de error del compilador”

Se enumeran los mensajes de error, ordenados por número, seguidos de una descripción.

Para la definición de términos Java desconocidos, consulte los “Glosarios en línea” en *Procedimientos iniciales con Java*.

# Convenciones de la documentación

---

En la documentación de Borland para JBuilder, el texto con significado especial se identifica mediante la tipografía y los símbolos descritos en la siguiente tabla.

**Tabla 1.1** Convenciones tipográficas y de símbolos

Tipo de letra	Significado
Letra monoespaciada	<p>El tipo monoespaciado representa lo siguiente:</p> <ul style="list-style-type: none"> <li>• texto tal y como aparece en la pantalla</li> <li>• cualquier cosa que debe escribir, como “Escriba <code>Hola</code> a todos en el campo Título del Asistente para aplicaciones”.</li> <li>• nombres de archivos</li> <li>• nombres de vías de acceso</li> <li>• nombres de directorios y carpetas</li> <li>• comandos, como <code>SET PATH</code>.</li> <li>• código Java</li> <li>• tipos de datos de Java, como <code>boolean</code>, <code>int</code> y <code>long</code>.</li> <li>• los identificadores de Java, como nombres de variables, clases, nombres de paquetes, interfaces, componentes, propiedades, métodos y sucesos.</li> <li>• nombres de argumentos</li> <li>• nombres de campos</li> <li>• palabras clave de Java, como <code>void</code> y <code>static</code>.</li> </ul>
<b>Negrita</b>	La negrita se utiliza para las herramientas <code>java</code> , <code>bmj</code> (Borland Make for Java), <code>bcj</code> (Borland Compiler for Java) y opciones del compilador. Por ejemplo: <b>javac</b> , <b>bmj</b> , <b>-vía de acceso a clases</b> .
<i>Cursiva</i>	Las palabras en cursiva indican los términos nuevos que se definen y los títulos de libros; ocasionalmente se usan para indicar énfasis.
<i>Nombres de tecla</i>	Este tipo de letra indica una tecla, como “Pulse <code>Esc</code> para salir de un menú”.
[ ]	Los corchetes, en las listas de texto o sintaxis, encierran elementos optativos. En estos casos no se deben escribir los corchetes.

**Tabla 1.1** Convenciones tipográficas y de símbolos (continuación)

Tipo de letra	Significado
< >	Los corchetes se utilizan para indicar las variables de las vías de acceso a los directorios, las opciones de comando y los ejemplos de código. Por ejemplo, <nombredearchivo> puede utilizarse para indicar dónde tiene que incluir el nombre de un archivo (incluida la extensión) y <usuario> indica normalmente que debe indicar su nombre de usuario.
	Cuando se sustituyen las variables en las vías de acceso a los directorios, comandos y ejemplos de código, se sustituye la variable completa, incluidos los corchetes (< >). Por ejemplo, reemplazaría <nombredearchivo> con el nombre de un archivo, como <code>employee.jds</code> y omitiría los corchetes.
	<b>Nota:</b> Los corchetes se utilizan en HTML, XML, JSP y otros archivos basados en etiquetas para demarcar los elementos del documento, como <color de fuente=red> y <ejb-jar>. Las siguientes convenciones describen cómo se especifican las cadenas de variables dentro del ejemplo de código que ya está utilizando corchetes como delimitadores.
<i>Cursiva, serif</i>	Este formato se utiliza para indicar las cadenas de variables en los ejemplos de código que ya están usando corchetes como delimitadores. Por ejemplo, <url="jdbc:borland:jbuilder\\samples\\guestbook.jds">
...	En los ejemplos de código, los puntos suspensivos (...) indican código que se ha omitido en el ejemplo para ahorrar espacio y aumentar la claridad. Si están en un botón, los puntos suspensivos indican que éste conduce a un cuadro de diálogo de selección.

JBuilder se puede utilizar con diversas plataformas. Consulte la siguiente tabla para ver una descripción de las convenciones de plataforma utilizadas en la documentación.

**Tabla 1.2** Convenciones de las plataformas

Elementos	Significado
Vías de acceso	Las vías de acceso a los directorios en la documentación se indican con una barra normal (/). Para la plataforma Windows se utiliza la barra invertida (\).
Directorio de inicio	La ubicación del directorio inicial varía según la plataforma y se indica con una variable <home>. <ul style="list-style-type: none"> <li>• En UNIX y Linux, el directorio inicial puede variar. Por ejemplo, puede ser /user/&lt;nombre de usuario&gt; o /home/&lt;nombre de usuario&gt;</li> <li>• En Windows NT, el directorio inicial es C:\Winnt\Profiles\&lt;nombre de usuario&gt;</li> <li>• En Windows 2000 y XP, el directorio inicial es C:\Documents and Settings\&lt;nombredeusuario&gt;</li> </ul>
Imágenes de pantalla	Las imágenes o capturas de pantalla utilizan el aspecto Metal en diversas plataformas.

# Asistencia a los desarrolladores

---

Borland ofrece una amplia gama de opciones de asistencia técnica y recursos de información para ayudar a los desarrolladores a obtener lo máximo de sus productos Borland. Estas opciones incluyen un rango de programas de asistencia técnica de Borland, así como servicios gratuitos en Internet, donde es posible efectuar búsquedas en nuestra amplia base de información y ponerse en contacto con otros usuarios de productos Borland.

## Cómo ponerse en contacto con el servicio técnico de Borland

---

Borland ofrece varios programas de asistencia para clientes y clientes potenciales. Se puede elegir entre varios tipos de asistencia, que van desde la asistencia para la instalación de los productos Borland hasta el asesoramiento de expertos y la asistencia pormenorizada (servicios no gratuitos).

Si desea más información sobre el servicio al desarrollador de Borland, visite nuestra página web, en <http://www.borland.com/devsupport>.

Cuando se ponga en contacto con el servicio técnico tenga a mano la información completa sobre el entorno, la versión del producto utilizada y una descripción detallada del problema.

Si necesita más información sobre las herramientas o la documentación de otros proveedores, póngase en contacto con ellos.

## Recursos en línea

---

También puede obtener información de los siguientes recursos en línea:

<b>World Wide Web</b>	<a href="http://www.borland.com/">http://www.borland.com/</a>
<b>FTP</b>	<a href="ftp://ftp.borland.com/">ftp://ftp.borland.com/</a>
	Documentación técnica disponible por ftp anónimo.
<b>Listserv</b>	Para suscribirse a circulares electrónicas, rellene el formulario en línea que aparece en: <a href="http://info.borland.com/contact/listserv.html">http://info.borland.com/contact/listserv.html</a> y para el servidor de listas internacional Borland: <a href="http://info.borland.com/contact/intlist.html">http://info.borland.com/contact/intlist.html</a>

## World Wide Web

---

Visite periódicamente [www.borland.com/jbuilder](http://www.borland.com/jbuilder). El equipo de desarrollo de productos Java publica en esta página documentación técnica, análisis de competitividad, respuestas a preguntas frecuentes, aplicaciones de ejemplo, software actualizado e información sobre productos nuevos y antiguos.

En particular, pueden resultar interesantes las siguientes direcciones:

- <http://www.borland.com/jbuilder/> (actualizaciones de software y otros archivos)
- <http://www.borland.com/techpubs/jbuilder/> (actualizaciones de documentación y otros archivos)
- <http://community.borland.com/> (contiene nuestra revista de noticias para desarrolladores en formato web)

## Grupos de noticias de Borland

---

Puede registrar JBuilder y participar en los grupos de debate sobre JBuilder, estructurados en hilos. Los grupos de noticias de Borland proporcionan los medios a todos los clientes de la comunidad Borland para intercambiar sugerencias y técnicas acerca de los productos, herramientas relacionadas y tecnologías Borland.

Puede encontrar grupos de noticias, moderados por los usuarios, sobre JBuilder y otros productos de Borland, en <http://www.borland.com/newsgroups>.

## Usenet, grupos de noticias

---

En Usenet existen los siguientes grupos dedicados a Java y temas relacionados:

- news:comp.lang.java.advocacy
- news:comp.lang.java.announce
- news:comp.lang.java.beans
- news:comp.lang.java.databases
- news:comp.lang.java.gui
- news:comp.lang.java.help
- news:comp.lang.java.machine
- news:comp.lang.java.programmer
- news:comp.lang.java.security
- news:comp.lang.java.softwaretools

**Nota** Se trata de grupos moderados por usuarios; no son páginas oficiales de Borland.

## Información sobre errores

---

Si encuentra algún error en el software, comuníquelo en la página Support Programs, en <http://www.borland.com/devsupport/namerica/>. Pulse el enlace "Reporting Defects" para llegar al formulario Entry.

Cuando informe sobre un fallo, incluya todos los pasos necesarios para llegar a él, así como toda la información posible sobre la configuración, el entorno y las aplicaciones que se estaban utilizando junto con JBuilder. Intente explicar con la mayor claridad posible las diferencias entre el comportamiento esperado y el obtenido.

Si desea enviar felicitaciones, sugerencias o quejas al equipo de documentación de JBuilder, envíe un mensaje a [jpgpubs@borland.com](mailto:jpgpubs@borland.com). Envíe únicamente comentarios sobre la documentación. Tenga en cuenta que los asuntos relacionados con el servicio técnico se deben enviar al departamento de asistencia técnica para programadores.

JBuilder es una herramienta creada por desarrolladores y para desarrolladores. Valoramos sumamente sus aportaciones.



## Creación y gestión de proyectos

JBuilder realiza todas las tareas dentro del contexto de un *proyecto*. En este manual, el término “proyecto” incluye todos los archivos que constituyen un trabajo definido por un usuario, la estructura de directorios en la que residen esos archivos y las vías de acceso, opciones y recursos necesarios.

El proyecto es una herramienta de organización, no un lugar de almacenamiento. Esto significa que los archivos de un proyecto pueden estar en cualquier carpeta. La reestructuración de un árbol de proyecto no afecta al árbol de directorios. Así, se ofrece un control independiente de los proyectos y de la estructura de directorios.

Cada proyecto se administra mediante un archivo de proyecto. El nombre del archivo del proyecto es el nombre del proyecto con la extensión `jpx`. El archivo de proyecto contiene una lista de los archivos del proyecto y mantiene las propiedades del proyecto; éstas incluyen una plantilla de proyecto, las vías de acceso por defecto, las bibliotecas de clase y las configuraciones de conexión. JBuilder utiliza esta información cuando carga, guarda, genera y ejecuta proyectos. Los archivos de proyecto se modifican cuando se utiliza el entorno de desarrollo de JBuilder para añadir o eliminar archivos o para definir o cambiar las propiedades del proyecto. El archivo de proyecto se ve como un nodo en el panel del proyecto. A continuación se enumeran todos los paquetes y archivos del proyecto.

**Nota**

Es una función de  
JBuilder SE y  
Enterprise.

Si la recopilación automática de fuentes está activada, también aparecen en el panel del proyecto los nodos de paquetes de código fuente. Éstos muestran los archivos y paquetes que aparecen en la vía de acceso a fuentes del proyecto. Consulte “[Recopilación automática de paquetes fuente](#)” en la página 6-25.

A pesar de que pueden incluir archivos de cualquier tipo en los proyectos de JBuilder, hay ciertos tipos que JBuilder reconoce automáticamente y

para los que dispone de visualizadores. Se pueden añadir archivos de tipos binarios, personalizar la gestión del tipo de archivos y ver los iconos asociados a los tipos de archivos seleccionando Herramientas | Opciones del IDE, y la pestaña Tipos de archivos.

La primera vez que se inicia JBuilder se solicita que se configuren las asociaciones de archivos y se asocien a JBuilder los archivos .class y .java, así como los archivos de proyecto y de grupo de proyectos. De este modo, JBuilder se convierte en el programa por defecto para abrir y ver estos archivos. Para modificar esta configuración, seleccione Herramientas | Configurar asociaciones de archivos. Se abre el cuadro de diálogo del mismo nombre.

## Creación de proyectos

---

Para crear un proyecto, utilice el Asistente para proyectos de JBuilder para generar automáticamente la estructura básica de archivos, directorios, vías de acceso, y preferencias. El Asistente para proyectos crea automáticamente el archivo de notas de proyecto, que recoge las notas y comentarios. Cuando se utilizan los asistentes de JBuilder para crear archivos Java, los campos de la clase Javadoc que se llenan en el Asistente para proyectos se utilizan en el archivo de notas del proyecto en forma de comentarios de encabezado Javadoc, y por tanto, se incluyen en la documentación generada por Javadoc. Estos comentarios pueden modificarse en la ficha General de Propiedades de proyecto.

En muchos casos, cuando se abre un asistente de JBuilder sin que haya un proyecto abierto, aparece el Asistente para proyectos, que ofrece la posibilidad de crear un proyecto.

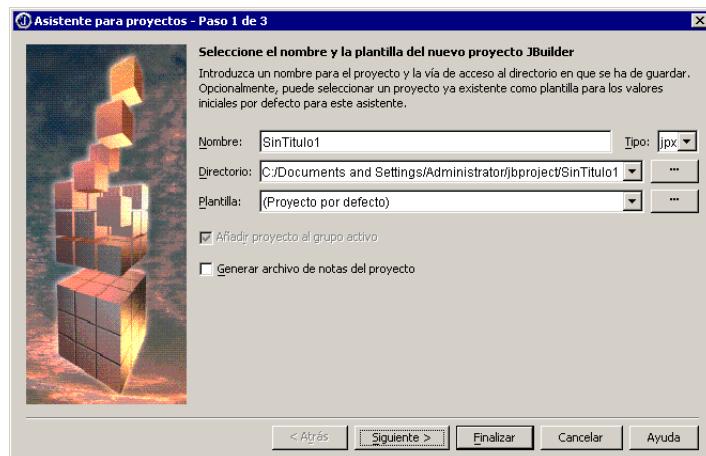
### Creación de proyectos con ayuda del Asistente para proyectos

---

Para crear un proyecto con el Asistente para proyectos, seleccione Archivo | Nuevo proyecto. También puede seleccionar Archivo | Nuevo, elegir la pestaña Proyecto y hacer doble clic en el ícono Proyecto. Se abre el asistente para proyectos.

## Selección del nombre del proyecto y la plantilla

Utilice el Paso 1 para definir el nombre, el tipo, el directorio raíz y la plantilla del proyecto.



**1** Escriba un nombre para el nuevo proyecto.

JBuilder utiliza el nombre del proyecto como nombre de paquete por defecto.

Cualquier nombre de archivo válido para el sistema de archivos se puede utilizar como nombre del proyecto. Sin embargo, hay otros nombres que se derivan de este nombre de archivo y tienen ciertas restricciones que hay que cumplir:

- a** El nombre del directorio del proyecto puede aparecer en una vía de acceso a clases Java. Dado que los espacios incrustados pueden ocasionar problemas, se sustituyen por guiones bajos.
  - b** El asistente utiliza el nombre del proyecto como nombre de paquete por defecto. Por tanto, debe ser un nombre de paquete Java válido. Esto significa que los números se eliminan del nombre del archivo, los espacios se sustituyen por guiones bajos, las mayúsculas se cambian por minúsculas y el nombre puede cortarse si es demasiado largo.
- 2** Seleccione el directorio del proyecto. El directorio del proyecto es el que contiene el archivo del proyecto. Casi todas las demás vías de acceso del proyecto, como las vías de acceso a fuentes y a copias de seguridad, se derivan de este directorio.
- Pulse la tecla de flecha abajo para seleccionar un directorio superior utilizado previamente o para elegir uno del mismo árbol y modificarlo.
  - Puede escribir en el campo directamente o pulsar el botón de puntos suspensivos para buscar un directorio ya creado.

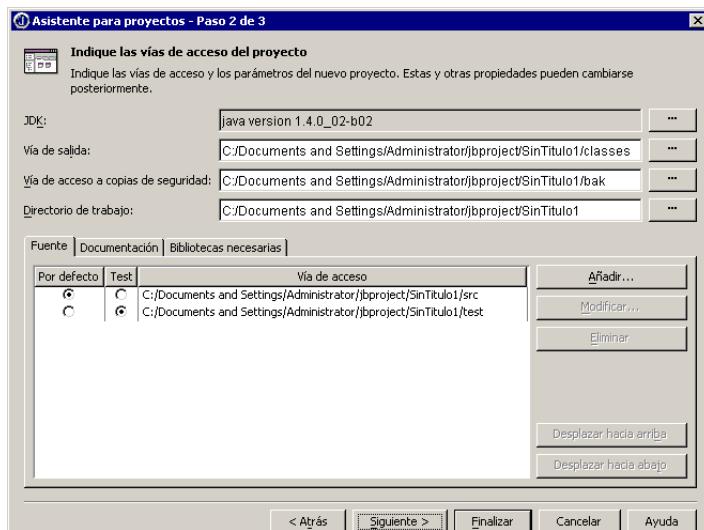
**Nota** Si la vía de acceso tiene errores de sintaxis, no podrá continuar.

- 3 Acepte Proyecto por defecto como el valor del campo Plantilla. (Puede pulsar el botón de ayuda si desea leer más acerca de las plantillas de proyectos.)
- 4 Para añadir el proyecto que está creando a un grupo de proyectos ya creado (el grupo de proyectos debe estar activo), marque la casilla de selección Añadir proyecto al grupo activo. Sólo se puede activar esta casilla si tiene abierto y activo un grupo de proyectos. Los grupos de proyectos solamente están disponibles en JBuilder Enterprise. Si desea obtener más información sobre los grupos de proyectos, consulte el [Capítulo 3, "Los grupos de proyectos"](#).
- 5 Para generar un archivo de notas de un proyecto HTML, marque la casilla de selección Generar archivo de notas del proyecto. Este archivo es opcional.
- 6 Pulse Siguiente para pasar al Paso 2.

Si está habilitado el botón Finalizar, cuando se pulsa se aceptan las opciones por defecto de JBuilder para el resto del asistente y el proyecto se crea inmediatamente.

## Configuración de las vías de acceso del proyecto

En el Paso 2 se establecen todas las vías de acceso del proyecto, incluida la versión del JDK que se utiliza para compilar. Si lo cree necesario, puede cambiar esta configuración más adelante mediante la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto).



JBuilder propone como directorio de trabajo el directorio de proyecto elegido en el Paso 1. El directorio de trabajo es el directorio inicial que JBuilder proporciona a los programas cuando se inician. Se puede configurar cualquier directorio como directorio de trabajo.

Si desea modificar alguna de las vías de acceso de esta ficha, puede escribir la nueva o desplazarse hasta ella mediante el botón de puntos suspensivos contiguo al campo apropiado.

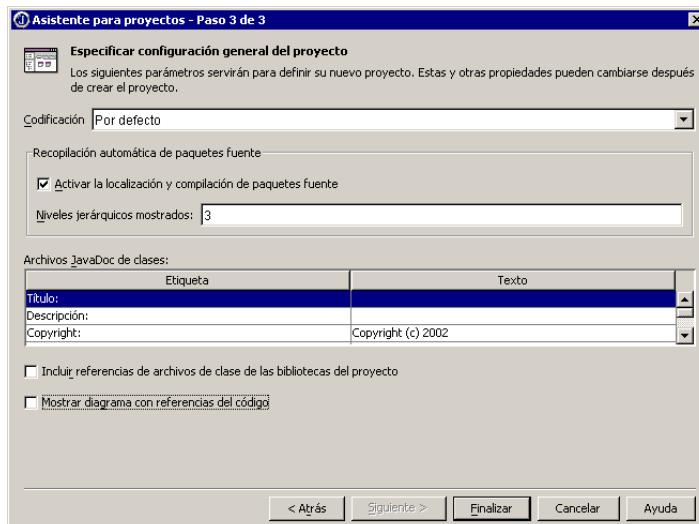
**Nota** Si acaba de empezar a utilizar JBuilder, sólo tiene que aceptar los valores por defecto de esta ficha. Si se escribe una vía de acceso con errores de sintaxis no se puede continuar. Los usuarios con más experiencia quizás prefieran cambiar estos directorios por los de su elección. Si desea obtener información más completa acerca del uso de esta ficha y sobre los directorios, pulse el botón de ayuda del Asistente para proyectos.

Pulse Siguiente para continuar con el Paso 3 o bien, pulse Finalizar para crear su proyecto. Los usuarios con más experiencia quizás prefieran continuar hasta el Paso 3.

## Configuración de las opciones generales del proyecto

El Paso 3 del Asistente para proyectos incluye las opciones generales del proyecto, como la codificación, la configuración para la ejecución por defecto, la recopilación automática de paquetes fuente, los campos de clases Javadoc y las referencias de las bibliotecas del proyecto.

Esta información se puede modificar más adelante en la ficha General y la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto).



- 1 Seleccione una codificación o acepte la que se ofrece por defecto. La codificación determina cómo gestiona JBuilder los caracteres que no pertenecen al conjunto de caracteres ASCII. La opción por defecto es la página de códigos del sistema operativo.

### Consulte

- “Elección de una codificación nativa para el compilador” en la página 16-13
- “Internationalization Tools: native2ascii” at <http://java.sun.com/products/jdk/1.4/docs/tooldocs/tools.html#intl>

- 2 Seleccione la opción Activar assert como palabra clave si desea que assert se reconozca como una palabra clave.

JBuilder admite aserciones JDK 1.4. Antes de JDK 1.4, assert era una palabra clave reservada para un uso posterior. Con JDK 1.4, la palabra clave assert se ha añadido al lenguaje y se utiliza para las órdenes. assert toma un valor booleano, que comprueba una condición que debe cumplir la expresión asociada para ejecutarse. Las órdenes se activan y se desactivan durante la ejecución.

### Consulte

- <http://java.sun.com/j2se/1.4/docs/guide/lang/assert.html>

Es una función de  
JBuilder SE y Enterprise.

- 3 Seleccione Opciones de recopilación automática de paquetes fuente.

- a La opción Activar la localización y compilación de paquetes fuente está activada por defecto. Si está activada tienen lugar varias acciones:

- Todos los paquetes de la vía de acceso a fuentes del proyecto aparecen en el panel del proyecto (panel superior izquierdo) del IDE.
- Los paquetes que contienen archivos Java se compilan de forma automática.
- Los archivos generados en esta compilación se copian en la vía de archivos generados del proyecto.

No se muestran todos los paquetes, solo un subconjunto lógico determinado por el nivel de profundidad con el que JBuilder expone los paquetes.

- b Indique hasta qué nivel desea que se expongan los paquetes.

JBuilder expone los paquetes hasta el nivel que se defina aquí, excepto si la adición de niveles a un paquete no modifica la longitud de la lista de paquetes. Por ejemplo, si cuenta con estos tres paquetes:

```

uno.dos.tres.cuatro
uno.dos.tres.cinco
uno.dos.cuatro.seis

```

y define el nivel de paquetes en tres, esto es lo que aparece en el panel del proyecto:

```

uno.dos.tres
uno.dos.cuatro.seis

```

Los paquetes uno.dos.tres.cuatro y uno.dos.tres.cinco están en uno.dos.tres, por lo que JBuilder sólo representa el paquete superior y permite ampliar el nodo para tener acceso a los paquetes y archivos que contiene.

El paquete uno.dos.cuatro.seis se expone en el cuarto nivel, ya que si se acorta la representación no se acorta la lista de paquetes, por lo que se presenta de todas formas.

### Consulte

- “Paquetes” en la página 4-6

**4** Especifique los campos de la clase Javadoc. Se pueden utilizar en el archivo de notas del proyecto y en el cuadro de diálogo Ayuda | Acerca de, de la aplicación, y se pueden insertar como comentarios de cabecera Javadoc en los archivos generados por asistentes.

Seleccione el campo que desea modificar y escriba el texto deseado en la columna Texto.

**5** Active la opción Incluir referencias de archivos de clase de las bibliotecas del proyecto si desea que sea posible buscar referencias a todas las bibliotecas del proyecto. El comando Buscar referencias del menú contextual del editor permite detectar todos los archivos fuente en los que se utiliza un símbolo determinado. Actívela también si desea que los diagramas UML del proyecto muestren las referencias de las bibliotecas del proyecto.

### Consulte

- “Búsqueda de referencias a símbolos” en la página 12-9
- Capítulo 11, “Presentación de código con UML”

**6** Si tiene JBuilder Enterprise y desea incluir referencias tales como archivos IIOP o stubs EJB en los diagramas UML del proyecto, seleccione la opción Mostrar diagrama con referencias del código.

### Consulte

- Capítulo 11, “Presentación de código con UML”

Buscar referencias es una característica de JBuilder SE y Enterprise

UML es una característica de JBuilder Enterprise

**7** Pulse Finalizar cuando termine.

JBuilder crea un nuevo proyecto que aparece en el panel del proyecto.

Si más adelante desea cambiar la configuración que especificó para su proyecto mediante el Asistente para proyectos, puede hacerlo con la ayuda de la ficha Vías de acceso y la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto).

## Creación de proyectos a partir de archivos anteriores

---

Es una función de JBuilder SE y Enterprise.

El asistente Proyecto para código existente permite crear un proyecto en JBuilder a partir del trabajo ya realizado. Cuando se utiliza este asistente, JBuilder examina el directorio actual y crea vías de acceso para compilar, buscar, depurar y realizar otros procesos. Todos los archivos JAR y ZIP que no se encuentran en bibliotecas se colocan en una nueva biblioteca y se añaden al proyecto. Las bibliotecas del proyecto se recogen en la pestaña Bibliotecas necesarias de la ficha Vías de acceso de Propiedades de proyecto (Proyecto | Propiedades de proyecto).

Para abrir el asistente Proyecto para código existente:

- 1** Seleccione Archivo | Nuevo. Se abre la galería de objetos.
- 2** Seleccione la pestaña Proyecto.
- 3** Haga doble clic sobre el ícono Proyecto para código existente.

## Selección del directorio fuente y el nombre del nuevo proyecto de JBuilder

---

En el Paso 1 se configuran el directorio, el nombre, el tipo y la plantilla del proyecto.

- 1** Seleccione el directorio donde se encuentra el proyecto o el árbol de fuente. Pulse el botón de puntos suspensivos para buscarlo. JBuilder examina el directorio seleccionado en busca de archivos de clase, de fuente, JAR y ZIP, y los coloca en los subdirectorios adecuados de ese directorio.
- 2** Escriba un nombre para el nuevo proyecto. JBuilder utiliza por defecto el nombre del proyecto como nombre del paquete, por lo que si existe un nombre de paquete anterior es conveniente utilizarlo como nombre del proyecto. Los asistentes de JBuilder también utilizan el nombre del proyecto como nombre del paquete, que se puede modificar en los asistentes.
- 3** Seleccione la plantilla del proyecto.

- Pulse la tecla de flecha abajo para seleccionar un proyecto que se haya utilizado anteriormente como plantilla.
- Pulse el botón de puntos suspensivos para utilizar como plantilla un proyecto diferente en el cuadro de diálogo Abrir proyecto.

Las plantillas de proyectos ofrecen valores por defecto para las opciones descritas en el cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto). Si ya cuenta con un proyecto de JBuilder de propiedades parecidas a las que se necesitan en el nuevo, selecciónelo aquí. Así se reducen al mínimo las tareas repetitivas que conlleva la configuración de un nuevo proyecto en un entorno ya establecido.

- 4** Indique si desea crear un archivo HTML de notas del proyecto. La información inicial de este archivo, como el título, el autor y la descripción, se genera desde los campos de la clase Javadoc configurados en el Paso 3 del asistente Proyecto para el código existente. En caso necesario también es posible añadir notas y otra información en este archivo.

- 5** Pulse Siguiente para pasar al Paso 2.

Los pasos 2 y 3 del asistente Proyecto para el código existente son los mismos que en el Asistente para proyectos. Estos pasos también coinciden con la ficha Vías de acceso y la ficha General de Propiedades de proyecto. Consulte “[Creación de proyectos con ayuda del Asistente para proyectos](#)” en la página 2-2.

Si el proyecto necesita alguna biblioteca determinada, puede añadirla en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto. Para configurar la clase principal para ejecutar el proyecto, seleccione la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto.

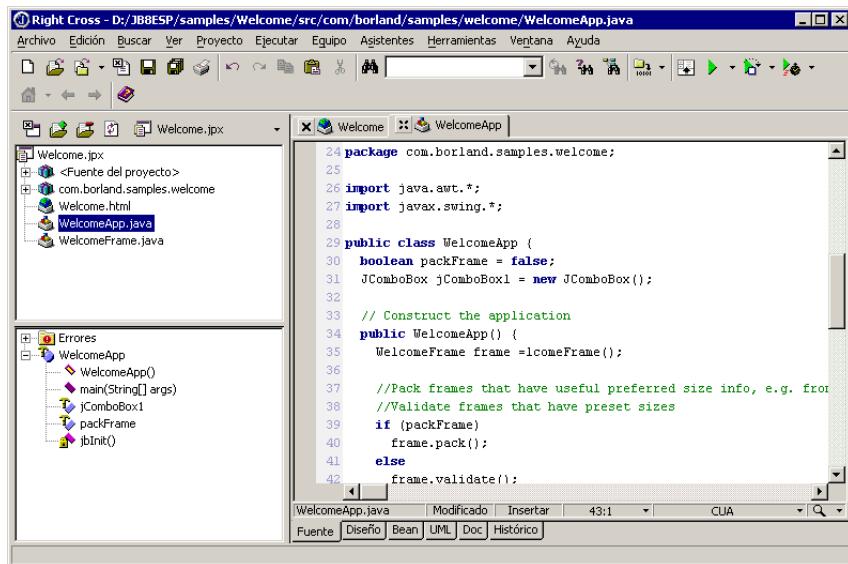
### Consulte

- “[Configuración de vías de acceso a bibliotecas necesarias](#)” en la página 2-25
- “[El comando Ejecutar](#)” en la página 7-4

# Presentación de los archivos

JBuilder muestra todos los archivos abiertos de un proyecto en el panel de contenido del Visualizador de aplicaciones. Haga doble clic en un archivo para abrirlo en el panel de contenido. En la parte superior aparece una pestaña con el nombre del archivo.

La siguiente figura muestra en el panel del proyecto un archivo de proyecto, Welcome.jpr, con los archivos fuente enumerados debajo. Este proyecto incluye un paquete y tres archivos fuente. Aparecen dos archivos abiertos en el panel de contenido, y el archivo WelcomeApp.java seleccionado se muestra en el panel fuente.



Haga clic con el botón derecho en el archivo de proyecto para mostrar un menú con opciones como Abrir, Añadir archivos/paquetes, Eliminar del proyecto, Cerrar proyecto, Ejecutar Make, Generar de nuevo y Propiedades. También se puede acceder a muchas de estas opciones de menú desde el menú Proyecto.

## Cambio entre archivos

Cuando hay muchos archivos abiertos no siempre es fácil examinar todas las pestañas de archivos abiertos en busca de la que desea utilizar. Existen dos modos de cambiar rápidamente entre los archivos abiertos:

- Seleccionar Ventana | Cambiar a o pulsar *Ctrl+B* para abrir el cuadro de diálogo, que enumera todos los archivos abiertos del proyecto. Desplácese para seleccionar el archivo que desea. O empiece a escribir el nombre del archivo y se seleccionará la primera coincidencia en la lista,

continúe escribiendo hasta que se seleccione el archivo que desea. Una vez seleccionado, pulse Aceptar.

- Seleccione Ventana para abrir el menú Ventana y seleccione el archivo que desee en la lista de archivos abiertos enumerados en el menú. El archivo activo se identifica mediante una marca.

El menú Ventana también permite cambiar entre *proyectos* abiertos.

<b>Sugerencia</b>	Si lo prefiere, puede tener pestañas de los archivos abiertas verticalmente a la derecha del panel de contenido en lugar de en la parte superior. Seleccione Herramientas   Opciones del IDE y en las opciones de la pestaña Panel de contenido, seleccione Vertical en la lista desplegable Orientación. Las opciones de la ficha Panel de contenido también ofrecen otras opciones para mostrar las pestañas.
-------------------	---

## Almacenamiento de proyectos

---

Cuando se trabaja con un proyecto, se puede guardar en la ubicación propuesta o en un directorio diferente. Por defecto, JBuilder guarda los proyectos en el subdirectorio `jbproject` del directorio inicio, aunque esto depende de la configuración del sistema. Cada proyecto se guarda dentro de su propio directorio, dentro de `jbproject`. Los directorios de proyecto incluyen un archivo del proyecto, un archivo `.html` optativo para las notas del proyecto, un subdirectorio `classes` para los archivos generados (como los archivos `.class`), un subdirectorio `src` para los archivos de código fuente, un subdirectorio `bak` para los archivos de copia de seguridad y un subdirectorio `doc` para la documentación.

### Almacenamiento y cierre de proyectos

Para guardar un proyecto, seleccione Archivo | Guardar todos, Archivo | Guardar proyecto o haga clic en el botón Guardar todos de la barra de herramientas principal.

Para cerrar un proyecto, seleccione Archivo | Cerrar proyectos o haga clic en el botón Cerrar proyecto de la barra de herramientas del proyecto.

### Consulte

- “Cómo construye JBuilder las vías de acceso” en la página 4-10
- “Localización de los archivos” en la página 4-13



## Apertura de proyectos

Sólo hay un modo de abrir un proyecto la primera vez: mediante Archivo | Abrir proyecto. Hay dos modos de abrir un proyecto que se haya abierto previamente: con el comando Archivo | Abrir proyecto y con Archivo | Abrir de nuevo.

Para abrir un proyecto mediante el comando Archivo | Abrir proyecto:

- 1 Seleccione Archivo | Abrir proyecto. Se muestra el cuadro de diálogo Abrir archivo.
- 2 Desplácese hasta el directorio que contenga el archivo de proyecto que desea abrir.
- 3 Seleccione el archivo de proyecto y pulse Aceptar o *Intro*. También puede hacer doble clic en el archivo de proyecto para abrirlo.

Para abrir un proyecto previamente abierto, con el comando Archivo | Abrir de nuevo:

- 1 Elija Archivo | Abrir de nuevo.
- 2 Seleccione el proyecto que desea abrir en la lista de los proyectos abiertos con anterioridad.

El archivo de proyecto y sus archivos fuente se muestran en el panel del proyecto.

Para abrir un archivo en el panel de contenido puede seguir uno de estos tres métodos:

- Hacer doble clic en el archivo, en el panel del proyecto.
- Seleccionar el archivo en el panel del proyecto y pulsar *Intro*.
- Hacer clic con el botón derecho del ratón en el archivo, en el panel del proyecto, y seleccionar Abrir.

Para ver un proyecto en un nuevo Visualizador de aplicaciones:

- 1 Seleccione Ventana | Nueva ventana.
- 2 Seleccione Archivo | Abrir proyecto y desplácese hasta el archivo en el cuadro de diálogo Abrir proyecto.

Si hay varios visualizadores abiertos y en todos aparecen los mismos archivos, los cambios realizados en un visualizador se reflejan inmediatamente en el mismo archivo abierto, en los otros visualizadores. Así se mantiene la coherencia de todas las versiones de trabajo de un archivo.

**Nota** Todos los proyectos abiertos están disponibles en la lista desplegable de proyectos de los Visualizadores de aplicaciones.

# Creación de un archivo fuente Java

Hay varias formas de crear archivos fuente de Java con JBuilder. La mayoría de los asistentes pueden crear archivos. Se accede a algunos de ellos mediante la galería de objetos (Archivo | Nuevo), y a otros, desde el menú Asistentes. Más concretamente, el Asistente para clases genera el marco de una nueva clase Java.

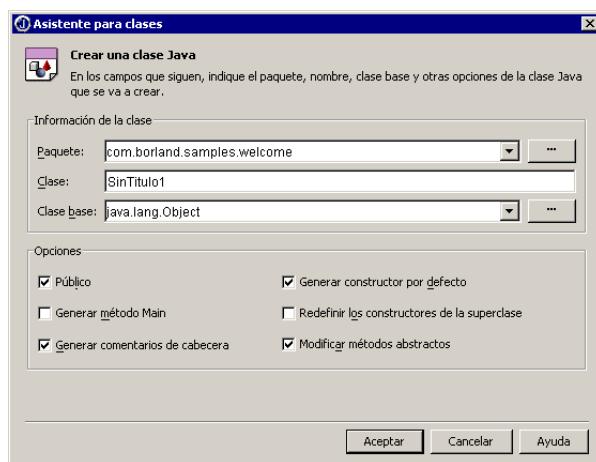
Para crear un archivo fuente Java vacío:

- 1 Seleccione Archivo | Archivo nuevo para abrir el cuadro de diálogo Crear archivo.
- 2 Escriba el nombre del archivo en el campo Nombre.
- 3 Seleccione el tipo de archivo .java del menú desplegable o incluya la extensión cuando escriba el nombre.
- 4 Si desea modificar el directorio en el que se guardan los archivos, escriba el nuevo directorio en el campo Directorio o pulse el botón de puntos suspensivos para seleccionar el directorio que deseé.
- 5 Pulse Aceptar.

JBuilder crea el archivo y lo abre en el panel de contenido.

Para crear un archivo fuente Java con el Asistente para clases:

- 1 Cree un proyecto, según se describe en “[Creación de proyectos](#)” en la [página 2-2](#).
- 2 Seleccione Archivo | Nueva clase para iniciar el Asistente para clases.



- 3 Escriba los nombres de paquete, clase y clase base en el Asistente para clases.
- 4 Configure las opciones de exposición, gestión de métodos y comentarios de cabecera.
- 5 Pulse Aceptar.

El archivo .java se crea y se añade al proyecto (su nodo aparece en el panel del proyecto). El archivo se abre en el panel de contenido del editor.

### Consulte

- “Cómo añadir elementos a un proyecto” en la página 2-14
- “Paquetes” en la página 4-6
- “Configuración de las opciones generales del proyecto” en la página 2-5

Para guardar un archivo tras haberlo modificado, seleccione Archivo | Guardar o haga clic sobre el icono Guardar. La vía de acceso y el directorio superior del archivo se muestran en la parte superior de la ventana del Visualizador de aplicaciones cuando se selecciona y se abre el archivo. También puede guardar los archivos del proyecto seleccionando Archivo | Guardar todo o pulsando el icono Guardar todo.



## Gestión de proyectos

---

JBuilder tiene como finalidad asistir a los desarrolladores en la realización de la mayor cantidad posible de tareas de desarrollo. Las herramientas de proyectos de JBuilder, el completo IDE y las amplias funciones del editor automatizan y simplifican la tarea del desarrollo y permiten concentrarse en el código.

### Cómo añadir elementos a un proyecto

---

En el Visualizador de aplicaciones se puede añadir al proyecto carpetas, archivos nuevos y archivos ya existentes. Muchos de estos comandos están disponibles tanto en el menú contextual del panel del proyecto como en el menú principal.

En el caso de proyectos más grandes se pueden utilizar carpetas de proyecto para organizar la jerarquía del proyecto.

**Nota** Las carpetas de proyectos sólo se utilizan para fines organizativos y no se corresponden con los directorios del disco.

## Cómo añadir carpetas

Las carpetas de proyecto no afectan al árbol de directorios. Son herramientas de organización que permiten ordenar los elementos de un proyecto de una forma útil sin afectar a la estructura de directorios.

Para añadir una carpeta de proyecto:

- 1** Seleccione el proyecto en el panel del proyecto (superior izquierdo).
- 2** Elija Archivo | Nueva carpeta en el menú principal o haga clic con el botón derecho del ratón en el panel del proyecto.
- 3** Seleccione Nueva carpeta.

### Sugerencia

Para anidar la nueva carpeta dentro de otra ya creada, seleccione esa carpeta antes de elegir Archivo | Nueva carpeta.

- 4** Escriba el nombre de la carpeta.
- 5** Pulse Aceptar o *Intro*.

Para añadir un archivo a una carpeta:

-  **1** Haga clic en la carpeta con el botón derecho del ratón y pulse Añadir archivos/paquetes para abrir el cuadro de diálogo Añadir archivos o paquetes al proyecto, o seleccione la carpeta y pulse el botón Añadir archivos/paquetes de la barra de herramientas del panel del proyecto.
- 2** Desplácese por la ficha Explorador del cuadro de diálogo Añadir archivo o paquetes al proyecto para acceder al directorio que contiene el archivo que desea añadir.
- 3** Seleccione el archivo y pulse Abrir.

## Cómo añadir archivos y paquetes

La detección automática de las vías de acceso a fuente, una función de JBuilder SE y Enterprise, añade archivos y paquetes al proyecto sobre la marcha. Puede configurar esta opción en la ficha General del cuadro de diálogo Propiedades de proyecto. Está activada por defecto.

Si no se utiliza esta función, para que JBuilder considere que los archivos y paquetes pertenecen al proyecto es necesario añadirlos expresamente. Puede añadir archivos y paquetes al proyecto actual mediante el cuadro de diálogo Añadir archivos o paquetes al proyecto.

Este cuadro de diálogo se abre de dos formas: Utilice la que prefiera:

- 
- Pulse el botón Añadir Archivos/Paquetes en la barra de herramientas del proyecto.
  - Haga clic con el botón derecho del ratón en cualquier nodo del panel del proyecto y seleccione Añadir archivos/paquetes en el menú contextual.

Cuando aparezca el cuadro de diálogo, siga estos pasos:

- 1 Seleccione la ficha Explorador para añadir un archivo, la ficha Paquetes si desea añadir un paquete o la ficha Clases para añadir una clase. Todas estas fichas admiten la selección múltiple.
- 2 Desplácese hasta el archivo, clase o paquete que desea importar
- 3 y selecciónelo. Cuando se encuentre en el directorio superior de un archivo puede escribir su nombre en lugar de seleccionarlo.
- 4 Haga doble clic en la selección, pulse el botón Aceptar o pulse la tecla *Intro*.

El nuevo nodo aparece en el directorio del proyecto, dentro del panel del proyecto.

## Eliminación de partes de un proyecto

---

Es posible eliminar carpetas, archivos, clases y paquetes del proyecto sin borrarlos de la unidad de almacenamiento mediante el cuadro de diálogo Eliminar del proyecto.

Para eliminar un nodo de su proyecto, seleccione una de las siguientes opciones:

- Pulse con el botón derecho del ratón sobre el nodo que desee eliminar, seleccione Eliminar del proyecto y, a continuación, pulse Aceptar.
- Seleccione el nodo que desee eliminar, pulse el botón Eliminar del proyecto de la barra de herramientas del panel del proyecto y, a continuación, pulse Aceptar.



**Nota** Si una carpeta contiene archivos, éstos también serán eliminados del proyecto.

También es posible seleccionar varios nodos y eliminarlos del proyecto.

## Eliminación de partes del proyecto

---

También es posible eliminar del disco archivos, clases y paquetes. Para ello, haga clic en el archivo con el botón derecho del ratón y seleccione Borrar.



**Advertencia** Este comando elimina permanentemente los archivos del proyecto y del disco duro del ordenador.

## Apertura de archivos fuera de un proyecto

---

Utilice el comando Archivo | Abrir archivo para abrir un archivo en el Visualizador de aplicaciones sin añadirlo al proyecto abierto. Para que esta opción esté disponible debe haber un proyecto abierto.

Para abrir un archivo sin añadirlo al proyecto:

- 1 Seleccione Archivo | Abrir archivo. Se muestra el cuadro de diálogo Abrir archivo.
- 2 Seleccione el archivo que desea abrir.
- 3 Pulse Aceptar. En el Visualizador de aplicaciones aparece el contenido del archivo.

**Es una característica de JBuilder SE y Enterprise.**

También se puede abrir un archivo en el proyecto abierto en otro Visualizador de aplicaciones.

- 1 Haga clic con el botón derecho del ratón en un archivo, en el panel del proyecto.
- 2 Seleccione Abrir en nueva ventana.

## Cómo cambiar el nombre a proyectos y archivos

---

Para cambiar el nombre de un proyecto o archivo:

- 1 Seleccione el proyecto o el archivo en el panel del proyecto.
- 2 Seleccione Proyecto | Renombrar o Archivo | Renombrar, o haga clic con el botón derecho en el panel del proyecto y seleccione Renombrar.
- 3 Escriba el nuevo nombre en el cuadro de diálogo Cambiar nombre y pulse Aceptar.

También puede cambiar el nombre de un archivo abierto mediante la pestaña del archivo de la parte superior del panel de contenido:

- 1 Haga clic con el botón derecho en la pestaña del archivo, situada en la parte superior del panel de contenido.
- 2 Seleccione Cambiar nombre.
- 3 Escriba el nuevo nombre en el cuadro de diálogo Cambiar nombre y pulse Aceptar.

**Nota** El cambio de nombre de un archivo no modifica su tipo. Si desea cambiar la extensión del archivo, utilice Archivo | Guardar como.

**Advertencia** Al renombrar proyectos y archivos no se cambian los nombres de paquete y archivo referenciados dentro del código. JBuilder SE y Enterprise ofrecen la función de perfeccionamiento por cambio de nombre, que

sustituye el nombre antiguo por el nuevo en todos los lugares donde aparece.

### Consulte

- [Capítulo 12, “Perfeccionamiento de símbolos de código”](#)

## Adición de un nuevo directorio de navegación

---

Es una función de JBuilder SE y Enterprise.

Se puede añadir un nuevo nodo al panel del proyecto que apunte al directorio de su elección. Por ejemplo, puede tener código que se encuentre enterrado en el fondo de la estructura de su proyecto. Sólo tiene que añadir el directorio que contenga ese código como un nodo de proyectos del panel del proyecto. Cuando abra el nodo, tendrá acceso inmediato al código, en lugar de tener que navegar por una complicada estructura de directorios.

Un directorio de navegación es una vista de su directorio o árbol de directorios, que muestra todos los tipos de archivos. Una vez creado una vista de directorios y cuando los cambios se han producido en el contenido actual de el directorio o sus subdirectorios, la vista de directorio se actualiza en el panel del proyecto (es posible que se necesite pulsar el botón Actualizar de la barra de proyectos para actualizar el proyecto).

Si se extrae un proyecto mediante CVS, se crea automáticamente un nodo de directorios de navegación que muestra el árbol de directorios completo del proyecto con los subdirectorios de CVS ocultos. Esto proporciona la posibilidad de encontrar en el proyecto todo tipo de archivos, sin tener en cuenta el tipo, y seleccionarlos para realizar operaciones en CVS.

Para añadir un directorio de navegación a un proyecto:

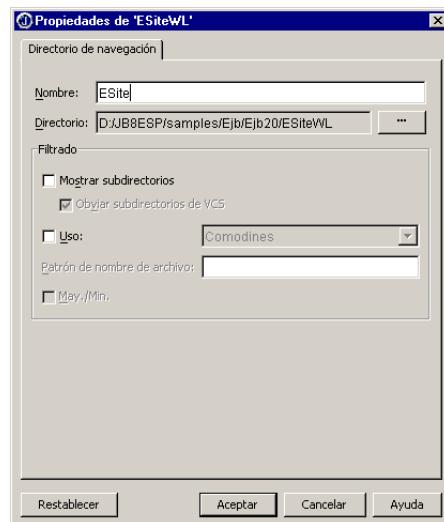
- 1 Seleccione Proyecto | Nuevo directorio de navegación, o bien, pulse con el botón derecho del ratón sobre el nodo de proyectos (el archivo .jpx) en el panel del proyecto y, a continuación, seleccione Nuevo directorio de navegación.
- 2 Desplácese hasta el directorio que desee para que aparezca directamente en el panel del proyecto.
- 3 Pulse Aceptar, y el directorio que haya seleccionado se añade como un nodo al panel del proyecto.
- 4 Abra el nodo para ver los archivos que se incluyen en el directorio.

Puede personalizar los archivos y subdirectorios que aparecen cuando se abre el directorio de navegación del nodo de proyectos. Por ejemplo, puede elegir que aparezcan sólo los archivos .java o los archivos .html. También puede tener varios directorios de navegación en un mismo proyecto, para que uno de ellos muestre los archivos .java, otro los archivos .html y otro todos los archivos que empiecen con la letra 'a'.

Para personalizar el directorio de navegación:

- 1 Pulse con el botón derecho del ratón sobre el directorio de navegación del nodo de proyectos en el panel del proyecto.
- 2 Seleccione Propiedades.

Aparece el cuadro de diálogo Propiedades con una ficha Directorio de navegación:



- 3 Escriba un nombre descriptivo para el directorio de navegación del nodo de proyectos o mantenga el nombre por defecto, que es el nombre del directorio.
- 4 Utilice las opciones de filtrado para filtrar los archivos y los directorios que deben de aparecer al abrir el directorio de navegación del nodo de proyectos. Estos son ejemplos de los modelos de archivos:

```
*.java
myfile?.java
file??.*
```

**Nota**

No puede encadenar varios modelos de archivos juntos, como \*.java; \*.cpp.

Si desea obtener información más completa sobre el uso de estas opciones, pulse el botón de ayuda de este cuadro de diálogo.

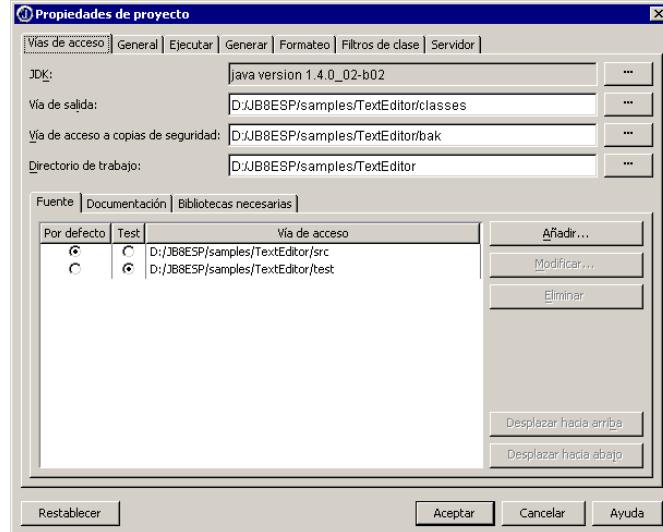
- 5 Pulse Aceptar.

## Definición de las propiedades del proyecto

Las propiedades del proyecto determinan la forma en que se compila. Mediante el cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto), puede definir la configuración de las vías de acceso, establecer una configuración de ejecución para el proyecto, definir el modo en que se ha de crear el proyecto, personalizar la apariencia de los diagramas UML, especificar las opciones del servidor y mucho más.

Para establecer las propiedades del proyecto:

- 1 Haga clic con el botón derecho en el nombre de archivo con extensión .jpx, dentro del panel del proyecto, y seleccione Propiedades para abrir el cuadro de diálogo Propiedades de proyecto. También puede seleccionar el proyecto y elegir Proyecto | Propiedades de proyecto. Aparece el cuadro de diálogo Propiedades de proyecto.
- 2 Seleccione la pestaña correspondiente a las opciones que desea configurar. En esta imagen, se ha seleccionado la ficha Vías de acceso:



A continuación, se incluye una breve descripción de lo que se puede hacer en cada una de las fichas del cuadro de diálogo Propiedades de proyecto:

- Ficha Vías de acceso: configuración de las vías de acceso del proyecto a la versión JDK, vía de salida, vía a copia de seguridad, directorio de trabajo, vías a archivos fuente, vía a test, a documentación y vías a bibliotecas necesarias.
- Ficha General: establece opciones para la codificación, la activación de la recopilación automática de paquetes fuente y la modificación los campos de los campos de clase Javadoc que generan los asistentes, así

como la configuración de la opción de incluir referencias de las bibliotecas del proyecto.

- Ficha Ejecutar: selecciona o crea la configuración que se utiliza durante la ejecución o la depuración.
- Ficha Generar: define las opciones del compilador para la creación de un proyecto. Esto incluye la información de depuración y la copia selectiva de recursos.
- Ficha Formateo: establece las opciones de formato del código. JBuilder puede acelerar la codificación formateando el código automáticamente de acuerdo con sus indicaciones. Es una característica de JBuilder Enterprise.
- Filtros de clase: establece las opciones para los filtros de clase.
- Ficha Servidor establece las opciones del servidor. Es una característica de JBuilder Enterprise.

**Nota** También se pueden definir estas opciones para los proyectos futuros, por medio del cuadro de diálogo Propiedades por defecto para proyectos (Proyecto | Propiedades por defecto para proyectos), o seleccionando el proyecto por defecto como plantilla de documento en el Asistente para proyectos.

## Configuración del JDK

---

En la ficha Vías de acceso se pueden configurar la versión del JDK, distintas vías de acceso para el proyecto y las vías de acceso a bibliotecas necesarias.

JBuilder SE y Enterprise incorporan la posibilidad de cambiar de JDK, mientras que JBuilder Personal solamente permite modificar un JDK. JBuilder compila y ejecuta con todos los JDK de Sun y muchos otros.

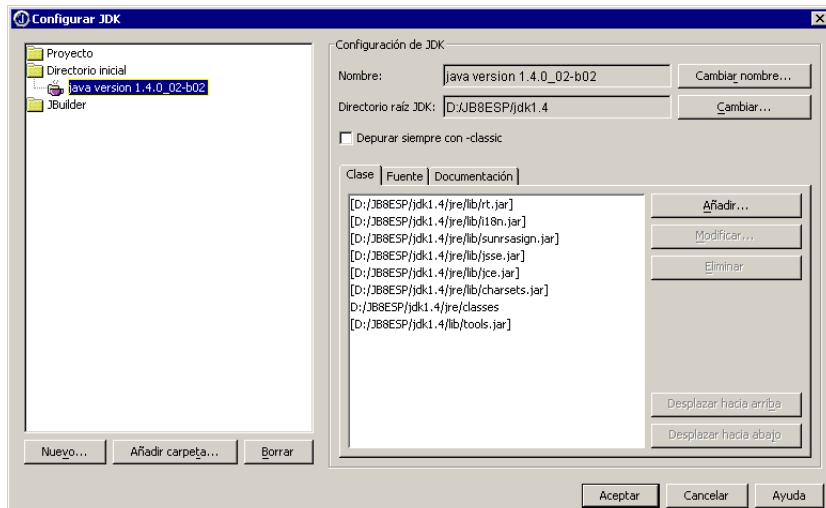
Con JBuilder SE y Enterprise, se puede configurar la versión JDK para el proyecto en la ficha Vías de acceso en el cuadro de diálogo Propiedades de proyecto, y añadir, modificar y eliminar los JDK en el cuadro de diálogo Configurar JDK. Consulte “[Configuración del JDK en SE y Enterprise](#)” en la página 2-23.

En JBuilder Personal se puede modificar el JDK en el cuadro de diálogo Configurar JDK (Herramientas | Configurar JDK). Consulte “[Modificación del JDK](#)” en la página 2-22.

## Modificación del JDK

La versión actual del JDK se puede modificar de la siguiente manera:

- 1 Seleccione Herramientas | Configurar JDK para abrir el cuadro de diálogo Configurar JDK.



- 2 Pulse el botón Cambiar, que se encuentra a la derecha del campo Directorio del JDK. Aparece el cuadro de diálogo Seleccione un directorio.
  - 3 Desplácese hasta el JDK de destino.
  - 4 Pulse Aceptar para cambiar el JDK.
- Tome nota del nuevo nombre del JDK y de la vía de acceso de raíz que aparecen en el cuadro de diálogo Configurar JDK.
- 5 Pulse Aceptar para cerrar el cuadro de diálogo Configurar JDK.

### Depuración con -classic

La opción Depurar siempre con -classic del cuadro de diálogo Configurar JDK proporciona un mejor rendimiento a los usuarios con versiones de JVM anteriores a la 1.3.1. JBuilder comprueba automáticamente si esta opción va a mejorar el rendimiento, y activa o desactiva la casilla según el resultado. Esta función está disponible en todas las versiones de JBuilder.

Cuando se efectúa la evaluación, JBuilder lleva a cabo dos comprobaciones:

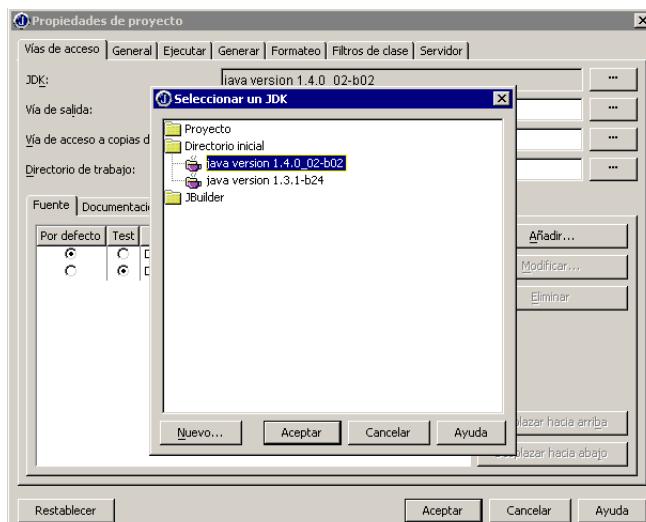
- 1 ¿Tiene instalada la MV Classic?
- 2 Si es así, ¿la versión de JVM es anterior a la 1.3.1?

Esta selección se modifica cuando se definen parámetros de MV como native, hotspot, green y server.

## Configuración del JDK en SE y Enterprise

JBuilder SE y Enterprise permiten cambiar de JDK. También es posible añadir, modificar y eliminar JDK. Para cambiar de JDK, siga estos pasos:

- 1 Seleccione Proyecto | Propiedades de proyecto y elija la pestaña Vías de acceso.
- 2 Pulse el botón de puntos suspensivos, a la derecha de la versión de JDK. Aparece el cuadro de diálogo Seleccione un JDK:



- 3 Si en la lista aparece el JDK de destino, selecciónelo y pulse Aceptar. Si no aparece en la lista, seleccione Nuevo para abrir el Asistente para JDK.



- a** Haga clic en el botón de puntos suspensivos y desplácese hasta el directorio raíz del JDK que desea añadir a la lista. Pulse Aceptar. El campo Nombre del JDK se rellena automáticamente.
  - b** Seleccione el lugar donde desea guardar las especificaciones JDK:
    - **Directorio inicial:** guarda las especificaciones JDK en un archivo `.library`, en el subdirectorio `.jbuilder` del directorio raíz del usuario. Guarde en esta ubicación si desea que el JDK esté disponible para todos los proyectos.
    - **JBuilder:** guarda las especificaciones JDK en un archivo `.library`, en el directorio `jbuilder`. Los diferentes usuarios que utilizan JBuilder en una red o lo comparten en una máquina pueden acceder a los JDK de esta carpeta. Es una característica de JBuilder SE y Enterprise.
    - **Proyecto:** guarda las especificaciones JDK en un archivo `.library`, en el directorio del proyecto actual. Guarde en esta ubicación si desea que el JDK sólo esté disponible para este proyecto. Es una característica de JBuilder SE y Enterprise.
    - **Carpeta definida por el usuario:** guarda las especificaciones JDK en una carpeta definida por el usuario o un directorio compartido. Para que la carpeta aparezca en la lista desplegable es necesario añadirla (elija Herramientas | Configurar JDK y pulse Añadir carpeta). Es una característica de JBuilder Enterprise.
  - c** Pulse Aceptar. La especificación JDK se añade al directorio especificado en el cuadro de diálogo Seleccione un JDK.
- 4 Pulse Aceptar para cerrar el cuadro de diálogo Seleccione un JDK. La vía de acceso al JDK se actualiza con la nueva.
- 5 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.
- 6 Guarde el proyecto. La versión de JDK se actualiza en el archivo de proyecto.

#### Sugerencia

Se pueden añadir, modificar y eliminar los archivos JDK seleccionando Herramientas | Configurar JDK. También se pueden modificar las propiedades de proyecto por defecto (Proyecto | Propiedades por defecto para proyectos), lo que cambia el JDK de todos los proyectos posteriores.

## Configuración de los JDK

---

La adición y el borrado de JDK son funciones de JBuilder SE y Enterprise.

Se pueden añadir, modificar y eliminar archivos JDK en el cuadro de diálogo Configurar JDK (Herramientas | Configurar JDK). Los usuarios de JBuilder Personal pueden modificar JDKs como se explica en “Modificación del JDK” en la página 2-22.

Este cuadro de diálogo permite:

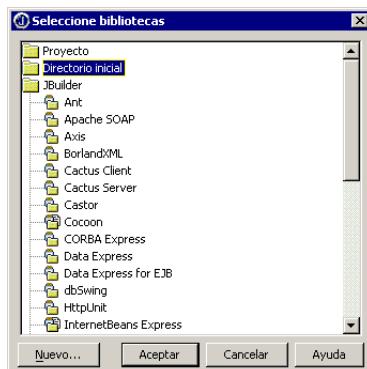
- Asignar un nombre al JDK por medio del botón Renombrar.
- Añadir, modificar, eliminar y reorganizar archivos de clase, fuente y documentación de JDK.
- Abrir el Asistente para JDK y añadir JDK por medio del botón Nuevo.
- Añadir una carpeta que otros usuarios puedan utilizar. Es una característica de JBuilder Enterprise.
- Eliminar un JDK de la lista.

### Consulte

- Botón Ayuda del cuadro de diálogo Configurar JDK.
- Botón Ayuda del Asistente para JDK.

## Configuración de vías de acceso a bibliotecas necesarias

En la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto se puede configurar qué bibliotecas se utilizarán al compilar. JBuilder coloca las bibliotecas seleccionadas en la vía de acceso a clases. Para añadir, modificar, eliminar y reordenar bibliotecas, seleccione la pestaña Bibliotecas necesarias.

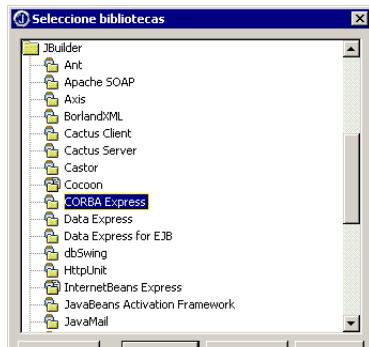


Se pueden seleccionar bibliotecas en la lista Bibliotecas necesarias de la ficha Vías de acceso y modificarlas, eliminarlas o cambiar su orden en la lista.

**Nota** Las bibliotecas se buscan por el orden en que figuran en la lista. Para cambiar el orden de las bibliotecas, seleccione una y pulse Desplazar hacia arriba o Desplazar hacia abajo.

Al hacer clic en el botón Añadir se abre el cuadro de diálogo Seleccione bibliotecas, en el que se puede elegir qué bibliotecas añadir al proyecto.

Seleccione Nuevo en el cuadro de diálogo para abrir el Asistente para bibliotecas y cree una biblioteca.



Puede configurar bibliotecas seleccionando Herramientas | Configurar bibliotecas.

#### Consulte

- “[Las bibliotecas](#)” en la página 4-1

## Trabajo con varios proyectos

---

Es posible trabajar simultáneamente con varios proyectos dentro del entorno de desarrollo de JBuilder. Se pueden abrir en un solo Visualizador de aplicaciones o en varios. La lista desplegable Proyecto de cualquier Visualizador de aplicaciones abierto proporciona acceso a todos los proyectos abiertos. Cualquier cambio realizado en un Visualizador de aplicaciones se refleja en los otros visualizadores en los que aparezca el mismo proyecto. Para abrir otro Visualizador de aplicaciones de JBuilder basta con seleccionar Ventana | Nueva ventana.

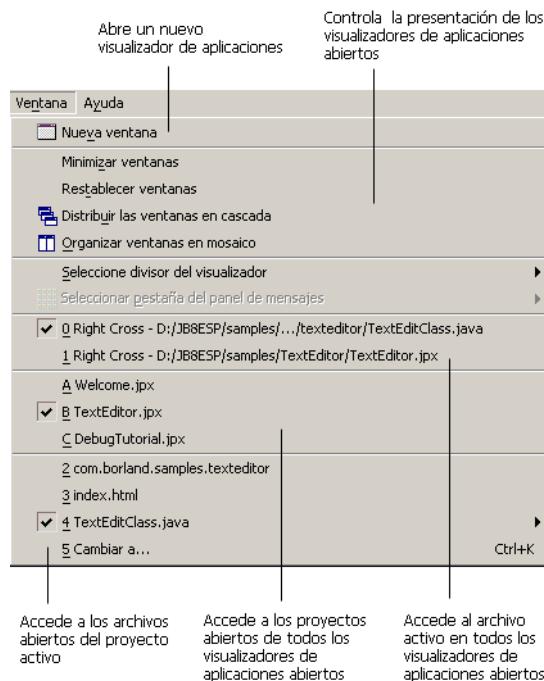
Si tiene JBuilder Enterprise, puede también agrupar varios proyectos en grupos de proyectos. Los grupos de proyectos son particularmente útiles cuando se está trabajando con proyectos relacionados. Si desea obtener más información sobre los grupos de proyectos, consulte el [Capítulo 3, “Los grupos de proyectos”](#).

#### Cambio de un proyecto a otro

---

Si en el Visualizador de aplicaciones hay varios proyectos abiertos, sólo uno es visible en el panel del proyecto. Para pasar a otro proyecto abierto, selecciónelo en la lista desplegable Proyectos de la barra de herramientas que está por encima del panel del proyecto.

El menú Ventana permite acceder a los visualizadores de aplicaciones, proyectos y archivos abiertos:



Seleccione Nueva ventana para abrir otro Visualizador de aplicaciones. Las ventanas del visualizador se controlan con los cuatro comandos siguientes. Seleccione el archivo abierto o el archivo del proyecto de un visualizador distinto para cambiar entre ventanas de visualizador.

En este menú también se puede acceder a los archivos abiertos en el visualizador actual. Esto ofrece un modo alternativo de tener acceso a los archivos abiertos, y resulta muy útil cuando se prefiere no mostrar los nombres de los archivos en las pestañas.

## Guardar varios proyectos

Para guardar los cambios realizados en todos los archivos y proyectos, seleccione Archivo | Guardar todo. Se guardan todos los archivos de todos los Visualizadores de aplicaciones abiertos.

## Información adicional acerca de los proyectos

---

Mientras trabaja con los proyectos en JBuilder, debe comprender cómo utiliza JBuilder las vías de acceso, para que pueda sacar mayor provecho de funciones tales como las bibliotecas o CodeInsight. Consulte [Capítulo 4, “Gestión de las vías de acceso”](#).

La capacidad de agrupar  
proyectos es una función  
de JBuilder Enterprise

JBuilder permite agrupar los proyectos en grupos de proyectos. Si desea obtener más información sobre los grupos de proyectos, consulte el [Capítulo 3, “Los grupos de proyectos”](#).

# Los grupos de proyectos

Es una función de JBuilder Enterprise.

Los grupos de proyectos son contenedores para proyectos, y pueden resultar de gran utilidad cuando se trabaja con proyectos relacionados. Por ejemplo, puede tener dos proyectos que tengan dependencias entre ellos, como un cliente y un servidor. Otra agrupación lógica podría incluir proyectos que utilizan los mismos archivos fuente pero que tienen configuraciones diferentes, como servidores de aplicaciones de destino distintos o diferentes JDK. Además, los grupos de proyectos proporcionan otras ventajas, como la fácil navegación entre proyectos y la generación de proyectos como grupos.

Los grupos de proyectos sólo pueden contener proyectos, y no otros grupos de proyectos. El grupo de proyectos se guarda como un archivo XML con la extensión .jpgr y, por defecto, en la raíz del directorio jbproject. Al contrario que los proyectos, los grupos de proyectos no se guardan en una carpeta para grupos de proyectos, sino como un archivo. Los proyectos, por sí mismos, no son conscientes de la existencia de los grupos de proyectos, y se pueden abrir simultáneamente de forma autónoma y dentro de un grupo de proyectos. Los cambios que realice en un proyecto dentro de un grupo también se realizan en el proyecto abierto de forma autónoma.

## Creación de grupos de proyectos

JBuilder proporciona el Asistente para grupos de proyectos, que se encuentra disponible en la ficha Proyecto de la galería de objetos (Archivo | Nuevo), para crear grupos de proyectos. Puede crear un grupo de proyectos vacío o rellenarlo con proyectos al completar el asistente. Puede añadir proyectos a un grupo de proyectos en cualquier momento según se describe en “[Adición y eliminación de proyectos de los grupos de proyectos](#)” en la página 3-3. Una vez creado el grupo de proyectos, puede añadir también

nuevos proyectos con el Asistente para proyectos. Al crear el nuevo proyecto, seleccione la opción Añadir proyecto al grupo activo para que se incluya en el grupo de proyectos.

Los proyectos de un grupo se generan en el orden en que aparecen en el panel del proyecto. El orden de generación se indica en el Paso 2 del Asistente para proyectos, y se puede modificar en cualquier momento en el cuadro de diálogo Propiedades del grupo de proyectos. Si desea obtener más información sobre la generación de grupos de proyectos, consulte “[Generación de grupos de proyectos](#)” en la página 6-6.

Para crear un grupo de proyectos con la ayuda del Asistente para grupos de proyectos, siga los pasos siguientes:

- 1 Elija Archivo | Nuevo y abra la ficha Proyecto de la galería de objetos.
- 2 Haga doble clic en el ícono Grupo de proyectos para iniciar el Asistente para grupos de proyectos.
- 3 Modifique el nombre del archivo de grupo de proyectos y/o la ubicación del archivo en el campo Nombre del archivo.
- 4 Pulse Siguiente para pasar al siguiente paso.
- 5 Realice una de las operaciones siguientes:
  - a Pulse el botón Añadir, busque un proyecto y selecciónelo. Pulse Aceptar para añadir el proyecto. Repita esta acción para añadir otro proyecto.
  - b Pulse el botón Añadir recursivamente, seleccione el directorio que va a examinar y, a continuación, pulse Aceptar. JBuilder examina el directorio seleccionado y todos sus subdirectorios, y añade todos los archivos de proyecto (.jpx) al grupo de proyectos.



- 6 Seleccione un proyecto de la lista y utilice los botones Desplazar hacia arriba o Desplazar hacia abajo para reorganizar la lista de proyectos. Los proyectos se crean y aparecen en el panel del proyecto en el orden de la lista.

- 7 Pulse Aceptar para cerrar el asistente.

El archivo de grupo de proyectos aparece en la parte superior del panel del proyecto y, junto a él, los proyectos en el orden en que se han añadido. Haga doble clic sobre el nodo del grupo de proyectos para ampliarlo y contraerlo. Sólo puede haber un proyecto activo en el grupo en cada momento. Un proyecto se puede abrir simultáneamente de forma independiente o dentro de un grupo de proyectos. Haga doble clic sobre un proyecto para convertirlo en el proyecto activo dentro de un grupo. Amplíe el nodo del proyecto para ver el contenido. Observe que el proyecto activo y abierto aparece en negrita en el panel del proyecto, y en negrita o cursiva en la lista desplegable de proyectos, según el aspecto elegido para la interfaz. Si desea obtener más información sobre el desplazamiento entre los grupos de proyectos, consulte “[Desplazamiento por los grupos de proyectos](#)” en la página 3-4.

Una vez creado un grupo de proyectos, también puede cerrarlo mediante Archivo | Cerrar proyectos, con el botón Cerrar de la barra de herramientas del panel del proyecto panel del proyecto o haciendo clic con el botón derecho en el nodo del grupo de proyectos en el panel del proyecto y seleccionando Cerrar el grupo de proyectos <Nombre.jpgr>. Para abrir un grupo de proyectos, utilice Archivo | Abrir proyecto o Archivo | Abrir archivo. Por defecto, los grupos de proyectos se guardan en el directorio `jbproject` por lo que tiene que buscar los grupos de proyectos (archivos con una extensión `.jpgr`) en `jbproject` a menos que indicara una ubicación diferente para el grupo de proyectos cuando lo creó con el Asistente para grupos de proyectos.

## Adición y eliminación de proyectos de los grupos de proyectos

---

Puede añadir proyectos a o eliminarlos de un grupo de proyectos en cualquier momento. Existen varias formas de hacer esto:

- Menú Proyecto
- Menú contextual del panel del proyecto
- Barra de herramientas del panel del proyecto
- Propiedades del grupo de proyectos

Para añadir un proyecto al grupo de proyectos abierto, lleve a cabo una de las siguientes acciones:

- Seleccione el grupo de proyectos en el panel del proyecto y siga una de las siguientes opciones:
  - Seleccione Proyecto | Añadir proyecto.
  - Pulse el botón Añadir en la barra de herramientas del panel del proyecto y busque el proyecto que desee añadir.
- Pulse el grupo de proyectos con el botón derecho del ratón y seleccione Añadir proyecto.
- Seleccione Proyecto | Propiedades del grupo de proyectos, pulse el botón Añadir y elija el proyecto que desee añadir.

Para eliminar un proyecto de un grupo de proyectos abierto, lleve a cabo una de las siguientes acciones:

- Seleccione el proyecto en el panel del proyecto y, a continuación, Proyecto | Eliminar del grupo de proyectos.
- Pulse con el botón derecho del ratón sobre el proyecto en el panel del proyecto y seleccione Eliminar del proyecto en el menú contextual.
- Seleccione el proyecto en el panel del proyecto y pulse el botón Eliminar de la barra de herramientas del panel del proyecto.
- Seleccione Proyecto | Propiedades del grupo de proyectos, elija el proyecto que desee eliminar y, a continuación, pulse el botón Eliminar.

## Desplazamiento por los grupos de proyectos

---

Una de las ventajas que se deriva de la colocación de proyectos en grupos es el fácil desplazamiento. Aunque sólo puede haber un proyecto activo en cada momento, puede desplazarse rápidamente de un proyecto abierto a otro dentro del grupo. Puede seleccionar otro proyecto en la lista desplegable del panel del proyecto. Al igual que ocurre con los proyectos, también puede abrir los grupos de proyectos en varias ventanas del Visualizador de aplicaciones. Seleccione Ventana | Nueva ventana para abrir otra ventana del Visualizador de aplicaciones.

## Adición de proyectos como bibliotecas necesarias

---

A menudo, los proyectos dentro de los grupos de proyectos tienen dependencias entre ellos. Si tiene un proyecto que depende de otro, puede añadir el proyecto del cual depende el suyo a la lista de bibliotecas necesarias de su proyecto.

Para añadir un proyecto como una biblioteca necesaria:

- 1 Seleccione Proyecto | Propiedades de proyecto y abra la ficha Vías de acceso.
- 2 Haga clic en la pestaña Bibliotecas necesarias.
- 3 Pulse el botón Añadir proyecto.
- 4 Seleccione el proyecto que desea añadir.
- 5 Pulse Aceptar.

El proyecto que haya indicado se añade en la parte inferior de la lista de bibliotecas necesarias.

- 6 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

Compruebe que el proyecto que indicó como biblioteca necesaria se recoge delante del proyecto que depende de él en el grupo de proyectos. De ese modo se sabe que el proyecto necesario se genera primero. Consulte “[Generación de grupos de proyectos](#)” en la página 6-6 para obtener más información sobre la generación de grupos de proyectos.



## Gestión de las vías de acceso

Las vías de acceso son la infraestructura del desarrollo de programas en Java. Las vías de acceso proporcionan al programa los elementos necesarios para su ejecución. Cuando se establece un JDK se indica al programa su ubicación. Cuando se crea una biblioteca se recopila un conjunto de vías de acceso que va a necesitar el programa. Siempre que un archivo hace referencia a otro utiliza una vía de acceso para llegar a él.

En esta sección se explica cómo estructura JBuilder las vías de acceso, cómo se deben manipular en el cuadro de diálogo Propiedades de proyecto y cómo se utilizan las herramientas basadas en vías de acceso, como las bibliotecas y las funciones de CodeInsight.

### Las bibliotecas

---

JBuilder utiliza las bibliotecas para buscar los elementos que necesita para ejecutar un proyecto, así como para examinar el código fuente, ver los comentarios Javadoc, utilizar el diseñador visual, aplicar CodeInsight y compilar el código. Las bibliotecas son recopilaciones de vías de acceso que incluyen archivos de clase, código fuente y documentación. Las bibliotecas son estáticas, no dinámicas. Las vías de acceso de las bibliotecas se suelen incluir en archivos JAR o ZIP, pero también pueden residir en directorios.

Cuando se añaden bibliotecas a JBuilder, se añaden a la vía de acceso a clases, de modo que JBuilder pueda encontrarlas. Las bibliotecas se buscan por el orden en que figuran en la lista. El orden de las bibliotecas se puede cambiar en el cuadro de diálogo Configurar bibliotecas (Herramientas | Configurar bibliotecas) y en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto).

## Consulte

- “Cómo construye JBuilder las vías de acceso” en la página 4-10

La configuración de las bibliotecas se guarda en archivos .library, que pueden almacenarse en varias ubicaciones:

- **Directorio inicial**

Guarda el archivo .library en el subdirectorio <.jbuilder> del directorio inicial del usuario.

- **JBuilder**

Es una función de JBuilder SE y Enterprise.

Guarda el archivo .library en el directorio <jbuilder>/lib. Los usuarios que utilizan JBuilder en red o lo comparten en una única máquina pueden acceder a las bibliotecas de esta carpeta.

- **Proyecto**

Es una función de JBuilder SE y Enterprise.

Guarda el archivo .library en el directorio del proyecto actual. Si se utiliza el control de versiones en Enterprise, el archivo .library aparece junto con los demás archivos de proyecto.

- **Carpetas definidas por el usuario**

Es una función de JBuilder Enterprise.

Guarda el archivo .library en una carpeta definida por el usuario o en un directorio compartido. Para que la carpeta nueva aparezca en la lista desplegable es necesario añadirla en el cuadro de diálogo Configurar bibliotecas.

## Adición y configuración de bibliotecas

---

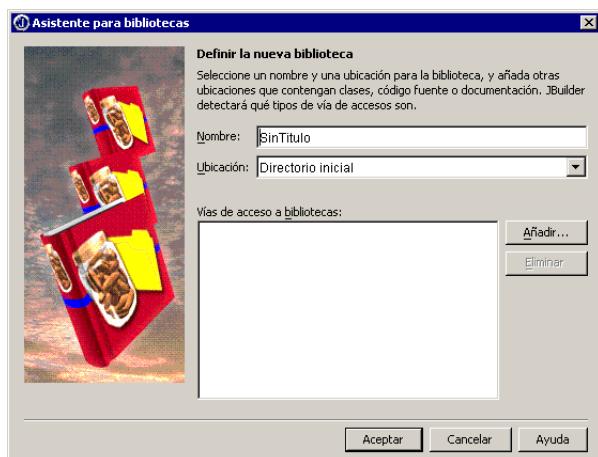
Existen varias maneras de añadir bibliotecas al proyecto. Es conveniente empezar por reunir los archivos en recopilatorios JAR, sobre todo si se tiene intención de distribuir el programa.

Después de crear la biblioteca, añádala a JBuilder de la siguiente manera:

- 1 Seleccione Herramientas | Configurar bibliotecas. Se abre el cuadro de diálogo Configurar bibliotecas.

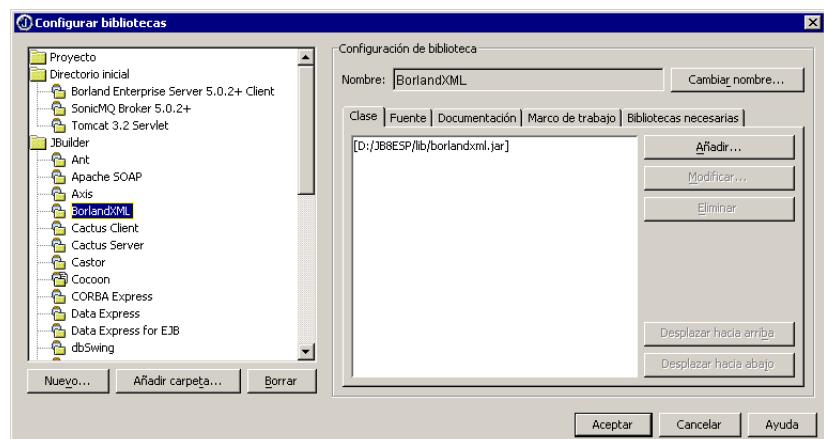
El panel de la izquierda permite desplazarse por las bibliotecas disponibles; el de la derecha muestra la configuración de la biblioteca seleccionada.

- 2** Pulse el botón Nuevo bajo el panel izquierdo para abrir el Asistente para bibliotecas.



- 3** Escriba un nombre en el campo Nombre.
- 4** Seleccione en la lista desplegable el lugar donde desea guardar la configuración de la biblioteca: Proyecto, Directorio inicial, JBuilder o Definido por el usuario. Las carpetas definidas por el usuario son una función de Enterprise.
- 5** Pulse el botón Añadir y seleccione vías de acceso que contengan archivos de clase, fuente y documentación. JBuilder determina automáticamente la vía de acceso correcta de los archivos. Pulse Aceptar. Observe que la selección aparece en la lista Vía de acceso a bibliotecas.
- 6** Pulse Aceptar para cerrar el Asistente para bibliotecas. Observará que la biblioteca se guarda en las vías de acceso a clases, fuente y documentación adecuadas, en el cuadro de diálogo Configurar bibliotecas. Este cuadro de diálogo también permite añadir, modificar, eliminar y reordenar las listas de bibliotecas. JBuilder Enterprise también incluye una función Añadir carpeta que permite añadir un marco como biblioteca. Si desea obtener más información acerca de la

adicción de marcos como bibliotecas, consulte “Marcos de trabajo JSP” en la *Guía del desarrollador de aplicaciones Web*.



- 7 Pulse Aceptar o *Intro* para cerrar el cuadro de diálogo Configurar bibliotecas.

Para añadir una biblioteca al proyecto, consulte “[Configuración de vías de acceso a bibliotecas necesarias](#)” en la página 2-25.

También es posible añadir bibliotecas en el cuadro de diálogo Propiedades de proyecto.

- 1 Seleccione Proyecto | Propiedades de proyecto.
- 2 Seleccione la pestaña Bibliotecas necesarias de la ficha Vías de acceso y pulse Añadir.
- 3 Pulse el botón Nuevo para abrir el Asistente para bibliotecas.

### Consulte

- “[Configuración de vías de acceso a bibliotecas necesarias](#)” en la página 2-25
- “Using JAR Files: The Basics” en <http://java.sun.com/docs/books/tutorial/jar/basics/index.html>
- “Asistente para bibliotecas” en la ayuda en línea

## Modificación de bibliotecas

---

Para modificar una biblioteca,

- 1 Seleccione Herramientas | Configurar bibliotecas.
- 2 Seleccione la biblioteca que desea modificar de la lista de bibliotecas.
- 3 Seleccione la pestaña Clase, Fuente o Documentación, Marco de trabajo o Bibliotecas necesarias para elegir la vía de acceso de la biblioteca que desea modificar.
- 4 Seleccione la biblioteca y pulse Modificar.
- 5 Busque el archivo o la carpeta en el cuadro de diálogo Seleccionar directorio. Pulse Aceptar.
- 6 Pulse Añadir para buscar la biblioteca que desea añadir.
- 7 Para borrar una vía de acceso a una biblioteca, selecciónela y pulse Eliminar.
- 8 Para reorganizar las vías de acceso a bibliotecas, seleccione una de ellas y pulse Desplazar hacia arriba o Desplazar hacia abajo.

**Sugerencia** JBuilder busca las bibliotecas por el orden con que figuran en la lista.

- 9 Pulse Aceptar o *Intro* para cerrar el cuadro de diálogo Configurar bibliotecas.

## Adición de proyectos como bibliotecas necesarias

---

Los proyectos pueden tener con otros proyectos. Si tiene un proyecto que depende de otro, puede añadir el proyecto del cual depende el suyo a la lista de bibliotecas necesarias de su proyecto.

Para añadir un proyecto como una biblioteca necesaria:

- 1 Seleccione Proyecto | Propiedades de proyecto y abra la ficha Vías de acceso.
- 2 Haga clic en la pestaña Bibliotecas necesarias.
- 3 Pulse el botón Añadir proyecto.
- 4 Seleccione el proyecto que desea añadir.
- 5 Pulse Aceptar.

El proyecto que haya indicado se añade en la parte inferior de la lista de bibliotecas necesarias.

- 6 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

## Presentación de la lista de bibliotecas

---

Hay tres colores posibles para las bibliotecas que aparecen en las listas de los cuadros de diálogo de JBuilder:

**Tabla 4.1** Colores de las listas de bibliotecas

Color	Descripción	Resolución de problemas
Negro	La biblioteca se ha definido correctamente.	
Rojo	La definición de la biblioteca no se encuentra.	Normalmente indica que el proyecto hace referencia a una biblioteca que no se ha definido todavía. También puede significar que la definición de la biblioteca es defectuosa: se ha definido sin vías de acceso o existen varias bibliotecas con ese nombre.
Gris	La utilización de esta biblioteca requiere una actualización.	Es necesario actualizar su edición de JBuilder para utilizar esta biblioteca. Por ejemplo, si posee JBuilder Personal, la utilización de la biblioteca dbSwing necesita que actualice a JBuilder Enterprise.

## Paquetes

---

Java agrupa los archivos .java y .class en un *paquete*. Los archivos que constituyen el código fuente de un paquete de Java se encuentran en el subdirectorio (`src`), y los archivos compilados, en el subdirectorio (`clases`). En la generación de aplicaciones JBuilder utiliza el nombre del proyecto como nombre por defecto para el paquete del Asistente para aplicaciones o el Asistente para applets. Por ejemplo, si el proyecto se llama `sintítulo1.jpr`, el Asistente para aplicaciones y el Asistente para applets proponen la utilización del nombre `sintítulo1`. Los nombres de paquete propuestos se basan siempre en el nombre del proyecto.

Vamos a tomar como referencia un proyecto de ejemplo para ver cómo influye el nombre del paquete en la estructura del archivo.

**Nota** En estos ejemplos, las vías de acceso son un reflejo de la plataforma UNIX. Si desea información sobre la forma en que se documentan aquí las vías de acceso, consulte “[Convenciones de la documentación](#)” en la página 1-5.

Para organizar el proyecto, supongamos que se encuentra en una carpeta llamada Proyecto de ejemplo. Esta carpeta de proyectos contiene un archivo de proyecto (africa.jpr), un directorio `clases` y un directorio `src`:

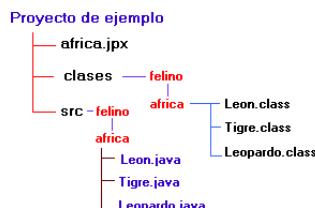


Al crear este proyecto puede ser conveniente crear paquetes propios que contengan los archivos de código fuente y clase relacionados. En este ejemplo, `africa.jpr` contiene un nombre de paquete de `felino.africa`. Este paquete contiene archivos de código fuente sobre varios felinos africanos: Leones, Tigres y Leopardo.

Los archivos de clase, guardados en una estructura de directorios que coincide con el nombre del paquete, se guardan en el subdirectorio `clases`, dentro del proyecto. El subdirectorio `src`, que contiene los archivos `.java`, tiene la misma estructura que el subdirectorio de clases.



Si las clases que contiene este proyecto son `Leon.class`, `Tigre.class` y `Leopardo.class`, se encuentran en `clases/felino/africa`. Los archivos fuente `Leon.java`, `Tigre.java` y `Leopardo.java` se encuentran en `src/felino/africa`, tal y como indicamos.



## Ubicación del archivo .java = vía de acceso a archivos fuente + vía de acceso a paquetes

Es importante comprender qué datos utiliza JBuilder para generar el directorio en que se encuentra un archivo `.java` determinado. La primera parte de la vía de acceso se determina por la vía de fuentes. La vía de fuentes se define en el nivel de proyecto y puede modificarse en la página de vías de acceso del cuadro de diálogo Propiedades de proyecto.

Prosiguiendo con Proyecto de ejemplo, la vía de acceso a archivos fuente de Leon.java es:

/<inicio>/<nombredesusuario>/jbproject/Proyecto de ejemplo/src.

**Nota** Para obtener más información sobre la definición del directorio <inicio>, consulte “[Convenciones de la documentación](#)” en la página 1-5.

La segunda parte de la vía de acceso al directorio está determinada por el nombre del paquete, que en este caso es felino.africa.

**Nota** En la nomenclatura de Java se utiliza un punto (.) para separar las partes de los paquetes.

La ubicación del archivo .java en el caso de Leon.java es:

/<inicio>/<nombredesusuario>/jbproject/Proyecto de ejemplo/src/felino/africa/  
Leon.java

### Consulte

- “[Cómo construye JBuilder las vías de acceso](#)” en la página 4-10

## Ubicación del archivo .class = vía de salida + vía de acceso a paquetes

---

La ubicación del directorio del archivo .class está formada por la vía de archivos generados y el nombre del paquete. La vía de salida es la “raíz” a la que añadirá JBuilder añade las vías de acceso de los paquetes, para crear la estructura de directorios de los archivos .class generados por el compilador. La vía de acceso a archivos fuente se define en cada proyecto y se puede modificar en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto.

En Proyecto de ejemplo, la vía de salida de Leon.class es:

/<inicio>/<nombredesusuario>/jbproject/Proyecto de ejemplo/classes

La segunda parte de la vía de acceso al directorio está determinada por el nombre de paquete, que en este caso es felino.africa.

Como se indica a continuación, la ubicación .class de Leon.class es:

/<inicio>/<nombredesusuario>/jbproject/Proyecto de ejemplo/classes/felino/  
africa/Leon.class

### Consulte

- “[Cómo construye JBuilder las vías de acceso](#)” en la página 4-10

## Utilización de paquetes en JBuilder

Cuando se hace referencia a clases desde un paquete se puede utilizar una sentencia import. Las sentencias de importación permiten hacer referencia a cualquier clase del paquete importado utilizando solamente el nombre corto en el código. (Los diseñadores y asistentes de JBuilder añaden automáticamente las sentencias de importación.) A continuación se muestra un ejemplo de una sentencia import:

```
import felino.africa.*;
```

Si esta sentencia de importación se incluye en el código fuente es posible referirse a la clase `Leon` simplemente como `Leon` en el cuerpo del código.

Si no se importa el paquete es necesario hacer referencia a una clase determinada en el código fuente, con su nombre de clase completo. Como se indica en este esquema, el nombre de clase completo de `Leon.java` es `felino.africa.Leon`. (nombre de paquete + nombre de clase sin la extensión).



Se pueden excluir paquetes del proceso de creación de forma selectiva.

### Consulte

- “[Filtrado de paquetes](#)” en la página 6-27

## Directrices de nomenclatura de paquetes

Se recomienda utilizar las siguientes directrices de nomenclatura de paquetes en todos los programas en Java. Con el fin de mantener la coherencia, la legibilidad y la capacidad de mantenimiento, los nombres de los paquetes deben seguir estas reglas:

- Una palabra
- Singular mejor que plural
- Todo en minúsculas, incluso si hay más de una palabra (por ejemplo, `nombreproyectocuatropalabras` y **no** `NombreProyectoCuatroPalabras`)

Si los paquetes se van a utilizar fuera del grupo, los nombres han de comenzar con el nombre de un dominio de Internet invertido. Por ejemplo, si desea utilizar el nombre de dominio `foo.dominio.com`, los nombres de los paquetes deben ir precedidos por `com.dominio.foo`.

## Cómo construye JBuilder las vías de acceso

---

El IDE de JBuilder utiliza varias vías de acceso durante el proceso:

- Vía de acceso a archivos fuente
- Vía de acceso a archivos generados
- Vía de acceso a clases
- Vía de búsqueda
- Vía de acceso a documentos
- Vía de acceso a las copias de seguridad
- Directorio de trabajo

Las vías de acceso se establecen para cada proyecto. Para establecerlas, utilice el cuadro de diálogo Propiedades de proyecto. Consulte ["Definición de las propiedades del proyecto"](#) en la página 2-20 para obtener más información.

Al construir las vías de acceso, JBuilder elimina los nombres repetidos. Así se evitan los posibles problemas de las limitaciones de DOS en Windows.

**Nota** En estos ejemplos, las vías de acceso son un reflejo de la plataforma UNIX. Consulte ["Convenciones de la documentación"](#) en la página 1-5.

### Vía de acceso a archivos fuente

---

La vía de acceso a archivos fuente determina el lugar donde el compilador busca los archivos fuente. Se constituye a partir de lo siguiente:

- La vía de acceso definida en la pestaña Fuente de la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto.
- El directorio de los archivos generados. Este directorio contiene los archivos fuente que genera automáticamente el IDE, como los archivos de servidor IDL y los de esqueleto. Este directorio se encuentra en la vía de salida. Esta opción puede modificarse en la ficha Generar del cuadro de diálogo Propiedades de proyecto.

La vía de acceso a archivos fuente completa consta de estos dos elementos, por este orden:

vía de acceso a archivos fuente + vía de archivos generados/código fuente generado

En Proyecto de ejemplo, la vía de acceso a archivos fuente del proyecto africa.jpx es:

/<inicio>/<nombredeusuario>/jbproject/Proyecto de ejemplo/src.

## Vía de acceso a archivos generados

---

La vía de salida contiene los archivos .class creados por JBuilder. Se constituye a partir de la vía definida en el cuadro de texto Vía de salida, situado en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto.

Los archivos se envían a un directorio cuya vía de acceso coincide con la vía de salida más el nombre del paquete. Sólo existe una vía de salida en cada proyecto.

Por ejemplo, en el caso de Proyecto de ejemplo, la vía de salida de felino.africa.jpr es:

```
/<inicio>/<nombredelusuario>/jbproject/Proyecto de ejemplo/classes
```

## Vía de acceso a clases

---

La vía de acceso a clases se emplea en las compilaciones. Esta vía se construye a partir de lo siguiente:

- La vía de salida
- La vía de acceso a clases de las bibliotecas enumeradas en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto (el orden en el que aparecen)
- La versión de JDK seleccionada en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto

La vía de acceso a clases completa consta de estos elementos, por este orden:

vía de archivos generados + vía de acceso a clases de la biblioteca (por el orden en que aparecen las bibliotecas en el cuadro de diálogo Propiedades de proyecto) + versión JDK destino

Por ejemplo, la vía de acceso completa a Leon.class es:

```
/<inicio>/<nombredelusuario>/jbproject/Proyecto de ejemplo/classes:  
/user/jbuilder/lib/dbswing.jar:/
```

La vía de acceso a clases se muestra en el panel de mensajes cuando ejecuta el proyecto.

## Vía de búsqueda

---

El IDE utiliza la vía de búsqueda en los siguientes casos:

- Cuando se utiliza CodeInsight
- Cuando se selecciona Buscar definición en el menú desplegable del editor

- Cuando se selecciona Buscar | Buscar clases
- Cuando se ejecuta el depurador

La vía de búsqueda se constituye a partir de lo siguiente:

- La vía de acceso a archivos fuente
- La vía de acceso a archivos fuente de las bibliotecas enumeradas en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto (en el orden por el que aparecen).
- La vía de acceso a archivos fuente de la versión del JDK seleccionada en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto.

La vía de búsqueda completa consta de estos elementos, por este orden:

vía de acceso a archivos fuente + vías de acceso a fuente de las bibliotecas (en el orden por el que aparecen las bibliotecas en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto) + vía de acceso a fuente de la versión del JDK destino.

Por ejemplo, la vía de búsqueda completa en el caso de Leon.class es:

```
/<inicio>/<nombredeusuario>/jbproject/Proyecto de ejemplo/src:  
/user/jbuilder/src/dbswing-src.jar:  
/user/jbuilder/src/dx-src.jar
```

## Vía de acceso a documentos

---

La vía de acceso a documentos contiene los archivos HTML de documentación de los archivos de clase de la API. Esto permite que la documentación de referencia se muestre en la ficha Doc del panel de contenido.

La vía de acceso a documentos se puede configurar en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto. Las vías se buscan en la lista por orden.

## Vía de acceso a las copias de seguridad

---

JBuilder almacena en la vía de acceso a copias de seguridad las versiones de respaldo de los archivos de código fuente. El directorio de copia de seguridad por defecto es:

```
/<inicio>/<nombredeusuario>/jbproject/Proyecto de ejemplo/bak
```

**Importante** los archivos JSP y HTML, entre otros, se tratan como archivos fuente. Sus copias de seguridad se guardan en los directorios originales.

No obstante, también se pueden incluir sus copias de seguridad en el directorio de copias de seguridad. Para ello:

- 1** Seleccione Proyecto | Propiedades de proyecto y active la pestaña Vías de acceso.
- 2** En ella, seleccione la pestaña Fuente.
- 3** En la ficha Fuente, seleccione Añadir. Se abre el cuadro de diálogo Seleccionar directorio.
- 4** Desplácese el directorio de copias de seguridad del proyecto y pulse Aceptar.

## Directorio de trabajo

---

El directorio de trabajo es el directorio inicial que JBuilder proporciona a los programas cuando se inician. Se puede configurar cualquier directorio como directorio de trabajo. Por defecto, tiene el mismo nombre que el archivo del proyecto.

Normalmente es el directorio superior del directorio fuente. También es el directorio superior por defecto de los directorios de salida (archivos generados), copia de seguridad, documentación y bibliotecas.

## Localización de los archivos

---

Todos los archivos de un proyecto se almacenan con una vía de acceso relativa a la ubicación del archivo .jpr. JBuilder busca y guarda los archivos en las vías de acceso a archivos fuente, de comprobación, de acceso a clases, de búsqueda y de archivos generados.

Esta lista explica el propósito de cada tipo de vía de acceso:

- La vía de acceso a archivos fuente determina el lugar donde el compilador busca los archivos fuente.
- La vía de comprobación es la vía de acceso a archivos fuente cuando se utiliza los tests de módulos.
- La vía de acceso a clases se utiliza en la compilación y durante la ejecución, y en determinadas funciones del editor de Enterprise.
- El IDE utiliza la vía de búsqueda cuando se emplea CodeInsight o el comando Buscar definición del editor, así como la búsqueda y la depuración.
- La vía de salida incluye los archivos .class creados por JBuilder durante la compilación del proyecto.

### Consulte

- “Cómo construye JBuilder las vías de acceso” en la página 4-10
- “Las bibliotecas” en la página 4-1

## Cómo encuentra JBuilder los archivos al profundizar

---

Cuando se profundiza para explorar el código fuente, JBuilder busca los archivos .java en la vía de búsqueda. Si desea información adicional sobre la exploración y la profundización, consulte “Desplazamiento por el código fuente” en *Introducción a JBuilder*.

## Cómo encuentra JBuilder los archivos al compilar

---

Cuando se compila el proyecto, JBuilder utiliza las siguientes vías de acceso:

- vía de acceso a clases
- vía de acceso a archivos fuente
- vía de archivos generados

JBuilder busca en la vía de acceso a clases los archivos .class, las bibliotecas que ha de emplear y la versión del JDK en que debe compilar. El compilador compara los archivos .class de la vía de acceso a clases con sus archivos fuente, ubicados en la vía de acceso a archivos fuente, para determinar si es necesario volver a compilar los archivos .class para actualizarlos. Los archivos .class resultantes se almacenan en la vía de salida que se haya definido.

Para obtener más información acerca de la compilación de archivos, consulte el [Capítulo 6, “Generación de programas en Java”](#) y el [Capítulo 5, “Compilación de programas en Java”](#).

## Cómo encuentra JBuilder los archivos de clase al ejecutar o depurar

---

Cuando se ejecuta y se depura el programa, JBuilder utiliza la vía de acceso a clases para localizar todas las clases que emplea el programa.

Cuando se inspecciona línea a línea el código fuente con el depurador, JBuilder utiliza la vía de búsqueda para localizar los archivos de código fuente.

Para obtener más información acerca de la depuración de archivos, consulte el [Capítulo 8, “Depuración de programas en Java”](#).

# Compilación de programas en Java

Los compiladores de Java leen los archivos fuente de Java y otros archivos fuente pasados a Java, determinan qué archivos adicionales deben compilarse y generan el programa Java en forma de archivos .class que contienen los bytecodes que constituyen el código máquina para la máquina virtual (MV) de Java. La compilación genera un archivo .class por cada declaración de clase o de interfaz de un archivo fuente. Cuando se ejecuta el programa de Java resultante en una plataforma concreta, como por ejemplo Windows NT, el intérprete de Java de dicha plataforma ejecuta los bytecodes de los archivos .class. Para obtener información general acerca de la compilación en Java, consulte la descripción general del compilador del Kit de desarrollo de Java (JDK), “javac - The Java programming language compiler” en <http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html>.

El compilador por defecto para el IDE de JBuilder, Make de Borland para Java (**bmj**), admite el lenguaje Java sin limitaciones. El compilador de JBuilder utiliza la comprobación inteligente de dependencias, por lo que el ciclo de compilación y recompilación es más rápido y eficaz. El verificador de dependencias determina la naturaleza de los cambios del código fuente y sólo compila los archivos necesarios. Para obtener más información, consulte “Comprobación inteligente de dependencias” en la página 5-2. Para comprender como trabaja el compilador de JBuilder, consulte “Make de Borland para Java (**bmj**)” en la página B-12.

Si prefiere compilar desde la línea de comandos, JBuilder también le proporciona las siguientes herramientas de línea de comandos en las ediciones JBuilder SE y Enterprise:

- La opción de línea de comandos de JBuilder **-build** para generar proyectos
- Make de Borland para Java (**bmj**), que utiliza el verificador de dependencias
- El compilador de Borland para Java (**bcj**)

JBuilder también ofrece la opción de cambiar los compiladores. Para aprovechar todas las ventajas de las funciones de JBuilder, como la comprobación independiente de dependencias, UML y el perfeccionamiento, es recomendable que utilice el Make de Borland. No obstante, si desea utilizar `javac`, puede cambiar de compilador en la ficha Generar del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto). Para obtener más información, consulte ["Definición de opciones del compilador" en la página 5-7](#).

La compilación es sólo una de las fases del sistema de generación de JBuilder. Entre otras se incluyen la precompilación, postcompilación, limpieza, empaquetado y distribución. Si desea obtener más información sobre estas fases y el sistema de generación de JBuilder, consulte [Capítulo 6, "Generación de programas en Java"](#).

## Comprobación inteligente de dependencias

---

El compilador Make de Borland realiza una compilación rápida pero completa, utilizando la comprobación inteligente de dependencias, que reduce el número de compilaciones innecesarias de los archivos fuente relacionados entre sí, acelerando así el ciclo de edición y recompilación. Durante la compilación, en lugar de decidir la recompilación de un archivo en función del momento en el que se produjo la modificación del archivo, el Make de Borland analiza la naturaleza de los cambios realizados en los archivos fuente.

Existen varias razones posibles para recompilar el archivo fuente:

- Faltan uno o más archivos de clase que produciría el archivo fuente.
- Se ha modificado el archivo fuente después de compilar.
- Una o más de las clases que produce el archivo fuente depende de un miembro de otra clase que se ha cambiado.

Al cambiar un archivo fuente, puede que cambie el modo en que se compilan otros archivos fuente. JBuilder no sólo es capaz de detectar esta situación, sino, además, de advertir si un cambio en un archivo fuente no va a afectar a otros archivos, ya que se refieren a partes que no han sufrido cambios. Si es éste el caso, JBuilder **no** vuelve a compilar los archivos.

Cuando se compilan los archivos fuente por primera vez, se crea automáticamente para cada paquete un archivo de dependencias, que se

guarda en el directorio de salida, junto con los archivos de clases. Este archivo de dependencias contiene información detallada, para todas las clases de un paquete, relacionada con la utilización que unas clases hacen de otras. Este archivo tiene la extensión `.dep2` y se guarda en una carpeta de nombre `package cache` en el mismo directorio que las clases.

Los archivos de dependencias deben encontrarse en la vía de acceso a clases, para que el compilador pueda encontrarlos. Cuando la compilación se realiza desde el IDE, la vía de acceso a clases se encuentra correctamente definida, por defecto. Para obtener información sobre cómo se construye la vía de acceso a clases, consulte “[Vía de acceso a clases](#)” en la página 4-11.

Si se compila desde la línea de comandos puede ser necesario asignar un valor a la variable de entorno `CLASSPATH`. Para obtener más información, consulte “[Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos](#)” en la página B-2.

El Make de Borland del IDE y el make de línea de comandos **bmj** utilizan la comprobación inteligente de dependencias, a diferencia del compilador de línea de comandos **bcj**, que no lo utiliza. **bmj** y **bcj** se encuentran disponibles en las versiones SE y Enterprise de JBuilder.

**Importante** Los archivos de revisiones se consideran “estables” y el verificador de dependencias no los comprueba.

### Consulte

- “[JBuilder Dependency Checker](#)” en la página web de Blake Stone, en <http://homepages.borland.com/bstone/articles/depchecker.html>

---

## Compilación de un programa

El IDE de JBuilder utiliza el Make de Borland para Java (**bmj**) para compilar los archivos fuente de Java. Debido a que el compilador de JBuilder utiliza la comprobación inteligente de dependencias, el ciclo de compilación y recompilación es más rápido y eficaz. Para obtener más información, consulte “[Comprobación inteligente de dependencias](#)” en la página 5-2. Para comprender como trabaja el compilador de JBuilder, see “[Make de Borland para Java \(bmj\)](#)” en la página B-12.

Es posible compilar las siguientes partes de un proyecto:

- La totalidad del proyecto
- Paquetes
- Los archivos java

Para comprender cómo encuentra JBuilder los archivos para compilar el programa, consulte “[Cómo construye JBuilder las vías de acceso](#)” en la [página 4-10](#), y “[Localización de los archivos](#)” en la [página 4-13](#).

Para compilar los archivos fuente de un programa:

**1** Abra el proyecto que contiene el programa o abra un archivo de Java.

**2** Realice una de las operaciones siguientes:

- Seleccione Proyecto | Ejecutar Make del proyecto.
- Haga clic con el botón derecho del ratón sobre los archivos Java del panel del proyecto y seleccione Ejecutar Make <nombredelproyecto>.
- Haga clic con el botón derecho de ratón sobre la pestaña de archivo de un archivo Java abierto, en el editor y seleccione Ejecutar Make <nombredelarchivo>.
- Pulse el botón Ejecutar Make de la barra de herramientas.

## Menús de generación de JBuilder

---

JBuilder ofrece comandos de menú para generar proyectos: Ejecutar Make, Generar de nuevo y Limpiar.

Ejecutar Make es una fase del sistema de generación de JBuilder que establece dependencias entre otras fases autónomas: Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir. El comando Ejecutar Make de JBuilder no debe confundirse con el make de compilación de Java, que sólo compila los archivos fuente de Java. El comando Ejecutar Make genera archivos fuente Java y otros archivos generables en su proyecto, como archivos recopilatorios, WebApps y otros nodos generables.

JBuilder también proporciona el comando Generar de nuevo para que pueda volver a generar completamente su proyecto. El comando Generar de nuevo tiene como dependencias Limpiar y Ejecutar Make. En primer lugar, toda la salida generada se elimina y, a continuación, se ejecuta el Make. Por último, el comando Limpiar elimina toda la salida generada.

En JBuilder Enterprise se pueden configurar los menús de generación. Para obtener más información, consulte “[Configuración del menú Proyecto](#)” en la [página 6-21](#).

### Consulte

- “[El comando Ejecutar Make](#)” en la [página 6-4](#)
- “[El comando Generar de nuevo](#)” en la [página 6-5](#)
- “[El comando Limpiar](#)” en la [página 6-5](#)

## Generación de proyectos con el comando Ejecutar

El comando Ejecutar se puede configurar para que ejecute un tipo de generación antes de ejecutar el proyecto. El funcionamiento por defecto del comando Ejecutar el proyecto (Ejecutar | Ejecutar el proyecto) y del botón Ejecutar el proyecto es ejecutar el Make de la aplicación y, después, ejecutarla. El funcionamiento por defecto se puede cambiar en las configuraciones de ejecución del proyecto. Por ejemplo, puede generar de nuevo su proyecto cada vez que vaya a ejecutarlo, en lugar de utilizar el Make por defecto. Los tipos de generación disponibles varían según el tipo de proyecto en el que esté trabajando. Entre los tipos disponibles pueden estar Generar de nuevo, Limpiar, Ninguno, tareas externas de generación, tipos Ant y cualquier tarea de generación personalizada que haya agregado al ampliar el sistema de generación por medio de Open Tools. Si desea obtener más información sobre como cambiar el tipo de generación, consulte “[Tipos de generación](#)” en la página 7-11.

### Consulte

- “[El comando Ejecutar](#)” en la página 7-4
- “[Generación de proyectos Ant con el comando Ejecutar](#)” en la página 6-14

## Errores de sintaxis y mensajes de error

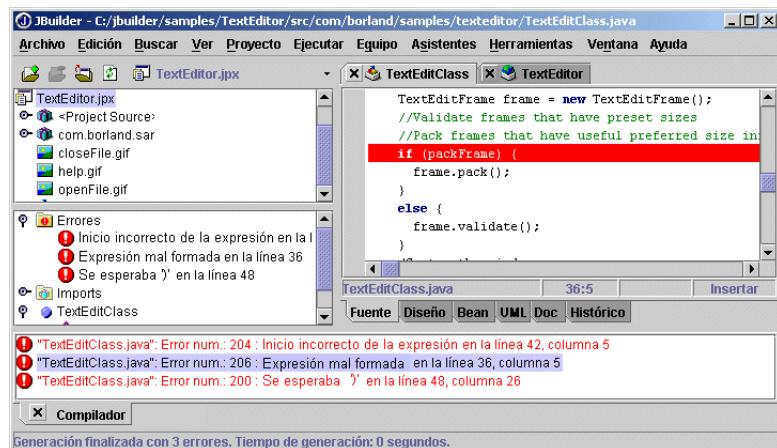
---

Los errores de sintaxis son errores que violan las reglas sintácticas del lenguaje de programación Java. El editor captura estos errores en el momento en el que ocurren antes de la compilación. Los errores de sintaxis se muestran en una carpeta en la parte superior del panel de estructura. Para ubicar la línea de código que contiene el error, es necesario ampliar la carpeta Errores en el panel de estructura y hacer doble clic sobre el error. La línea del error, está resaltada en el editor.

Los mensajes de error también se presentan en la pestaña Generar del panel de mensajes durante la compilación. Si desea ayuda sobre un mensaje de error, selecciónelo y pulse *F1*. Las teclas de desplazamiento permiten moverse por los mensajes de error del compilador. Haga clic en



uno de ellos para resaltar el código del archivo abierto. Haga doble clic en el error para desplazar el cursor a la línea de código, en el editor.



### Consulte

- “Errores y mensajes de advertencia” en la ayuda en línea
- En “Mensajes de error del compilador” en la ayuda el línea, los mensajes de error aparecen listados por número

## Problemas de compilación al abrir proyectos

Si abre un proyecto y no se compila, compruebe que esté bien definida la vía de acceso en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto. JBuilder utiliza los valores de la vía de acceso para construir la vía de acceso a clases y la vía de acceso a archivos fuente, los lugares donde el compilador busca los archivos.

Además, compruebe la lista de Bibliotecas necesarias de la ficha Vías de acceso. Si alguna de las bibliotecas está resaltada en rojo, significa que no se ha definido al instalar JBuilder. Haga doble clic sobre el nombre de la biblioteca o selecciónelo y, a continuación, seleccione Modificar para redefinirlo. Por último, recompile el proyecto.

Los proyectos con WebApps pueden necesitar un servidor web correctamente configurado para poder compilar. Consulte “Configuración del servidor web” en *Guía del desarrollador de aplicaciones Web*.

Para definir las vías de acceso por defecto de proyectos nuevos (a fin de evitar posibles problemas en el futuro), acceda al cuadro de diálogo de Propiedades por defecto para proyectos (Proyecto | Propiedades por defecto para proyectos). Consulte “[Definición de las propiedades del proyecto](#)” en la página 2-20 y “[Cómo construye JBuilder las vías de acceso](#)” en la página 4-10.

## Comprobación de correspondencia entre paquetes y directorios

---

JBuilder cuenta con una comprobación de protección de definiciones duplicadas de clases dentro de un proyecto y de correspondencia entre paquetes y directorios. El compilador **bmj**, que es el compilador por defecto del IDE, verifica que la sentencia package de un archivo fuente se corresponda con el directorio del paquete, y también que una misma clase no esté definida por dos archivos fuente.

La primera vez que se construye un proyecto, se verifican y compilan todos los archivos .java que se encuentran dentro de un directorio de paquete. Si tiene fuentes temporales que no desea compilar, utilice una extensión que no sea .java. Por ejemplo, si el proyecto contiene una versión anterior de un archivo en el que se está trabajando y que contiene una definición diferente de la misma clase, aparecerá un mensaje de “definición duplicada de una clase”. Esta comprobación impide la aparición de problemas sutiles que serían difíciles de encontrar.

## Definición de opciones del compilador

---

Puede definir las opciones del compilador del proyecto actual en la ficha Java de la ficha Generar de Propiedades de proyecto (Proyecto | Propiedades de proyecto). Las opciones, que varían según el compilador seleccionado, se aplican a todos los archivos del árbol del proyecto. Si cambia las opciones del compilador, debe generar de nuevo los paquetes o todo el proyecto en lugar de utilizar solamente Ejecutar Make. Las opciones del proyecto se aplican a todas las clases que vuelvan a generarse, tanto fuera como dentro del árbol del proyecto.

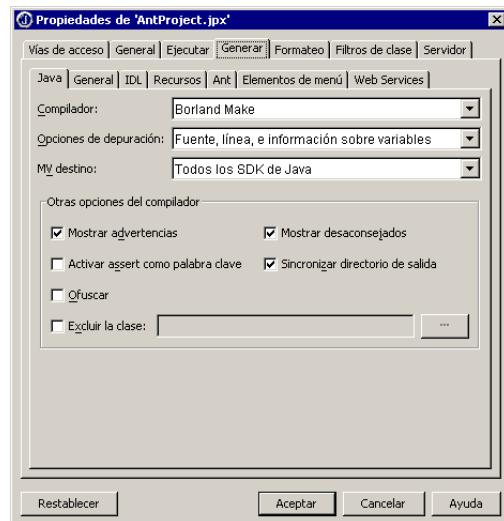
No puede definir opciones del compilador para cada archivo. Sin embargo, dos proyectos que tengan opciones diferentes de compilación pueden utilizar un mismo archivo. No se pueden activar opciones de forma individual a clases y paquetes, dado que en Java las cabeceras y módulos no se compilan de forma independiente. Si falta información de importación (por ejemplo acerca de un archivo de clase), la clase importada se compila al mismo tiempo que la clase que la importa, utilizando las mismas opciones aplicadas a todo el proyecto.

También se pueden configurar las opciones del compilador para proyectos futuros en el cuadro de diálogo Propiedades de proyecto por defecto (Proyecto | Propiedades de proyecto por defecto). Después de configurar las propiedades por defecto para proyectos, siempre que cree un proyecto con el Asistente para proyectos, se aplican las opciones por defecto.

Si compila su proyecto con el Make de Borland, las opciones del compilador se aplican a todos los archivos del árbol del proyecto y a todos los archivos a los que se hace referencia en estos archivos, y se detiene en los paquetes marcados como estables y que no tengan clases en el árbol del proyecto. Make de Borland ofrece opciones adicionales del compilador, como Confundir, Sincronizar directorio de salida y Excluir la clase. Si desea más información acerca de estas opciones, pulse el botón Ayuda de la ficha Java, en la ficha Generar.

Para definir las opciones del compilador para su proyecto, siga los pasos siguientes:

- 1** Seleccione Proyecto | Propiedades de proyecto o, bien, pulse con el botón derecho del ratón sobre el nodo de proyecto .jpx del panel del proyecto y, a continuación, seleccione Propiedades. Se abre el cuadro de diálogo de Propiedades de proyecto.
- 2** Haga clic en la pestaña Generar para mostrar la ficha Generar. Después, seleccione la pestaña Java.



- 3** Seleccione un compilador y las opciones de depuración y compilación que deseé. Las opciones del compilador disponibles varían según el compilador seleccionado. Si desea obtener más información acerca de las opciones, consulte la ficha Generar del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto). El cambio entre compiladores es una función de JBuilder Enterprise.
- 4** Defina las opciones que deseé en las demás fichas de la ficha Generar.
- 5** Seleccione Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto y guardar la configuración.

- 6 Seleccione Proyecto | Generar de nuevo el proyecto para generar el proyecto con las opciones revisadas.
- 7 Pulse Aceptar para cerrar el cuadro de diálogo.

## Definición de un compilador

---

Es una función de JBuilder Enterprise.

Por defecto, JBuilder compila los proyectos con el Make de Borland para Java (**bmj**). Normalmente, se recomienda compilar con el compilador de Borland para aprovechar todas las ventajas de las funciones de JBuilder, como la comprobación de dependencias y el perfeccionamiento. Los demás compiladores disponibles incluyen **javac** y **jjavac** del proyecto. Cuando seleccione **javac** como el compilador, el proyecto se compila mediante el host **javac** de Java, que se encuentra en el directorio de JBuilder. Si se selecciona el **javac** del proyecto, JBuilder utiliza el **Jjavac** del JDK especificado para el proyecto en la ficha Vías de acceso (Proyecto | Propiedades de proyecto). Al seleccionar un compilador, sólo están disponibles las opciones de ese compilador.

Para cambiar el compilador del proyecto, siga los pasos siguientes:

- 1 Seleccione Proyecto | Propiedades de proyecto.
- 2 Seleccione la ficha Generar y abra la pestaña Java.
- 3 Seleccione un compilador de la lista desplegable Compilador.
- 4 Pulse Aceptar para cerrar el cuadro de diálogo.

## Configuración de las opciones adicionales de compilación y generación

---

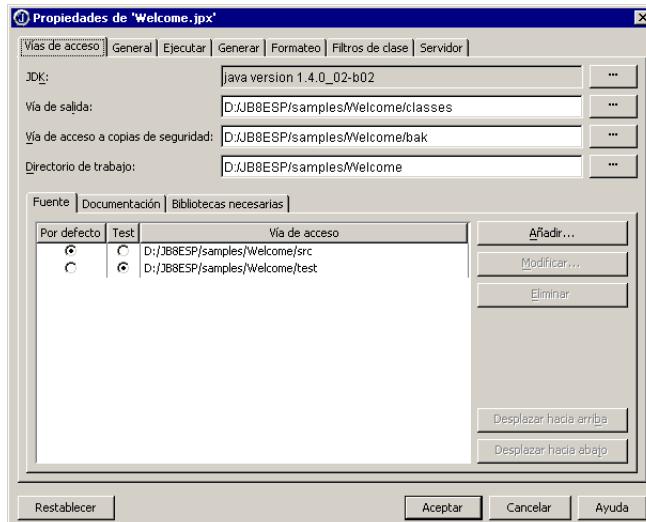
En la ficha Java de la ficha Generar de Propiedades de proyecto (Proyecto | Propiedades de proyecto), puede seleccionar un compilador, opciones de depuración, la MV de destino y otras opciones del compilador, como Mostrar advertencias, Mostrar desaconsejados y Activar assert como palabra clave. Si desea más información acerca de estas opciones, pulse el botón Ayuda de la ficha Java, en la ficha Generar. También hay opciones adicionales que tienen que ver con la generación en la ficha General de la ficha Generar.

# Configuración de la vía de salida

En el cuadro de diálogo Propiedades de proyecto se puede determinar la vía de salida de los archivos de clase compilados.

Para configurar la vía de salida,

- 1 Seleccione Proyecto | Propiedades de proyecto.
- 2 Seleccione la pestaña Vías de acceso para mostrar la ficha Vías de acceso.



- 3 Pulse el botón puntos suspensivos (...), que se encuentra a la derecha del campo Vía de salida.
- 4 Busque el directorio en el que desea guardar los archivos de clase compilados y selecciónelo. Si el directorio no existe, seleccione el botón Nueva carpeta para crearlo. Pulse Aceptar.
- 5 Pulse Aceptar para cerrar el cuadro de diálogo.

## Consulte

- “Cómo construye JBuilder las vías de acceso” en la página 4-10
- “Localización de los archivos” en la página 4-13

# Compilación de proyectos en un grupo de proyectos

---

**Es una función de JBuilder Enterprise.** Si desea obtener más información sobre la compilación de proyectos en un grupo de proyectos, consulte “[Generación de grupos de proyectos](#)” en la [página 6-6](#).

## Compilación desde la línea de comandos

---

**Son características de JBuilder SE y Enterprise** Puede compilar desde la línea de comandos si utiliza los comandos **bmj** o **bcj**. Para ver la sintaxis y una lista de las opciones, escriba **bmj** o **bcj** en la línea de comandos del directorio <jbuilder>/bin. También es posible generar proyectos desde la línea de comandos de JBuilder. Es posible que necesite ejecutar **CLASSPATH** para definir la variables de entorno de la línea de comandos, de tal forma que se encuentren las clases necesarias.

### **bmj (Make de Borland para Java)**

---

**Es una función de JBuilder SE y Enterprise.** **bmj** es la utilidad Make de Borland para Java. **bmj** compila todos los archivos.java del nodo seleccionado que tienen archivos.class no actualizados o no existentes. **bmj** también compila las clases importadas que tienen archivos.class no actualizados o no existentes.

**bmj** busca archivos de dependencias en la vía de acceso a clases y comprueba las dependencias. Si se especifica un conjunto de archivos fuente, es posible que no se recompilen todos ellos. Por ejemplo, podría determinarse que los archivos de clase están actualizados si se han guardado pero no se han editado desde la última compilación. Es posible forzar la recompilación por medio de la opción **-rebuild**.

Para comprobar un conjunto de módulos interdependientes (o “gráfico”), basta con llamar a **bmj** en el archivo fuente raíz (o en varios archivos fuente raíz, si no están uno debajo de otro). Puede definir este argumento utilizando nombres de fuente, nombres de paquetes, nombres de clases o una combinación de ellos.

#### Consulte

- “[Make de Borland para Java \(bmj\)](#)” en la [página B-12](#)
- “[Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos](#)” en la [página B-2](#)

## bcj (Compilador de Borland para Java)

---

Es una función de JBuilder SE y Enterprise.

El compilador **bcj** es el compilador de Borland para Java. **bcj** compila el código fuente especificado, independientemente de que los archivos .class correspondientes estén actualizados o no. **bcj** compila además todos los archivos .java importados directamente que no tengan un archivo .class. Los archivos .java importados que ya tengan archivos .class no se vuelven a compilar, incluso si los archivos .class no están actualizados. Después de utilizar **bcj**, algunas clases importadas pueden, todavía, tener archivos .class sin actualizar.

**bcj** no comprueba las dependencias, y no utiliza ni genera un archivo de dependencia. **bcj** se limita a compilar los elementos especificados.

### Consulte

- “El compilador de Borland para Java (bcj)” en la página B-7
- “Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos” en la página B-2

## Creación de proyectos desde la línea de comandos

---

Esta es una opción de JBuilder SE y Enterprise

Es posible generar archivos de proyecto y especificar tipos de generación desde la línea de comandos de JBuilder por medio de la opción **-build** del directorio <jbuilder>/bin. Para obtener más información, consulte “Interfaz de la línea de comandos de JBuilder” en la página B-4.

## Cambio entre la línea de comandos y el IDE

---

Si edita un archivo fuera del IDE, asegúrese de incluir en el proyecto el paquete correspondiente, o al menos uno de sus archivos fuente, para que el paquete se compruebe durante la compilación. Si no lo hace así, el cambio no se detecta y el paquete fuente no se recompila.

# Generación de programas en Java

Las funciones de generación disponibles varían según la edición de JBuilder

El sistema de generación de JBuilder, basado en la herramienta de generación en Java, Ant, incluye varias fases de generación. Las fases de generación, que son tipos especiales que siempre crea el sistema de generación para cada proceso de generación, pueden incluir tareas de generación tales como la preparación de archivos ajenos a Java para su compilación, la compilación de archivos Java de código fuente, la recopilación, la distribución, etc... El sistema de generación se puede personalizar y ampliar con la clase del *Creador* de OpenTool.

El compilador de JBuilder, Make de Borland para Java (**bmj**), admite el lenguaje Java sin limitaciones, incluidas las clases internas y los archivos JAR. Debido a que el compilador de JBuilder utiliza la comprobación inteligente de dependencias, el ciclo de compilación y recompilación es más rápido y eficaz. El verificador de dependencias determina la naturaleza de los cambios y sólo compila los archivos necesarios.

## Consulte

- “Comprobación inteligente de dependencias” en la página 5-2
- “Compilación de programas en Java” en la página 5-1
- “Make de Borland para Java (bmj)” en la página B-12

# El sistema de generación de JBuilder

---

El sistema de generación de JBuilder utiliza Ant, una herramienta de generación de código abierto basada en Java para ejecutar generaciones como a la utilización de archivos de generación Ant estáticos. El sistema de generación, ampliable también como una OpenTool, tiene varias ventajas y permite realizar lo siguiente:

- Ampliar el sistema con una OpenTool y realizar un seguimiento de la salida.
- Especificar dependencias entre los tipos de generación, una función de JBuilder Enterprise.
- Generar grupos de proyectos, una función de JBuilder Enterprise.
- Generar proyectos Ant ya existentes en JBuilder, una función de JBuilder Enterprise.
- Filtrar paquetes y eliminarlos del proceso de generación, una función de JBuilder SE y Enterprise.

## Consulte

- “JBuilder Build System Concepts” en la ayuda en línea de OpenTools
- “[Generación de grupos de proyectos](#)” en la página 6-6
- “[Generación con archivos Ant externos](#)” en la página 6-9
- “[Filtrado de paquetes](#)” en la página 6-27

## Términos del sistema de generación

---

Para hacer referencia al sistema de generación, se utilizan los siguientes términos.

**Tabla 6.1** Términos del sistema de generación

Término	Definición
Tarea de generación	Parte del código que se puede ejecutar durante la generación, como, por ejemplo, la compilación en java, FTP, la generación de un archivo JAR, etc.
Destino	Colección de tareas de generación que se van a ejecutar. Los tipos pueden depender de otros tipos. Por ejemplo, si el tipo A depende de los tipos B y C, se ejecutarán B y C antes de que se ejecute A.
Fase	Tipos especiales que crea el sistema de generación de JBuilder para cada generación. Existen ocho fases: seis fases autónomas sin dependencias (Limpiar, Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir) y dos fases que establecen dependencias entre fases (Ejecutar Make y Generar de nuevo). Cada fase cuenta con sus tipos específicos.

---

## Fases de generación

---

Las fases de generación en el IDE de JBuilder incluyen seis fases autónomas sin dependencias y dos fases que establecen dependencias entre otras fases. Cada uno de los proyectos de JBuilder incluyen las siguientes fases autónomas: Limpiar, Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir. Ya que son fases autónomas, una fase se puede ejecutar sin necesidad de ejecutar otra. Cada fase cuenta con sus propios tipos como dependencias. Por ejemplo, SQLJ es una dependencia de la fase de postcompilación.

Existen dos fases adicionales que establecen dependencias entre las seis fases autónomas: Ejecutar Make y Generar de nuevo. Ejecutar Make cuenta con las siguientes dependencias: Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir. Generar de nuevo tiene como dependencias Limpiar y Ejecutar Make.

Ya que el sistema de generación de JBuilder se muestra como una OpenTool puede crear sus propias tareas de generación y especificar las fases ya existentes como dependencias, o bien, no ateniéndose en absoluto a ellas. Consulte "JBuilder build system concepts" en la ayuda en línea de OpenTools para obtener más información sobre cómo ampliar el sistema de generación. Si desea un ejemplo para hacer Creadores, consulte el ejemplo del ofuscador en el directorio samples/opentoolsAPI/Build de JBuilder.

**Tabla 6.2** Fases del sistema de generación

<b>Fases autónomas</b>	
<b>Término</b>	<b>Definición</b>
Limpiar	Elimina todas las salidas generadas, como los archivos .class y los archivos JAR.
Precompilar	Tareas que tienen lugar antes de la compilación. Los archivos IDL, que se convierten en archivos fuente Java antes de la compilación, son ejemplos de un tipo de la precompilación.
Compilar	Generación de archivos de clase Java a partir de archivos fuente Java.
Postcompilar	Tareas que tienen lugar después de la compilación. Es necesario que en esta fase se ejecuten los archivos de clase Java. Pueden ser tipos de esta fase java2iiop y el código enmascarado.
Paquete	Tareas que generan archivos recopilatorios.
Distribuir	Tareas que mueven a una ubicación distinta los archivos ya distribuidos. Por ejemplo, esta fase debe tener una tarea para los archivos FTP.

**Tabla 6.2** Fases del sistema de generación (continuación)

Fases autónomas	
Fases que establecen dependencias	
Término	Definición
Ejecutar Make	Ejecutar Make establece dependencias entre las fases autónomas en el orden siguiente: Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir.
Generar de nuevo	Generar de nuevo tiene como dependencias Limpiar y Ejecutar Make.

## El comando Ejecutar Make

Ejecutar Make es una fase que establece dependencias entre las fases autónomas. Ejecutar Make cuenta con las siguientes dependencias listadas por orden: Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir.

El comando Ejecutar Make no debe confundirse con el make de compilación de Java, que sólo compila los archivos fuente de Java. Para obtener más información acerca del compilador de JBuilder, consulte “[Make de Borland para Java \(bmj\)](#)” en la página B-12 y “[Comprobación inteligente de dependencias](#)” en la página 5-2.

Ejecutar Make ejecuta varias tareas de ejecución, dependiendo de los nodos seleccionados. Los nodos seleccionados pueden ser un proyecto, paquetes, un archivo fuente de Java u otro tipo de nodo, como nodos de recopilatorios, de documentación, de WebApp, de tareas externas de generación o de tipos Ant. Por ejemplo, si ejecuta Make en un nodo de recopilatorio, se genera un archivo de recopilatorios. Si ejecuta Make en un paquete, se compilan los archivos fuente de Java y se copian los recursos de estos paquetes en la vía de salida del proyecto. Al ejecutar Make en un proyecto, se compilan los archivos fuente de Java del proyecto y se ejecutan las tareas de generación adecuadas en cualquier nodo que se pueda generar.

Existen varias formas de Ejecutar Make en un archivo, proyecto, paquete u otro nodo apropiado:

- Seleccione Proyecto | Ejecutar Make del proyecto.
- Seleccione Proyecto | Ejecutar Make <nombre del archivo>.
- Seleccione el botón Ejecutar Make del proyecto en la barra de herramientas, si está disponible.
- Pulse con el botón derecho del ratón en un nodo del panel del proyecto, y, a continuación, seleccione Ejecutar Make.
- Pulse con el botón derecho del ratón en la pestaña del archivo del panel de contenido, y, a continuación, seleccione Ejecutar Make <nombredelarchivo>.

Además, en JBuilder Enterprise se puede ejecutar Make en un grupo de proyectos. Si desea obtener más información sobre los grupos de proyectos, consulte el [Capítulo 3, “Los grupos de proyectos”](#).

## El comando Generar de nuevo

Generar de nuevo es otra fase que establece dependencias entre fases autónomas. Su dependencias son Limpiar y Ejecutar Make. Generar de nuevo borra todas las salidas de generación con Limpiar, para después ejecutar Make. El nodo seleccionado puede ser cualquier cosa que se pueda generar que admita Limpiar. Algunos ejemplos incluyen proyectos, paquetes, archivos fuente de Java, recopilatorios y recursos.

Debido a que Generar de nuevo ejecuta Limpiar y, posteriormente, Ejecutar Make, necesita más tiempo que Ejecutar Make. Pero puede resultar de gran utilidad si desea una generación limpia. Por ejemplo, si ha borrado los archivos fuente de Java, tendría que utilizar Generar de nuevo. Generar de nuevo ejecuta Limpiar, que elimina todas las salidas de generación, incluidos los archivos de clase. Después, se ejecuta Ejecutar Make. Si ejecuta Make después de borrar los archivos fuente de Java, se conservarían los archivos de clase.

**Importante** Si cambia cualquier opción de depuración o de confusión en la ficha Generar de Propiedades de proyecto, deberá generar de nuevo el proyecto para que se activen esos cambios.

Existen varias formas de generar de nuevo en un archivo, proyecto, paquete u otro nodo apropiado:

- Seleccione Proyecto | Generar el proyecto.
- Seleccione Proyecto | Generar de nuevo <nombre del archivo>.
- Pulse con el botón derecho del ratón en el panel del proyecto y seleccione Generar de nuevo.
- Pulse con el botón derecho del ratón en la pestaña del archivo en panel de contenido y seleccione Generar de nuevo < nombredelarchivo>.
- Abra la lista desplegable que se encuentra junto al botón Ejecutar Make de la barra de herramientas y seleccione Generar el proyecto.

Además, en JBuilder Enterprise, se puede generar de nuevo un grupo de proyectos. Si desea obtener más información sobre la generación de grupos de proyectos, consulte [“Generación de grupos de proyectos” en la página 6-6](#).

## El comando Limpiar

El comando Limpiar elimina de los otros tipos todas las salidas de generación, como el directorio `classes`, los archivos JAR, los archivos WAR, etc. Si se utiliza una única vía como vía de acceso y de salida, el directorio de salida no se borra, pero sí se borra la salida de generación. Lo

que hace el comando Limpiar es eliminar todos los elementos que dependen del nodo seleccionado:

- Nodos del proyecto: elimina de forma recursiva el directorio de salida. Esto solamente tiene lugar si el directorio de salida es un subdirectorio del proyecto. El comando Limpiar no elimina el directorio de salida si coincide con el directorio fuente o es un subdirectorio de éste.
- Nodos Java: elimina los archivos .class correspondientes y todos los archivos creados, como los java2iop. También elimina recursos.
- Nodos de paquete: borra los archivos .class correspondiente y cualquier recurso.
- Nodos de recursos: borra las copias en el directorio de salida.
- Nodos de documentación: borra todos los archivos HTML y HTM del directorio de salida de Javadoc.
- Nodos de recopilatorios: borra los archivos recopilatorios y los ejecutables.
- Nodos WebApp: borra los archivos WAR y los directorios WEB-INF/lib y WEB-INF/classes.

Existen varios modos de ejecutar el comando Limpiar:

- Pulse con el botón derecho del ratón en el archivo del proyecto del panel del proyecto y, a continuación, seleccione Limpiar.
- Pulse con el botón derecho del ratón sobre los nodos adecuados del panel del proyecto y seleccione Limpiar.

Al igual que los comandos Ejecutar Make y Generar de nuevo, el comando Limpiar solamente aparece en el menú contextual cuando se seleccionan los nodos adecuados.

## Generación de grupos de proyectos

Es una función de  
JBuilder Enterprise.

Los grupos de proyectos permiten controlar el orden de generación de los proyectos dentro del grupo. Esto resulta de gran utilidad si un proyecto depende de otro. En este caso, puede que le interese crear las dependencias en primer lugar. Por ejemplo, si el proyecto B depende del proyecto A, genere en primer lugar el proyecto A y, a continuación, el proyecto B.

El orden de generación de los proyectos dentro de un grupo de proyectos se puede modificar en la ficha Orden de generación del cuadro de diálogo Propiedades del grupo de proyectos (Proyecto | Propiedades del grupo de proyectos).

**Consulte**

- Capítulo 3, “Los grupos de proyectos”

## Definición del orden de generación de un grupo de proyectos

---

El orden de generación de un grupo de proyectos lo determina el orden de los proyectos en el panel del proyecto. Por ejemplo, si un grupo de proyectos cuenta con dos nodos de proyectos, project1.jpx y project2.jpx, y project1.jpx es el primer nodo dependiente del grupo de proyectos, JBuilder genera primero project1.jpx y, a continuación, project2.jpx. El orden de generación se puede modificar en el cuadro de diálogo Propiedades del grupo de proyectos.

Puede resultar muy útil controlar el orden de generación de un grupo de proyectos si se ha añadido un proyecto a otro en forma de biblioteca necesaria. Si desea que el proyecto necesario se genere en primer lugar, hay que colocar ambos proyectos en un grupo de proyectos y hacer que el necesario esté el primero en el grupo. Para obtener más información, consulte [“Adición de proyectos como bibliotecas necesarias” en la página 3-5](#).

Para cambiar el orden de generación en un grupo de proyectos:

- 1 Abra Propiedades del grupo de proyectos:
  - Seleccione Proyecto | Propiedades del grupo de proyectos.
  - Pulse con el botón derecho del ratón en el nodo del grupo de proyectos del panel del proyecto y seleccione Propiedades.
- 2 Abra la pestaña Orden de generación de la ficha Generar.
- 3 Seleccione un proyecto de la lista y utilice los botones Desplazar hacia arriba o Desplazar hacia abajo para reorganizar el orden.
- 4 Pulse Aceptar para cerrar el cuadro de diálogo. Observe que el orden de los proyectos en el panel del proyecto ha cambiado de acuerdo con el nuevo orden de generación que acaba de especificar.

**Sugerencia**

También se pueden añadir proyectos en el cuadro de diálogo Propiedades del grupo de proyectos. Seleccione el botón Ayuda si desea obtener más información.

## Generación de un grupo de proyectos

---

Para generar o generar de nuevo un grupo de proyectos:

- 1 Defina el orden de generación de los subproyectos en el grupo según se describe en “[Definición del orden de generación de un grupo de proyectos](#)” en la página 6-7.
- 2 Realice una de las operaciones siguientes:
  - Seleccione Proyecto | Ejecutar Make del grupo de proyectos o, bien, Proyecto | Generar de nuevo el grupo de proyectos.
  - En el panel del proyecto, pulse el archivo del grupo de proyectos (.jpgr) con el botón derecho del ratón y, a continuación, seleccione Ejecutar Make del grupo de proyectos o Generar de nuevo el grupo de proyectos.



Ejecutar Make del grupo de proyectos y Generar de nuevo el grupo de proyectos también se encuentran en la barra de herramientas. Por defecto, Ejecutar Make del grupo de proyectos es el botón de la barra de herramientas y Generar de nuevo el grupo de proyectos se encuentra en la lista desplegable junto al botón. Si añade tipos de generación personalizados, también aparecen en la lista desplegable. A las dos primeras opciones de menú de la parte de generación de grupos de proyectos del menú Proyecto se les asigna una combinación de teclas.



### Consulte

- “[El comando Ejecutar Make](#)” en la página 6-4
- “[El comando Generar de nuevo](#)” en la página 6-5

## Adición de tipos de generación de grupos de proyectos al menú Proyecto

---



JBuilder permite añadir nuevos tipos de generación al menú Proyecto para grupos de proyectos, al igual que personalizar el orden de los menús. Puede añadir el comando de menú Limpiar el grupo de proyectos, así como tipos de generación personalizados que especifiquen una colección de tipos de generación que se ejecuten con un comando de menú. Todos los tipos que se añadan al menú Proyecto también aparecen en el menú contextual. Para obtener más información, consulte “[Configuración del menú Proyecto para grupos de proyectos](#)” en la página 6-23.

# Generación con archivos Ant externos

Es una función de JBuilder Enterprise.

Si ya cuenta con un proyecto que utilice Ant, puede ejecutar Ant dentro de JBuilder. Ant es una herramienta de generación basada en Java que utiliza archivos de generación creados en XML. Los archivos de generación utilizan un árbol de tipos en el que se ejecutan varias tareas. Un tipo, que es el conjunto de tareas que hay que ejecutar, puede depender de otros tipos. Entre los ejemplos de tipos se incluyen la compilación, el empaquetado en archivos JARS para la distribución, la limpieza de directorios, etc.

Por ejemplo, el siguiente archivo de generación cuenta con dos tipos, init y compile. El tipo init ejecuta una tarea que crea un directorio build. El tipo compile, que depende del tipo init, ejecuta la tarea **javac** en el directorio src y envía las clases compiladas al directorio build. Como compile depende de init, init debe ejecutarse primero. El directorio build debe crearse antes, para poder compilar las clases. Los archivos de generación también cuentan con un tipo por defecto, que en este ejemplo es compile, y que se ejecuta si no se especifica tipo.

Los archivos de generación pueden tener un conjunto de propiedades con un nombre que distingue entre mayúsculas y minúsculas y con un valor. Las propiedades se pueden utilizar como valores en las tareas, y van entre \${ }. En este ejemplo, la propiedad build tiene un valor build. El tipo init, cuando ejecuta la tarea <mkdir dir="\${build}" />, crea un directorio build de acuerdo con el valor de la propiedad, build.

## Ejemplo de archivo de generación

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="MyProject" default="compile" basedir=".">
<!--
    [ property definitions ]
    [ path and patternset ]
    [ targets ]
-->
<property name="build" value="build"/>
<property name="src" value="src"/>
<target name="init">
    <mkdir dir="${build}" />
</target>
<target name="compile" depends="init">
    <javac srcdir="${src}" destdir="${build}" />
</target>
</project>
```

Si desea obtener más detalles acerca de los archivos de generación, consulte la documentación de Ant en el directorio de JBuilder extras/Ant/docs/ o en <http://jakarta.apache.org/ant/manual/using.html#buildfile>.

### Consulte

- [Capítulo 18, “Tutorial: Generación con archivos Ant”](#)
- El proyecto Jakarta en Apache: <http://jakarta.apache.org/ant>
- Documentación acerca de Ant en <http://jakarta.apache.org/ant/manual/index.html>
- Documentación sobre Ant en el directorio de JBuilder extras/ant/docs/

## Adición de archivos de generación Ant a los proyectos

---

Existen dos modos de añadir archivos de generación Ant a un proyecto: automáticamente, con el Asistente para Ant, o manualmente mediante Proyecto | Añadir archivos/paquetes. Si añade archivos de generación con el Asistente para Ant, JBuilder los reconoce automáticamente como nodos Ant, y muestra iconos Ant para los nodos del archivo de generación. Si añade archivos de generación manualmente mediante Proyecto | Añadir archivos/paquetes, los archivos de generación de nombre `build.xml` son los únicos que se reconocen como archivos de generación Ant. Puede utilizar otros nombres para los archivos de generación, pero debe configurar una opción en las propiedades del nodo para que JBuilder los reconozca como archivos Ant. Además, si el archivo de generación Ant tiene por nombre `build.xml`, la vía de acceso correspondiente al archivo aparece en el panel del proyecto. Para obtener más información sobre cómo cambiar propiedades de los nodos Ant, consulte “[Configuración de las propiedades Ant](#)” en la página 6-15.

### Adición de archivos Ant con el Asistente para Ant

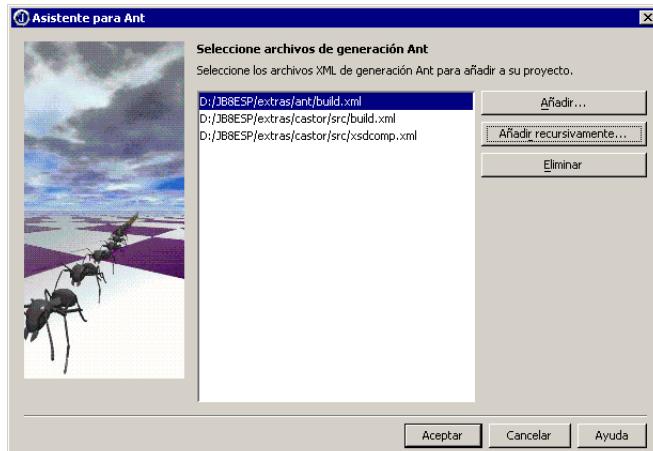
El modo más sencillo de añadir archivos de generación Ant a su proyecto es mediante el Asistente para Ant. El asistente asigna automáticamente el valor del archivo de generación Ant a la propiedad del archivo, para que JBuilder lo reconozca como un nodo Ant, sin tener en cuenta el nombre del archivo. Una vez que haya añadido un archivo de generación al proyecto con la ayuda del asistente, aparece en el panel del proyecto con un ícono Ant.

Para añadir un archivo de generación Ant mediante el asistente:

- 1 Seleccione Archivo | Nuevo, abra la pestaña Generar de la galería de objetos y, a continuación, haga doble clic sobre el ícono Ant. También puede seleccionar Asistentes | Ant.
- 2 Realice una de las operaciones siguientes:
  - Seleccione el botón Añadir, busque todos los archivos de generación XML que desee añadir y, a continuación, pulse Aceptar. Si utiliza el botón Añadir, todos los archivos XML que añada tienen asignado automáticamente el valor de archivo de generación Ant para que

JBuilder los reconozca como un nodo Ant, sin tener en cuenta el nombre del archivo. Una vez que haya añadido un archivo de generación mediante el botón Añadir, este aparece en el panel del proyecto con un ícono Ant.

- Seleccione el botón Añadir con subpaquetes, elija un directorio y pulse Aceptar. JBuilder examina todos los archivos que tengan por nombre `build*.xml` en el directorio seleccionado y en todos sus subdirectorios, y los añade al proyecto.



- 3 Pulse Aceptar para cerrar el asistente.

## Adición manual de archivos Ant

Si añade manualmente los archivos de generación Ant al proyecto, deben tener por nombre `build.xml` para que JBuilder los reconozca automáticamente. Si un archivo tiene un nombre diferente, aparece en el panel del proyecto con el ícono XML. Para que JBuilder lo reconozca como un archivo Ant, debe configurar las propiedades del nodo tal y como se describe en el último paso.

Para añadir manualmente un archivo de generación Ant:

- 1 Seleccione el botón Añadir archivos/paquetes de la barra de herramientas del panel del proyecto o, bien, seleccione Proyecto | Añadir archivos/paquetes.
- 2 Busque y seleccione el archivo de generación que desee añadir.
- 3 Pulse Aceptar.

**Nota**

Si el archivo de generación Ant no se llama `build.xml`, cambie las propiedades del nodo del archivo. Haga clic con el botón derecho del ratón sobre el archivo de generación en el panel del proyecto y seleccione Propiedades. Abra la pestaña Ant de la ficha Propiedades y seleccione la opción Archivo de generación Ant. Pulse Aceptar para

cerrar la ficha Propiedades. El archivo de generación aparece con un icono Ant.

## Creación y modificación de archivos de generación Ant

---

Si no cuenta con un archivo de generación, puede crearlo en el editor de JBuilder, que también ofrece resaltado de sintaxis. También puede utilizar el editor de JBuilder para modificar los archivos de generación Ant ya creados. Si desea generar nuevos archivos de generación en JBuilder:

- 1 Abra un proyecto o cree uno nuevo.
- 2 Seleccione Archivo | Archivo nuevo.
- 3 Escriba el nombre del archivo en el campo Nombre, seleccione XML como extensión de archivo en la lista desplegable Tipo, especifique un directorio para el archivo y pulse Aceptar. Si lo guarda con el nombre `build.xml`, lo reconoce automáticamente como un archivo de generación Ant. Si el archivo no tiene el nombre `build.xml`, debe activar la opción Archivo de generación Ant en Propiedades Ant para que JBuilder pueda reconocerlo como nodo Ant. Consulte “[Configuración de las propiedades Ant](#)” en la página 6-15.
- 4 Escriba texto en el nuevo archivo abierto en el editor.
- 5 Seleccione Proyecto | Añadir archivos/paquetes, seleccione el nuevo archivo y pulse Aceptar para añadirlo al proyecto.
- 6 Pulse Aceptar para guardar el archivo en el proyecto.
- 7 Introduzca la información de generación apropiada del nuevo archivo en el editor y guarde el archivo.
- 8 Pulse el botón de actualización y amplíe el nodo Ant del panel del proyecto para que aparezcan los tipos.

## Importación de proyectos Ant

---

Si ya tiene un proyecto Ant con el que le gustaría trabajar en JBuilder, utilice el asistente Proyecto para código existente. El asistente para proyecto con código existente crea un proyecto nuevo en JBuilder a partir de un proyecto ya creado, derivando las vías de acceso desde el árbol de directorios. JBuilder reconoce de forma automática los archivos `build.xml` de Ant, y los añade al nuevo proyecto en JBuilder. Si el archivo Ant no tiene el nombre `build.xml`, debe activar la opción Archivo de generación Ant en la ficha Propiedades Ant. Consulte “[Configuración de las propiedades Ant](#)” en la página 6-15. Si desea abrir el asistente Proyecto para código existente, seleccione Archivo | Nuevo, pulse la pestaña Proyecto y, a continuación, haga doble clic sobre el ícono Proyecto para código existente.

## Generación de proyectos en Ant

---

Mientras trabaja con un proyecto Ant, puede ejecutar Ant como parte del proceso de generación de JBuilder. Para poder hacer esto, debe activar la opción Ejecutar siempre Ant al generar el proyecto para cualquier nodo Ant que desee incluir en el proceso de generación. Consulte

["Configuración de las propiedades Ant" en la página 6-15](#). Una vez que haya configurado esta opción, los comandos Ejecutar Make y Generar de nuevo ejecutarán Ant como parte del proceso de generación de JBuilder. Si esta opción está desactivada, los comandos Ejecutar Make y Generar de nuevo realizarán el proceso de generación de JBuilder sin ejecutar Ant. Consulte ["Generación de proyectos Ant con el comando Ejecutar" en la página 6-14](#).

### Sugerencia

También puede añadir tipos Ant al menú Proyecto y a la barra de herramientas. Consulte ["Configuración del menú Proyecto" en la página 6-21](#).

Ant se puede ejecutar manualmente desde el panel del proyecto. Pulse con el botón derecho del ratón en el nodo Ant y seleccione Ejecutar Make para ejecutar el tipo por defecto en el archivo de generación Ant. El tipo por defecto es un valor que se especifica en el elemento <project>. Si desea ejecutar varios tipos en el archivo de generación, seleccione uno o varios nodos de tipos, pulse con el botón derecho y, a continuación, seleccione Ejecutar Make.

### Nota

JBuilder puede utilizar distintas vías de acceso y directorios para los archivos fuente, archivos de clase y otros archivos. Puede modificar las vías de acceso a JBuilder para que coincidan con sus tipos Ant en la ficha Vías de acceso de Propiedades de proyecto. También puede modificar las vías de acceso a Ant si cambia las propiedades Ant. Consulte ["Configuración de las propiedades Ant" en la página 6-15](#).

El archivo de salida de Ant aparece en la pestaña Generar del panel de mensajes. Hay dos nodos que pueden mostrar mensajes:

- StdErr: muestra el flujo de salida de errores estándar.
- StdOut: muestra el flujo de salida estándar.

Si desea desplazarse por los archivos que contengan errores, pulse en los mensajes de error del panel de mensajes. Haga doble clic sobre cualquier error para desplazar el cursor al error del código.

Si desea introducir distintos parámetros de destino sin modificar el archivo de generación, pulse con el botón derecho del ratón sobre el nodo Ant, seleccione Propiedades y añada los parámetros. Consulte ["Configuración de las propiedades Ant" en la página 6-15](#).

## Definición del JDK

Por defecto, Ant utiliza el JDK que se distribuye con JBuilder para generar los proyectos. En algunos casos, puede que su proyecto utilice un JDK diferente. Si desea que Ant utilice el mismo JDK que el proyecto, puede definir la opción Utilizar JDK del proyecto al ejecutar Ant en Propiedades de proyecto.

Para configurar Ant para que utilice el JDK del proyecto:

- 1 Seleccione Proyecto | Propiedades de proyecto.
- 2 Abra la pestaña Generar y, a continuación, la pestaña Ant.
- 3 Marque la opción Utilizar JDK del proyecto al ejecutar Ant.
- 4 Pulse Aceptar para cerrar el cuadro de diálogo.

## Generación de proyectos Ant con el comando Ejecutar

Cuando se ejecuta un proyecto Ant project en JBuilder con el comando Ejecutar el proyecto (Ejecutar | Ejecutar el proyecto), JBuilder ejecuta el tipo de generación por defecto, Ejecutar Make. A continuación, JBuilder ejecuta el proyecto sin ejecutar Ant. Si desea ejecutar también Ant con este comando, debe activar la opción Ejecutar siempre Ant al generar el proyecto para cualquier nodo Ant que desee incluir en el proceso de generación. En ese momento, JBuilder ejecuta Make para el proceso de generación de JBuilder y para Ant, utilizando el tipo Ant por defecto en el archivo de generación. Consulte “[Configuración de las propiedades Ant](#)” en la página 6-15. Tras haber activado esta opción, el comando Ejecutar el proyecto genera el proyecto con Ant en el nodo especificado como parte del proceso de generación de JBuilder. A continuación, se ejecuta el programa.

Además, puede cambiar el tipo de generación por defecto, Ejecutar Make, antes de ejecutar el programa. Por ejemplo, quizás desee ejecutar el tipo Ant antes de ejecutar el proyecto. En este caso, no necesitaría seleccionar la opción Ejecutar siempre Ant al generar el proyecto. El tipo de generación se especifica en las configuraciones para la ejecución en el cuadro de diálogo Propiedades para la ejecución (Ejecutar | Configuraciones). Si desea obtener más información sobre como cambiar el tipo de generación, consulte “[Tipos de generación](#)” en la página 7-11.

### Consulte

- “[Generación de proyectos con el comando Ejecutar](#)” en la página 5-5
- “[El comando Ejecutar](#)” en la página 7-4

## Configuración de las propiedades Ant

---

Un proyecto Ant puede tener un conjunto de propiedades. La mayoría de los tipos y tareas Ant están enlazados a la propiedad. Por ejemplo, la propiedad `build.compiler` especifica qué compilador utiliza la tarea `javac`. También se puede especificar si desea que una tarea se ejecute dependiendo de la existencia o ausencia de una determinada propiedad. Las propiedades también se utilizan para pasar parámetros a las tareas sin necesidad de redefinir las propiedades del archivo de generación.

En el cuadro de diálogo Propiedades del archivo de generación se pueden pasar parámetros y controlar las opciones para iniciar Ant. Pulse con el botón derecho del ratón en el nodo Ant del panel del proyecto, seleccione Propiedades y abra la pestaña Ant. En el cuadro de diálogo Propiedades, puede definir las siguientes opciones:

- **Archivo de generación Ant**

Seleccione esta opción para identificar el archivo XML como un archivo de generación Ant. Si se llama a un archivo de generación `build.xml`, se selecciona esta opción por defecto y se desactiva.

- **Mostrar vía de acceso relativa**

Seleccione esta opción para que aparezca en el panel del proyecto la vía de acceso relativa de todos los archivos de generación Ant que tengan por nombre `build.xml`.

- **Nivel del histórico**

Puede elegir entre silencioso, normal, verbose o depurar para la salida del mensaje.

- **Utilizar archivo de registro**

Envía los mensajes de salida a un archivo histórico en lugar de al panel de mensajes.

- **Propiedades**

Cómo añadir nuevas propiedades y modificar las ya existentes sin sobreescribirlas en el archivo de generación. Las modificaciones se guardan en el archivo `<project>.jpx`, no en el archivo XML.

- **Utilizar el compilador Java de Borland**

Le permite utilizar el compilador Borland de Java (`bmj`) para las tareas `javac`.

- **Parámetros de la MV**

Especifica parámetros adicionales de la MV cuando se ejecuta Ant como un proceso externo desde dentro de JBuilder. Por ejemplo, si desea que la memoria dinámica máxima de la MV de Java en que está

ejecutando Ant sea de 256MB, escriba `-Xmx256m` en el campo Parámetros de la MV.

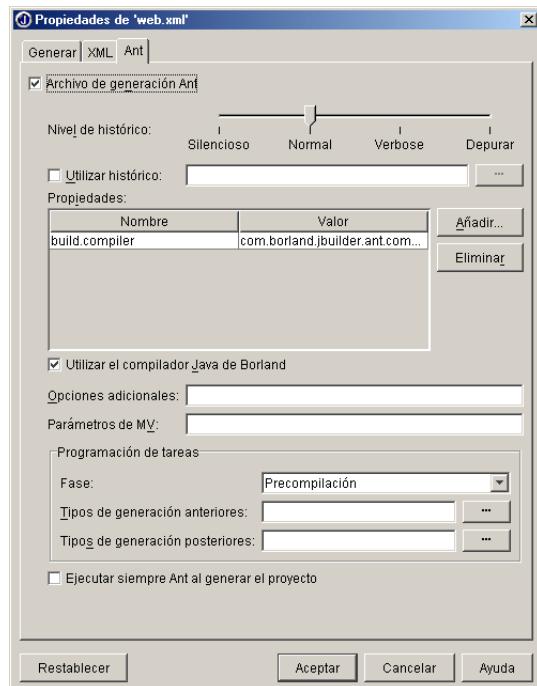
- **Programación de tareas:**

Le permite especificar la fase de generación en que se ha de ejecutar el archivo de generación, y seleccionar los tipos que se han de ejecutar antes y después de que se genere el archivo Ant.

- **Ejecutar siempre Ant al generar el proyecto**

Le permite ejecutar Ant en cualquier nodo Ant que tenga esta opción seleccionada durante la generación del proyecto.

**Importante** Cuando se utilizan estas funciones de JBuilder como, por ejemplo, el perfeccionamiento, se recomienda que acepte la opción por defecto Utilizar el compilador Borland de Java. Si está opción está seleccionada y tiene alguna tarea `javac` en su archivo `build.xml`, esas tareas utilizarán `bmj`. Por ejemplo, `bmj` añade información adicional en los archivos de dependencias que permitan el perfeccionamiento para poder trabajar en determinados casos en los que no se puede recabar la información necesaria para el perfeccionamiento desde los archivos `.class` por sí solos.



Si desea información adicional acerca de la configuración de propiedades Ant, seleccione el botón Ayuda en el cuadro de diálogo Propiedades Ant.

## Opciones Ant

Puede introducir opciones Ant adicionales en el campo Opciones adicionales del cuadro de diálogo Propiedades.

Las opciones son:

-help	imprime el mensaje
-projecthelp	imprime la ayuda del proyecto
-version	imprime los datos de la versión y sale de la aplicación
-emacs	produce información de seguimiento sin adornos
-logger classname	muestra la clase que ha de llevar a cabo el seguimiento
-listener classname	añade una instancia de clase como monitor del proyecto
-find file	busca el archivo de generación en dirección a la raíz del sistema de generación
	y utiliza el primero que encuentra

## Adición de bibliotecas Ant personalizadas

Si sus tipos Ant requieren bibliotecas Ant, puede añadirlas a su proyecto en la ficha Generar del cuadro de diálogo Propiedades de proyecto.

- 1 Seleccione Proyecto | Propiedades de proyecto, y a continuación pulse la pestaña Generar.
- 2 Seleccione la pestaña Ant
- 3 Añada las bibliotecas necesarias y pulse Aceptar.
- 4 Reorganice las bibliotecas de la lista mediante los botones Desplazar hacia arriba o Desplazar hacia abajo.

**Nota** Las bibliotecas se buscan por el orden con que figuran en la lista.

- 5 Pulse Aceptar para cerrar el cuadro de diálogo.

Puede también utilizar una versión diferente de Ant si añade una biblioteca con los JAR de Ant. Si no especifica ningún JAR de Ant, JBuilder utiliza el Ant que se suministra en el directorio `lib` de JBuilder.

## Generación de archivos SQLJ

Es una función de JBuilder Enterprise.

El sistema de generación de JBuilder admite la generación de archivos SQLJ. SQLJ combina el lenguaje de programación Java con SQL (Lenguaje de consulta estructurado), que se utiliza para acceder a bases de datos relacionales. SQLJ, complementario a JDBC, permite que un programa en Java tenga acceso a una base de datos mediante sentencias SQL incrustadas. Una vez que se han incrustado las sentencias SQL, se ejecuta en el programa un traductor de SQLJ. Este traductor cambia el programa SQLJ a Java y sustituye las sentencias SQL por llamadas a la ejecución SQLJ. A continuación, el programa Java se compila y se ejecuta sobre la base de datos. Ya que SQLJ solamente es compatible con el SQL estático,

puede combinarlo con JDBC para que también pueda funcionar con el SQL dinámico.

Para obtener más información sobre SQL y bases de datos, consulte *Guía del desarrollador de aplicaciones de bases de datos*.

JBuilder reconoce los archivos .sqlj del proceso de generación. Para generar archivos .sqlj, en primer lugar debe configurar un traductor y, después, especificar uno para su proyecto tal y como se explica a continuación:

- 1 Configure DB2 o Oracle SQLJ en el cuadro de diálogo Configurar Enterprise tal y como se detalla a continuación:
  - a Seleccione Herramientas | Configurar Enterprise y abra la pestaña SQLJ.
  - b Seleccione una configuración SQLJ, como Oracle o DB2.
  - c Navegue hasta la ubicación del archivo ejecutable.
  - d Introduzca las opciones adicionales.
  - e Añada cualquier biblioteca SQLJ dependiente y controladores JDBC que requiera el traductor SQLJ. Examine la documentación de DB2 u Oracle para ver qué archivos JAR se necesitan en el CLASSPATH. Cree una biblioteca con estos JARs con ayuda del Asistente para bibliotecas y añádala a la configuración de SQLJ.
  - f Haga clic en Aceptar para cerrar el cuadro de diálogo Configurar Enterprise.
- 2 Especifique el traductor de SQLJ que va a utilizar en el proyecto:
  - a Seleccione Proyecto | Propiedades de proyecto y abra la pestaña Generar.
  - b Abra la pestaña General de la ficha Generar.
  - c Seleccione un traductor de SQLJ para el proyecto.
  - d Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

Una vez que su proyecto tenga un traductor SQLJ activo, SQLJ se ejecuta contra los archivos .sqlj de su proyecto como parte del proceso de generación, y los archivos .java generados aparecen como descendientes del nodo SQLJ. Los archivos .java generados son posteriormente compilados como parte del proceso global de generación.

### Consulte

- SQLJ.org en <http://www.sqlj.org>

# Creación de tareas externas de generación

**Es una función de JBuilder Enterprise.**

En algunos casos puede que desee ejecutar tareas externas cada vez que genere un proyecto. Por ejemplo, puede tener un archivo .bat o .exe para Windows o un archivo .sh o un ejecutable para Linux o UNIX que desee ejecutar cada vez que se genere un proyecto. Algunos ejemplos de tareas externas son: el paso de archivos compilados a un ofuscador después de la fase Empaquetar, la distribución de un programa y su incorporación a CVS o el paso de un programa a un preverificador tras la compilación. Con la ayuda del Asistente para tareas externas de generación podrá crear tareas externas que le permitan ejecutar comandos externos shell o de consola como parte del proceso de generación.

## Asistente para tareas externas de generación

Con la ayuda del Asistente para tareas externas de generación se puede generar una tarea externa de generación. Para abrir el asistente, seleccione Archivo | Nuevo, a continuación la ficha Generar, y haga doble clic sobre el ícono Tareas externas de generación.



En el Asistente para tareas externas de generación se pueden configurar las siguientes opciones:

- **Nombre**

Escriba un nombre para el nodo que se muestra en el panel del proyecto.

- **Mostrar salida de consola**

Muestra en el panel de mensajes de JBuilder toda la información de salida de la consola. Si esta opción está desactivada no se muestra ninguna información de salida.

- **Programa**

Puede desplazarse al archivo de programa externo que deseé incluir en la generación.

- **Parámetros**

Le permite introducir parámetros y/o seleccionarlos en la lista Macros.

- **Directorio de ejecución**

Especifique el directorio desde el que se inicia la tarea externa de generación.

- **Planificación de tareas**

Indique la fase de la generación en la que debe ejecutarse la tarea externa y seleccione los elementos que se deben ejecutar antes y después de ella.

Una vez completado el asistente, aparece un nodo en el panel del proyecto. Puede haber varios nodos de tareas externas en un proyecto. Cuando se coloca el ratón encima de cada uno de ellos, aparece la ayuda inmediata con el nombre del ejecutable.

Se pueden añadir tareas externas de generación al menú Proyecto. Consulte “[Configuración del menú Proyecto](#)” en la página 6-21. También se pueden definir como tipos de generación para que se ejecuten antes que el proyecto. Consulte “[Tipos de generación](#)” en la página 7-11.

## Generación de tareas externas

---

Si desea generar solamente el nodo de tareas externas, pulse con el botón derecho del ratón en el panel del proyecto y seleccione Ejecutar Make. Si se selecciona la opción Mostrar salida de consola, se envían todos los mensajes a la pestaña Generar del panel de mensajes. Hay dos nodos que pueden mostrar mensajes:

- StdErr: muestra el flujo de salida de errores estándar.
- StdOut: muestra el flujo de salida estándar.

Seleccione Proyecto | Ejecutar Make del proyecto para generar todo el proyecto y el nodo de tareas externas.

## Configuración de propiedades de las tareas externas de generación

Las tareas externas de generación poseen un conjunto de propiedades que se configuran de forma inicial en el Asistente para tareas externas de generación. Entre estas propiedades se incluyen el nombre, el ejecutable que se va a utilizar, la programación de tareas y los parámetros. La propiedad Programación de tareas determina en qué fase se va a ejecutar la tarea. Por ejemplo, si su tarea externa de generación es un ofuscador, puede configurarla en la fase Empaquetar. También puede especificar los tipos que tienen lugar antes y después de esa fase.

Si desea modificar cualquiera de estas propiedades después de crear la tarea externa de generación, pulse con el botón derecho del ratón en el nodo del panel del proyecto y seleccione Propiedades.

## Configuración del menú Proyecto

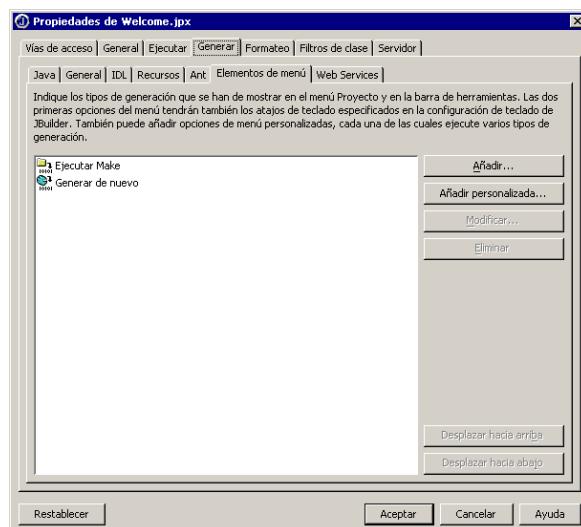
Es una función de JBuilder Enterprise.

Para mayor comodidad, JBuilder le permite configurar el primer grupo del menú Proyecto. Por defecto, Ejecutar Make y Generar de nuevo son los tipos del menú Proyecto. Además, puede añadir más tipos y tareas de generación, como Limpiar, tareas externas de generación y los tipos Ant, si su proyecto lo requiere.

En el menú Proyecto, se le asignan teclas por defecto a los dos primeros elementos. El primer elemento del menú aparece en la barra de herramientas. Junto al botón del menú de la barra de herramientas, hay un menú desplegable que contiene la opción Generar de nuevo, a menos que la haya eliminado del menú y haya añadido otros tipos personalizados al menú Proyecto.

Se pueden modificar estas opciones por defecto y añadir tipos personalizados en la ficha Elementos del menú de la ficha Generar, en Propiedades de proyecto. En el caso de los dos primeros tipos de la ficha Elementos de menú, las teclas asignadas pueden personalizarse. Las asignaciones de teclas de estos dos tipos se pueden modificar en el Editor de configuración de teclado, que se encuentra en la ficha Visualizador del cuadro de diálogo Opciones del IDE (Herramientas | Opciones del IDE | Visualizador | Personalizar). El primer tipo de la lista aparece, con el icono

adecuado, en la barra de herramientas principal con una lista desplegable del resto de tipos añadidos al menú Proyecto.



Si desea añadir un tipo de proyecto al menú Proyecto:

- 1 Seleccione Proyecto | Propiedades de proyecto, y a continuación pulse la pestaña Generar.
- 2 Pulse en la pestaña Elementos de menú.
- 3 Realice una de las operaciones siguientes:
  - Pulse el botón Añadir y seleccione un tipo de la lista. Después, seleccione Aceptar para cerrar el cuadro de diálogo Añadir tipo de generación al menú.
  - Pulse el botón Añadir personalizada para añadir al menú una opción personalizada que ejecute varios tipos de generación. Escriba el nombre del menú en el campo Etiqueta de menú, pulse el botón Añadir y, a continuación, seleccione los tipos que deseé. Los tipos se ejecutan en el orden en el que figuran en la lista. Utilice los botones Desplazar hacia arriba o Desplazar hacia abajo para reorganizar la lista. Pulse Aceptar para cerrar el cuadro de diálogo Nuevo tipo personalizado.
- 4 Puede cambiar el orden de los tipos en la ficha Elementos de menú se selecciona un tipo y, a continuación, Desplazar hacia arriba o Desplazar hacia abajo. Esta operación cambia el orden de los tipos en el menú Proyecto. Tal y como se ha indicado anteriormente, el primer tipo de la lista aparece en la barra de herramientas, y a los dos primeros, se le puede asignar la tecla que se deseé.

- 5** Seleccione Aceptar para cerrar Propiedades de proyecto. Los nuevos tipos aparecen en el menú Proyecto y en el menú desplegable junto al tipo de la barra de herramientas.

**Nota** Para configurar el menú Proyecto para otros proyectos, realice los cambios en el cuadro de diálogo Propiedades por defecto para proyectos.

### Consulte

- “[Creación de tareas externas de generación](#)” en la página 6-19
- “Configuración de teclado de las emulaciones de editores” en *Introducción a JBuilder*

## Configuración del menú Proyecto para grupos de proyectos

---

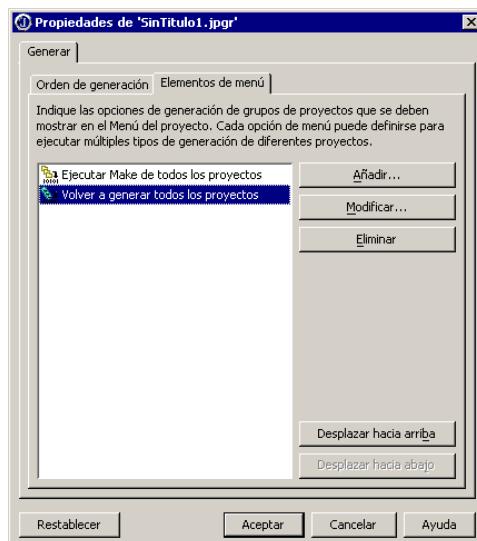
Es una función de JBuilder Enterprise.

JBuilder también permite configurar el menú Proyecto para grupos de proyectos. Puede añadir Limpiar el grupo de proyectos al menú Proyecto, así como tipos que contengan colecciones de tipos contenidos en los subproyectos. Por defecto, Ejecutar Make del grupo de proyectos y Generar de nuevo el grupo de proyectos son los tipos de generación de grupos de proyectos que se encuentran en el menú Proyecto. Todos los tipos que se añadan al menú Proyecto también aparecen en el menú contextual.

En el menú Proyecto, se le asignan teclas por defecto a los dos primeros elementos de menú para grupos de proyectos. El primer elemento de menú para grupos de proyectos aparece en la barra de herramientas. Junto al botón del menú de la barra de herramientas, hay un menú desplegable que contiene la opción Generar de nuevo el grupo de proyectos, a menos que la haya eliminado del menú y haya añadido otros tipos personalizados al menú Proyecto para grupo de proyectos.

Se pueden modificar estas opciones por defecto y añadir tipos personalizados en la ficha Elementos del menú de la ficha Generar, en Propiedades del grupo de proyectos. En el caso de los dos primeros tipos de la ficha Elementos de menú, las teclas asignadas pueden personalizarse. Las asignaciones de teclas de estos dos tipos se pueden modificar en el Editor de configuración de teclado, que se encuentra en la ficha Visualizador del cuadro de diálogo Opciones del IDE (Herramientas | Opciones del IDE | Visualizador | Personalizar). El primer tipo de la lista aparece, con el ícono adecuado, en la barra de herramientas

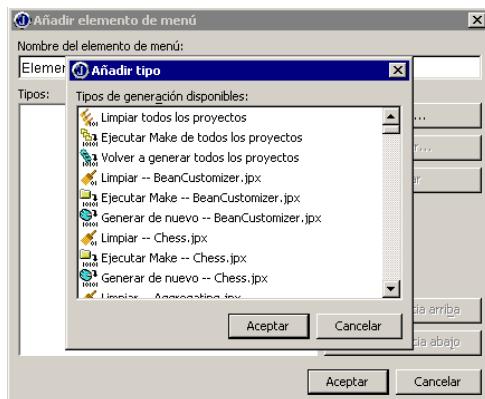
principal con una lista desplegable del resto de tipos añadidos al menú Proyecto.



Si desea añadir un tipo de grupo de proyectos al menú Proyecto:

- 1 Elija Proyecto | Propiedades de grupo de proyectos para abrir el cuadro de diálogo Propiedades del grupo de proyectos. También se puede conseguir haciendo clic con el botón derecho del ratón sobre el nodo del grupo de proyectos en el panel del proyecto y seleccionar Propiedades.
- 2 Seleccione la pestaña Generar y, a continuación, la pestaña Elementos de menú.
- 3 Pulse Añadir para abrir el cuadro de diálogo Añadir elemento de menú.
- 4 Escriba un nombre de menú para el tipo.
- 5 Pulse el botón Añadir, seleccione los tipos que desee añadir y, a continuación, pulse Aceptar.
- 6 Seleccione un tipo de la lista y utilice los botones Desplazar hacia arriba o Desplazar hacia abajo en el cuadro de diálogo Añadir elemento de menú para reorganizar los tipos de la lista. Los tipos se ejecutan en el orden en el que figuran en la lista.

- 7 Haga clic en Aceptar para cerrar el cuadro de diálogo Añadir tipo.



- 8 Seleccione en la lista un elemento de menú y utilice los botones Desplazar hacia arriba o Desplazar hacia abajo de la ficha Elementos de menú para cambiar el orden del tipo en el menú Proyecto.

- 9 Haga clic en Aceptar para cerrar el cuadro de diálogo.

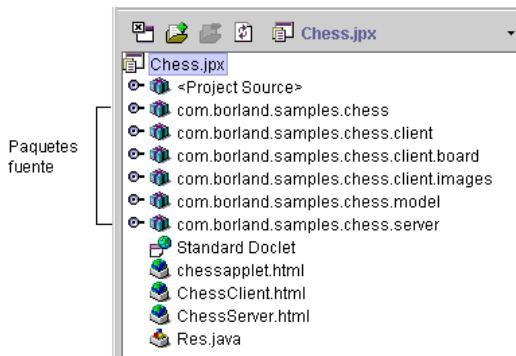
El nuevo tipo aparece ahora en el menú Proyecto con los demás tipos de generación de grupos de proyectos.

## Recopilación automática de paquetes fuente

Es una función de JBuilder SE y Enterprise.

Si se activa la recopilación automática de paquetes fuente, todos los paquetes de la vía de fuentes del proyecto se muestran automáticamente en el panel del proyecto. Durante la generación con esta opción activada, cualquier paquete que contenga archivos que se puedan generar se genera automáticamente y se copia conjuntamente con los recursos en la vía de salida del proyecto. Por ejemplo, si un proyecto contiene archivos Java y SQLJ, los archivos Java son compilados y SQLJ se ejecuta contra cualquier archivo SQLJ. Por defecto, JBuilder considera que los recursos son archivos de imagen, sonido y propiedades. Los recursos se pueden definir en archivos individuales y por extensión de archivos en todo el proyecto. Consulte “[Copia selectiva de los recursos](#)” en la página 6-29 para obtener más información sobre los recursos. Si desea más información acerca de

las vías de salida de un proyecto, consulte “[Configuración de la vía de salida](#)” en la página 5-10.



La recopilación automática de paquetes fuente está activada por defecto y se encuentra en la ficha General del cuadro de diálogo Propiedades de proyecto. Si desea cambiar la configuración, seleccione Proyecto | Propiedades de proyecto y abra la ficha General. Active o desactive la opción Activar la localización y compilación de paquetes fuente. También es posible controlar el número de niveles de paquetes que se muestran en el panel del proyecto modificando el valor en el campo Niveles jerárquicos mostrados. Para más información acerca de esta característica, pulse el botón Ayuda en la ficha General (Proyecto | Propiedades de proyecto).

Para minimizar el número de nodos de paquete en el listado, JBuilder muestra automáticamente un subconjunto de los paquetes del proyecto. Aunque es posible que algunos de los paquetes fuente no aparezcan en los primeros niveles del proyecto, JBuilder los genera.

**Importante** Despu  s de añadir nuevos archivos fuente al proyecto, seleccione el bot  n Actualizar en la barra de herramientas del proyecto para actualizar la lista de recopilaci  n autom  tica de fuentes.

También se muestra un nodo <Fuente del proyecto> en la parte superior del panel del proyecto cuando est   activada la caracter  stica Recopilaci  n autom  tica de paquetes fuente. Este nodo contiene todos los paquetes fuente y todos los archivos fuente del proyecto, excepto los paquetes y filtros que se han añadido manualmente. Puede utilizar este nodo para filtrar r  pidamente paquetes fuente. Si desea m  s informaci  n sobre el filtrado, consulte “[Filtrado de paquetes](#)” en la p  gina 6-27.

**Nota** Si en su proyecto hay tipos de archivos que JBuilder no reconoce, puede añadirlos como archivos de recursos gen  ricos. Para obtener m  s informaci  n, consulte “[Adici  n de tipos de archivos no reconocidos como archivos de recursos gen  ricos](#)” en la p  gina 6-31.

Para obtener más información sobre los paquetes Java, consulte el “Chapter 7: Packages” en Java Language Specification en [http://java.sun.com/docs/books/jls/second\\_edition/html/packages.doc.html#60384](http://java.sun.com/docs/books/jls/second_edition/html/packages.doc.html#60384).

### Consulte

- “Cómo construye JBuilder las vías de acceso” en la página 4-10
- Ficha General del cuadro de diálogo Propiedades de proyecto

## Filtrado de paquetes

---

**Es una función de JBuilder SE y Enterprise.**

JBuilder cuenta con una característica de filtrado que permite excluir paquetes del proceso de generación. Sin embargo, si el verificador de dependencias de JBuilder determina que existe alguna dependencia entre las clases de los paquetes excluidos, esas clases se compilan. Cuando se excluyen paquetes de un proyecto, aparece la carpeta Filtros de paquetes en el panel del proyecto. Esta carpeta contiene una descripción general de los filtros que se aplican a los paquetes del proyecto. Los iconos de la carpeta indican qué filtro se ha utilizado. Consulte la [Tabla 6.3, “Iconos del filtrado de paquetes”, en la página 6-29](#) para obtener más información sobre las definiciones.

**Importante**

La función recopilación automática de paquetes fuente, que es la opción por defecto, debe activarse en la ficha General de Propiedades de proyecto. Si ha anidado profundamente los paquetes y sólo ha excluido algunos, es recomendable que aumente el número en el campo Niveles jerárquicos mostrados para que puedan mostrarse más paquetes en el panel del proyecto. Para obtener más información sobre la recopilación automática de paquetes, consulte [“Recopilación automática de paquetes fuente” en la página 6-25](#).

También se muestra un nodo <Fuente del proyecto> en la parte superior del panel del proyecto cuando está activada la característica Recopilación automática de paquetes fuente. Este nodo contiene todos los paquetes fuente y todos los archivos fuente del proyecto, excepto los paquetes y filtros que se han añadido manualmente. Puede utilizar este nodo para filtrar rápidamente paquetes fuente. Una vez que se ha excluido el nodo, aparece en la carpeta Filtros de paquetes y no en la parte superior del panel del proyecto.

Los paquetes y archivos añadidos manualmente no se pueden excluir mediante el comando Aplicar filtro. Tiene que eliminarlos del proyecto para poder excluirlos del proceso de generación. Asimismo, los nodos generables descendientes del nodo del proyecto, tales como los archivos de código fuente Java añadidos por los asistentes, también se compilan, y no se pueden excluir a menos que se eliminen del proyecto. En resumen,

cualquier archivo o paquete que aparezca sobre la carpeta Filtros de paquetes no se filtran y se incluyen en el proceso de generación.

## Exclusión de paquetes

---

Para excluir paquetes del proceso de generación:

- 1 Seleccione en el panel del proyecto los paquetes que deseé excluir.
- 2 Pulse con el botón derecho del ratón y seleccione Aplicar filtro, o bien, Proyecto | Aplicar filtro.
- 3 En el submenú, elija uno de los siguientes comandos de menú:
  - Excluir paquete y subpaquetes: excluye los paquetes seleccionados y sus subpaquetes.
  - Excluir paquete: excluye el paquete seleccionado, pero no sus subpaquetes.

**Nota** Si se excluyen el paquete y los subpaquetes, el paquete no aparece en la parte superior del panel del proyecto. Solamente aparecerá en la carpeta Filtros de paquetes. Si se excluye un paquete, pero se incluye un subpaquete, aparece con el ícono correspondiente en la parte superior del panel del proyecto y en la carpeta Filtros de paquetes.

Si desea excluir todos los paquetes fuente del proyecto:

- 1 Pulse con el botón derecho del ratón en el nodo <Fuente del proyecto> del panel del proyecto, y seleccione Aplicar filtro.
- 2 En el submenú, seleccione Excluir paquete y subpaquetes.

En algunos casos, el proceso de filtrado puede componerse de dos pasos. Por ejemplo, si desea excluir todos los paquetes excepto algunos subpaquetes, debe realizar lo siguiente:

- 1 Seleccione el nodo <Fuente del proyecto> en el panel del proyecto y, a continuación, Proyecto | Aplicar filtro.
- 2 En el submenú, seleccione Excluir paquete y subpaquetes.
- 3 Abra la carpeta Filtros de paquetes.
- 4 De la lista desplegable, seleccione los subpaquetes que deseé incluir.
- 5 Pulse con el botón derecho y seleccione Aplicar filtro | Incluir paquete.

La configuración del filtro se guarda de forma local en el archivo del proyecto. Cualquier nueva configuración del filtro que se aplique a un paquete, sustituye a la configuración anterior para ese paquete.

## Inclusión de paquetes

---

Una vez que se han excluido los paquetes, se pueden volver a incluir en el proceso de generación.

- Seleccione Proyecto | Eliminar todos los filtros.
- Pulse con el botón derecho del ratón en la carpeta Filtros de paquetes y seleccione Eliminar todos los filtros.
- Ampliación de la carpeta Filtros de paquetes. Pulse con el botón derecho del ratón sobre uno o varios paquetes y seleccione Aplicar filtro. En el submenú, elija uno de los siguientes comandos de menú:
  - Incluir paquete y subpaquetes: incluye los paquetes seleccionados junto con sus subpaquetes.
  - Incluir paquete: incluye el paquete seleccionado, pero no los subpaquetes.

**Tabla 6.3** Iconos del filtrado de paquetes

Icono	Comando Menú	Descripción
	Excluir paquete y subpaquetes	Excluye del proceso de generación los paquetes seleccionados y todos sus subpaquetes.
	Incluir paquete y subpaquetes	Incluye en el proceso de generación los paquetes seleccionados y todos sus subpaquetes.
	Excluir paquete	Excluye del proceso de generación solamente los paquetes seleccionados, pero no excluye los subpaquetes.
	Incluir paquete	Incluye en el proceso de generación solamente los paquetes seleccionados, pero no incluye los subpaquetes.

## Copia selectiva de los recursos

---

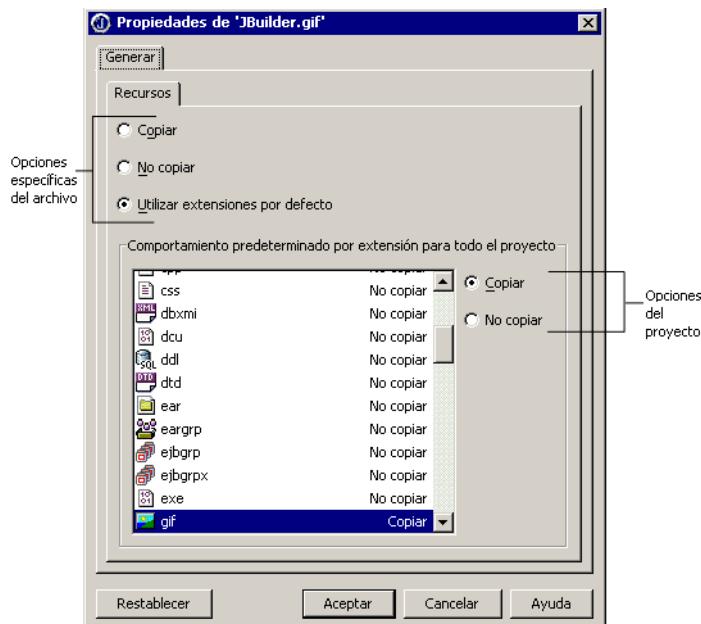
Es una función de JBuilder SE y Enterprise.

JBuilder copia todos los tipos de recursos conocidos de la vía de acceso a archivos fuente del proyecto a la vía de salida durante el proceso de compilación. Por defecto, JBuilder reconoce todos los archivos de imagen, sonido y propiedades como recursos, y los copia en la vía de salida. Estas definiciones de recursos por defecto se pueden modificar en archivos individuales o en todo el proyecto según la extensión de archivo. Consulte “Configuración de la vía de salida” en la página 5-10 para obtener más información sobre la vía de salida.

## Propiedades de recursos individuales

Si desea cambiar las opciones predeterminadas para archivos individuales en un proyecto, seleccione un archivo en el panel del proyecto, haga clic en él con el botón derecho, a continuación, seleccione Propiedades y abra la pestaña Recursos de la ficha Generar.

**Figura 6.1** Ficha Recursos del cuadro de diálogo Propiedades de proyecto



### Opciones específicas de archivos

Los tres botones superiores de radio son opciones específicas de los archivos que controlan los archivos seleccionados. Estas opciones son:

- Copiar: copia el archivo o archivos seleccionados en la vía de salida.
- No copiar: no copia el archivo o archivos seleccionados en la vía de salida.
- Utilizar extensiones por defecto: utiliza las extensiones por defecto tal y como aparecen en la lista Comportamiento predeterminado por extensión para todo el proyecto.

Las opciones Copiar y No copiar realizan una acción absoluta: copiar siempre o no copiar nunca en la vía de salida cuando se genera el proyecto, sin tener en cuenta de si normalmente se considera o no al tipo de archivo como un recurso.

La tercera opción, Usar extensiones de archivo por defecto, permite que JBuilder distribuya el archivo según la extensión definida en la lista

inferior. Este es el comportamiento por defecto de todos los archivos nuevos y los archivos de proyectos existentes. Las extensiones correctas de los archivos seleccionados se eligen automáticamente en la lista para resaltar el comportamiento por defecto.

- Importante** Si los archivos o extensiones seleccionados no comparten la misma configuración, no se selecciona **ninguno** de los botones de radio del grupo correspondiente. Al seleccionar uno de los botones de radio, se cambia todo al mismo valor, mientras que si no se selecciona ninguno, los valores distintos se dejan aparte.
- Si cambia las opciones para archivos individuales y desea que vuelvan a ser las opciones por defecto del proyecto, seleccione de nuevo los archivos y elija la opción Utilizar extensiones por defecto.

### Opciones específicas del proyecto

A continuación de las tres opciones específicas de los archivos, se encuentra una lista de todos los comportamientos predeterminados por extensión y su comportamiento de distribución por defecto. Estas opciones por defecto pueden cambiar según el proyecto. Seleccione una o más extensiones y utilice los botones de radio a la derecha para cambiar el comportamiento por defecto de esas extensiones en el proyecto actual. Entre las opciones específicas del proyecto se incluyen:

- Copiar: copia el archivo o archivos seleccionados en la vía de salida.
- No copiar: no copia el archivo o archivos seleccionados en la vía de salida.

Utilice el botón Restablecer para que todos los archivos de la lista de extensión de archivos vuelvan al estado en el que se encontraban cuando se mostró, inicialmente, el cuadro de diálogo. Recuerde que esta acción no devuelve las opciones de sus archivos individuales a las predeterminadas.

También se pueden configurar estas opciones para los proyectos futuros en el cuadro de diálogo Propiedades por defecto para proyectos (Proyecto | Propiedades por defecto para proyectos).

### Adición de tipos de archivos no reconocidos como archivos de recursos genéricos

Si cuenta con tipos de archivos que no reconoce JBuilder, puede asociarlos con el tipo de archivo de recursos genérico. De este modo, JBuilder los reconoce y los incluye en el proceso de localización automática de fuentes. También puede agruparlos en recopilatorios. Por ejemplo, JBuilder no reconoce archivos Flash, pero puede tener acceso a ellos en su proyecto y distribuirlos en un recopilatorio con el resto del proyecto. Si desea agrupar un tipo de archivo no reconocido en un recopilatorio, debe seguir un proceso de dos pasos. En primer lugar, es necesario que lo añada como un

tipo de archivo reconocido. A continuación, debe configurarlo para que se copie en la vía de salida cuando se genere el recopilatorio.

- 1** Para añadir tipos de archivos no reconocidos a la lista de tipos reconocidos para JBuilder, lleve a cabo estos pasos:
  - a** Seleccione Herramientas | Opciones del IDE y abra la pestaña Tipos de archivo.
  - b** Seleccione Archivo de recursos genéricos en la lista Tipos de archivo reconocidos.
  - c** Pulse el botón Añadir, escriba la extensión del nuevo archivo y, a continuación, pulse Aceptar.
  - d** Haga clic en Aceptar para cerrar el cuadro de diálogo Opciones del IDE.

**Importante**

Por defecto, todos los tipos de archivo que se añadan como archivos de recursos genéricos **no** se incluyen en los recopilatorios. Es necesario que configure el nuevo tipo de archivo para que se copie en la vía de salida en la pestaña Recursos de la ficha Generar de Propiedades de proyecto. Pase al paso siguiente para llevarlo a cabo.

- 2** Para incluir el nuevo tipo de archivo en su recopilatorio, siga estos pasos:
  - a** Seleccione Proyecto | Propiedades de proyecto, y a continuación pulse la pestaña Generar. Para incluir el nuevo tipo de archivo para todos los proyectos venideros, seleccione Proyecto | Propiedades por defecto para proyectos.
  - b** Seleccione la pestaña Recursos y el nuevo tipo de archivo en la lista Tipos de archivo reconocidos. Observe que, por defecto, se activa No copiar.
  - c** Seleccione el botón circular Copiar para hacer que este tipo de archivo se copie en la vía de salida.
  - d** Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

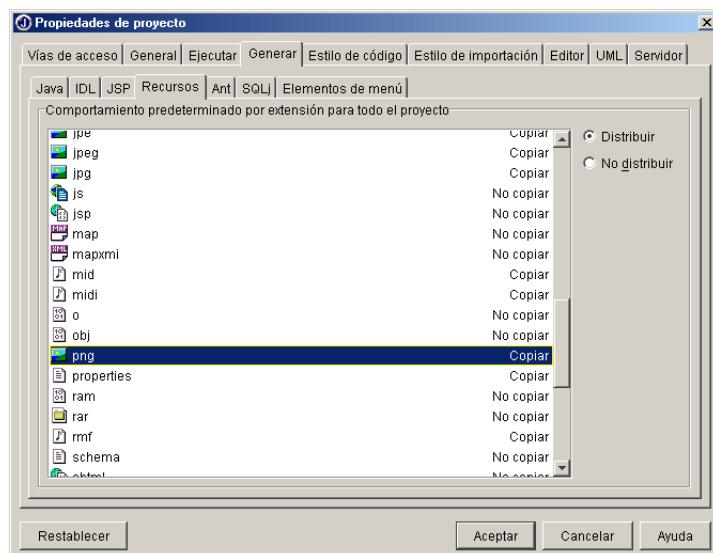
## La ficha Recursos del cuadro de diálogo Propiedades de proyecto

---

Es una función de JBuilder SE y Enterprise.

El cuadro de diálogo Propiedades de proyecto posee la correspondiente pestaña Recursos en el ficha Generar, que proporciona control sobre los comportamientos por defecto de las extensiones de archivo del proyecto completo, en lugar de hacerlo siguiendo un criterio de archivo individual. Utilice el botón Restablecer para que todos los archivos de la lista de

extensión de archivos vuelvan al estado en el que se encontraban cuando se mostró el cuadro de diálogo.





# Ejecución de programas en Java

Cuando esté preparado para comprobar el programa, puede optar por limitarse a ejecutarlo o depurarlo y ejecutarlo simultáneamente. Cuando se ejecuta el programa, JBuilder utiliza la vía de acceso a clases para localizar todas las clases que emplea el programa. Para entender como JBuilder encuentra los archivos para ejecutar el programa, consulte “[Cómo construye JBuilder las vías de acceso](#)” en la página 4-10 y “[Localización de los archivos](#)” en la página 4-13.

Existen varias formas de ejecutar los archivos. Es posible ejecutar un sólo archivo, como un archivo HTML applet o un archivo de aplicación haciendo clic con el botón derecho sobre el archivo en el panel del proyecto y seleccionando el comando Ejecutar. También se puede ejecutar un proyecto si se selecciona el botón Ejecutar proyecto en la barra de herramientas principal.

El botón Ejecutar proyecto puede configurarse con parámetros de ejecución predeterminados y guardarse como una selección de menú desplegable. Consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7. Debe tener creada al menos una configuración de ejecución para poder ejecutar un proyecto utilizando F9 o el botón Ejecutar de la barra de herramientas. Si tiene varias configuraciones, puede seleccionar una de ellas para utilizarla como configuración por defecto en la ejecución.

# Ejecución de archivos de programa

Para ejecutar un archivo de programa en un proyecto,

- 1 Guarde el archivo `.java`.
- 2 Seleccione el archivo `.java` que contiene el método `main()` y realice una de las acciones siguientes:
  - Pulse el botón derecho del ratón sobre el archivo en el panel del proyecto y seleccione Ejecutar. Si tiene varias configuraciones de ejecución también debe seleccionar en el submenú la que desea utilizar.
  - Haga doble clic sobre el archivo del panel del proyecto para abrirlo o, si ya está abierto, selecciónelo para convertirlo en el archivo activo. Haga clic con el botón derecho del ratón sobre la pestaña Nombre del archivo en el panel de contenido y seleccione Ejecutar.
  - Pulse la flecha contigua al ícono Ejecutar de la barra de herramientas y elija una configuración de la lista desplegable.
  - Seleccione Ejecutar | Ejecutar proyecto (*F9*) en el menú principal. De esta forma se ejecuta la clase principal indicada en la configuración por defecto de la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto. Consulte “[Ejecución de proyectos](#)” en la página 7-3. Si no se elige ningún valor por defecto y sólo hay una configuración, se ejecuta la clase principal indicada en ella.

En JBuilder SE o Enterprise:

- Si el proyecto tiene varias configuraciones de ejecución pero no se ha establecido una por defecto, se pide al usuario que seleccione una y a continuación prosigue la ejecución.
- Si no existe ninguna configuración para el proyecto, aparece la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto, donde se puede crear una configuración de ejecución cuando se pulse el botón Ejecutar o se pulse *F9*. Puede ejecutar sin configuraciones de ejecución haciendo clic con el botón derecho en el archivo que se va a ejecutar en el panel del proyecto y seleccionar Ejecutar utilizando la configuración por defecto.

**Nota** Si no hay configuración de ejecución, después de crear una debe volver a ejecutar la aplicación.

La aplicación se compila y, si no se producen errores, se ejecuta. El avance de la generación se muestra en la barra de estado, y el panel de mensajes muestra los errores de compilación. Antes de la compilación, los mensajes de error se muestran en la carpeta `Errores` del panel de estructura. Si el programa se compila correctamente, se muestra la línea de comandos de Java en el panel de mensajes, y el programa se ejecuta.

## Ejecución de archivos Web

---

**Las applets son una función de todas las ediciones de JBuilder.**

**Es una característica de JBuilder SE y Enterprise.**

Con JBuilder también se pueden ejecutar archivos Web, tales como JSP, servlet, SHTML y HTML por medio de un servidor Web para obtener una vista dinámica de la aplicación Web. Para ejecutar un applet, haga clic con el botón derecho del ratón en el archivo HTML que contiene la etiqueta <applet> y elija Ejecutar. Para ejecutar archivos JSP, SHTML, y HTML, pulse el botón derecho sobre el archivo en el panel del proyecto y seleccione Ejecutar web.

En el caso de los servlets, primero debe configurar la opción de menú Activar Ejecutar/Depurar/Optimizar en el menú contextual. Si se crean las servlets con el Asistente para servlets, de JBuilder, esta opción se activa automáticamente. Para activar esta opción, pulse sobre el archivo servlet con el botón derecho en el panel del proyecto y seleccione Propiedades. Active la opción de menú Activar Ejecutar/Depurar/Optimizar en el menú contextual en la ficha Ejecutar web. Tras hacer esto, pulse el archivo servlet con el botón derecho y seleccione Ejecutar web.

### Consulte

- “Las applets”, en la *Guía del desarrollador de aplicaciones Web*
- Consulte “Las aplicaciones web en JBuilder” en la *Guía del desarrollador de aplicaciones Web*
- “Creación de servlets en JBuilder” en la *Guía del desarrollador de aplicaciones Web*
- “Páginas JavaServer (JSP)” en la *Guía del desarrollador de aplicaciones Web*

## Ejecución de proyectos

---



El proyecto puede ejecutarse seleccionando Ejecutar | Ejecutar proyecto, pulsando *F9* o haciendo clic en el botón Ejecutar proyecto en la barra de herramientas principal. De esta forma se ejecuta la clase principal seleccionada en la configuración de ejecución por defecto. Si no se elige ningún valor por defecto y sólo hay una configuración, se utiliza de forma predeterminada. Si se tienen varias configuraciones y no hay ninguna por defecto, JBuilder solicita que se elija una configuración de ejecución.

Las configuraciones de ejecución se definen en la ficha Ejecutar del cuadro de diálogo Propiedades del proyecto, seleccionado Ejecutar | Configuraciones en el menú principal o seleccionando Configuraciones en el menú desplegable del icono Ejecutar de la barra de herramientas. También se puede crear una configuración de ejecución por defecto en el último paso de algunos asistentes, como los de aplicaciones, applets y servlets y tests.

Si aún no se ha seleccionado una clase main, se muestra el cuadro de diálogo para seleccionarla. Si el archivo se ha creado con el Asistente para aplicaciones, la clase main se selecciona automáticamente.

Con JBuilder SE y Enterprise también es posible definir configuraciones de ejecución y depuración para los botones Ejecutar proyecto y Depurar proyecto. Dichas configuraciones se añaden a los menús desplegables a los que se puede acceder desde la flecha que se encuentra a la derecha de cada botón y desde los elementos del menú Ejecutar.

### Consulte

- “Definición de las configuraciones para la ejecución” en la página 7-7
- “Ejecución de tests” en la página 13-15

## El comando Ejecutar

---

Cuando se elige el comando Ejecutar (Ejecutar | Ejecutar Proyecto) o se pulsa el botón Ejecutar proyecto de la barra de herramientas principal, JBuilder ejecuta el programa según la configuración de ejecución por defecto o la seleccionada. Muchos de los asistentes de JBuilder pueden crear una configuración para la ejecución y establecer la clase principal de forma automática si decide hacerlo así. El comando Ejecutar y el botón Ejecutar proyecto ejecutan también el Tipo de generación definido en la configuración de ejecución que se utiliza.

Para ejecutar el proyecto sin depurarlo:

- 1 Guarde los archivos.
- 2 Compruebe que se ha seleccionado una clase principal en la configuración de ejecución que se va a utilizar. Si las configuraciones de ejecución no están definidas, se pueden añadir y modificar desde la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto.
- 3 Elija Ejecutar | Ejecutar proyecto, pulse *F9* o haga clic en el botón Ejecutar en la barra de herramientas.



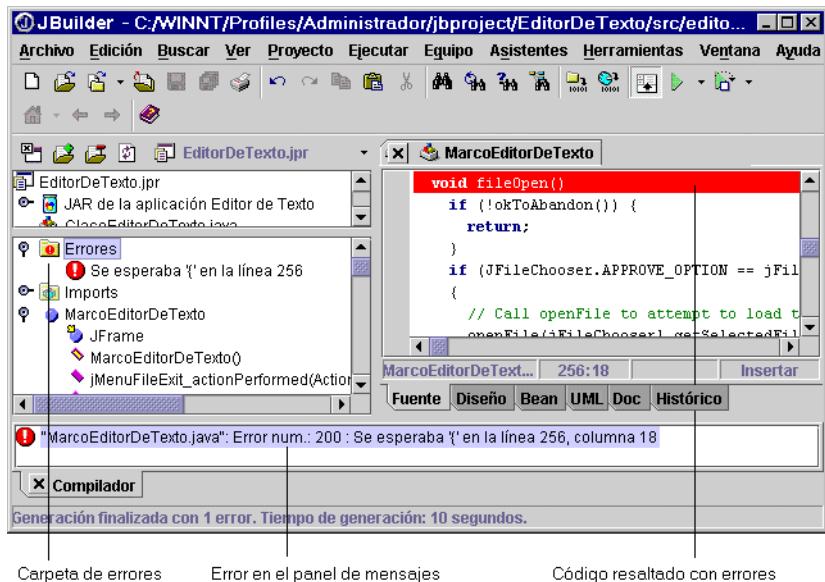
### Sugerencia

Otra posibilidad consiste en pulsar sobre el proyecto con el botón derecho del ratón, en el panel del proyecto, elegir Limpiar, Make o Generar de nuevo y pulsar a continuación el botón Ejecutar de la barra de herramientas.

JBuilder ejecuta el Tipo de generación elegido y el programa. Los errores que se produzcan durante la compilación se muestran en el panel de mensajes de la parte inferior del Visualizador de aplicaciones. Si se producen errores, la compilación se detiene para que pueda solucionarlos e intentarlo de nuevo. Seleccione un error en el panel de mensajes o en la carpeta Errores del panel de estructura para resaltar el código en el panel

fuente. Si necesita ayuda con un mensaje de error, seleccione el mensaje en el panel y pulse *F1* (Ayuda).

**Figura 7.1** Mensajes de error en el Visualizador de aplicaciones



### Consulte

- [Capítulo 6, “Generación de programas en Java”](#)
- [Capítulo 5, “Compilación de programas en Java”](#)

## Ejecución de proyectos agrupados

Para ejecutar los proyectos de un grupo de proyectos (sólo JBuilder Enterprise), pulse el proyecto con el botón derecho del ratón en el panel del proyecto y seleccione Ejecutar. El botón Ejecutar de la barra de herramientas sólo ejecuta el proyecto activo, no el grupo de proyectos. La lista desplegable Configuración de ejecución de la barra de herramientas muestra las configuraciones de ejecución del proyecto activo y las demás configuraciones de ejecución que se hayan especificado como disponibles para el grupo de proyectos.

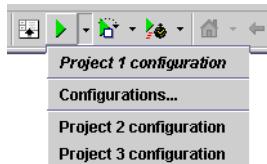
Puede especificar qué configuraciones de ejecución del grupo de proyectos se han de añadir a la lista desplegable Configuración de ejecución al crear o modificar cada configuración de ejecución de los proyectos del grupo.

Para que aparezcan las configuraciones de ejecución del proyecto:

- 1 Abra el grupo de proyectos.

- 2** Pulse con el botón derecho del ratón un proyecto del grupo y seleccione Propiedades.
- 3** Abra la pestaña Ejecutar del cuadro de diálogo Propiedades de proyecto y, a continuación, seleccione la configuración de ejecución que desee. (Consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7 para obtener información sobre la creación y edición de configuraciones de ejecución.)
- 4** Asegúrese de que el cuadro Grupo está seleccionado para esta configuración. (Grupo está seleccionado por defecto.)
- 5** Repita este paso para cualquier otra configuración del proyecto, o para otros proyectos que desee que estén disponibles en la lista desplegable Configuraciones de ejecución y, a continuación, pulse Aceptar.

Ahora, si pulsa la flecha hacia abajo junto al ícono Ejecutar de la barra de herramientas, además de las configuraciones de ejecución del proyecto actual, se ven todas las demás configuraciones de ejecución de los demás proyectos del grupo.



Si desea obtener más información sobre los grupos de proyectos, consulte el [Capítulo 3, “Los grupos de proyectos”](#).

## Ejecución de OpenTools

---

JBuilder cuenta con un tipo de configuración de ejecución OpenTool en JBuilder Enterprise que permite ejecutar, depurar y optimizar su proyecto OpenTool en JBuilder como cualquier otro proyecto. Ya no tiene que cerrar JBuilder, crear un archivo JAR, copiarlo en el directorio <JBuilder home>\lib\ext o <JBuilder home>\patch ni reiniciar JBuilder.

Al ejecutar un proyecto OpenTool mediante el ejecutor OpenTool, se genera un archivo de configuración temporal en el directorio <outputpath>\\.\\ config-temp, y se inicia una nueva instancia de JBuilder mediante ese archivo de configuración temporal. Este archivo de configuración se elimina cuando se cierra el seguimiento.

Para utilizar este ejecutor OpenTool, debe crear una nueva configuración para el proyecto OpenTool que sea del tipo OpenTool mediante los siguientes pasos:

- 1** Abra su proyecto OpenTool en JBuilder.

- 2** Seleccione Ejecutar | Configuraciones, o bien, pulse la flecha hacia abajo del botón Ejecutar y seleccione Configuraciones para abrir la ficha Ejecutar del cuadro de diálogo Propiedades del proyecto.
- 3** Pulse Nuevo para abrir el cuadro de diálogo Propiedades de configuración de ejecución.
- 4** Escriba un nombre para la configuración y seleccione un Tipo de generación.
- 5** Indique la ubicación de los archivos que se han de utilizar durante la ejecución: los que se encuentran en el directorio del proyecto OpenTool, o los que están en el archivo JAR.
- 6** Defina los parámetros que desea que se pasen a la MV o a JBuilder durante la ejecución, y la ubicación de la vía de acceso a <JBuilder Home>.
- 7** Seleccione Redefinir las clases y recursos si desea que las clases y recursos OpenTool sean los primeros en la vía de acceso a clases
- 8** Pulse Aceptar para volver a la ficha Ejecutar del cuadro de diálogo Propiedades del proyecto.
- 9** Marque cualquiera de las casillas que se aplican a esta configuración (Por defecto, Menú contextual o Grupo), y utilice los botones Desplazar hacia arriba o Desplazar hacia abajo para desplazarse a la ubicación que prefiera en la lista. Este es el mismo orden en que las configuraciones aparecen en los menús y listas desplegables de JBuilder.
- 10** Pulse Aceptar cuando haya terminado.

Si necesita ayuda para definir estas configuraciones de ejecución, pulse el botón de la ayuda de la parte inferior del cuadro de diálogo Propiedades de configuración de ejecución.

## Definición de las configuraciones para la ejecución

---

En la versión de JBuilder Personal solamente hay una configuración para la ejecución.

Las configuraciones de ejecución (Ejecutar | Configuraciones) son parámetros de tiempo de ejecución predefinidos. La utilización de parámetros prestablecidos ahorra mucho tiempo durante la ejecución y la depuración, porque así sólo se definen una vez. Estas configuraciones prestablecidas permiten, cada vez que ejecute o depure la aplicación, simplemente seleccionar la configuración deseada.

Las configuraciones para la ejecución se gestionan en la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto. Desde ahí, puede añadir, modificar, copiar o eliminar las configuraciones, y controlar el orden en que aparecen en los menús de selección. También puede designar una de las configuraciones como la configuración por defecto para utilizarla cuando ejecute o depure sin elegir una configuración, y definir qué

configuraciones han de aparecer en los menús contextuales de un proyecto en concreto, o de un grupo de proyectos.

Para definir las configuraciones de ejecución, abra la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto de uno de estos modos:

- Seleccione Proyecto | Propiedades de proyecto y, a continuación, pulse la pestaña Ejecutar.
- Seleccione Ejecutar | Configuraciones.
- Pulse la flecha abajo que se encuentra junto a los botones Ejecutar o Depurar y, a continuación, seleccione Configuraciones en el menú desplegable.
- Pulse con el botón derecho del ratón sobre el archivo de proyecto del panel del proyecto y seleccione Propiedades y, a continuación, abra la pestaña Ejecutar.

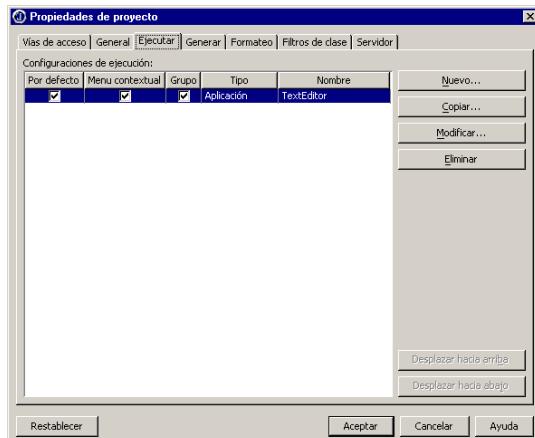
Algunos de los asistentes de JBuilder, como los asistentes para aplicaciones, applets, servlets o tests, le ofrecen la opción de crear una configuración para la ejecución.

- Si solamente cuenta con una configuración para la ejecución, JBuilder utiliza esa configuración cuando se selecciona Ejecutar | Ejecutar el proyecto, (*F9*), o Ejecutar | Depurar proyecto (*Mayús-F9*).
- Si especifica una configuración por defecto, JBuilder muestra esa configuración en negrita (o cursiva dependiendo del aspecto) en las listas desplegables de los botones Ejecutar y Depurar de la barra de herramientas, y la utiliza al ejecutar o depurar el proyecto.
- Si no ha especificado una configuración por defecto, al seleccionar Ejecutar o Depurar, se le solicita que elija qué configuración va a utilizar.
- Si necesita unas configuraciones inexistente para que se ejecute el tipo de archivo, cuando seleccione Ejecutar en el menú, pulse *F9* o haga clic sobre el botón Ejecutar de la barra de herramientas con el fin de que se abra la ficha Ejecutar de Propiedades de proyecto para que pueda crear una configuración en ese momento. Sin embargo, si hace clic con el botón derecho del ratón sobre el archivo ejecutable del panel del proyecto y selecciona Ejecutar, el Creador lo ejecutará automáticamente basándose en las configuraciones por defecto que considera apropiadas para ese tipo de archivo.

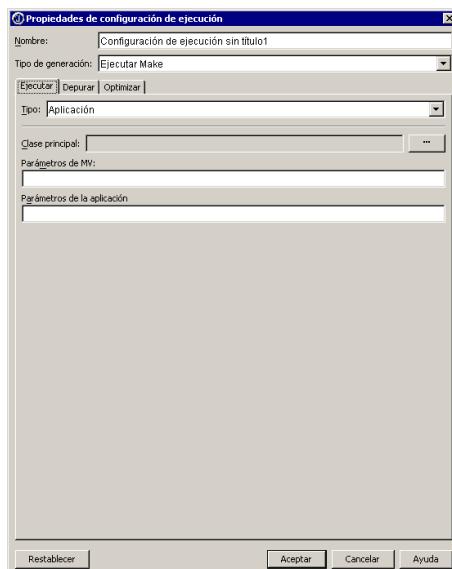
## Creación de una configuración de ejecución

Para crear una configuración de ejecución:

- 1 Seleccione Ejecutar | Configuraciones, o bien, seleccione Proyecto | Propiedades de proyecto, y abra la pestaña Ejecutar.



- 2 Pulse Nuevo para abrir el cuadro de diálogo Propiedades de configuración de ejecución.

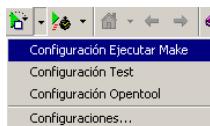


- 3 Escriba un nombre para la configuración si desea modificar el nombre que aparece por defecto. Este nombre se añade a las listas desplegables y a los menús contextuales Ejecutar el proyecto y Depurar proyecto.

- 4 Seleccione un Tipo de generación para esta configuración en la lista desplegable de las opciones de generación.
- 5 Seleccione el tipo de configuración de ejecución adecuado para su proyecto, como Aplicación, Applet, Servidor, Test, OpenTool o MIDlet si tiene MobileSet instalado. (Los tipos de configuraciones para la ejecución se excluyen mutuamente unos a otros.) Especifique la clase principal que va a ejecutar, si es aplicable, y escriba los parámetros que desee de la MV y de la aplicación. Si desea obtener más información acerca de las opciones disponibles para un tipo de ejecutor específico, seleccione ese tipo y, a continuación, pulse el botón de ayuda en la parte inferior del cuadro de diálogo.

Si desea ver una descripción de los tipos de configuración, consulte “[Tipos de configuraciones de ejecución](#)” en la página 7-13.
- 6 Abra la pestaña Depurar y seleccione las opciones de depuración que desee.
- 7 Pulse la pestaña Optimizar si ha instalado Optimizeit y ha configurados sus opciones.
- 8 Pulse Aceptar para volver a la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto y realice los cambios en las casillas de selección de esa ficha.
  - Marque la casilla Por defecto para la configuración de ejecución que va a utilizar cuando pulse el botón Ejecutar o Depurar, o bien, pulse (*F9*) o (*Mayús-F9*).
  - Marque la casilla Menú contextual para cada configuración que desee que aparezca en el menú contextual del proyecto.
  - Pulse el botón Grupo si este proyecto pertenece a un grupo y desea que la configuración esté disponible en la lista de configuraciones del grupo de proyectos.
- 9 Añada las configuraciones adicionales que desee. Cuando haya instalado todas las configuraciones, colóquelas en el orden que desee mediante los botones Desplazar hacia arriba o Desplazar hacia abajo. El modo en que aparecen en esta lista es el modo en que aparecen en los menús desplegables Ejecutar/Depurar.
- 10 Seleccione la configuración por defecto en la columna Por defecto. Si tiene varias configuraciones y no selecciona una como la configuración por defecto, JBuilder le advertirá de que elija una cuando ejecute la aplicación.
- 11 Marque en la columna Menú contextual todas aquellas configuraciones que desee que aparezcan en los menús contextuales de configuración.

Para seleccionar una configuración de ejecución, pulse la flecha hacia abajo que se encuentra junto a los iconos Ejecutar o Depurar de la barra de herramientas principal.



Si se producen errores, excepciones en tiempo de ejecución u otro mal comportamiento del programa, es posible que desee depurar el programa conforme se ejecuta, para detectar problemas. Esto se consigue ejecutando el programa desde el Depurador integrado de JBuilder.

### Consulte

- [Capítulo 8, “Depuración de programas en Java”](#)

## Modificación de una configuración de ejecución

---

Se puede modificar todo lo relacionado con una configuración de ejecución existente, menos su tipo. Si desea un tipo de configuración diferente, debe crear una nueva.

Para modificar una configuración de ejecución ya creada:

- 1 Seleccione Ejecutar | Configuraciones, o bien, pulse la flecha hacia abajo junto al botón Ejecutar de la barra de herramientas y, a continuación, seleccione Configuraciones.
- 2 Seleccione una configuración de ejecución en la ficha Ejecutar y pulse Modificar o haga doble clic en el nombre de la configuración para abrir el cuadro de diálogo Propiedades de configuración de ejecución.
- 3 Realice los cambios deseados a la configuración en el cuadro de diálogo Propiedades de configuración de ejecución, y pulse Aceptar para salir de él.
- 4 Realice modificaciones adicionales a las configuraciones de ejecución en la ficha Ejecutar, como, por ejemplo, desplazarlas en la lista para cambiar el orden en que aparecen en la lista desplegable o cambiar la configuración por defecto. (Si desea más detalles sobre estas opciones, pulse el botón de ayuda en la parte inferior de la ficha Ejecutar.)
- 5 Haga clic en Aceptar cuando haya finalizado.

## Tipos de generación

---

Para cada configuración de ejecución que cree, puede especificar en el cuadro de diálogo Propiedades de configuración de ejecución qué tipo de generación desea ejecutar en primer lugar. JBuilder proporciona un conjunto de tipos estándar entre los que elegir, y dependiendo de su

proyecto, se añaden otros a la lista. A continuación se recogen los elementos que aparecen en la lista Tipos de generación:

- **<Ninguno>**

Se ejecuta sin compilar en primer lugar.

- **Ejecutar Make**

Ejecutar Make establece dependencias entre las fases autónomas en el orden siguiente: Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir.

- **Generar de nuevo**

Generar de nuevo tiene como dependencias Limpiar y Ejecutar Make.

- **Limpiar**

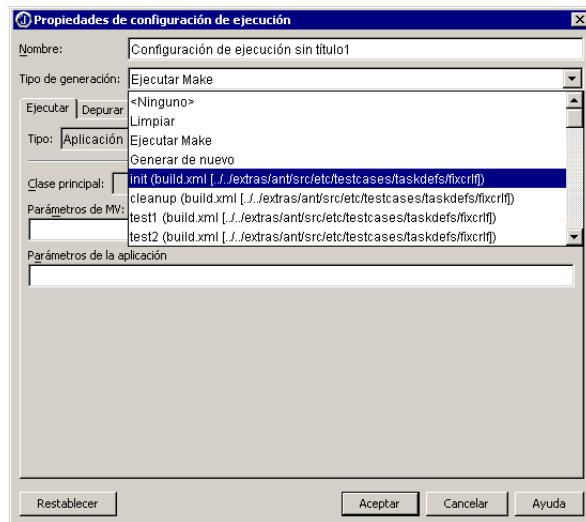
Limpiar elimina todos los archivos de salida de los otros tipos, como el directorio `clases`, los archivos JAR, WAR, etc. Los elementos que elimina Limpiar dependen del nodo seleccionado.

- **Tareas externas de generación** (sólo en JBuilder Enterprise)

Si ha generado una tarea externa de generación, aparecerá en la lista de tipos disponibles entre los que puede seleccionar la configuración para la ejecución. Para obtener más información, consulte “[Creación de tareas externas de generación](#)” en la página 6-19.

- **Tipos de archivos Ant** (sólo en JBuilder Enterprise)

Si tiene en su proyecto algún archivo de generación Ant, los tipos dentro del archivo Ant se añaden a la lista de tipos disponibles del cuadro de diálogo Propiedades de configuración de ejecución.



Además, si ha ampliado su sistema de generación con la ayuda de OpenTools, en la lista aparece cualquiera de esos tipos de tareas de generación.

### Consulte

- [Capítulo 6, “Generación de programas en Java”](#)
- [Capítulo 5, “Compilación de programas en Java”](#)

## Tipos de configuraciones de ejecución

---

Los tipos de configuraciones de ejecución disponibles varían según la versión de JBuilder. Cada tipo cuenta con un conjunto diferente de propiedades de ejecución en este cuadro de diálogo.

- **Aplicación**

Seleccione el tipo Aplicación para una aplicación y pulse el botón de puntos suspensivos. Busque el archivo clase que contiene el método `main()`. La clase principal debe residir en el proyecto actual.

- **Applet**

Seleccione el tipo Applet para un proyecto applet y lleve a cabo una de las siguientes acciones:

- Seleccione Clase principal y haga clic sobre el botón puntos suspensivos para buscar la clase que contenga el método `init()`. Esta opción ejecuta el applet en el visualizador de applets de JBuilder, AppletTestbed.
- Seleccione el archivo HTML y haga clic sobre el botón puntos suspensivos para buscar el archivo HTML de applet que contiene la etiqueta `<applet>`. La opción HTML ejecuta el applet en el `appletviewer` de Sun.

- **Servidor**

Seleccione el tipo Servidor y configure los parámetros del servidor que se utiliza. Consulte “Las aplicaciones web en JBuilder” en la *Guía del desarrollador de aplicaciones Web*. Si desea más información sobre los parámetros de ejecución del servidor, pulse el botón Ayuda.

- **Test**

Si tiene una configuración de ejecución para el test o para el conjunto de tests, seleccione el tipo Test. Elija la clase principal de la clase del conjunto de pruebas, configure los parámetros de la MV y seleccione el ejecutor de tests deseado. Si desea más información sobre los parámetros de ejecución de tests, pulse el botón Ayuda.

- **OpenTool**

Seleccione el tipo OpenTool para establecer una configuración de ejecución para poder ejecutar, depurar y optimizar un proyecto OpenTool directamente desde JBuilder.

El tipo de configuración de ejecución OpenTool permite ejecutar, depurar y optimizar el proyecto OpenTool en JBuilder del mismo modo que otros proyectos. Ya no tiene que cerrar JBuilder, crear un archivo JAR, copiarlo en el directorio <JBuilder home>\lib\ext or <JBuilder home>\patch y reiniciar JBuilder.

Si ejecuta un proyecto OpenTool mediante el ejecutor OpenTool, se genera un archivo de configuración temporal en el directorio <outpath>\...\config-temp, y se inicia una nueva ventana de JBuilder mediante este archivo temporal. Este archivo de configuración se elimina cuando se cierra el seguimiento.

- **MIDlet**

Seleccione el tipo MIDlet (si tiene MobileSet instalado) para establecer una configuración de ejecución para poder ejecutar y depurar un MIDlet J2ME en JBuilder.

## Ejecución de programas desde la línea de comandos

---

Para ejecutar programas fuera de JBuilder, todas las bibliotecas que necesita el programa se deben poner en la vía de acceso a clases o añadirse al argumento **-classpath** del comando `java`. Por ejemplo:

```
java -classpath /<jbuilder>/lib/dbswing.jar  
/<inicio>/jbproject/mypackage/classes/mypackage.application1
```

En el ejemplo:

- <jbuilder> es el nombre del directorio JBuilder
- <inicio> = el directorio inicial, como por ejemplo, `c:\winnt\profiles\<nombre de usuario>`

## Ejecución del programa distribuido desde la línea de comandos

---

Tras la distribución del programa por medio del Creador de recopilatorios o la herramienta **jar**, es posible ejecutar el archivo JAR desde la línea de comandos.

- 1 Abra la ventana de línea de comandos.

**Sugerencia** En Windows, utilice barras invertidas (\) con todos los comandos mencionados.

- 2** Introduzca el siguiente comando en una línea del indicador desde cualquier ubicación:

```
java -classpath nombre-de-paquete.nombre-de-clase-principal
```

**Nota** El directorio <jdk>/bin/ debe estar en la vía de acceso. <jdk> representa el nombre del directorio raíz de JDK.

Por ejemplo, el comando podría tener el siguiente aspecto:

```
java -classpath /<inicio>/jbproject/hello/classes/HelloWorld.jar  
hello.HelloWorldClass
```

En este ejemplo, <inicio> representa el directorio inicial, como por ejemplo, c:\winnt\profiles\<nombre de usuario>.

Puede utilizar también la opción **-jar**:

```
java -jar HolaATodos.jar
```

**Nota** Primero, debe cambiarse al directorio que contiene el archivo .jar para poder ejecutar este comando con la opción **-jar**.

### Consulte

- [Capítulo 6, “Generación de programas en Java”](#)
- [Capítulo 5, “Compilación de programas en Java”](#)
- [Capítulo 15, “Distribución de programas en Java”](#)
- El tutorial JAR en <http://java.sun.com/docs/books/tutorial/jar/index.html>
- [Apéndice B, “Las herramientas de línea de comandos”](#)



# 8

## Depuración de programas en Java

La depuración es el proceso de búsqueda y solución de errores en el programa. El depurador integrado de JBuilder permite depurar dentro del entorno de JBuilder. El menú Ejecutar permite acceder a numerosas funciones del depurador. También se puede acceder a las funciones de depuración por medio de los menús contextuales del editor y el depurador.

Cuando el depurador interrumpe el programa es posible comprobar los valores actuales de los elementos de datos del programa. La modificación de los valores de los datos durante una sesión de depuración proporciona una manera de comprobar posibles soluciones a los errores durante la ejecución. Si encuentra una modificación que soluciona un error de programa, puede salir de la sesión de depuración, implementar la solución necesaria en el código del programa y recompilar para que la solución sea permanente. Si está depurando con JDK 1.4 o superior, no necesita salir de la sesión de depuración para que la solución de errores tenga efecto. Consulte “[Modificación del código durante la depuración](#) en la página 8-72 para obtener más información.

Si desea ver un tutorial sobre depuración, consulte el [Capítulo 17, “Tutorial: Tutorial de compilación, ejecución y depuración”](#). Si tiene JBuilder Personal, consulte *Creación de aplicaciones con JBuilder* en la ayuda en línea para ver una versión del tutorial específicamente diseñada para las características de JBuilder Personal.

Existe información adicional y sugerencias sobre estos temas relacionados con la depuración:

- Si el programa utiliza un componente `JDataStore`, consulte el capítulo “Resolución de problemas” de la *Guía del desarrollador de JDataStore*. (JBuilder Enterprise)
- Si está depurando una aplicación distribuida, consulte el [Capítulo 9, “Depuración remota”](#). (JBuilder Enterprise)
- Si va a depurar un test de módulos, consulte el [Capítulo 13, “Test de módulos”](#). (JBuilder Enterprise).

Para efectuar la depuración fuera de JBuilder, utilice la herramienta `jdb` del directorio `<jdk> /bin`. Consulte la documentación JDK en <http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html> si desea obtener más información sobre esta herramienta.

## Tipos de errores

---

El depurador puede ayudar a encontrar errores de ejecución y errores lógicos. Si encuentra o sospecha que existe un error de ejecución o un error lógico en el programa, se puede comenzar la sesión de depuración ejecutando el programa bajo el control del depurador.

### Errores de ejecución

---

Si su programa contiene sentencias válida, pero producen errores cuando se están ejecutando, se trataría de un error en el tiempo de ejecución. Por ejemplo, el programa podría estar intentando abrir un archivo que no existe o dividir un número por cero.

Sin un depurador, los errores de ejecución pueden ser difíciles de localizar, porque el compilador no proporciona información sobre ellos. A menudo, la única pista de la que se dispone es el resultado de la ejecución, como por ejemplo el aspecto de la pantalla y el mensaje de error generado.

Aunque puede encontrar los errores de ejecución buscándolos en el código fuente del programa, el depurador le ayudará a hacerlo más rápidamente. El depurador, permite ejecutar un programa hasta una ubicación determinada. Desde ahí, puede comenzar la ejecución del programa una sentencia tras otra, observando el comportamiento del programa en cada paso. Cuando se ejecuta la sentencia que hace fallar al programa, se detecta el error. A partir de ahí, puede solucionar los problemas del código fuente y reanudar el test.

Si el programa produce una excepción de ejecución, imprimirá un seguimiento de la pila en la Vista de salida, entrada y errores de consola. Puede pulsar sobre el nombre del archivo subrayado y el número de línea

en el seguimiento de la pila para desplazarse a la línea de código en el archivo fuente que aparece en dicha pila. (Es una función de JBuilder SE y Enterprise.)

## Errores lógicos

---

Los errores lógicos son errores en el diseño o la implementación del programa. Las sentencias del programa son sintácticamente correctas (realizan operaciones), pero las acciones que éstas realizan no son exactamente las que el programador tenía en mente. Por ejemplo, los errores lógicos pueden producirse cuando las variables contienen valores incorrectos, cuando los gráficos no son los adecuados o cuando los datos producidos por el programa son incorrectos.

Los errores lógicos son con frecuencia los más difíciles de encontrar porque pueden aparecer en los lugares más insospechados. Para asegurarse de que el programa funciona como estaba previsto en su diseño, es necesario comprobar exhaustivamente todos sus aspectos. Sólo si se comprueban con detalle todos los elementos de la interfaz de usuario y el resultado del programa, puede tener garantías de que su comportamiento se ajusta a lo esperado. Al igual que con los errores de ejecución, el depurador ayuda a encontrar errores lógicos al permitirle hacer un seguimiento de los valores de las variables del programa y de los objetos de datos durante la ejecución.

## Descripción general del proceso de depuración

---

Después de la fase de diseño del programa, el desarrollo de éste se basa en ciclos de codificación y depuración. Sólo después de una comprobación exhaustiva del programa, se puede pensar en distribuirlo. Para asegurarse de que comprueba todos los aspectos del programa, es mejor tener un plan minucioso de comprobación y depuración.

Un buen método de depuración incluye la división del programa en secciones diferentes que puedan depurarse sistemáticamente. Mediante el seguimiento cuidadoso de las sentencias de cada una de las secciones del programa, es posible comprobar que todas las áreas del programa funcionan como se espera. Si encuentra un error de programación, puede corregir el problema en el código fuente, recomilar el programa y reanudar la comprobación a continuación.

JBuilder Enterprise permite depurar código fuente distinto de Java, incluidas la Páginas JavaServer (JSP). Puede establecer un punto de interrupción en un archivo fuente distinto de Java y depurar ese archivo de forma local o remota. También puede alternar entre la visión del código fuente distinto de Java y la del código Java generado.

Para obtener más información, consulte “[Depuración de fuente distinta de Java](#)” en la página 8-30.

- Nota** Es posible depurar con un JDK que admita la API de depuración JPDA. Por lo general, se depura con la versión de JDK que acompaña a JBuilder. (Esta es la JDK por defecto que se selecciona para el proyecto, si no se ha definido y seleccionado ninguna otra.)

## Creación de una configuración de ejecución

---

Antes de ejecutar o depurar, necesita crear una configuración para la ejecución. Esta configuración está compuesta por un conjunto de parámetros preconfigurados. La utilización de parámetros prestablecidos ahorra mucho tiempo durante la ejecución y la depuración, porque así sólo se definen una vez. Estas configuraciones prestablecidas permiten, cada vez que ejecute o depure la aplicación, simplemente seleccionar la configuración deseada. Se establecen las opciones del depurador, tales como los parámetros del Paso inteligente o las opciones de depuración remota, a través de la configuración para la ejecución.

Para crear y gestionar configuraciones, ha de utilizar el cuadro de diálogo Propiedades de configuración de ejecución. Para obtener más información sobre las configuraciones de ejecución, consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7. Para obtener más información sobre las opciones del depurador, consulte “[Configuración de las opciones de depuración](#)” en la página 8-76. (Las configuraciones para la ejecución múltiple es una función de JBuilder SE y Enterprise.)

## Compilación del proyecto con información simbólica de depuración

---

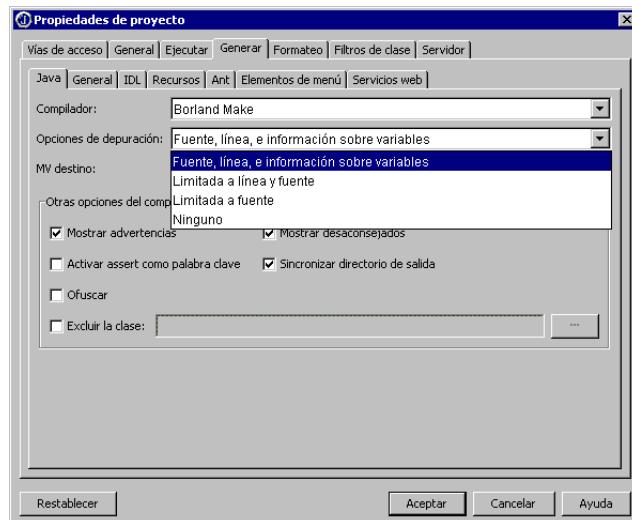
La eliminación de la información de depuración es una función de JBuilder SE y Enterprise.

Antes de comenzar una sesión de depuración, es necesario compilar el proyecto con información simbólica de depuración. La información simbólica de depuración, permite al depurador hacer conexiones entre el código fuente del programa y el bytecode de Java que genera el compilador.

Por defecto, JBuilder incluye información simbólica de depuración cuando se compila. Para comprobar que esta opción se encuentra activada en el proyecto actual:

- 1 Seleccione Proyecto | Propiedades de proyecto para abrir el cuadro de diálogo Propiedades de proyecto.

- 2** Seleccione la pestaña Generar y, a continuación, la pestaña Java. La ficha Generar | Java tiene la siguiente apariencia (se muestran las Opciones de depuración):



- 3** Seleccione una de las siguientes opciones en la lista desplegable Opciones de depuración:

- Fuente, línea, e información sobre variables: incluye información de depuración simbólica con el nombre del archivo fuente, número de línea e información de la variable local en el archivo .class cuando se compila, crea o reconstruye un nodo.
- Limitada a fuente y línea: incluye información de depuración simbólica sólo con el nombre del archivo fuente y el número de línea en el archivo .class cuando se compila, crea o reconstruye un nodo.
- Limitada a fuente: incluye información de depuración simbólica en el archivo .class cuando se compila, crea o reconstruye un nodo.
- Ninguno: no se incluye ninguna información de depuración. De todas formas, se puede depurar con esta opción, ya que el objeto `this` se encuentra disponible. Si se selecciona esta opción, se reduce el tamaño de la clase al mínimo posible.

**Sugerencia** Para que esta opción esté activada en los proyectos que se creen posteriormente, seleccione Proyecto | Propiedades por defecto para proyectos y seleccione la opción Incluir información de depuración en la ficha Generar. (El cambio de valores de propiedades por defecto no altera la configuración de proyectos creados anteriormente.)

**Nota** No podrá ver la información variable en las clases API de Java porque no se compilaron con información de depuración. No obstante, sí puede inspeccionar su código. Para obtener más información sobre cómo

inspeccionar estas clases, consulte “[Determinación de las clases que se han de inspeccionar](#)” en la página 8-42.

Cuando se genera información simbólica de depuración, el compilador almacena esta información en el archivo .class asociado. Esto puede agrandar considerablemente el archivo .class si se compila sin la información de depuración. Puede ser mejor desactivar esta opción antes de la distribución.

Para que la compilación se realice de forma automática antes de la depuración, asigne a Tipo de generación, en la parte inferior del cuadro de diálogo Propiedades de configuración de ejecución (cuando se asignan los valores de las opciones de depuración para la configuración de ejecución) el valor Make. Esta opción compila automáticamente el proyecto antes de ejecutarlo bajo el control del depurador. Si esta opción se configura como <Ninguno>, JBuilder no compila el programa antes de depurarlo, con lo que es posible que los archivos fuente y clase no estén sincronizados. Para obtener más información sobre los tipos de generación, consulte “[Tipos de generación](#)” en la página 7-11.

## Inicio del Depurador

Una vez haya creado una configuración para la ejecución y haya compilado el proyecto con información de depuración puede iniciar la depuración mediante una de las siguientes opciones de menú. Para obtener más información sobre las configuraciones de ejecución, consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7.

**Tabla 8.1** Comandos de menú para iniciar el depurador

Comando	Atajo de teclado	Descripción
Ejecutar   Depurar proyecto	Mayús + F9 o 	Inicia el programa en el depurador utilizando la configuración por defecto o la seleccionada. Se suspenderá la ejecución en un punto de interrupción o en la primera línea de código en que se requiera que el usuario introduzca algún dato que siempre tendrá prioridad.
Ejecutar   Omitir inspección	F8	Interrumpe la ejecución en la primera línea de código ejecutable.
Ejecutar   Inspeccionar	F7	Interrumpe la ejecución en la primera línea de código ejecutable.

Para depurar:

- Un único archivo de clases del proyecto y no el proyecto en su totalidad, seleccione el archivo fuente en el panel del proyecto y haga clic con el botón derecho. Seleccione el comando Depurar para obtener la configuración deseada.

- Una aplicación web, haga clic con el botón derecho del ratón sobre el servlet o el archivo JSP y seleccione Depurar web para obtener la configuración deseada. Si desea más información, consulte “Depuración web del servlet o de la JSP” en el capítulo “Las aplicaciones web” de la *Guía del desarrollador de aplicaciones Web*. (El desarrollo de aplicaciones web es una función de JBuilder Enterprise.)
- Un test, pulse con el botón derecho del ratón en ese test del panel del proyecto y, a continuación, seleccione Depurar test. Para obtener más información sobre los tests de módulos, consulte el [Capítulo 13, “Test de módulos”](#). (Los tests de módulos son una función de JBuilder Enterprise.)

Cada vez que se inicia el depurador comienza una sesión de depuración. Para obtener más información, consulte [“Sesiones de depuración” en la página 8-10](#).

**Nota**



Si desea seleccionar una configuración de depuración para una sesión, haga clic en la flecha de lista desplegable, que se encuentra a la derecha del botón Depurar proyecto de la barra de herramientas principal antes de empezar. Si no selecciona una configuración concreta, se utiliza la configuración por defecto definida en la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto. (Las configuraciones para la ejecución múltiple es una función de JBuilder SE y Enterprise.)

## Inicio del depurador con la opción -classic

En las versiones de la MV de Java anteriores a la 1.3.1, la opción `-classic` mejoró el rendimiento del depurador. Esta opción no se aplica a las MV más recientes; por ejemplo, JDK 1.4x y JDK 1.3.1 de Solaris no necesitan utilizar la opción `-classic`. En estas versiones, la MV utiliza la depuración “a toda velocidad”, que permite el funcionamiento mejorado.

Si está utilizando una Máquina virtual de Java inferior a la versión 1.3.1 para el proyecto, la opción Depurar siempre con `-Classic` del cuadro de diálogo Configurar JDK (Herramientas | Configurar JDK) proporciona un mejor rendimiento. JBuilder lo comprueba automáticamente para ver si esta opción aumenta el desarrollo, a continuación selecciona o elimina la selección de esta casilla de acuerdo con lo que proporcione mejores resultados. Esta función está disponible en todas las versiones de JBuilder.

Cuando se efectúa la evaluación, JBuilder lleva a cabo dos comprobaciones:

- 1 ¿Tiene instalada la MV Classic?
- 2 Si es así, ¿la versión de JVM es anterior a la 1.3.1?

Esta selección se modifica cuando se definen parámetros de MV como native, hotspot, green y server.

## Ejecución bajo el control del depurador

---

Cuando se ejecuta un programa bajo el control del depurador, su comportamiento es el habitual: crea ventanas, acepta información del usuario, calcula valores y presenta resultados. El depurador detiene el programa, de forma que se pueden utilizar las vistas del depurador para examinar su estado actual. Visualizando los valores de las variables, los métodos de la pila de llamadas y el resultado del programa, es posible cerciorarse de que la parte de código que se está examinando funciona según lo deseado.

Conforme se ejecuta el programa bajo el control del depurador, puede observar el comportamiento de la aplicación en las ventanas que se crean. Coloque las ventanas de manera que vea simultáneamente la ventana del depurador y la de la aplicación durante la depuración. Para impedir que las ventanas parpadeen cuando el foco pasa de las vistas del depurador a la de la aplicación y viceversa, ordene las ventanas de forma que no se solapen.

## Pausar la ejecución del programa

---

Cuando se está utilizando el depurador, el programa puede estar en uno de dos estados posibles: *en ejecución* o *en interrupción*.



- Cuando el programa está en modo de ejecución, el botón Pausa de la barra de herramientas del depurador está disponible.
- El programa se interrumpe cuando hace clic sobre el botón Pausa. Cuando se interrumpe el programa es posible analizar y modificar los valores de los datos. Entonces se habilitan los botones de inspección de la barra de herramientas del depurador. La ejecución también se interrumpe al alcanzar un punto de interrupción o cuando se realiza una inspección.



Para continuar ejecutando el programa, seleccione el botón Reanudar el programa en la barra de herramientas del depurador. Cuando la sesión del depurador finaliza, este botón se convierte en el botón Reiniciar el programa y reinicia la sesión.



Mientras el programa está interrumpido, puede modificar el código y continuar con la ejecución en todos los marcos activos. Para obtener más información, consulte “[Modificación del código durante la depuración](#)” en la página 8-72.

## Finalización de una sesión de depuración

---



Para finalizar la sesión de depuración actual y liberar la memoria, seleccione el ícono Reanudar el programa.

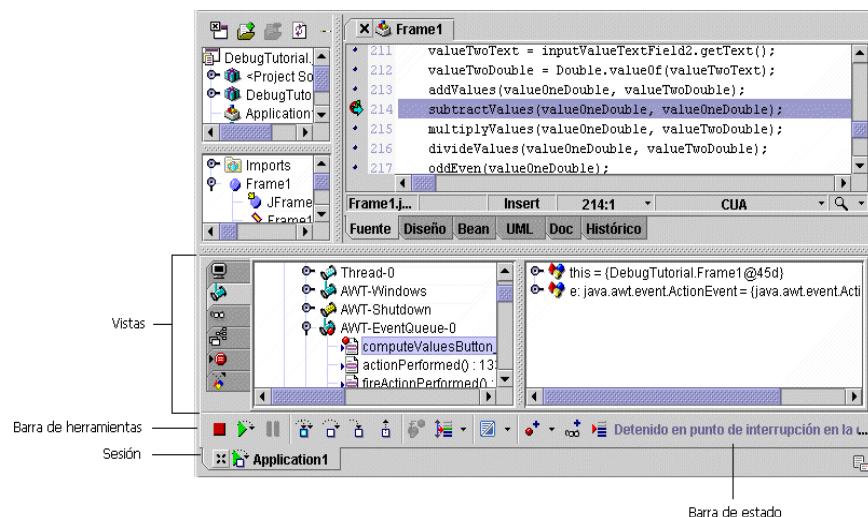
También es posible salir de la aplicación para cerrar la sesión de depuración. Para cerrar la pestaña de la sesión de depuración, haga clic con el botón derecho del ratón sobre la pestaña y seleccione Eliminar o pulse sobre la X de la pestaña.

## Interfaz del depurador

Si el proyecto o la clase se compila con éxito, se muestra el depurador en la parte inferior del Visualizador de aplicaciones.

- Las pestañas horizontales de la parte inferior del Visualizador de aplicaciones representan sesiones de depuración. Cada pestaña representa una sesión.
- Las pestañas verticales de la parte inferior izquierda del Visualizador de aplicaciones representan vistas de depuración. Las vistas corresponden a la sesión de depuración seleccionada actualmente. Las distintas vistas tienen iconos que indican el estado y el tipo del elemento seleccionado.
- La barra de herramientas del depurador corresponde a la sesión de depuración seleccionada actualmente.

**Figura 8.1** Interfaz del depurador



## Sesiones de depuración

---

El depurador permite depurar varios procesos en varias sesiones de depuración. Los procesos pueden encontrarse en el mismo proyecto de JBuilder o en diferentes. Es posible depurar simultáneamente un proceso servidor y otro cliente durante una sesión de JBuilder.

Los puntos de observación, de interrupción, y las clases que no tengan desactivada la inspección se almacenan por proyecto. Todos los puntos de observación e interrupción son aplicables a todos los procesos de un proyecto. Los puntos de interrupción pueden desactivarse de forma selectiva en cada configuración de ejecución.

Cuando se utilizan comandos del menú Ejecutar que no sean Ejecutar proyecto, Depurar proyecto ni Configuraciones, se continúa con la sesión de depuración seleccionada. Cuando se utilizan los botones de la barra de herramientas del depurador también se continúa con la sesión seleccionada.

Para finalizar la sesión de depuración actual y liberar la memoria,

■ seleccione el ícono Reanudar el programa. También puede finalizar la sesión haciendo clic en X en la pestaña de sesión de depuración o haciendo clic con el botón derecho del ratón en la pestaña y seleccionando Eliminar pestaña. A pesar de que el programa preguntará si desea detener el proceso antes de eliminar la pestaña, resulta conveniente utilizar primero Ejecutar | Terminar el programa.

## Vistas del depurador

---

La vista del depurador permite analizar el interior del programa. Mediante las vistas del depurador es posible examinar y modificar valores de datos, rastrear el programa hacia delante o hacia atrás, examinar los procesos internos de un método y de la llamada a dicho método, así como seguir un hilo dentro del programa.

Las vistas del depurador aparecen a lo largo de la parte izquierda del depurador de la interfaz del usuario. Para seleccionar una vista, elija la pestaña correspondiente en la parte izquierda del depurador. Las vistas (excepto la vista Salida, entrada y errores de la consola) también se pueden mostrar como ventanas flotantes. Las ventanas flotantes permiten ver varias vistas del depurador simultáneamente, en lugar que tener que alternar entre las diferentes vistas. (Las ventanas flotantes son una característica de JBuilder SE y Enterprise.)

- Para mostrar una vista como ventana flotante, haga clic con el botón derecho sobre una zona vacía de la vista y seleccione Ventana flotante.

- Para cerrar dicha ventana, haga clic sobre el botón Cerrar en la ventana flotante, o haga clic con el botón derecho sobre una zona vacía de la vista y desactive Ventana flotante.
- Si desea restaurar el orden de vista por defecto una vez haya cerrado una ventana flotante, haga clic con el botón derecho sobre una zona vacía de la vista y seleccione Restablecer orden predeterminado.

Las vistas del depurador también tienen menús contextuales. Los comandos de estos menús a menudo duplican aquellos que aparecen en los menús Ejecutar y Ver para controlar el depurador con más facilidad.

Además, cada depurador muestra una gama de iconos que indican el estado del elemento seleccionado. Por ejemplo, un punto de interrupción puede estar desactivado, verificado o sin verificar, o no ser válido; cada uno de estos estados se indica de forma visual por medio de un icono situado en el margen izquierdo de la vista.

A continuación se describen los iconos y vistas del depurador.

**Tabla 8.2** Vistas del depurador

Pestaña	Ver	Descripción
	Vista Consola	Muestra el resultado y los errores del programa. También permite introducir los datos necesarios. La imagen del ícono cambia si hay datos de resultado del programa y si se muestran mensajes de error.
	Vista Hilos, pilas de llamada y datos	Muestra los grupos de hilos del programa. Los grupos de hilos se expanden para mostrar sus hilos y contienen un seguimiento de marcos en la pila representando la secuencia de llamadas actual. Todos los marcos de pila se pueden ampliar y mostrar los elementos de datos disponibles en el ámbito. (Los datos estáticos no se muestran en esta vista, sino en la vista Datos estáticos y clases cargadas.)
	Vista Monitores de sincronización	Muestra los monitores de sincronización que utilizan los hilos y su estado actual, lo que resulta útil para detectar bloqueos y conflictos. Esta vista sólo se encuentra disponible si la MV actual la admite. La MV HotSpot no es compatible con esta función. La posibilidad de detectar hilos en conflicto o bloqueados es una función de JBuilder Enterprise.
	Vista Puntos de observación de datos	Muestra los valores actuales de los miembros de datos que está siguiendo.
	Vista Clases cargadas y datos estáticos	Muestra las clases cargadas actualmente por el programa. Si una clase tiene datos estáticos, se muestran cuando se amplía la clase. Si se muestra un paquete en el árbol, aparece el número de clases cargadas para ese paquete.

**Tabla 8.2** Vistas del depurador (continuación)

Pestaña	Ver	Descripción
	Vista Puntos de interrupción de datos y código	Muestra todos los puntos de interrupción del archivo y su estado actual. Esta vista también está disponible desde Ejecutar   Ver puntos de interrupción antes de que comience la sesión de depuración.
	Vista Clases con inspección desactivada	Muestra una lista alfabética de las clases y paquetes que no se deben inspeccionar. También se puede acceder a esta vista desde Ejecutar   Mostrar clases con inspección desactivada antes de que comience la sesión de depuración.

**Sugerencia** Es posible hacer flotantes todas las vistas del depurador, excepto la de la Consola. Para ello, haga clic con el botón derecho del ratón en el panel de mensajes y elija Ventana flotante. (Es una función de JBuilder SE y Enterprise.)

### Vista Salida, entrada y errores de consola



La vista de salida, entrada y errores de consola muestra las salidas de datos desde el programa y los errores del programa. También permite introducir datos requeridos por el programa. Cuando la pestaña Consola no se encuentra seleccionada, el ícono cambia si hay alguna salida del programa o si ha aparecido algún mensaje de error.

Las excepciones producidas durante la ejecución se muestran en esta vista. Para abrir el archivo en el que ha tenido lugar la excepción de ejecución y poder colocar el cursor en el número de línea de la excepción, pulse sobre el nombre del archivo que se encuentra subrayado. (Es una función de JBuilder SE y Enterprise.)

En esta vista, la salida de error aparece en color rojo. La salida estándar aparece en negro.

**Tabla 8.3** Iconos de la vista Consola

Icono	Descripción
	Se han escrito mensajes de salida en la vista.
	Se han escrito mensajes de error en la vista.
	No hay ninguna salida en la vista o la vista está en primer plano.

**Tabla 8.4** Menú contextual de la vista Consola

Comando	Descripción
Borrar todo	Borra todos los mensajes de la vista.
Copiar todo	Copia al portapapeles el contenido de la vista.

**Tabla 8.4** Menú contextual de la vista Consola (continuación)

Comando	Descripción
Copiar los seleccionados	Copia el contenido seleccionado al portapapeles.
Ajuste de palabras	Ajustas las líneas largas en la salida.

## Vista Clases con inspección desactivada



La Vista de clases con inspección desactivada muestra una lista ordenada alfabéticamente con las clases y paquetes que no se inspeccionan. Esta información está disponible antes de iniciar la depuración desde el comando Ejecutar | Mostrar clases con inspección desactivada.

Cuando se inicia una sesión de depuración, la inspección de todas las clases que aparecen en la vista está desactivada por defecto. Esto evita que el depurador inspeccione las bibliotecas proporcionadas con JBuilder así como las clases JDK, y que se concentre en el nuevo código en lugar de en el código que ya se ha depurado.

Para más información acerca del control de qué clases son inspeccionadas consulte “[Determinación de las clases que se han de inspeccionar](#)” en la [página 8-42](#).

### Nota

En JBuilder Personal, solamente se añaden a esta vista tres clases (`java.lang.Object`, `java.lang.String` y `java.lang.ClassLoader`). No se pueden añadir, modificar ni eliminar elementos de la lista; sin embargo, puede elegir entre inspeccionar o no esas clases. Consulte “[Paso inteligente](#)” en la [página 8-39](#) para obtener más información.

**Tabla 8.5** Iconos en la vista Clases con inspección desactivada

Icono	Descripción
(Gris)	La inspección está desactivada para la clase o el paquete seleccionado.
(De colores)	La inspección está activada para la clase o el paquete seleccionado.

**Tabla 8.6** Menú contextual con clase o paquete seleccionado en la vista Clases con inspección desactivada

Comando	Descripción
Modificar clase/ paquete	Muestra el cuadro de diálogo Cambiar clase/paquete con inspección desactivada, en el que puede utilizar comodines para modificar la clase o el paquete y para abrir el cuadro de diálogo Seleccionar clase o paquete. Si se selecciona un paquete, no se inspeccionan todas las clases del paquete. (JBuilder SE y Enterprise).

**Tabla 8.6** Menú contextual con clase o paquete seleccionado en la vista Clases con inspección desactivada (continuación)

Comando	Descripción
Inspeccionar clase/ paquete	Permite inspeccionar la clase o paquete. Si se selecciona un paquete, se inspeccionan las clases de ese paquete.
Eliminar clase/ paquete	Elimina la clase o paquete seleccionado de la vista. Esto permite inspeccionar la clase o paquete seleccionados. (JBuilder SE y Enterprise.)

**Tabla 8.7** Menú contextual sin selección en la vista Clases con inspección desactivada

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)
Añadir clase o paquete	Muestra el cuadro de diálogo Seleccionar clase o paquete, donde se elige la clase o paquete que se desea añadir a la vista. Esto evita que el depurador inspeccione esa clase o paquete. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)
Eliminar todos	Elimina todas las clases y paquetes de la vista. Se inspeccionarán todos los paquetes y clases, incluidos los de JBuilder y las bibliotecas del JDK. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)

## Vista Puntos de interrupción de datos y código



La Vista de puntos de interrupción en datos y códigos muestra todos los puntos de interrupción establecidos en el archivo y su estado actual. Esta información también está disponible antes de dar comienzo a la depuración por medio del comando Ver | Puntos de interrupción.

Para obtener más información sobre los puntos de interrupción, consulte “[Puntos de interrupción](#)” en la página 8-46.

**Tabla 8.8** Iconos de la vista Puntos de interrupción por datos y código.

Icono	Descripción
(rojo)	Un punto de interrupción sin verificar.
(verde)	Un punto de interrupción verificado.
(rojo)	Un punto de interrupción no válido.
(gris)	Un punto de interrupción desactivado para una configuración. (JBuilder SE y Enterprise.)
(campo)	Un punto de interrupción de campo. Un campo es una variable Java que está definida en un objeto Java. (JBuilder SE y Enterprise.)

**Tabla 8.9** Menú contextual con punto de interrupción seleccionado en la vista Puntos de interrupción de datos y código

Comando	Descripción
Ir al punto de interrupción	Va al punto de interrupción seleccionado del código fuente. Esto resulta útil si hay varios archivos abiertos en el editor y se desea encontrar la línea de código interrumpida con rapidez. Este comando está disponible cuando hay un punto de interrupción seleccionado.
Activar punto de interrupción	Activa el punto de interrupción seleccionado.
Desactivar en la configuración	Desactiva el punto de interrupción seleccionado en la configuración seleccionada. Este comando sólo está disponible si se cuenta con más de una configuración de ejecución. Por defecto, todos los puntos de interrupción se aplican a todas las configuraciones definidas. Este comando está disponible cuando hay un punto de interrupción seleccionado. (JBuilder SE y Enterprise.)
Eliminar punto de interrupción	Elimina el punto de interrupción seleccionado.
Propiedades de punto de interrupción	Muestra el cuadro de diálogo Propiedades de punto de interrupción, donde se definen las propiedades del punto de interrupción seleccionado.
Interrumpir al leer	Obliga al depurador a detenerse cuando está a punto de leer el punto de interrupción del campo seleccionado. Un campo es una variable Java que está definida en un objeto Java. Este comando está disponible cuando hay un punto de interrupción de campo seleccionado. (JBuilder SE y Enterprise.)
Interrumpir al escribir	Obliga al depurador a detenerse cuando está a punto de escribir en el punto de interrupción del campo seleccionado. Un campo es una variable Java que está definida en un objeto Java. Este comando está disponible cuando hay un punto de interrupción de campo seleccionado. (JBuilder SE y Enterprise.)

**Tabla 8.10** Menú contextual sin selección en la vista Puntos de interrupción de datos y código

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)
Añadir punto de interrupción de línea	Muestra el cuadro de diálogo Añadir punto de interrupción de línea, lo que permite añadir un punto de interrupción de línea.

**Tabla 8.10** Menú contextual sin selección en la vista Puntos de interrupción de datos y código (continuación)

Comando	Descripción
Añadir punto de interrupción por excepción	Muestra el cuadro de diálogo Añadir punto de interrupción por excepción, lo que permite añadir un punto de interrupción por excepción. (JBuilder SE y Enterprise.)
Añadir punto de interrupción de clase	Muestra el cuadro de diálogo Añadir punto de interrupción de clase, dónde añade un punto de interrupción de clase. (JBuilder SE y Enterprise.)
Añadir punto de interrupción de método	Muestra el cuadro de diálogo Añadir punto de interrupción de clase, dónde añade un punto de interrupción de método. (JBuilder SE y Enterprise.)
Añadir punto de interrupción interproceso	Muestra el cuadro de diálogo Añadir punto de interrupción interproceso, lo que permite añadir un punto de interrupción de interproceso. (JBuilder Enterprise.)
Desactivar todos	Desactiva todos los puntos de interrupción.
Activar todos	Activa todos los puntos de interrupción.
Eliminar todos	Elimina todos los puntos de interrupción.

## Vista Hilos, pilas de llamada y datos



La Vista hilos, pilas de llamada y datos muestra el estado actual de los grupos de hilos del programa. Los grupos de hilos se expanden para mostrar sus hilos y contienen un seguimiento de marcos de pila representando la secuencia actual de llamadas a métodos. Cada marco de pila puede ampliarse para mostrar los elementos de datos disponibles en ámbito. Los iconos identifican el tipo de elemento de datos. (Los datos estáticos no se muestran en esta vista, sino en la vista Datos estáticos y clases cargadas.) Los elementos oscurecidos son heredados.

Por defecto, esta vista se divide en dos paneles. El panel izquierdo se puede ampliar para que muestre los marcos de pila. El panel derecho muestra el contenido del elemento seleccionado a la izquierda, y muestra todo, desde un grupo de hilos hasta una variable. Por ejemplo, si se selecciona un hilo en el panel de la izquierda, el panel de la derecha mostrará los marcos de pilas de ese hilo. Por otra parte, si se selecciona un marco de pila en el panel de la izquierda, el panel de la derecha mostrará las variables disponibles en esa vista. (El panel dividido es una característica de JBuilder SE y Enterprise.)

Para obtener más información sobre los hilos, consulte “[Gestión de hilos](#)” en la página 8-34.

**Tabla 8.11** Iconos en la vista Hilos, pilas de llamadas y datos

Icono	Descripción
	Un hilo de inspección actual.
	Un grupo de hilos.
	(Amarillo) Un hilo bloqueado.
	Un hilo suspendido.
	(Gris) Un hilo muerto.
	Una clase.
	Una interfaz.
	(De colores) Un objeto.
	(Sombreado) Un objeto null.
	Seguimiento de marcos de pilas.
	Los marcos de pilas seleccionados.
	Una matriz.
	Una primitiva.
	(Rojo) Un error.
	(Gris) Un mensaje de información.

**Tabla 8.12** Menú contextual con selección en la vista Hilos, pilas de llamadas y datos

Comando	Descripción
Mantener hilo suspendido	La ejecución del hilo seleccionado no se reanuda cuando se selecciona Ejecutar   Reanudar el programa. Esto le permite observar el comportamiento de otros hilos. (JBuilder Enterprise)
Establecer hilo inspeccionado	El hilo seleccionado se inspecciona al seleccionar Ejecutar   Reanudar el programa. Permite observar el funcionamiento de este hilo. (JBuilder Enterprise)
Establecer punto de ejecución	Establece el marco de pila en el que se llevan a cabo las operaciones de reanudación. (JBuilder Enterprise)
Cortar	Elimina el valor de la variable y la coloca en el portapapeles. Este comando está disponible cuando hay una variable seleccionada. (JBuilder SE y Enterprise.)
Copiar	Copia al portapapeles el valor de la variable. Este comando está disponible cuando hay una variable seleccionada. (JBuilder SE y Enterprise.)

**Tabla 8.12** Menú contextual con selección en la vista Hilos, pilas de llamadas y datos

Comando	Descripción
Pegar	Pega el contenido del portapapeles en otra variable. Cuando se utiliza el comando Pegar, tanto la variable cortada o copiada del objeto como la variable pegada apuntan al mismo objeto. Este comando está disponible cuando hay una variable seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en una variable local	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la variable local seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una variable o una matriz de variables seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la matriz seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una matriz seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en un componente de una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el componente de matriz seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un componente seleccionado en una matriz. (JBuilder SE y Enterprise.)
Ajustar rango de visualización	Muestra el cuadro de diálogo Ajustar rango, donde se puede reducir el número de elementos de matriz que se muestran en la vista. Este comando está disponible cuando hay una matriz seleccionada. (JBuilder SE y Enterprise.)
Crear punto de observación 'this'	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto <i>this</i> seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto <i>this</i> seleccionado. (JBuilder SE y Enterprise.)
Crear punto de observación de clases	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la clase seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una clase seleccionada. (JBuilder SE y Enterprise.)

**Tabla 8.12** Menú contextual con selección en la vista Hilos, pilas de llamadas y datos

Comando	Descripción
Crear punto de observación de objetos	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto seleccionado. Un punto de observación de objetos vigila el objeto Java seleccionado. Se amplía para mostrar los miembros de datos de la instancia actual. (JBuilder SE y Enterprise.)
Crear punto de observación de cadenas	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la cadena seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una cadena seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en un campo estático	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el campo estático seleccionado. El punto de observación se añade a la vista de Observación de datos. Un campo estático es una variable Java definida como estática (una variable de clase). Este comando está disponible cuando hay un campo estático seleccionado. (JBuilder Enterprise.)
Crear un punto de observación en un campo	Crea un punto de observación en el campo seleccionado y añade automáticamente el punto de observación a la vista Puntos de observación de datos. Un campo es una variable Java que está definida en un objeto Java. Este comando está disponible cuando hay un campo seleccionado. (JBuilder SE y Enterprise.)
Crear un punto de interrupción en un campo	Crea un punto de interrupción en el campo seleccionado y añade automáticamente el punto de interrupción a la vista Puntos de interrupción de datos y código. Un campo es una variable Java que está definida en un objeto Java. Para activar el punto de interrupción, vaya a la vista de Puntos de interrupción de datos y código y haga clic con el botón derecho sobre el punto de interrupción. Seleccione Interrumpir al leer para que el depurador detenga la ejecución del programa justo antes de leer el campo, o Interrumpir al escribir para que lo haga justo antes de escribir en el campo. Este comando está disponible cuando hay un campo seleccionado. (JBuilder SE y Enterprise.)
Mostrar valor decimal/hexadecimal	Modifica la base numérica en la que se realiza la visualización de una variable numérica o de carácter. Este comando está disponible cuando hay una variable numérica o de carácter seleccionada. En el caso de las matrices, la selección de este comando modificará la base de sus elementos.

**Tabla 8.12** Menú contextual con selección en la vista Hilos, pilas de llamadas y datos

Comando	Descripción
Mostrar/Ocultar valores nulos	Conmuta la visualización de los valores nulos en una matriz. Este comando resulta útil cuando se depura un objeto tabla de colisiones. Está disponible cuando se selecciona una matriz de tipo Object. (JBuilder SE y Enterprise).
Cambiar valor	Muestra el cuadro de diálogo Modificar valor, donde puede modificar directamente el valor de una variable. Este comando está disponible cuando hay una variable seleccionada. (JBuilder SE y Enterprise).

**Tabla 8.13** Menú contextual sin selección en la vista Hilos, pilas de llamadas y datos

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise).
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise).
Mostrar sólo el hilo actual	Muestra únicamente las pilas de llamadas y los datos del hilo actual. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise).
Dividir las vistas de hilos y datos	Divide la vista en dos paneles. La parte izquierda se amplía y muestra marcos de pilas; la derecha muestra el contenido del elemento seleccionado en la parte izquierda. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise).

## Vista Puntos de observación de datos



La vista Puntos de observación de datos muestra los valores actuales de los componentes de los datos que se desea controlar. Algunos tipos de expresiones de punto de observación pueden ampliarse, de manera que muestren los elementos de datos en ámbito. Si los elementos no están en el ámbito, aparece en la vista el mensaje <fuera del ámbito>. Los elementos que aparecen en gris son heredados.

Para obtener más información sobre puntos de observación de datos, consulte “[Observación de expresiones](#)” en la página 8-66.

**Tabla 8.14** Iconos en la vista Puntos de observación de datos

Icono	Descripción
	Un objeto
	Una matriz

**Tabla 8.14** Iconos en la vista Puntos de observación de datos (continuación)

Icono	Descripción
	Un primitivo
	(Rojo) Un error
	(Gris) Un mensaje de información

**Tabla 8.15** Menú contextual con punto de observación seleccionado en la vista Puntos de observación de datos

Comando	Descripción
Eliminar punto de interrupción	Elimina el punto de observación seleccionado.
Crear un punto de observación en una variable local	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la variable local seleccionada. Este comando está disponible cuando hay una variable o una matriz de variables seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la matriz seleccionada. Este comando está disponible cuando hay una matriz seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en un componente de una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el componente de matriz seleccionado. Este comando está disponible cuando hay un componente seleccionado de una matriz. (JBuilder SE y Enterprise.)
Ajustar rango de visualización	Muestra el cuadro de diálogo Ajustar rango, donde se puede reducir el número de elementos de matriz que se muestran en la vista. Este comando está disponible cuando hay una matriz seleccionada. (JBuilder SE y Enterprise.)
Crear punto de observación 'this'	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto <i>this</i> seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto <i>this</i> seleccionado. (JBuilder SE y Enterprise.)
Crear punto de observación de clases	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la clase seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una clase seleccionada. (JBuilder SE y Enterprise.)

**Tabla 8.15** Menú contextual con punto de observación seleccionado en la vista Puntos de observación de datos (continuación)

Comando	Descripción
Crear punto de observación de objetos	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto seleccionado. Un punto de observación de objetos vigila el objeto Java seleccionado. Se amplía para mostrar los miembros de datos de la instancia actual. (JBuilder SE y Enterprise.)
Crear punto de observación de cadenas	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la cadena seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una cadena seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en un campo estático	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el campo estático seleccionado. El punto de observación se añade a la vista de Observación de datos. Un campo estático es una variable Java definida como estática (una variable de clase). Este comando está disponible cuando hay un campo estático seleccionado. (JBuilder Enterprise.)
Crear un punto de observación en un campo	Crea un punto de observación en el campo seleccionado y añade automáticamente el punto de observación a la vista Puntos de observación de datos. Un campo es una variable Java que está definida en un objeto Java. Este comando está disponible cuando hay un campo seleccionado. (JBuilder SE y Enterprise.)
Crear un punto de interrupción en un campo	Crea un punto de interrupción en el campo seleccionado y añade automáticamente el punto de interrupción a la vista Puntos de interrupción de datos y código. Un campo es una variable Java que está definida en un objeto Java. Para activar el punto de interrupción, vaya a la vista de Puntos de interrupción de datos y código y haga clic con el botón derecho sobre el punto de interrupción. Seleccione Interrumpir al leer para que el depurador detenga la ejecución del programa justo antes de leer el campo, o Interrumpir al escribir para que lo haga justo antes de escribir en el campo. Este comando está disponible cuando hay un campo seleccionado. (JBuilder SE y Enterprise.)
Mostrar/Ocultar valores nulos	Conmuta la visualización de los valores nulos en una matriz. Este comando resulta útil cuando se depura un objeto tabla de colisiones. Está disponible cuando se selecciona una matriz de tipo Object. (JBuilder SE y Enterprise.)

**Tabla 8.15** Menú contextual con punto de observación seleccionado en la vista Puntos de observación de datos (continuación)

Comando	Descripción
Cambiar valor	Muestra el cuadro de diálogo Modificar valor, donde puede modificar directamente el valor de una variable. Este comando está disponible cuando hay un tipo de datos primitivos seleccionado. (JBuilder SE y Enterprise.)
Cambiar punto de observación	Muestra el cuadro de diálogo Cambiar punto de observación, con el que se puede cambiar la descripción y la expresión de observación.

**Tabla 8.16** Menú contextual sin selección en la vista Puntos de observación de datos

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)
Añadir punto de observación	Muestra el cuadro de diálogo Añadir punto de observación, en el que se pueden añadir puntos de observación. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista.
Eliminar todos	Elimina todos los puntos de observación. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista.

## Vista Clases cargadas y datos estáticos



La vista Clases cargadas y datos estáticos muestra las clases cargadas en ese momento por el programa. Si una clase tiene datos estáticos, se muestran cuando se amplía la clase. Si se muestra un paquete en el árbol, aparece el número de clases cargadas para ese paquete.

Las clases de esta vista que contengan \$ seguido de un número representan clases internas. El compilador crea clases internas para los manejadores de sucesos definidos como Adaptadores anónimos en la ficha Generados del cuadro de diálogo Propiedades de proyecto.

Para obtener más información, consulte “[Presentación de las variables en el depurador](#)” en la página 8-63.

**Tabla 8.17** Iconos en la vista Clases cargadas y datos estáticos

Icono	Descripción
	Un paquete.
	Una clase.
	Una interfaz.
	Una clase bloqueada.
	(De colores) Un objeto.
	(Sombreado) Un objeto null.
	Una matriz.
	Una primitiva.

**Tabla 8.18** Menú contextual con selección de la vista Clases cargadas y datos estáticos

Comando	Descripción
Cortar	Elimina el valor de la variable y la coloca en el portapapeles. Este comando está disponible cuando hay una variable seleccionada. (JBuilder SE y Enterprise.)
Copiar	Copia al portapapeles el valor de la variable. Este comando está disponible cuando hay una variable seleccionada. (JBuilder SE y Enterprise.)
Pegar	Pega el contenido del portapapeles en otra variable. Cuando se utiliza el comando Pegar, tanto la variable cortada o copiada del objeto como la variable pegada apuntan al mismo objeto. Este comando está disponible cuando hay una variable seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en una variable local	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la variable local seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una variable seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la matriz seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una matriz seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en un componente de una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el componente de matriz seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un componente seleccionado en una matriz. (JBuilder SE y Enterprise.)

**Tabla 8.18** Menú contextual con selección de la vista Clases cargadas y datos estáticos

Comando	Descripción
Ajustar rango de visualización	Muestra el cuadro de diálogo Ajustar rango, donde se puede reducir el número de elementos de matriz que se muestran en la vista. Este comando está disponible cuando hay una matriz seleccionada. (JBuilder SE y Enterprise.)
Crear punto de observación 'this'	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto <code>this</code> seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto <code>this</code> seleccionado. (JBuilder SE y Enterprise.)
Crear punto de observación de clases	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la clase seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una clase seleccionada. (JBuilder SE y Enterprise.)
Crear punto de observación de objetos	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto seleccionado. Un punto de observación de objetos vigila el objeto Java seleccionado. Se amplía para mostrar los miembros de datos de la instancia actual. (JBuilder SE y Enterprise.)
Crear punto de observación de cadenas	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la cadena seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una cadena seleccionada. (JBuilder SE y Enterprise.)
Crear un punto de observación en un campo estático	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el campo estático seleccionado. El punto de observación se añade a la vista de Observación de datos. Un campo estático es una variable Java que está definida como estática en un objeto Java. Este comando está disponible cuando hay un campo estático seleccionado. (JBuilder Enterprise.)
Crear un punto de observación en un campo	Crea un punto de observación en el campo seleccionado y añade automáticamente el punto de observación a la vista Puntos de observación de datos. Un campo es una variable Java que está definida en un objeto Java. Este comando está disponible cuando hay un campo seleccionado. (JBuilder SE y Enterprise.)
Cambiar valor	Muestra el cuadro de diálogo Modificar valor, donde puede modificar directamente el valor de una variable. Este comando está disponible cuando hay una variable seleccionada. (JBuilder SE y Enterprise.)

**Tabla 8.19** Menú contextual sin selección de la vista Clases cargadas y datos estáticos

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)

## Vista Monitores de sincronización

Es una función de JBuilder SE y Enterprise.



### Nota

La vista Monitores de sincronización muestra los monitores de sincronización utilizados por los hilos y su estado actual, lo cual resulta útil para detectar situaciones de bloqueo. En JBuilder Personal, aparece la pestaña, pero no se muestra el estado de bloqueo.

Algunas MV, como HotSpot, no proporcionan esta información. Si la vista de los monitores de sincronización no se encuentra disponible y su MV admite classic, es necesario añadir -classic como un parámetro MV de la siguiente manera:

- 1 Abra el cuadro de diálogo Proyecto | Propiedades de proyecto del proyecto que se está depurando.
- 2 Abra la ficha Ejecutar, seleccione la configuración para la ejecución y pulse sobre Editar para editarlo.
- 3 En la pestaña Ejecutar, escriba -classic en el campo Parámetros de la MV.
- 4 Pulse dos veces sobre Aceptar para cerrar los cuadros de diálogo.

Para obtener más información sobre los hilos, consulte “[Gestión de hilos](#)” en la página 8-34.

**Tabla 8.20** Iconos en la vista Monitores de sincronización

Icono	Descripción
(Amarillo)	El monitor de sincronización utilizado por el hilo especificado no está bloqueado.
(Rojo)	El monitor de sincronización utilizado por el hilo especificado está bloqueado.

**Tabla 8.21** Menú contextual de la vista Monitores de sincronización

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (JBuilder SE y Enterprise.)

## Barra de herramientas del depurador

La barra de herramientas de la parte inferior del depurador proporciona un acceso rápido a las acciones más utilizadas por el depurador. La parte derecha de la barra de herramientas, que corresponde con la barra de estado del depurador, muestra mensajes de estado.

**Figura 8.2** Barra de herramientas del depurador

La tabla que aparece a continuación explica detalladamente los iconos de la barra de herramientas.

**Tabla 8.22** Botones de la barra de herramientas

Botón	Acción	Descripción
[ ]	Terminar el programa	Pone fin a la ejecución actual de la aplicación y libera la memoria. Su efecto es el mismo que el de Ejecutar   Terminar programa.
[ ] / [ ]	Reanudar el programa	Reinicia el depurador que se ha apagado o reiniciado o continua con el actual. Su efecto es el mismo que el de Ejecutar   Reanudar programa.
[ ]	Pausar el programa	Detiene provisionalmente la sesión de depuración actual. Su efecto es el mismo que el de Ejecutar   Pausar el programa.
[ ]	Paso inteligente	Controla el uso de los parámetros de Paso inteligente de la vista Clases con inspección desactivada y las opciones de Paso inteligente de la ficha Depurar del cuadro de diálogo Propiedades de ejecución. (JBuilder SE y Enterprise.)
[ ]	Omitir inspección	Omite la inspección de la línea de código actual. Su efecto es el mismo que el de Ejecutar   Omitir inspección.
[ ]	Inspeccionar	Inspecciona la línea de código actual. Su efecto es el mismo que el de Ejecutar   Inspeccionar.

**Tabla 8.22** Botones de la barra de herramientas (continuación)

Botón	Acción	Descripción
	Abandonar inspección	Sale del método actual y vuelve al elemento que ha efectuado la llamada. Su efecto es el mismo que el de Ejecutar   Abandonar inspección.
	Intercambio inteligente	Compila los archivos modificados y actualiza las clases compiladas. Su efecto es el mismo que el de Ejecutar   Intercambio inteligente. (JBuilder Enterprise)
	Establecer punto de ejecución	Define el lugar en el que el programa está para iniciarse. El punto de ejecución se traslada a la nueva ubicación. Su efecto es el mismo que el de Ejecutar   Establecer punto de ejecución. (JBuilder Enterprise)
	Fuente inteligente	Establece el tipo de archivo fuente, basado en el lenguaje fuente original distinto de Java. Coloca el cursor en la vista del nuevo archivo del marco de pila actual. Su efecto es el mismo que el de Ejecutar   Fuente inteligente. (JBuilder Enterprise)
	Añadir punto de interrupción	Añade un punto de interrupción a la sesión de depuración actual. Haga clic en la flecha de lista desplegable que aparece a la derecha del botón para elegir el tipo de punto de interrupción. Su efecto es el mismo que el de Ejecutar   Añadir punto de interrupción.
	Añadir punto de observación	Añade un punto de observación a la sesión de depuración actual. Su efecto es el mismo que el de Ejecutar   Añadir punto de observación.
	Mostrar marco actual	Muestra la pila de llamadas del hilo actual y resalta en el código fuente el punto de ejecución.

## Métodos abreviados para el depurador

Las siguientes teclas de método abreviado permiten acceder fácilmente a las funciones de depuración.

**Tabla 8.23** Métodos abreviados para el depurador

Contraseña	Acción
<b>Mayús+F9</b>	Depurar proyecto
<b>Ctrl+F2</b>	Terminar el programa
<b>F4</b>	Ejecutar hasta el cursor
<b>F5</b>	Conmutar punto de interrupción en el editor
<b>F7</b>	Inspeccionar
<b>F8</b>	Omitir inspección
<b>F9</b>	Reanudar el programa (reanuda la sesión de depuración actual)

**Tabla 8.23** Métodos abreviados para el depurador (continuación)

Contraseña	Acción
<i>Ctrl+clic con el botón izquierdo del ratón en el margen sobre el punto de interrupción</i>	Mostrar el cuadro de diálogo Propiedades del punto de interrupción
<i>Ctrl+botón derecho en el editor y sobre la expresión</i>	Mostrar la ventana ExpressionInsight de esa expresión

## ExpressionInsight

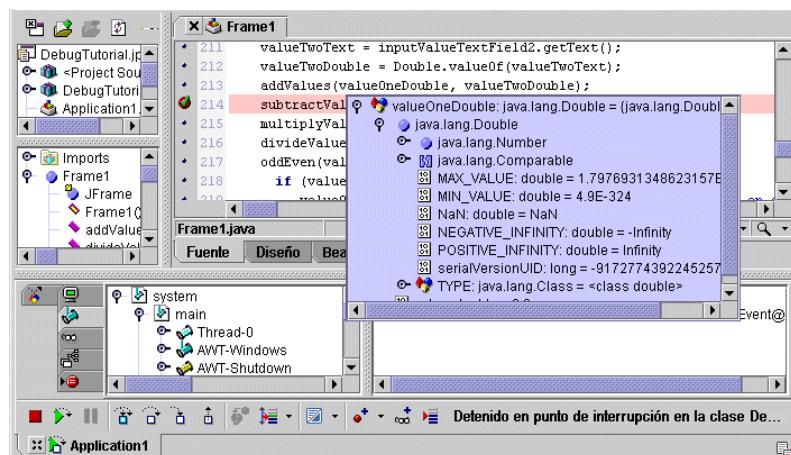
Es una función de JBuilder SE y Enterprise.

Cuando se interrumpe el depurador es posible acceder a ExpressionInsight - una pequeña ventana emergente que muestra, mediante una representación en árbol, el contenido de la expresión seleccionada. Para mostrar la ventana ExpressionInsight,

- Mantenga pulsado el botón *Ctrl* (botón *Comando* en Macintosh) y lleve el ratón sobre el código en el editor. La ventana ExpressionInsight se muestra cuando el ratón pasa sobre una expresión con sentido.
- Mueva el ratón a la expresión que se desea analizar en más detalle y pulse *Ctrl* junto con el botón derecho del ratón.

La ventana ExpressionInsight se abre hasta que pulsa una tecla para cerrar.

La ventana ExpressionInsight permite descender por los miembros de la expresión. Si la expresión es un objeto, el menú contextual muestra los mismos comandos de menú que los que están disponibles en la vista Hilos, pilas de llamadas y datos cuando se selecciona un objeto. También puede pulsar con el botón derecho un descendiente de la ventana para que aparezca un menú contextual.

**Figura 8.3** Ventana ExpressionInsight

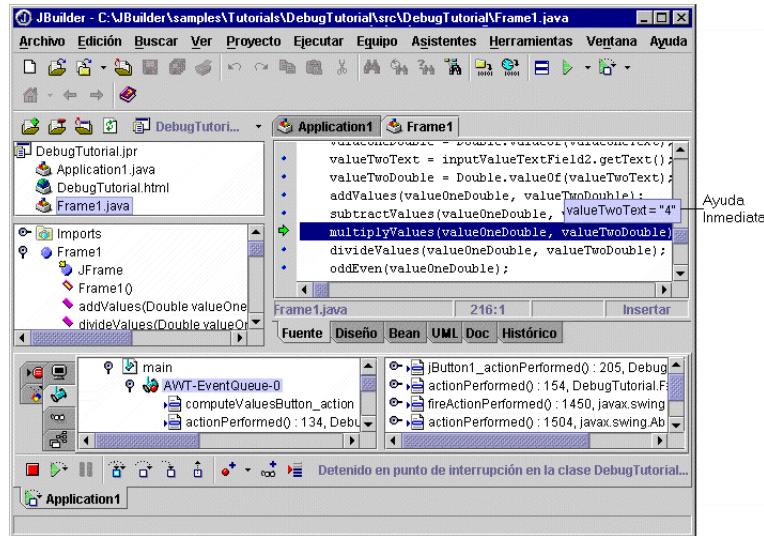
La ventana ExpressionInsight está desactivada cuando no ha finalizado o no se ha interrumpido la sesión de depuración.

## Ayuda inmediata

**Es una función de JBuilder SE y Enterprise.**

Cuando se detenga la depuración, puede colocar, en el editor, el cursor del ratón sobre una variable, para ver su valor. El valor aparece en una pequeña ventana emergente llamada ventana de ayuda inmediata. Si se selecciona texto, se ve su valor.

**Figura 8.4** Ventana Ayuda inmediata



La ayuda inmediata se desactiva cuando la sesión de depuración finaliza o no se ha suspendido.

## Depuración de fuente distinta de Java

**Es una función de JBuilder Enterprise.**

JBuilder se puede utilizar para depurar código fuente distinto de Java, incluidos JSP, SQLJ y código LegacyJ. Se puede depurar de forma local y de forma remota. Para conseguir esto, JBuilder utiliza la información de mapeo que se guarda en el archivo de clase (consulte JSR-45). Esto le permite depurar del modo en que lo haga normalmente. Puede ejecutar e interrumpir el programa, establecer y ejecutar puntos de interrupción, inspeccionar el código y examinar y cambiar los valores de los datos.

Al interrumpir el programa, puede cambiar la vista de su código, lo que permite ver el código fuente Java o el código distinto de Java. Por ejemplo, si está depurando una JSP y se ha detenido en un punto de interrupción, puede ver el código fuente Java de esa JSP o la propia JSP.

-  Para intercambiar las vistas, utilice el botón Fuente inteligente de la barra de herramientas del depurador. Una ventana emergente muestra la vista fuente seleccionada actualmente y las vistas fuente disponibles. Si selecciona una vista fuente diferente de la actual, el editor vuelve a dibujar el archivo asociado con la nueva vista fuente. El estado de la vista fuente se mantiene en cada sesión de depuración, con lo que si se cambia la fuente se cambia el archivo que aparece cuando se suspende la MV.
- Nota** La vista fuente por defecto es la que JBuilder determina como la mejor para el código actual.  
Si cambia las vistas fuente, también se cambian el marco de la pila actual y la ubicación del cursor. Por ejemplo, si está depurando una JSP y ve el código de la JSP, debe haber establecido el punto de interrupción en la línea 25. Sin embargo, si cambia al código fuente Java, el cursor cambia a la línea 75. Esto se produce porque los marcos de pila análogos no se encuentran en las mismas líneas de código en los dos archivos.

## Control de la ejecución del programa

---

La característica más importante de un depurador es que permite controlar la ejecución del programa. Es posible controlar si el programa ejecuta una sola línea de código, un método completo o un bloque de programa completo. Controlando manualmente cuándo debe ejecutarse y cuándo debe detenerse el programa, puede pasar rápidamente por las secciones que sabe que funcionan correctamente y concentrarse en las secciones que causan los problemas.

### Ejecución e interrupción del programa

---

Cuando el programa se ejecuta en el depurador es necesario detenerlo momentáneamente para analizar los valores de los datos. La detención temporal hace que el depurador suspenda la ejecución del programa. Desde aquí puede utilizar el depurador para examinar el estado del programa respecto a la ubicación del programa.

Cuando se está utilizando el depurador, el programa puede estar en uno de dos estados posibles: *en ejecución* o *en interrupción*.

- 
- Cuando el programa está en modo de ejecución, el botón Pausa de la barra de herramientas del depurador está disponible.
  - El programa se interrumpe cuando hace clic sobre el icono Pausa. Cuando se interrumpe el programa es posible analizar los valores de los datos. Entonces se habilitan los botones de inspección de la barra de herramientas del depurador.

- ▶ Para continuar ejecutando el programa, seleccione el botón Reanudar el programa en la barra de herramientas del depurador. Cuando la sesión del depurador finaliza, este botón se convierte en el botón Reiniciar el programa y reinicia la sesión.
- ▶ Mientras el programa está interrumpido, puede modificar el código y continuar con la ejecución en todos los marcos de pilas activos. Para obtener más información, consulte "["Modificación del código durante la depuración"](#) en la página 8-72.

## Reinicio del programa

---

Durante la depuración, puede resultar necesario reinicializar el programa, por ejemplo, si se sobrepasa la posición de un error o si las variables o estructuras de los datos se han dañado y contienen valores no deseados.

Para detener la ejecución actual del programa, efectúe una de las siguientes acciones:

- Seleccione Ejecutar | Terminar el programa.
- Haga clic en el botón Reiniciar el programa de la barra de herramientas del depurador.

El reinicio de un programa libera recursos y borra la configuración de todas las variables. Sin embargo, reinicializar un programa no borra los puntos de interrupción ni los puntos de observación que haya definido, por lo que resulta sencillo reanudar la sesión de depuración.

- ▶ Para reiniciar el programa, pulse el botón Reiniciar el programa de la barra de herramientas del depurador.

## El punto de ejecución

---

Durante una sesión de depuración interrumpida, la línea de código que se esté ejecutando en ese momento en un hilo de proceso aparece resaltada en el editor mediante una flecha verde en el margen izquierdo.

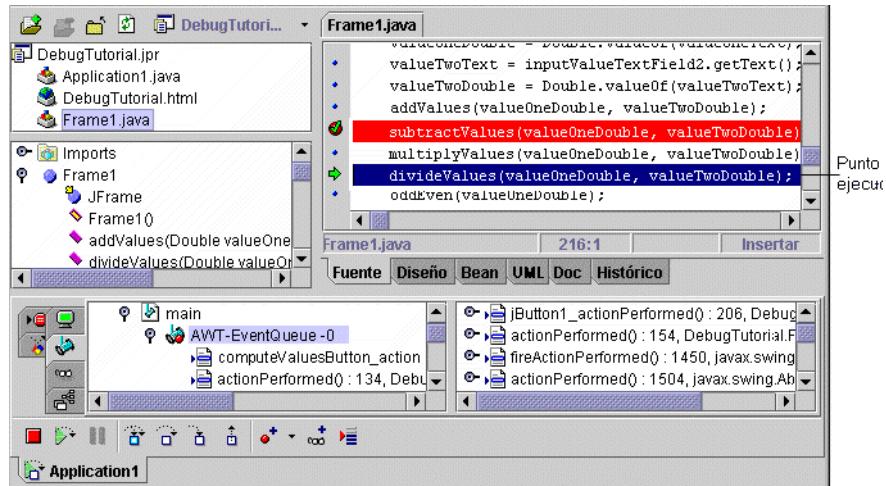
El punto de ejecución marca la actual línea del código fuente que va a ejecutar el depurador. Cuando detiene la ejecución del programa dentro del depurador, se resalta el punto de ejecución actual en el hilo de proceso seleccionado. El punto de ejecución siempre muestra la línea actual de código que va a ejecutarse, tanto si desea omitir la inspección, inspeccionar código o ejecutar el programa sin detenerse.

Para encontrar el punto de ejecución actual, utilice uno de estos métodos:

- Seleccione Ejecutar | Mostrar punto de ejecución.
- ▶ Haga clic en el botón Mostrar marco de la pila actual en la barra de herramientas del depurador.

El editor muestra el bloque de código en el área donde se encuentre en ese momento el punto de ejecución. El punto de ejecución se marca con una flecha en el margen izquierdo del editor, y se resalta esa línea de código. La ejecución del programa continúa a partir de ese punto.

**Figura 8.5** El punto de ejecución



Durante la depuración, es posible abrir, cerrar y desplazarse por cualquier archivo en el editor. Por ello resulta fácil perder la pista de qué sentencia de programa se ejecutará a continuación, o de la ubicación del ámbito del programa actual. Para volver rápidamente al punto de ejecución, seleccione Ejecutar | Mostrar punto de ejecución o haga clic en el botón Mostrar marco de la pila actual de la barra de herramientas del depurador.

## Definición del punto de ejecución

Es una función de JBuilder Enterprise.

Cuando el programa se suspende, se puede definir el punto de ejecución del hilo de inspección actual. Esto cambia el punto de ejecución de su ubicación actual. También es posible definir el punto de ejecución una vez que se ha utilizado el botón Intercambio inteligente. (Para obtener más información sobre el Intercambio inteligente, consulte “[Modificación del código durante la depuración](#)” en la página 8-72.)

Para definir el punto de ejecución del hilo de inspección actual:

- 1 Abra la vista Hilos, pilas de llamadas y datos.
- 2 Seleccione el marco de pila en el que desea continuar las operaciones, pulse con el botón derecho del ratón y, a continuación, seleccione Establecer punto de ejecución.
- 3 El editor muestra el código fuente del área del nuevo punto de ejecución. Si este es el hilo de inspección, el punto de ejecución se

resalta en el margen izquierdo del editor con una flecha, y se resalta la línea de código. La ejecución del programa continúa a partir de ese punto.



Puede utilizar el botón Establecer punto de ejecución de la barra de herramientas del Depurador para definir el punto de ejecución. El botón muestra una ventana emergente que enumera los marcos de pilas del hilo de inspección. Puede elegir el que desee. Observe que la selección del marco de pila actual está atenuada. El marco de pila donde va a continuar la ejecución del programa se marca en la vista Hilos, pilas de llamada y datos con el botón de inspección. También puede utilizar el comando de menú Ejecutar | Establecer punto de ejecución para establecer un nuevo marco de pila para continuar con las operaciones.

**Nota**

La definición del punto de ejecución no redefine el valor de las variables de objeto que se han modificado en las pilas de llamadas.

## Gestión de hilos

---

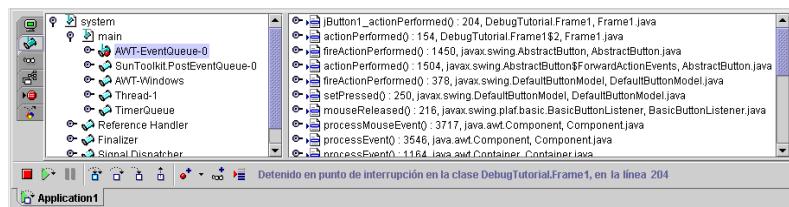
Para gestionar los hilos del programa con el depurador se pueden utilizar la vista Hilos, pila de llamadas y datos y la vista Monitores de sincronización.

- La vista Hilos, pila de llamadas y datos muestra el estado actual de todos los grupos de hilos del programa. También muestra todas las llamadas a métodos que ha realizado el programa, y el orden en el que han sido llamadas. Esto permite conocer las llamadas que se han realizado y que han conducido al error actual. Mediante este panel también se puede retroceder al lugar desde donde se llamó al método.
- La vista de Monitores de sincronización muestra todos los monitores de sincronización que utilizan todos los hilos del programa depurado, así como su estado actual.

## Utilización del panel dividido

Es una función de JBuilder SE y Enterprise.

La visualización por defecto de los Hilos, pilas de llamadas y datos se divide en dos paneles. El panel izquierdo se puede ampliar para que muestre los marcos de pila. El panel derecho muestra el contenido del elemento seleccionado a la izquierda, y muestra todo, desde un grupo de hilos hasta una variable. Por ejemplo, si se selecciona un hilo en el panel izquierdo, el panel derecho muestra los marcos de pila de ese hilo. Por otra parte, si se selecciona un marco de pila en el panel de la izquierda, el panel de la derecha mostrará las variables disponibles en esa vista.

**Figura 8.6** Panel dividido de la vista Hilos, pilas de llamadas y datos

## Presentación del hilo actual

**Es una función de JBuilder SE y Enterprise.**

Para mostrar únicamente las pilas de llamadas y los datos del hilo actual:

- 1 Abra la vista Hilos, pilas de llamadas y datos.
- 2 Haga clic con el botón derecho del ratón en una zona vacía de la vista.
- 3 Seleccione Mostrar sólo el hilo actual. Todos los hilos excepto el actual se eliminan de la vista.
- 4 Para mostrar de nuevo todos los hilos, haga clic con el botón derecho del ratón en una zona vacía de la vista y seleccione Mostrar sólo el hilo actual.

## Presentación del marco de pila superior

Para mostrar el marco de pila superior del hilo actual, haga clic en el icono Mostrar marco de la pila actual de la barra de herramientas del depurador.

## Elección de un hilo para inspeccionarlo

Para elegir el hilo que se va a inspeccionar:

- 1 Compruebe que se muestran todos los hilos en la vista Hilos, pila de llamadas y datos. (Haga clic con el botón derecho del ratón y compruebe que la opción Mostrar sólo el hilo actual está desactivada.)
- 2 Seleccione el hilo que desea inspeccionar.
- 3 Haga clic con el botón derecho del ratón y seleccione Establecer hilo inspeccionado. El icono del nuevo hilo inspeccionado cambia a

## Interrupción prolongada de un hilo

Una vez se ha interrumpido la depuración y está preparado para continuar la ejecución, puede mantener interrumpido un hilo. Esto permite observar el comportamiento de determinados hilos, sin que los otros interfieran con ellos.

Para continuar ejecutando el programa, seleccione el botón Reanudar el programa en la barra de herramientas del depurador. Cuando continúe la

**Es una función de JBuilder Enterprise.**



ejecución del programa, sólo los hilos que no se mantengan interrumpidos reanudarán la ejecución.

**Advertencia** Esto puede dar lugar a una situación conflictiva.

Para mantener interrumpido un hilo,



- 1 Inicie el programa y detenga el depurador mediante el botón Pausa.
- 2 En la vista Hilos, pilas de llamadas y datos, haga clic con el botón derecho sobre el hilo que quiere mantener interrumpido.
- 3 Seleccione Mantener hilo interrumpido.
- 4 Pulse el botón Reanudar.
- 5 El hilo seleccionado no reanudará la ejecución.



Un hilo interrumpido aparece representado por un icono Un icono que muestra un hilo de computadora roto en dos partes, indicando que el hilo no está ejecutándose.

## Detección de conflictos

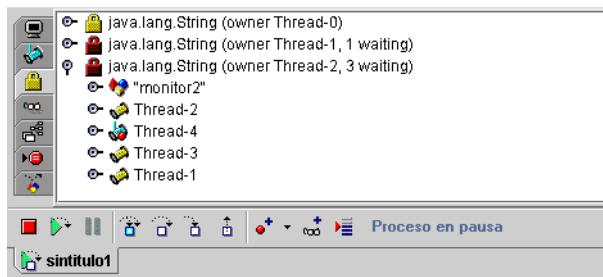
La posibilidad de detectar hilos en conflicto o bloqueados es una función de JBuilder Enterprise.

La vista Monitores de sincronización permite detectar los hilos bloqueados. Esta vista se puede utilizar para comprobar qué hilos esperan a qué monitores.

En esta vista, el monitor muestra el hilo que es su propietario, y los hilos, si los hay, que esperan para obtenerlo. Cuando un monitor se encuentra bloqueado, más de un hilo intenta obtenerlo. Sin embargo, no es posible liberarlo, porque el hilo que lo retiene espera a que se libere otro monitor. El icono (rojo) indica que un monitor está bloqueado.

Cuando se expande un monitor en la vista se ven todos los hilos que lo esperan, así como su propietario. Los iconos de esta vista muestran si el hilo es el que se encuentra activo actualmente. Los hilos se pueden expandir para mostrar su pila actual, como en la vista de Hilos, pila de llamadas y datos.

En el ejemplo siguiente, los hilos 1 y 2 tienen hilos en espera. Los hilos 4, 3 y 1 esperan al hilo 2; el hilo 2 es el propietario. El objeto de hilo se llama "monitor2". El hilo 4 es el que se está inspeccionando.

**Figura 8.7** Vista Monitores de sincronización

## Desplazamiento a través del código

Los comandos Ejecutar | Inspeccionar y Ejecutar | Omitir inspección ofrecen el método más sencillo para desplazarse en el código del programa. A pesar de que los dos comandos son muy similares, cada uno ofrece una manera diferente de controlar la ejecución del código.

**Nota** También es posible utilizar los botones Inspeccionar y Omitir inspección de la barra de herramientas del depurador.

El mínimo incremento con el que puede recorrer un programa es una sola línea de código. Si una línea de texto contiene varias sentencias de programa, se tratan como una sola línea de código; no es posible depurar individualmente varias sentencias contenidas en una sola línea de texto. Lo más sencillo es situar cada sentencia en su propia línea. Las sentencias que ocupan varias líneas de texto se tratan como una sola línea de código.

Conforme vaya depurando, puede inspeccionar el código de algunos métodos y excluir otros. Si está seguro de que un método funciona correctamente, puede omitir la inspección de dicho método, sabiendo que la llamada a métodos no provoca errores. Si no está seguro de que un método funciona bien, inspeccione el código del método y verifique si funciona. Debe omitir la inspección de métodos de bibliotecas proporcionadas por JBuilder u otros proveedores. Esto aumentará considerablemente la velocidad del ciclo de depuración.

## Inspección de código de llamadas a métodos

El comando Ejecutar | Inspeccionar código ejecuta las sentencias de programa de una en una. Si la función Paso inteligente se encuentra activada, no se entra en las clases de la vista Clases con inspección desactivada. Si Paso inteligente se encuentra desactivada, las clases de la Vista Clases con inspección desactivada se pasan por alto, de forma que es posible inspeccionarlas todas.

Si el punto de ejecución se sitúa en una llamada a un método, el comando Inspeccionar código entra en el método y coloca el punto de ejecución en

su primera sentencia. Los comandos Inspeccionar posteriores ejecutan las líneas de código del método una por una.

Si el punto de ejecución se encuentra en la última sentencia de un método, Inspeccionar hace que el depurador regrese desde el método, situando el punto de ejecución en la línea de código que sigue a la llamada al método del que regresa.

El término “avance paso a paso” se refiere a la utilización de Inspeccionar para ejecutar sucesivamente las sentencias del código del programa.

Existen varias maneras de ejecutar el comando Inspeccionar.

- Seleccione Ejecutar | Inspeccionar.
- Pulse *F7*.
- Pulse el botón Inspeccionar de la barra de herramientas del depurador.

### Omisión de inspección en las llamadas a métodos

El comando Ejecutar | Omitir inspección, igual que Ejecutar | Inspeccionar, permite ejecutar las sentencias de programa de una en una. Sin embargo, si ejecuta el comando Omitir inspección cuando el punto de ejecución se encuentra en una llamada a un método, el depurador ejecuta ese método sin detenerse (en lugar de seguir su ejecución), situando a continuación el punto de ejecución en la sentencia que sigue a la llamada al método.

Existen varias maneras de ejecutar el comando Omitir inspección.

- Seleccione Ejecutar | Omitir inspección.
- Pulse *F8*.
- Pulse el botón Omitir inspección de la barra de herramientas del depurador.

### Salida de métodos

El comando Ejecutar | Abandonar inspección permite abandonar la inspección de un método para pasar a la rutina de llamada.

Si la función Paso inteligente se encuentra activada, las clases de la vista Clases con inspección desactivada no se inspeccionan.

Existen varias maneras de ejecutar el comando Abandonar inspección:

- Seleccione Ejecutar | Abandonar inspección.
- Pulse el botón Abandonar inspección de la barra de herramientas del depurador.

## Paso inteligente

El commutador Paso inteligente permite determinar si la inspección es "inteligente". Para habilitar este commutador, elija Activar Paso inteligente en la ficha Depurar del cuadro de diálogo Propiedades de ejecución. Si lo prefiere, pulse el botón Paso inteligente en la barra de herramientas del depurador para activar Paso inteligente en la sesión actual.

Cuando esta función se encuentra activada, las operaciones de inspección inspeccionan las clases que aparecen en la vista Clases con inspección desactivada. En JBuilder SE y Enterprise, la inspección también se controla por medio de las opciones de Paso inteligente de la ficha Depurar del cuadro de diálogo Propiedades de ejecución.

- La vista Clases con inspección desactivada permite determinar las clases de las que no se efectúa un seguimiento. En JBuilder Personal sólo hay tres clases disponibles en esta vista: `java.lang.Object`, `java.lang.String` y `java.lang.ClassLoader`. No es posible añadir, modificar ni eliminar elementos de esta vista.
- En JBuilder SE y Enterprise, los parámetros de Paso inteligente de la ficha Depurar del cuadro de diálogo Propiedades de ejecución determinan la forma de inspección de las clases que son controladas. Estas opciones son:
  - Omitir métodos sintéticos  
Omite métodos sintéticos cuando inspecciona las clases.
  - Omitir constructores  
Omite constructores al inspeccionar las clases.
  - Omitir inicializadores estáticos  
Omite inicializadores estáticos al inspeccionar las clases.
  - Avisa si se detiene en una clase con la inspección desactivada (Es una función de todas las ediciones de JBuilder.)  
Muestra un mensaje de advertencia si hay un punto de interrupción en una clase que tenga la inspección desactivada. Para obtener más información, consulte "[Puntos de interrupción y configuración de Inspección desactivada](#)" en la página 8-45.

Si la opción Paso inteligente se encuentra desactivada, las clases de la vista Clases con inspección desactivada y las opciones de Paso inteligente se pasan por alto, de forma que es posible inspeccionarlas todas.

Paso inteligente se encuentra activado por defecto cuando se inicia una sesión de depuración.

- Para desactivarlo en la sesión actual, haga clic sobre el botón Paso inteligente en la barra de herramientas del depurador. Para desactivarlo al principio de la sesión de depuración, desactive la opción

Activar paso inteligente en la ficha Depurar en el cuadro de diálogo Propiedades de ejecución.

- El botón Paso inteligente de la barra de herramientas del depurador se atenúa cuando esta opción se encuentra desactivada. Para volver a activarlo, pulse el botón o active la opción Activar paso inteligente de la ficha Depurar del cuadro de diálogo Propiedades de configuración de ejecución.

## Ejecución hasta un punto de interrupción

---

Defina puntos de interrupción en las líneas de código fuente donde quiera que se detenga temporalmente la ejecución del programa. La ejecución hasta un punto de interrupción es similar a la ejecución hasta una posición del cursor, en ambos casos, el programa se ejecuta, hasta que alcanza un determinado punto del código fuente en el que se detiene.

El código puede tener varios puntos de interrupción. Puede personalizar cada uno de estos puntos para establecer las condiciones particulares bajo las que se debe detener el programa.

Si está depurando código distinto de Java y el programa está detenido en un punto de interrupción, puede cambiar las vistas. Pulse el botón Fuente inteligente de la barra de herramientas del depurador o, bien, seleccione Ejecutar | Fuente inteligente. Seleccione la vista del código que desee ver. La fuente se muestra en el editor, y el cursor se coloca en el marco de pila actual. Tenga en cuenta que se trata probablemente de un número de línea diferente que en la vista anterior. Por ejemplo, si está depurando una JSP, debe estar en la línea 120 del código fuente Java, y en la línea 55 del código fuente de la JSP (el código fuente distinto de Java). Para obtener más información, consulte “[Depuración de fuente distinta de Java](#)” en la [página 8-30](#).

Para obtener más información sobre los puntos de interrupción, consulte “[Puntos de interrupción](#)” en la [página 8-46](#).

## Ejecución hasta el final de un método

---

El comando Ejecutar | Ejecutar hasta el final del método ejecuta una aplicación hasta que llega al final del método actual. Este comando resulta muy útil cuando se inspecciona un método para el que se pretendía omitir la inspección.

## Ejecución hasta la posición del cursor

---

Puede ejecutar el programa hasta un punto justamente anterior a donde sospecha que puede encontrarse el problema. En ese punto, verifique que todos los valores de los datos sean correctos. A continuación, ejecute el programa hasta otra ubicación y verifique ahí los valores.

Para ejecutar el programa hasta una ubicación específica:

- 1 En el editor, sitúe el cursor en la línea de código en la que desea comenzar (o reanudar) la depuración.
- 2 Seleccione Ejecutar | hasta el cursor o haga clic con el botón derecho y elija hasta el cursor.

Con el comando Ejecutar hasta el cursor, el programa se ejecuta sin detenerse, hasta que alcanza la posición marcada por el cursor en el editor. Cuando la ejecución llega al código marcado por el cursor, el depurador recupera el control, interrumpe el programa y sitúa el punto de ejecución en esa línea del código. Para más información acerca del punto de ejecución, consulte “[El punto de ejecución](#)” en la página 8-32.

Este comando acelera el proceso de depuración, dado que permite desplazarse con rapidez a través de código sin errores.

## Visualización de llamadas a métodos

---

La vista de hilos, pila de llamadas y datos muestra todos los grupos de hilos del programa. Para cada hilo se muestra la secuencia de llamadas a métodos que llevaron al programa a su estado actual. Cada marco de pila se puede ampliar con el fin de mostrar los datos disponibles.

Si su programa se ha compilado con información de depuración (por defecto), esta vista también muestra los argumentos que se han pasado a una llamada de método. Cada método va seguido de una lista de los parámetros con los que se ha realizado la llamada. Esta vista también muestra el lugar en que se encuentran los métodos, la línea en que se activan las llamadas a métodos, el nombre de clase y el nombre de fuente.

Para ver el código fuente y el estado de los datos de una llamada a un método, haga clic en el método.

## Localización de una llamada a un método

---

Gracias a que es posible encontrar el lugar en el que se ha llamado a un método en el código fuente, se puede retroceder en una sesión de depuración.

Para buscar una llamada a un método, siga uno de estos procedimientos:

- Haga clic sobre el método en la vista Hilos, pilas de llamadas y datos. Esto le lleva al editor, en el cual aparece el cursor en la línea de código del archivo desde la que se llamó al método.
- Haga clic con el botón derecho sobre el editor y seleccione el comando Ejecutar hasta el cursor.

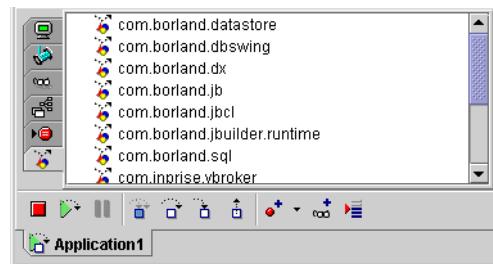
## Determinación de las clases que se han de inspeccionar

---

Si desea examinar detenidamente una parte del programa, puede indicar al depurador que inspeccione únicamente los archivos deseados. De esta forma, puede concentrarse en un área problemática conocida, en lugar de tener que avanzar manualmente paso a paso por todas y cada una de las líneas de código de todo el programa. Por ejemplo, normalmente no inspeccionará paso a paso las clases de la biblioteca Sun, porque no le hace falta resolver problemas en ellas; por lo general, solamente deseará inspeccionar y resolver los problemas de sus propias clases.

Para determinar las clases en las que se va a inspeccionar el código,

- Si está en sesión de depuración, seleccione la vista Clases con inspección desactivada. Este comando muestra las clases en las que no se debe inspeccionar el código.



- Si la sesión de depuración no ha comenzado, seleccione Ejecutar | Mostrar clases con inspección desactivada. Entonces se muestra el cuadro de diálogo Clases con inspección desactivada.



La vista y el cuadro de diálogo tienen el mismo funcionamiento.

**Nota** En JBuilder Personal, se añaden a esta vista tres clases básicas (`java.lang.Object`, `java.lang.String` y `java.lang.ClassLoader`). No se pueden añadir, modificar o eliminar elementos de la lista, aunque puede elegir entre inspeccionar o no esas clases. Consulte “[Paso inteligente](#)” en la [página 8-39](#) para obtener más información.

Las clases y paquetes se pueden activar y desactivar en cualquier momento desde la vista Clases con inspección desactivada: Basta con hacer clic con el botón derecho del ratón en la clase o el paquete y activar o desactivar la opción Inspeccionar clase/paquete. Si se encuentra desactivada (por defecto), la clase o el paquete no se inspecciona. Si se encuentra activada, se realiza la inspección. Observe que el ícono cambia según el estado.

- Cuando la inspección está activada, el ícono es:  (De colores)
- Cuando está desactivada, el ícono es:  (Gris)

**Nota** Cuando se desactiva la inspección de un paquete se desactiva la inspección de todas las clases que contiene.

En JBuilder SE y Enterprise, si desea eliminar una clase o un paquete de la lista, selecciónelo y pulse *Borrar*, o selecciónelo, haga clic con el botón derecho del ratón y seleccione Eliminar clase/paquete. Si desea borrar todas las clases y paquetes, haga clic con el botón derecho del ratón en una zona vacía de la vista y seleccione Eliminar todos. Si se borran todas las clases y paquetes de la vista, se activa automáticamente la inspección de todas las clases a las que llama el programa.

En JBuilder SE y Enterprise, se le ofrece la posibilidad de añadir clases y paquetes a la lista haciendo clic con el botón derecho del ratón en un área vacía de la vista y seleccionando Añadir clase o paquete. Aparece el cuadro de diálogo Seleccionar clase o paquete, en el que se puede introducir el nombre de la clase o el paquete que se desea excluir de la inspección.

En JBuilder SE y Enterprise, se le ofrece la posibilidad de modificar clases y paquetes de la lista haciendo clic con el botón derecho del ratón sobre una clase o paquete de la vista y seleccionando Modificar clase/paquete. Aparece el cuadro de diálogo Seleccionar clase o paquete, en el que se puede introducir el nombre de la clase o el paquete que se desea inspeccionar.

Los cambios se efectúan de forma inmediata. No es necesario reiniciar la sesión de depuración.

Las clases de la vista Clases con inspección desactivada, con su estado activado o desactivado, se guardan en el archivo del proyecto.

Después de elegir las clases que no desea inspeccionar, pulse el ícono Paso inteligente de la barra de herramientas del depurador, para controlar la inspección. Cuando esta función se encuentra activada, las operaciones de



inspección utilizan las clases que aparecen en la vista Clases con inspección desactivada y los parámetros de paso inteligente seleccionados en la ficha Depurar del cuadro de diálogo Propiedades de configuración de ejecución:

- La vista Clases con inspección desactivada permite determinar las clases de las que no se efectúa un seguimiento.
- Los parámetros de Paso inteligente de la ficha Depurar del cuadro de diálogo Propiedades de ejecución controlan la forma de inspección de las clases de las que se efectúa un seguimiento.

Si la opción Paso inteligente se encuentra desactivada, las clases de la vista Clases con inspección desactivada y los parámetros de Paso inteligente se pasan por alto, de forma que es posible inspeccionarlas todas.

### Inspección de clases cuando el archivo fuente no está disponible

Es una función de JBuilder SE y Enterprise.

Si se desactiva el Paso inteligente mientras se utiliza una clase cuyo archivo de código fuente no está disponible, se genera un archivo fuente stub, que aparece cuando se inspecciona el código. El archivo fuente stub sólo muestra las firmas de los métodos de esa clase. Si no desea que se muestre la fuente stub, conserve la clase en la vista Clases con inspección desactivada y deje activado Paso inteligente.

### Archivos fuente de stub

Si la fuente stub se ha generado a partir de archivos para los que existe una fuente, compruebe la vía de acceso a archivos fuente. El depurador busca archivos fuente en la vía de acceso a archivos fuente. La vía de acceso a archivos fuente se describe en el apartado “[Vía de acceso a archivos fuente](#)” en la página 4-10. El archivo .java que se depura debe existir en una bifurcación que sea equivalente a su nombre de paquete.

Por ejemplo, si la vía de acceso a archivos fuente contiene un elemento:

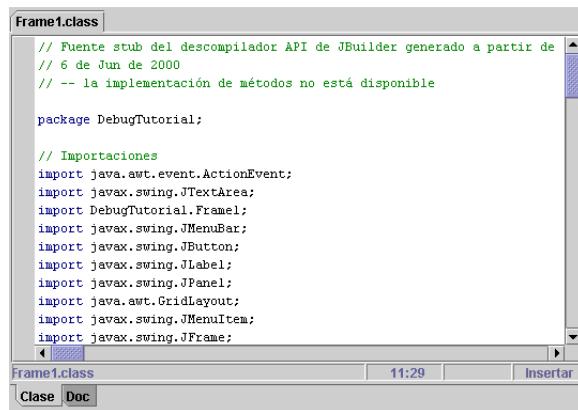
c:\MyProjects\Test\src

y el archivo .java se encuentra en un paquete llamado mypackage,, el depurador espera que el archivo .java exista en el directorio:

c:\MyProjects\Test\src\mypackage

El nombre del paquete se agrega al nombre del elemento fuente. Si cuenta con múltiples elementos fuente, el depurador intentará localizarlos todos mediante el sistema explicado anteriormente. Si el depurador no puede localizar el archivo fuente, genera una fuente stub.

Se muestra un archivo fuente stub en el panel de contenidos. Contiene una cabecera y los stubs del método.

**Figura 8.8** Archivo fuente stub


```
// Fuente stub del descompilador API de JBuilder generado a partir de
// 6 de Jun de 2000
// -- la implementación de métodos no está disponible

package DebugTutorial;

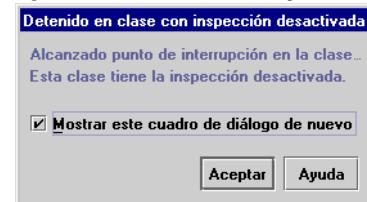
// Importaciones
import java.awt.event.ActionEvent;
import javax.swing.JTextArea;
import DebugTutorial.Frame1;
import javax.swing.JMenuBar;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import java.awt.GridLayout;
import javax.swing.JMenuItem;
import javax.swing.JFrame;
```

## Puntos de interrupción y configuración de Inspección desactivada

Cuando se define un punto de interrupción en una clase, en la vista Clases con inspección desactivada, se redefine la configuración de inspección y se detiene provisionalmente la clase, porque se ha indicado al depurador que acuda a ese punto.

El cuadro de diálogo de advertencia Detenido en clase con inspección desactivada se muestra si:

- el paso inteligente se encuentra activado y, además, si
- la opción Advertir si se detiene en una clase con la inspección desactivada se encuentra activada en la ficha Depurar del cuadro de diálogo Propiedades de configuración de ejecución.

**Figura 8.9** Cuadro de diálogo Detenido en clase con inspección desactivada

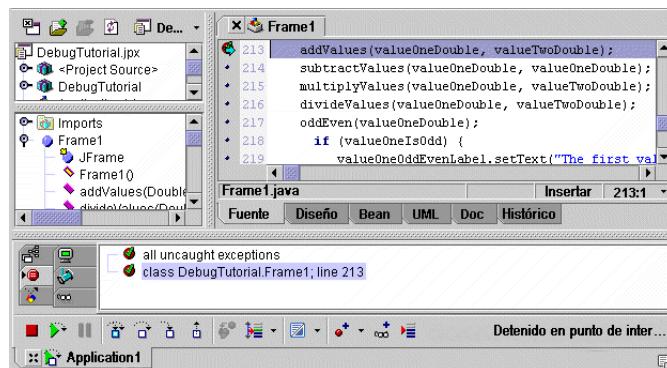
Si esto ocurre, cuando se efectúa la inspección después de alcanzar el punto de interrupción, el depurador sale de la clase. Para permanecer en la clase, desactive el paso inteligente y utilice los botones de inspección.

# Puntos de interrupción

Cuando la ejecución encuentra un punto de interrupción, el programa se interrumpe y el depurador muestra en el editor la línea que contiene el punto de interrupción. Puede utilizar entonces el depurador para ver el estado del programa. Los puntos de interrupción son flexibles ya que pueden definirse antes de comenzar la ejecución de programa o en cualquier momento en que el depurador tiene el control. Definiendo puntos de interrupción en áreas potencialmente problemáticas del código fuente, es posible ejecutar el programa sin pausa hasta que su ejecución llegue a una posición que desee depurar.

Los puntos de interrupción se presentan y se modifican en la vista Puntos de interrupción por datos y código. Se muestra el tipo de punto de interrupción y su estado, así como información relacionada con el tipo de punto de interrupción, como el número de línea, el nombre de la clase y del método. Puede activar, desactivar, añadir y eliminar puntos de interrupción desde el menú contextual.

**Figura 8.10** Vista Puntos de interrupción de datos y código



## Definición de puntos de interrupción

Los puntos de interrupción de la clase, del método, de la excepción y del campo son características de JBuilder SE y Enterprise.

Los puntos de interrupción interprocesales son una característica de JBuilder Enterprise.

Es posible definir puntos de interrupción de línea, excepción, clase, método, campo e interprocesales en el depurador:

- Los puntos de interrupción de línea se establecen en una línea determinada del código fuente Java o distinto de Java. El depurador se interrumpe en esa línea.
- Los puntos de interrupción por excepción hacen que el depurador se detenga cuando está a punto de lanzarse la excepción especificada.
- Los puntos de interrupción de clase hacen que el depurador se detenga cuando se llama a un método de la clase especificada o cuando se instancia la clase especificada.

- Los puntos de interrupción de método hacen que el depurador se detenga cuando se llama al método especificado de la clase especificada.
- Los puntos de interrupción de campo hacen que el depurador se detenga cuando se va a leer o escribir en el campo especificado. Un campo es una variable Java que está definida en un objeto Java.
- Un punto de interrupción interprocesal hace que se detenga el depurador cuando se encuentre con bien un método o el método especificado en la clase especificada en un proceso aparte.

## Definición de puntos de interrupción de línea

Los puntos de interrupción de línea hacen que el depurador se detenga cuando alcanza una línea de código determinada. Los puntos de interrupción de línea se pueden definir en código fuente Java o distinto de Java. Se pueden definir directamente en el editor o por medio del cuadro de diálogo Añadir punto de interrupción de línea.

Para definir un punto de interrupción de línea en el código fuente, haga clic en el margen izquierdo de la línea en la que se desea establecer el punto de interrupción. También puede pulsar *F5* en una línea de código fuente para comutar un punto de interrupción. Cuando el depurador tiene el foco, aparecen pequeños puntos azules • en el editor, a la izquierda de las líneas de código ejecutable, indicando así que se puede definir el punto de interrupción en esa línea.

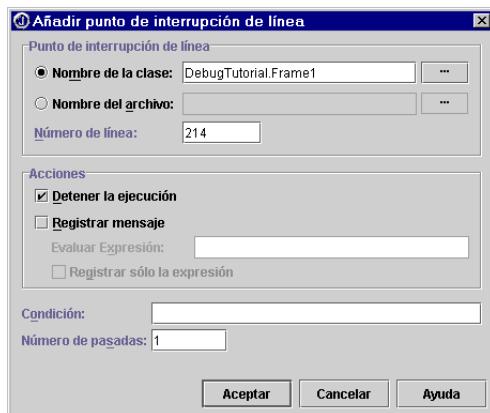
Las definiciones de puntos de interrupción en las líneas de comentarios, en las declaraciones y en otras líneas no ejecutables de código no son válidos. Los puntos de interrupción no válidos se indican mediante ✘ en el margen del editor cuando se ejecuta el programa.

Para definir un punto de interrupción de línea por medio del cuadro de diálogo Añadir punto de interrupción de línea, siga las instrucciones adecuadas a su caso:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar | Añadir punto de interrupción y seleccione Añadir punto de interrupción de línea.
- Durante la sesión de depuración, pulse la flecha de lista desplegable que se encuentra a la derecha del botón Añadir punto de interrupción de la barra de herramientas del depurador y seleccione Añadir punto de interrupción de línea.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista de puntos de interrupción de código y datos y seleccione Añadir punto de interrupción de línea.



Aparece el cuadro de diálogo Añadir punto de interrupción de línea.



Defina el punto de interrupción de línea, escogiendo entre las siguientes opciones:

- 1 Si se está definiendo un punto de interrupción en un archivo .class de Java, utilice el campo Nombre de la clase. Si el punto de interrupción se encuentra en un archivo que no es un archivo .class, utilice el campo Nombre del archivo.
  - Si se trata de un archivo .class, escriba el nombre o pulse el botón puntos suspensivos (...) para acceder a un archivo .class.
  - Si no se trata de un archivo .class, pulse el botón puntos suspensivos (...) para acceder al archivo.
- 2 En el campo Número de línea, escriba el número de la línea en la que se va a definir el punto de interrupción.
- 3 Seleccione las acciones del punto de interrupción. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Para obtener más información, consulte ["Definición de las acciones de puntos de interrupción"](#) en la página 8-57. (Las acciones son una función de JBuilder SE y Enterprise.)
- 4 En el campo Condición, asigne, si existe, la condición de punto de interrupción para este punto. Para obtener más información, consulte ["Creación de puntos de interrupción condicionales"](#) en la página 8-59.
- 5 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Para obtener más información, consulte ["Utilización de puntos de interrupción por número de pasadas"](#) en la página 8-60.
- 6 Pulse Aceptar para cerrar el cuadro de diálogo.

Cuando establezca un punto de interrupción en una línea válida (línea de código ejecutable), se resaltará la línea y aparecerá un ícono de



círculo rojo con una marca de selección en el margen izquierdo de la línea del punto de interrupción.

## Definición de puntos de interrupción por excepción

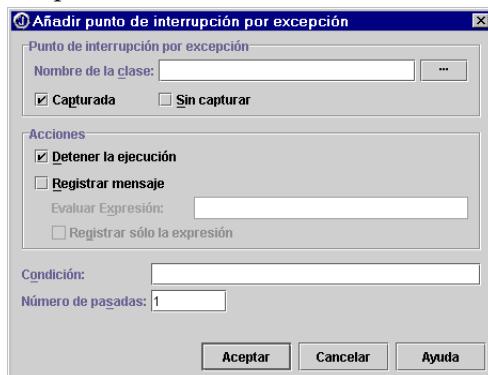
**Los puntos de interrupción por excepción son una característica de JBuilder SE y Enterprise**

Los puntos de interrupción por excepción hacen que el depurador se detenga cuando está a punto de lanzarse la excepción especificada. El depurador puede detenerse sobre excepciones capturadas o sin capturar. Para fijar un punto de interrupción por excepción, utilice el cuadro de diálogo Añadir punto de interrupción por excepción.

Para abrir el cuadro de diálogo Añadir punto de interrupción por excepción, haga una de las siguientes cosas:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar | Añadir punto de interrupción y seleccione Añadir punto de interrupción por excepción.
- Durante la sesión de depuración, pulse la flecha de lista desplegable que se encuentra a la derecha del botón Añadir punto de interrupción de la barra de herramientas del depurador y seleccione Añadir punto de interrupción de excepción.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista Puntos de interrupción de datos y código y seleccione Añadir punto de interrupción por excepción.

Aparece el cuadro de diálogo Añadir punto de interrupción por excepción.



Para definir un punto de interrupción por excepción:

- 1 En el campo Nombre de la clase, escriba el nombre del archivo de clase de excepción en que se va a detener el depurador. Puede escribir el nombre o pulsar el botón puntos suspensivos (...) para acceder a un archivo .class.
- 2 Seleccione cuándo debe detenerse el depurador:

- Seleccione la opción Capturada de modo que el depurador se detenga cuando se capture la excepción especificada.
- Seleccione la opción Sin capturar de modo que el depurador se detenga cuando no se capture la excepción especificada.

También es posible elegir las dos opciones para que el depurador se detenga en ambos casos.

- 3 Elija qué acciones realizará el depurador. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Para obtener más información, consulte “[Definición de las acciones de puntos de interrupción](#)” en la página 8-57. (Las acciones son una función de JBuilder SE y Enterprise.)
- 4 En el campo Condición, si existe alguna, asígnela al punto de interrupción. Para obtener más información, consulte “[Creación de puntos de interrupción condicionales](#)” en la página 8-59.
- 5 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Para obtener más información, consulte “[Utilización de puntos de interrupción por número de pasadas](#)” en la página 8-60.
- 6 Pulse Aceptar para cerrar el cuadro de diálogo.

## Definición de puntos de interrupción de clase

Los puntos de interrupción de clase son una característica de JBuilder SE y Enterprise

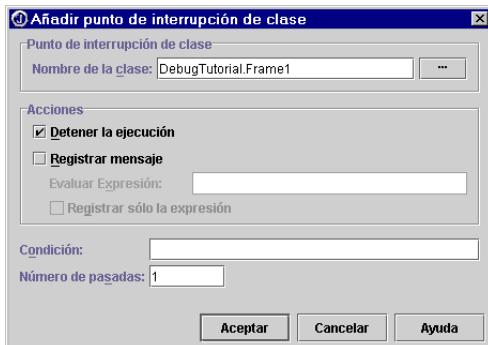
Los puntos de interrupción de clase hacen que el depurador se detenga cuando se llama a un método de la clase especificada o cuando se instancia la clase especificada. Para definir un punto de interrupción de clase, utilice el cuadro de diálogo Añadir punto de interrupción de clase.

Para abrir el cuadro de diálogo Añadir punto de interrupción de clase, haga una de las siguientes cosas:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar | Añadir punto de interrupción y seleccione Añadir punto de interrupción de clase.
- Durante la sesión de depuración, pulse la flecha de lista desplegable que se encuentra a la derecha del botón Añadir punto de interrupción de la barra de herramientas del depurador y seleccione Añadir punto de interrupción de clase.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista Puntos de interrupción por datos y código y seleccione Añadir punto de interrupción de clase.



Aparece el cuadro de diálogo Añadir punto de interrupción de clase.



Para definir un punto de interrupción de la clase:

- 1 En el campo Nombre de la clase, escriba el nombre del archivo de clase en el que desea que se detenga el depurador. Puede escribir el nombre o pulsar el botón puntos suspensivos (...) para acceder a un archivo .class.
- 2 Elija qué acciones realizará el depurador. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Para obtener más información, consulte “[Definición de las acciones de puntos de interrupción](#)” en la página 8-57. (La acciones son una función de JBuilder SE y Enterprise.)
- 3 En el campo Condición, si existe alguna, asígnela al punto de interrupción. Para obtener más información, consulte “[Creación de puntos de interrupción condicionales](#)” en la página 8-59.
- 4 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Para obtener más información, consulte “[Utilización de puntos de interrupción por número de pasadas](#)” en la página 8-60.
- 5 Pulse Aceptar para cerrar el cuadro de diálogo.

## Definición de puntos de interrupción de método

Los puntos de interrupción de método son una característica de JBuilder SE y Enterprise

Los puntos de interrupción de método hacen que el depurador se detenga cuando se llama al método especificado de la clase especificada. Para configurar un punto de interrupción de método, utilice el cuadro de diálogo Añadir punto de interrupción de método.

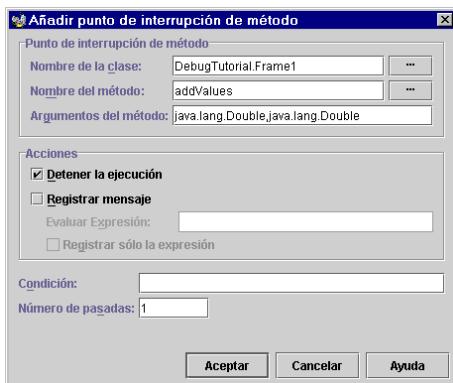
Para abrir el cuadro de diálogo Añadir punto de interrupción de método, haga una de las siguientes cosas:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar | Añadir punto de interrupción y seleccione Añadir punto de interrupción de método.



- Durante la sesión de depuración, pulse la flecha de lista desplegable que se encuentra a la derecha del botón Añadir punto de interrupción de la barra de herramientas del depurador y seleccione Añadir punto de interrupción de método.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista de puntos de interrupción de código y datos y elija Añadir punto de interrupción de método.

Aparece el cuadro de diálogo Añadir punto de interrupción de método.



Para definir un punto de interrupción de método:

- En el campo Nombre de la clase, escriba el nombre del archivo de clase que contiene el método en el que desea que se detenga el depurador. Puede escribir el nombre o pulsar el botón puntos suspensivos (...) para acceder a un archivo .class.
- En el campo Nombre del método, escriba el nombre del método en el que desea se detenga el depurador. Haga clic sobre el botón de puntos suspensivos para hallar el método que desea utilizar.
- En el campo Argumentos del método, introduzca una lista de argumentos del método separados por comas. Esto hace que el depurador se detenga únicamente cuando coinciden el nombre del método y la lista de argumentos. Esto resulta útil con los métodos sobrecargados. Si utiliza el visualizador para añadir el método, los argumentos de método se llenan automáticamente. Si no se especifican argumentos, el depurador detiene todos los métodos con el nombre de método especificado.
- Elija qué acciones realizará el depurador. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Para obtener más información, consulte ["Definición de las acciones de puntos de interrupción"](#) en la página 8-57. (La acciones son una función de JBuilder SE y Enterprise.)

- 5 En el campo Condición, si existe alguna, asígnela al punto de interrupción. Para obtener más información, consulte “[Creación de puntos de interrupción condicionales](#)” en la página 8-59.
- 6 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Para obtener más información, consulte “[Utilización de puntos de interrupción por número de pasadas](#)” en la página 8-60.
- 7 Pulse Aceptar para cerrar el cuadro de diálogo.

## Definición de puntos de interrupción de campo

**Los puntos de interrupción de campo son una característica de JBuilder SE y Enterprise**

Los puntos de interrupción de campo hacen que el depurador se detenga cuando se va a leer o escribir, según lo elegido, en el campo especificado. Un campo es una variable Java que está definida en un objeto Java. En el siguiente ejemplo:

```
class Test {
    private int x;
    private Object y;
}
Test myTest = new Test();
```

myTest es una variable. Las variables Java x e y representan campos.

Para añadir un punto de interrupción de campo, haga clic con el botón derecho del ratón en una variable de campo de la vista Hilos, pila de llamadas y datos y elija Añadir punto de interrupción de campo. El punto de interrupción se añade automáticamente a la vista Puntos de interrupción de datos y código. Un punto de interrupción de campo se indica con .

Para controlar si el depurador se interrumpe en una acción de lectura o escritura, abra la vista Puntos de interrupción por datos y código. Haga clic con el botón derecho del ratón en el punto de interrupción de campo que acaba de definir. Por defecto, los comandos Interrumpir al leer e Interrumpir al escribir del menú contextual se encuentran activados, lo que significa que el depurador se detendrá cuando se vaya a leer o escribir el campo especificado. Puede desactivar una o ambas opciones y permitir que continúe el depurador, en lugar de detener el depurador cuando está a punto de leer o escribir un campo.

## Configuración de un punto de interrupción interprocesal

**Los puntos de interrupción interprocesales son características de JBuilder Enterprise**

Un punto de interrupción interprocesal detiene el depurador cuando se inspecciona, en un proceso aparte, un método o el método especificado en la clase especificada. Esto permite inspeccionar un proceso servidor desde un proceso cliente, en lugar de tener que configurar puntos de interrupción en el cliente y en el servidor. Por lo general se configura un punto de interrupción de línea en el cliente y un punto de interrupción interprocesal en el servidor. Si desea seguir un tutorial que explica paso a

paso la inspección interprocesal, consulte el [Capítulo 19, “Tutorial: Depuración remota”](#).

Para activar un punto de interrupción interprocesal configurado en un proceso de servidor,

- 1 Inicie el proceso de servidor en el ordenador remoto en modo depuración.
- 2 En el ordenador cliente y desde JBuilder, conéctese con el servidor que ya está en ejecución en el ordenador remoto.
- 3 Configure un punto de interrupción en el código del cliente y comience a depurar el cliente. Cuando llegue al punto de interrupción, inspeccione el código del servidor. No utilice Omitir inspecciones, ya que la omisión de inspecciones hará que el depurador no se detenga en el punto de interrupción interprocesal.

**Nota** Puede utilizar los puntos de interrupción interprocesal para depurar de manera local, por ejemplo, una aplicación cliente/servidor que se ejecuta en un ordenador.

Para configurar un punto de interrupción interprocesal, utilice el cuadro de diálogo Añadir punto de interrupción interprocesal. Para abrir el cuadro de diálogo Añadir punto de interrupción interprocesal, haga una de las siguientes cosas:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar | Añadir punto de interrupción y seleccione Añadir punto de interrupción interprocesal.
- Durante la sesión de depuración, pulse la flecha de lista desplegable que se encuentra a la derecha del botón Añadir punto de interrupción () de la barra de herramientas del depurador y seleccione Añadir punto de interrupción interprocesal.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista Puntos de interrupción de datos y de código y seleccione Añadir punto de interrupción interprocesal.



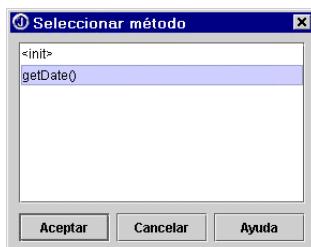
Aparece el cuadro de diálogo Añadir punto de interrupción interprocesal.



Si desea seguir un tutorial que explica paso a paso la inspección interprocesal, consulte [Capítulo 19, “Tutorial: Depuración remota”](#).

Para configurar un punto de interrupción interprocesal:

- 1 En el campo Nombre de clase, escriba el nombre de la clase del servidor que contiene el método en el que desea se detenga el depurador. Puede escribir el nombre o pulsar el botón puntos suspensivos (...) para acceder a un archivo .class.
- 2 En el campo Nombre del método, escriba el nombre del método en el que desea se detenga el depurador. Utilice el botón de puntos suspensivos para mostrar el cuadro de diálogo Seleccionar método, en el cual se puede acceder a la lista de métodos disponibles en la clase seleccionada.



No es necesario escribir el nombre del método. Si no se especifica un nombre de método, el depurador se detiene en todas las llamadas a métodos en la clase especificada.

**Nota**

No se puede seleccionar un método si la clase seleccionada contiene errores de sintaxis o de compilación.

- 3 En el campo Argumentos del método, introduzca una lista de argumentos del método separados por comas. Esto hace que el depurador se detenga cuando coinciden el nombre del método y la lista de argumentos. Esto resulta útil con los métodos sobrecargados.

- Si no se establece ningún argumento, el depurador se detiene en todos los métodos cuyo nombre coincide con el especificado.
  - Si se elige un nombre de método del cuadro de diálogo Seleccionar método, el campo Argumentos de método se rellena automáticamente.
- 4 Elija qué acciones realizará el depurador. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Para obtener más información, consulte “[Definición de las acciones de puntos de interrupción](#)” en la página 8-57. (Las acciones son una función de JBuilder SE y Enterprise.)
- 5 En el campo Condición, si existe alguna, asignela al punto de interrupción. Para obtener más información, consulte “[Creación de puntos de interrupción condicionales](#)” en la página 8-59.
- 6 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Para obtener más información, consulte “[Utilización de puntos de interrupción por número de pasadas](#)” en la página 8-60.
- 7 Pulse Aceptar para cerrar el cuadro de diálogo.
- 8 Fije un punto de interrupción de línea en el cliente, en el método que llama al punto de interrupción interprocesal.
- 9 Haga clic sobre el botón Inspeccionar en la barra del depurador para inspeccionar el método con punto de interrupción del servidor. (Si utiliza Omitir inspección, el depurador no se detendrá.)

## Definición de las propiedades de los puntos de interrupción

---

Después de crear un punto de interrupción se pueden definir y modificar sus propiedades. Para definir las propiedades de punto de interrupción,

- 1 Abra la vista Puntos de interrupción de datos y código.
  - 2 Elija el punto de interrupción que desea configurar. Haga clic con el botón derecho y seleccione Propiedades de punto de interrupción.
- Se muestra el cuadro de diálogo de Propiedades de punto de interrupción.

**Nota** El cuadro de diálogo Propiedades de punto de interrupción contiene las mismas opciones que el cuadro de diálogo utilizado para crear el punto de interrupción.

- 3 Es posible cambiar las siguientes propiedades:
    - Acciones
- Las acciones que se deben realizar cuando se alcanza un punto de interrupción. El depurador puede detener la ejecución en el punto de

interrupción, mostrar un mensaje o evaluar una expresión. Para obtener más información, consulte “[Definición de las acciones de puntos de interrupción](#)” en la página 8-57.

- Condición

La condición de este punto de interrupción, si la hay. Para obtener más información, consulte “[Creación de puntos de interrupción condicionales](#)” en la página 8-59.

- Número de pasadas

El número de veces que se debe pasar el punto de interrupción para que se active. Para obtener más información, consulte “[Utilización de puntos de interrupción por número de pasadas](#)” en la página 8-60.

Para mostrar el cuadro de diálogo Propiedades de punto de interrupción en modo de sólo lectura, coloque el ratón en el margen interior contiguo a la línea con punto de interrupción. Pulse **Ctrl** junto con el botón derecho del ratón. Se muestran las propiedades principales del punto de interrupción, pero no es posible modificarlas. Los campos Acciones, Condición y Número de pasadas pueden modificarse.

## Definición de las acciones de puntos de interrupción

Es una función de JBuilder SE y Enterprise.

Es posible seleccionar varias acciones que deben realizarse cuando llega un punto de ejecución. El depurador puede:

- Detener la ejecución del programa y mostrar un mensaje en la barra de estado del depurador (la barra por defecto).
- Registrar un mensaje en la vista Salida, entrada y errores de consola.
- Evaluar una expresión y guardar los resultados.

Las acciones se definen en la zona central del cuadro de diálogo Puntos de interrupción.

**Figura 8.11** Acciones de punto de interrupción



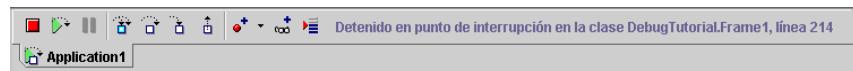
### Detención de la ejecución del programa

Para detener la ejecución del programa cuando se alcance el punto de interrupción especificado, seleccione la opción Detener la ejecución. Si detiene la ejecución del programa, el depurador se detendrá en el punto de interrupción especificado y se mostrará un mensaje de estado en la

barra de herramientas del depurador y la ventana Puntos de interrupción de datos y código.

El mensaje que aparezca en la barra de estado dependerá del tipo de punto de interrupción. El siguiente ejemplo muestra el mensaje de la barra de estado que aparece para un punto de interrupción de línea.

**Figura 8.12** Mensaje de la barra de estado de punto de interrupción

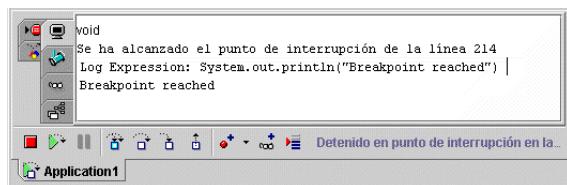


## Registro de mensajes

Para registrar un mensaje en la vista Salida, entrada y errores de la consola cuando se alcanza un punto de interrupción, seleccione la opción Registrar mensaje y escriba un mensaje en el cuadro Evaluar Expresión. Cuando el depurador alcance el punto de interrupción seleccionado, se registrará un mensaje en la vista. Si además se selecciona la opción Detener la ejecución, el programa se detendrá. Si no, continúa la ejecución del programa.

## Registro de un mensaje con una sentencia println

Es posible utilizar sentencias `println` para registrar mensajes de salida. El siguiente ejemplo muestra un mensaje en la vista Puntos de interrupción por datos y código que se registró con la sentencia -  
`System.out.println("Punto de interrupción")`. La sentencia se introdujo en el cuadro de entrada Evaluar Expresión y se activó la opción Registrar mensaje.

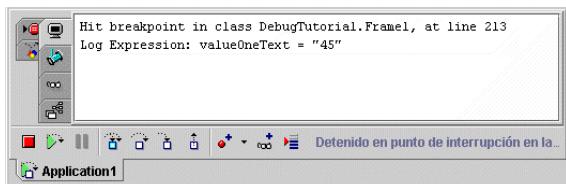


## Registro de un mensaje con evaluación de expresiones

También es posible utilizar la evaluación de expresiones, en lugar de las sentencias `println`, para registrar mensajes durante la depuración. Para hacerlo, escriba una expresión en el campo de entrada Evaluar Expresión. El depurador evalúa la expresión cuando se alcanza el punto de interrupción y escribe los resultados de la evaluación en la vista Salida, entrada y errores de la consola. La expresión puede ser cualquier sentencia válida en lenguaje Java.

Esta opción sólo está disponible si se selecciona la opción Registrar mensaje. Puede detener la ejecución del programa cuando se evalúa una expresión si selecciona la opción Detener la ejecución.

El siguiente ejemplo muestra los resultados de una expresión `valueOneText`. La expresión se introduce en el campo Evaluar expresión, con la opción Registrar mensaje marcada.



La opción Registrar sólo la expresión permite registrar sólo los resultados de esta expresión, de forma que el registro no se llene con otra información.

## Creación de puntos de interrupción condicionales

Cuando se crea un punto de interrupción, la ejecución del programa se interrumpe por defecto cada vez que se llega a él. Sin embargo, el cuadro de diálogo Propiedades de punto de interrupción permite personalizar los puntos de interrupción de forma que se activen sólo bajo determinadas condiciones.

Si se escribe una expresión booleana en el campo Condición, un punto de interrupción pasa a ser condicional y la ejecución del programa se detiene en este punto de interrupción sólo si la condición es `true`. También se puede basar un punto de interrupción en el número de pasadas, que se especifica en el campo Número de pasadas. Este campo resulta útil para los bucles de depuración. La ejecución del programa se detiene en el punto de interrupción cuando recorre el bucle el número de veces especificado.

Las condiciones se definen en la parte inferior del cuadro de diálogo Puntos de interrupción.

**Figura 8.13** Puntos de interrupción condicionales



### Definición de la condición de punto de interrupción

El cuadro de edición del cuadro de diálogo Propiedades de punto de interrupción permite escribir una expresión que se evalúa cada vez que alcanza el punto de interrupción durante la ejecución del programa.

- Si la condición da como resultado `true`, el depurador se detendrá en el punto de interrupción si está activada la opción Detener la ejecución.

- Si la evaluación de la expresión da como resultado `false`, el depurador no se detiene en el punto de interrupción.

Los puntos de interrupción condicionales permiten ver cómo se comporta el programa cuando una variable contiene valores comprendidos en un rango concreto o qué ocurre cuando se define un flag (indicador) determinado.

Por ejemplo, suponga que desea que un punto de interrupción se interrumpa en una línea de código sólo cuando la variable `mediumCount` supere 10. Para hacer esto:

- 1 Defina un punto de interrupción en una línea de código, haciendo clic a la izquierda de la línea en el editor.
- 2 Haga clic con el botón derecho y seleccione Propiedades de punto de interrupción.
- 3 Introduzca la expresión siguiente en el cuadro de edición Condición y haga clic en Aceptar:

```
mediumCount > 10
```

Puede escribir una expresión válida en lenguaje Java en el cuadro de edición Condición, pero todos los símbolos de la expresión deben ser accesibles desde el punto de interrupción.

## Utilización de puntos de interrupción por número de pasadas

La condición Número de pasadas del cuadro de diálogo Propiedades de punto de interrupción determina el número de veces que se debe pasar el punto de interrupción para que se active. El Depurador detiene el programa tras encontrar `n` veces el punto de interrupción durante la ejecución del programa. El valor por defecto de `n` es 1.

Los números de pasadas resultan útiles cuando se piensa que un bucle está fallando en la iteración número `n`. Cuando se utiliza el número de pasadas con condiciones booleanas, la ejecución del programa se suspende la `n` vez que la condición tiene el valor `true`.

## Desactivación y activación de puntos de interrupción

---

La desactivación de un punto de interrupción lo oculta de la ejecución actual del programa. Si desactiva un punto de interrupción, todos sus valores permanecen definidos, pero durante la ejecución del programa se obvia el punto de interrupción. El programa no se detiene en los puntos de interrupción desactivados. La desactivación de un punto de interrupción resulta útil si ha definido un punto de interrupción condicional que no necesita utilizar ahora, pero que podría necesitar más adelante.

- Para desactivar un solo punto de interrupción, haga clic en él con el botón derecho del ratón en la vista Puntos de interrupción de datos y código. Seleccione Activar punto de interrupción para conmutar el comando y desactivar el punto de interrupción. Si este conmutador está desactivado, también lo está el punto de interrupción, y la ejecución no se detiene al llegar a él. Si está activado, también lo está el punto de interrupción, y la ejecución se detiene al llegar a él. Los puntos de interrupción están activados por defecto.
- Para desactivar o activar todos los puntos de interrupción definidos para una sesión de depuración, abra la vista de Puntos de interrupción de datos y código. Haga clic con el botón derecho del ratón en una zona vacía de la vista y seleccione Desactivar todos o Activar todos.

También es posible desactivar los puntos de interrupción para una configuración de ejecución determinada. Si hay más de una configuración de ejecución definida en el proyecto actual, dispondrá del comando Desactivar en la configuración cuando haga clic con el botón derecho en un punto de interrupción. Basta con elegir las configuraciones para las que se desea desactivar el punto de interrupción. Cuando se efectúe la depuración con la configuración especificada, el punto de interrupción elegido estará desactivado.

## Eliminación de puntos de interrupción

---

Cuando ya no necesite examinar el código correspondiente a un punto de interrupción, puede eliminar la interrupción de la sesión. Los puntos de interrupción de línea se pueden eliminar en el editor. Los demás tipos de puntos de interrupción se eliminan con la vista de Puntos de interrupción de datos y código.

Tenga en cuenta que no es posible eliminar el punto de interrupción por defecto, definido en todas las excepciones sin capturar. Sin embargo, se puede desactivar.

Utilice uno de los métodos siguientes para borrar puntos de interrupción:

- En el editor, coloque el cursor en la línea que contenga el punto de interrupción, pulse *F5* o haga clic con el botón derecho del ratón y seleccione Comutar punto de interrupción.
- En la vista Puntos de interrupción de datos y código, resalte el punto de interrupción que desea eliminar, haga clic con el botón derecho del ratón y seleccione Eliminar punto de interrupción o pulse *Eliminar*.
- Para borrar todos los puntos de interrupción definidos para una sesión de depuración, abra la vista Puntos de interrupción de datos y código. Haga clic con el botón derecho del ratón en una zona vacía de la vista y seleccione Eliminar todos.

- Seleccione un grupo de puntos de interrupción en la vista de Puntos de interrupción de datos y código y pulse Eliminar.

**Advertencia** Los puntos de interrupción borrados no pueden recuperarse.

## Ubicación de los puntos de interrupción de línea

---

Si no encuentra un punto de interrupción de línea en el editor, puede utilizar la vista Puntos de interrupción de datos y código para encontrar rápidamente su ubicación en el código fuente.

Para encontrar un punto de interrupción de línea:

- 1 Seleccione un punto de interrupción de línea en la vista Puntos de interrupción de datos y código.
- 2 Haga clic con el botón derecho y seleccione Ir al punto de interrupción. También puede hacer doble clic sobre el punto de interrupción seleccionado.

El editor muestra la ubicación del punto de interrupción.

## Examen de los valores de datos del programa

---

A pesar de que puede descubrir muchas cosas interesantes acerca del programa ejecutándolo y recorriéndolo paso a paso, normalmente necesitará examinar los valores de las variables del programa para descubrir los errores. Por ejemplo, resulta útil conocer el valor de una variable de índice cuando se recorre un bucle `for`, o los valores de los parámetros que se pasan en una llamada a un método.

Tras detener el programa durante la depuración, puede examinar los valores de las variables de instancia, variables locales, propiedades, parámetros del método y elementos de las matrices.

La evaluación de los datos se realiza dentro de las expresiones. Las expresiones están formadas por constantes, variables y valores en estructuras de datos, que pueden ser combinados con los operadores del lenguaje. De hecho, casi cualquier cosa que se encuentre a la derecha de un operador de asignación puede utilizarse como una expresión de depuración.

JBuilder cuenta con varias funciones que permiten comprobar el estado del programa y se describen en la tabla siguiente.

**Tabla 8.24** Características del depurador

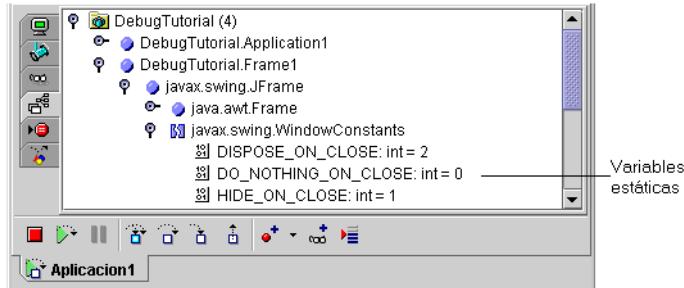
Función	Permite
Vista Clases cargadas y datos estáticos	Ver las clases cargadas actualmente por el programa y los datos estáticos que haya en ellas.
Vista Hilos, pilas de llamada y datos	Ver los grupos de hilos del programa. Los grupos de hilos se expanden para mostrar sus hilos y contienen un seguimiento de marcos de pila representando la secuencia actual de llamadas a métodos. Cada marco de pila puede ampliarse para mostrar los elementos de datos en ámbito.
Vista Puntos de observación de datos	Ver los valores actuales de variables de las que se desea efectuar un seguimiento. Los puntos de observación evalúan una expresión en función del contexto actual. Si se cambia de contexto, la expresión se evalúa otra vez, dentro del nuevo contexto. Si ya no está en ámbito no se podrá evaluar.
cuadro de diálogo Evaluar/Modificar	Evaluación de expresiones, llamadas a métodos y variables.
ExpressionInsight	Visualizar los valores de las expresiones.

## Presentación de las variables en el depurador

Las variables pueden tener diferentes ámbitos, existen variables estáticas o de clase, variables locales y variables de miembros. Una variable puede contener un único valor, como por ejemplo un escalar (un solo número) o varios valores, como las matrices.

### Visualización de variables estáticas

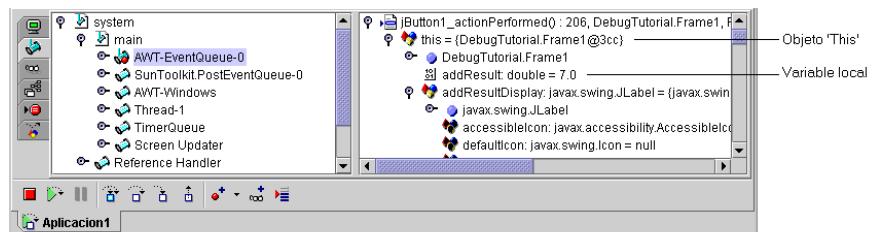
Las variables estáticas se muestran en la vista Clases cargadas y datos estáticos. Cuando se expande el árbol de clases cargadas, se muestran todas las variables estáticas definidas para la clase y sus valores. El menú contextual permite definir puntos de observación de estas variables, cambiar los valores de los tipos de datos originales y cambiar el valor de la base de presentación.

**Figura 8.14** Vista Clases cargadas y datos estáticos

### Formato de visualización de variables locales y de miembros

Las variables locales y de miembro se muestran en la vista Hilos, pila de llamadas y datos. Cuando se amplía un grupo de hilos, se muestra la pila de marcos que representan la secuencia de llamada al método actual. Para mostrar variables de miembro, amplíe el objeto `this` desde el interior de la clase. Cuando se amplía el nodo, se ven todas las variables pertenecientes a esa clase y la instanciaión con la que está trabajando. Los elementos que aparecen en gris son heredados.

A continuación se puede utilizar el menú contextual con el fin de definir puntos de observación para estas variables y, si la variable es una matriz, crear un punto de observación de la matriz y averiguar cuántos elementos se mostrarán en la vista. También se puede crear un punto de observación para el objeto `this`.

**Figura 8.15** Vista Hilos, pilas de llamadas y datos

**Nota** La ventana dividida es una característica de JBuilder SE y Enterprise.

### Cambio de valores de datos

Los valores de datos de las variables se pueden examinar y modificar por medio de las vistas Valores de datos; Hilos, pila de llamadas y datos y Clases cargadas y datos estáticos.

Se puede hacer clic con el botón derecho del ratón y elegir Modificar valor para modificar directamente el valor de una cadena o un tipo de datos

primitivo, como el numérico y el booleano. (Es una función de JBuilder SE y Enterprise.)

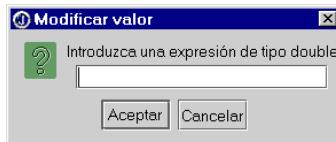
### Modificar los valores de las variables

**Es una función de JBuilder SE y Enterprise.**

Para modificar el valor de una variable:

- 1 Seleccione la variable cuyo valor desea modificar.
- 2 Haga clic con el botón derecho y seleccione Modificar valor.

Se muestra el cuadro de diálogo Modificar valor.



- 3 Introduzca el valor. Su tipo debe coincidir con el del anterior. Las instrucciones del cuadro de diálogo indican el tipo de valor que se espera obtener. Si el nuevo valor es de un tipo distinto, no se cambia. Una constante `String` debe estar rodeada de caracteres " de apertura y cierre, mientras que una constante `char` debe estar rodeada de caracteres ' de apertura y cierre.
- 4 Pulse Aceptar.

Para cambiar la base de presentación de una variable numérica, haga clic con el botón derecho del ratón y seleccione Mostrar valor hexadecimal o Mostrar valor decimal. Si el valor se presenta en formato hexadecimal, el comando permite pasar al formato decimal, y viceversa.

- Nota** En estos cuadros de diálogo se conserva la configuración anterior. Por ejemplo, el contenido del cuadro de diálogo no varía cuando se vuelve a trazar un nodo, se alcanza un punto de interrupción o se pulsa un botón de inspección.

### Modificación del valor de los objetos/primitivos

**Es una función de JBuilder SE y Enterprise.**

Las variables de objeto/primitivo se pueden cortar, copiar y pegar en otros objetos/primitivos por medio de los comandos Cortar, Copiar y Pegar de los menús contextuales. Si un objeto se pega en otra variable, las dos apuntan al mismo objeto. (Los comandos Cortar, Copiar y Pegar son características de JBuilder SE y Enterprise.)

### Modificación de los valores de una matriz de variables

Si desea cambiar el valor de un elemento de matriz, haga clic con el botón derecho del ratón sobre el elemento que desea modificar y elija Modificar. Consulte “Cambio de valores de datos” en la página 8-64 para obtener más información.

También es posible modificar la forma en que se muestra la matriz. Puede ver los detalles de la matriz si la amplía. Sin embargo, es posible que se muestren tantos elementos que se vea obligado a desplazarse dentro de la vista para ver todos los valores. Sin embargo, para verlos con más facilidad, puede reducir o aumentar el número de elementos mostrados con el cuadro de diálogo Ajustar rango. Por defecto, sólo se muestran los primeros 50 elementos de una matriz.

Para reducir el número de elementos de matriz que se muestran:

- 1 Haga clic con el botón derecho del ratón sobre la matriz (el elemento de la vista precedido por ) y elija Ajustar rango de visualización.
- 2 Se muestra el cuadro de diálogo Ajustar rango.



- 3 Introduzca el número de los elementos de matriz que desea ver.
- 4 Pulse Aceptar.

También es posible ocultar o mostrar un valor nulo en una variable matriz. Esto resulta muy útil cuando se depura un objeto con un mapa de direccionamiento calculado. Para activar esta característica, haga clic con el botón derecho del ratón sobre una matriz de tipo `Object` y elija Mostrar/Ocultar valor nulo. (Es una función de JBuilder SE y Enterprise.)

- Nota** En este cuadro de diálogo se conserva la configuración anterior. Por ejemplo, el contenido del cuadro de diálogo no varía cuando se vuelve a trazar un nodo, se alcanza un punto de interrupción o se pulsa un botón de inspección.

## Observación de expresiones

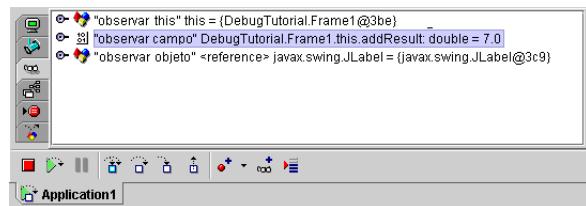
---

Los puntos de observación permiten hacer un seguimiento de los cambios de valor de las variables o las expresiones, durante la ejecución del programa. Al introducir una expresión de punto de observación, la vista Valores de Datos muestra el valor actual de la expresión. Los puntos de observación se evaluarán cuando estén dentro del ámbito.

La expresión de observación que devuelva un objeto o valor de matriz puede ampliarse para mostrar los elementos de datos que estén dentro del ámbito. Por ejemplo, si se define un punto de observación en un objeto `this` o en un único objeto, el punto de observación puede ampliarse. Sin embargo, si se asigna un punto de observación a un valor primitivo, el punto de observación no se puede ampliar debido a que es un elemento

único. Los elementos que aparecen en gris en la vista ampliada son heredados.

**Figura 8.16** Vista Puntos de observación de datos



Es posible definir dos tipos de puntos de observación:

- Puntos de observación de variables
- Puntos de observación de objetos

### Puntos de observación de variables

Existen dos tipos de puntos de observación de variables:

- Puntos de observación de variables con nombre
- Puntos de observación de variables de ámbito

### Puntos de observación de variables con nombre

Los puntos de observación de variables con nombre se añaden a un nombre; cuando se recorre el código, la variable que tenga el nombre seleccionado en el contexto actual será la evaluada por el punto de observación. Si no existe una variable con el nombre que ha seleccionado, el depurador mostrará el siguiente mensaje en la vista Puntos de observación de datos:

nombre de la variable = <está fuera de ámbito>

Para añadir puntos de observación con nombre, utilice el cuadro de diálogo Añadir punto de observación. Para presentar este cuadro de diálogo:

- Seleccione Ejecutar | Añadir punto de observación. Introduzca la expresión que desea observar en el campo Expresión. Si lo desea, puede introducir una descripción en el campo Descripción.
- En el editor, seleccione la expresión que quiere supervisar. Haga clic con el botón derecho y seleccione Añadir punto de observación. La expresión se introduce automáticamente en el cuadro de diálogo Añadir punto de observación. Si lo desea, puede introducir una descripción en el campo Descripción.

Es una función de JBuilder SE y Enterprise.

## Puntos de observación de variables de ámbito

Los puntos de observación de variables de ámbito efectúan un seguimiento de la variable en el contexto en que se han creado. Cuando se recorre el código, sólo se muestra el valor de la variable especificada. Los puntos de observación de variables de ámbito son características de JBuilder SE y Enterprise.

Si la expresión de la variable de ámbito no está en ámbito, el depurador mostrará el siguiente mensaje:

nombre de la variable = <está fuera de ámbito>

Si la expresión está en ámbito, el depurador mostrará su valor.

Para añadir un punto de observación de variables de ámbito, seleccione la variable deseada en la vista Hilos, pilas de llamada y datos y haga clic con el botón derecho del ratón. El menú permite añadir un punto de observación de ámbito para el tipo de variable seleccionado. Estos tipos son:

- Campo - Una variable Java definida en un objeto Java.
- Campo estático - Una variable Java definida como estática (una variable de clase).
- Variable local - Una variable que es local a un método o constructor.
- objeto 'this' - La instanciación de clase con la que se está trabajando.
- Matriz - Una serie de objetos idénticos.
- Componente de matriz - Un elemento de matriz.
- Cadena – Un tipo String Java.

La siguiente tabla muestra la forma en que se ven algunos de estos tipos de puntos de observación en la vista Puntos de observación de datos:

**Tabla 8.25** Tipos de variables de puntos de observación en ámbito

Tipos de observación	Presentación	Descripción
Punto de observación de campo	"addResult"DebugTutorial.Frame1.this.addResult: double=68.0	El campo en punto de observación, addResult, es de tipo primitivo. Se encuentra en DebugTutorial.Frame1. Su valor es 68.0.
Punto de observación de variables locales	"valueOneDouble" valueOneDouble: java.lang.double={java.lang.Double@354}	La variable local en punto de observación es valueOneDouble. Se define como un objeto Double.
Punto de observación de objetos	"DebugTutorial.Frame1" <reference>DebugTutorial.Frame1-{DebugTutorial.Frame1@353}	El objeto en punto de observación es DebugTutorial.Frame1. El objeto se amplía para mostrar los miembros de datos.

**Tabla 8.25** Tipos de variables de puntos de observación en ámbito (continuación)

Tipos de observación	Presentación	Descripción
Punto de observación <code>this</code>	"this" this:{DebugTutorial.Frame1@353}	La instancia actual de <code>DebugTutorial.Frame1</code> . El objeto se amplía para mostrar los miembros de datos de la instancia actual.
Punto de observación de matriz	"valueOneText "<reference>char []=char[2]	La matriz en punto de observación recibe el nombre <code>valueOneText</code> . Contiene dos elementos de matriz.
Punto de observación de componentes de matriz	"[0] = '3'	El primer elemento de la matriz <code>valueOneText</code> . Contiene el valor '3'.

## Puntos de observación de objetos

Es una función de JBuilder SE y Enterprise.

Los puntos de observación de objetos efectúan un seguimiento de un objeto Java determinado.

Para añadir un punto de observación de objetos:

- 1 Seleccione el objeto que desea observar. Puede hacerlo en la vista Puntos de observación de datos, en la vista Hilos, pilas de llamada y datos y en la vista Clases cargadas y datos estáticos.
- 2 Haga clic con el botón derecho del ratón y seleccione Crear punto de observación de objetos.

Un punto de observación del objeto `this` observa la instancia actual del objeto seleccionado.

## Edición de un punto de observación

Para modificar una expresión de observación, selecciónela en la vista Puntos de observación de datos y haga clic con el botón derecho del ratón. Elija Cambiar punto de observación.

- Para cambiar el nombre del punto de observación, escriba el nuevo nombre en el campo Expresión.
- Para cambiar la descripción, escriba el nuevo nombre en el campo Descripción.

## Eliminación de puntos de observación

Para cambiar una expresión de observación, selecciónela en la vista Puntos de observación de datos y seleccione Eliminar punto de observación o pulse *Eliminar*. Si desea borrar todos los puntos de observación, puede hacer clic con el botón derecho del ratón en una zona vacía de la vista Puntos de observación de datos y elegir Eliminar todos.

**Advertencia** El comando Eliminar todos no es reversible.

## Evaluación y modificación de expresiones

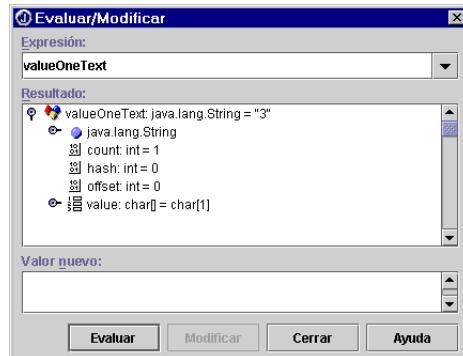
Mediante el cuadro de diálogo Evaluar/Modificar (Ejecutar | Evaluar/Modificar) es posible evaluar expresiones, cambiar los valores de los elementos de datos y evaluar llamadas a métodos. Esto puede resultar útil si cree que ha encontrado la solución a un error y quiere probar la corrección antes de salir del depurador, modificar el código fuente y volver a compilar el programa. CodeInsight y la sintaxis resaltada se muestran cuando se introduce una expresión en el campo correspondiente.

Para abrir el cuadro de diálogo Evaluar/Modificar, seleccione Ejecutar | Evaluar/Modificar.

### Evaluación de expresiones

Para evaluar una expresión, introduzcalo en el campo Expresión. Si ya está seleccionada en el editor, se introduce automáticamente en este campo. Haga clic en el botón Evaluar. Este cuadro de diálogo permite evaluar cualquier expresión válida del lenguaje, excepto las que están fuera del ámbito actual. Si el resultado es un objeto, se muestra el contenido del objeto.

**Figura 8.17** Evaluación de expresiones en el cuadro de diálogo Evaluar/Modificar.

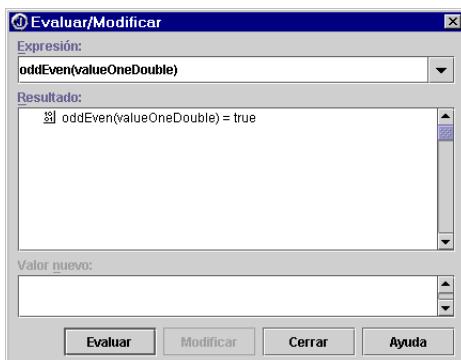


### Evaluación de las llamadas a métodos

Es una función de JBuilder SE y Enterprise.

Los resultados que se obtienen de una llamada a métodos también pueden evaluarse. Para evaluar una llamada a un método, escriba el nombre del método y los parámetros en el campo Expresión del cuadro de diálogo Evaluar/Modificar. Pulse sobre Evaluar.

En este caso, el valor de retorno del método es true.

**Figura 8.18** Evaluación de métodos en el cuadro de diálogo Evaluar/Modificar

## Modificación de los valores de las variables

**Es una función de JBuilder SE y Enterprise.** Es posible cambiar los valores de las variables durante la sesión de depuración con el fin de probar diferentes hipótesis de error y comprobar cómo se comporta una sección de código bajo diferentes circunstancias.

Si cambia el valor de una variable en el Depurador, la modificación sólo tendrá efecto en esta ejecución del programa; los cambios efectuados en el cuadro de diálogo Evaluar/Modificar no afectan al código fuente del programa ni al programa compilado. Si desea que el cambio sea permanente, debe modificar el código fuente del programa en el editor y volver a compilarlo.

Para modificar el valor de una variable, introduzca:

```
variable = <valor nuevo>
```

en el cuadro de edición Expresión. El depurador mostrará los resultados en el cuadro de visualización de resultados Resultado. El resultado debe ser un tipo compatible con la variable.

**Nota** Tanto el campo Expresión como el campo Nuevo valor son compatibles con CodeInsight.

También es posible modificar el valor de una variable siguiendo estos pasos:

- 1 Abra el cuadro de diálogo Evaluar/Modificar e introduzca en el cuadro de edición Expresión el nombre de la variable que desea modificar.
- 2 Haga clic en Evaluar para evaluar la variable.
- 3 Introduzca un valor en el cuadro de edición Valor nuevo (o seleccione un valor en la lista desplegable), y haga clic en Modificar para actualizar la variable.

La expresión del cuadro de entrada Expresión o el valor del cuadro Valor nuevo deben tener un tipo compatible con la variable a la que desea

asignarlos. En general, si la asignación diera lugar a un error durante la compilación o la ejecución, se trata de un valor de modificación no válido.

Por ejemplo, suponga que `valueOneText` es un objeto `String`. Si escribe:

```
valueOneText=34
```

en el campo de entrada Expresión, aparecerá el siguiente mensaje indicando un conflicto de tipos en el campo Resultado:

```
incompatible types; found int; required java.lang.String
```

Y deberá escribir:

```
valueOneDouble="34"
```

en el campo de entrada Expresión para que la expresión quede configurada con el nuevo valor.

## Modificación del código durante la depuración

---

**Es una función de JBuilder Enterprise.**

El depurador permite realizar cambios en el código fuente mientras se depura. También puede actualizar todos los archivos de clase del proyecto o actualizarlos de manera individual.

### Actualización de todos los archivos de clase

---

Cuando los archivos se modifican durante la depuración, el botón Intercambio inteligente se encuentra disponible en la barra de herramientas del depurador. Cuando se pulsa este botón, todos los archivos modificados en el proyecto se compilan y actualizan. Con Intercambio inteligente puede comprobar el código, realizar cambios y seguir depurando en la misma sesión de depuración, desde el punto de ejecución actual.

Para utilizar Intercambio inteligente:

- 1 Abra el código fuente de los archivos que desea modificar.
- 2 Modifique el código fuente.
- 3 Pulse el botón Intercambio inteligente o, bien, seleccione Ejecutar | Intercambio inteligente.

Intercambio inteligente compila todos los archivos modificados del proyecto. Puede continuar la depuración en la misma sesión.

**Importante**

Si la MV de destino del proyecto (Propiedades de proyecto | Generar | Java) está configurada en Todos los SDK de Java, el intercambio inteligente no funciona correctamente. La opción MV de destino de Todos los SDK de Java genera archivos de clase que pueden cargar todas las MV. Sin embargo, este archivo de clase incluirá el código que da instrucciones al cargador de clases para verificar todas las clases que se encuentran

referenciados en esta clase. Si la clase referenciada no se ha cargado todavía, la clase cargada cargará el archivo de clase en su memoria caché. Intercambio inteligente no puede acceder a este caché para actualizarlo. Por consiguiente, si está depurando y realiza un cambio en un archivo de clase que todavía no se ha cargado, el depurador no sabe que se ha llevado a cabo un cambio. Para solucionarlo, añada el siguiente parámetro de la MV al campo Parámetros de la MV de la ficha Ejecutar del cuadro de diálogo Configuraciones de ejecución. (Sólo es necesario que añada este parámetro si selecciona Todos los SDK de Java para la MV de destino.)

`-Xverify:none`

Este parámetro de la MV indica al cargador de clases que verifique sólo la clase actual.

## Actualización de archivos de clase individuales

---

Mientras depura también puede actualizar solo clases individuales del proyecto. Para actualizar un solo archivo de clase durante la depuración:

- 1 Antes de depurar, asegúrese de que la opción Redefinir clases después de compilar de la ficha Depurar del cuadro de diálogo Propiedades de configuración de ejecución se encuentra activada.
- 2 Mientras depura, abra el código fuente del archivo que desea modificar.
- 3 Haga clic con el botón derecho del ratón en el archivo, en el panel del proyecto, y seleccione Ejecutar Make. JBuilder compilará automáticamente el archivo y actualizará las clases. Continuará con la misma sesión de depuración.

**Nota**

Si la opción Redefinir clases después de compilar se encuentra desactivada (ficha Depurar | cuadro de diálogo Propiedades de configuración de ejecución) y la opción Advertir de la modificación de archivos está activada, el cuadro de diálogo Archivos modificados se abrirá y podrá elegir cómo proceder. Se puede:

- Compilar, actualizar las clases compiladas y continuar con la sesión de depuración
- Continuar con la sesión de depuración sin compilar ni actualizar
- Reiniciar la sesión de depuración

## Restablecimiento del punto de ejecución

---

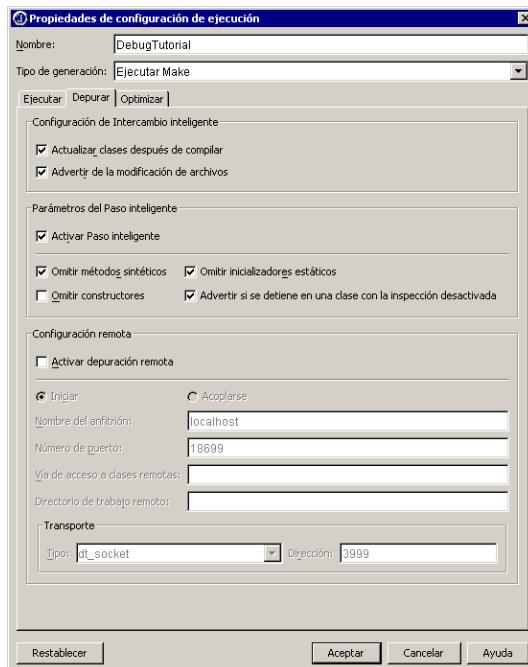
Una vez que ha modificado el código, puede restablecer el punto de ejecución (todavía seguirá en la misma sesión de depuración). El restablecimiento del punto de ejecución permite volver al punto anterior al valor cambiado, para que pueda volver a comprobarlo y ver si los

ajustes funcionan. Para obtener más información, consulte “[Definición del punto de ejecución](#)” en la página 8-33.

## Opciones para modificar el código

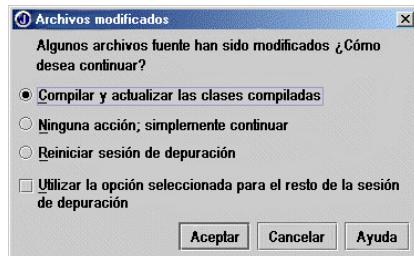
Las opciones en la parte superior de la ficha Depurar del cuadro de diálogo Propiedades de configuración de ejecución controlan cómo los gestionan los archivos modificados durante una sesión de depuración.

**Figura 8.19** Ficha Depurar del cuadro de diálogo Propiedades de configuración de ejecución



- La opción Actualizar clases después de compilar actualiza automáticamente todos los archivos modificados que se han compilado. Cuando esta opción está activa, no se le advertirá de que ha modificado el código fuente si compila los archivos modificados. Si esta opción no está activada y sí lo está la opción Advertir de la modificación de archivos, aparece el cuadro de diálogo Archivos modificados, y le permite decidir cómo continuar.
- La opción Advertir de la modificación de archivos abre el cuadro de diálogo Archivos modificados en el que selecciona cómo continuar. El

cuadro de diálogo Archivos modificados tendrá un aspecto parecido a éste:



Puede elegir:

- Actualizar los archivos y continuar en la misma sesión de depuración,
- Continuar la depuración sin actualizar los archivos, o bien,
- Iniciar una nueva sesión de depuración.

Si inicia una nueva sesión, se detiene la sesión actual y se restablece el punto de ejecución al principio del programa. Si continúa con la sesión actual, se comienza la ejecución en el punto de ejecución actual.

## Personalización del depurador

---

Puede personalizar los colores utilizados para indicar el punto de ejecución y activar, desactivar y convertir en no válidas las líneas de los puntos de interrupción.

## Personalización de la presentación del depurador

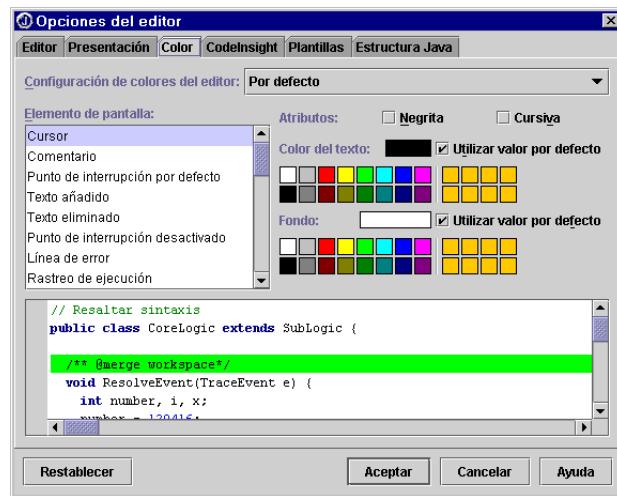
---

Para definir los colores de los puntos de interrupción y los puntos de ejecución:

**1** Seleccione Herramientas | Opciones del editor

Aparece el cuadro de diálogo Opciones del editor.

**2 Selecione la pestaña Color para mostrar la ficha Color.**



**3 En la lista Elemento de pantalla, seleccione un elemento relacionado con la depuración y, a continuación, los colores de fondo y de texto del elemento.**

Los elementos de pantalla que participan en la depuración son:

- Punto de interrupción por defecto
- Punto de interrupción desactivado
- Rastreo de ejecución
- Punto de ejecución
- Punto de interrupción no válido
- Punto de interrupción comprobado

## Configuración de las opciones de depuración

Las configuraciones para la depuración múltiple es una función de JBuilder SE y Enterprise

Las configuraciones de depuración se pueden crear de forma autónoma o como parte de la configuración de ejecución. Para obtener más información sobre las configuraciones de ejecución, consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7.

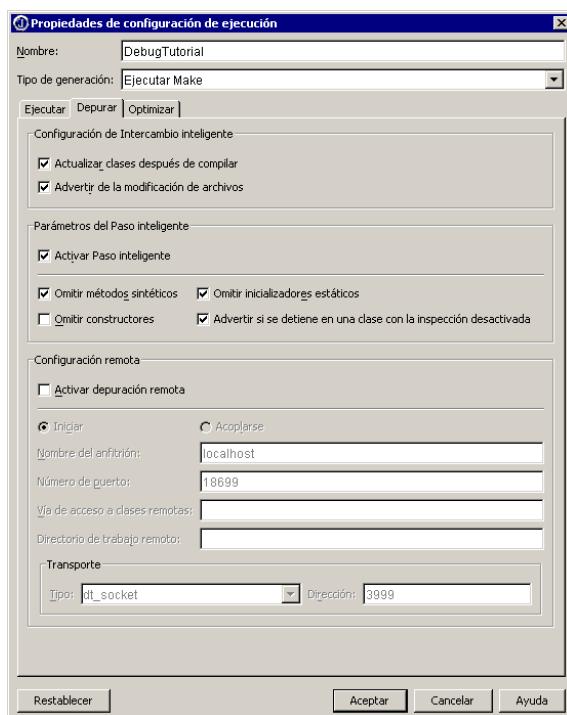
Para establecer las opciones de una configuración de depuración:

**1 Seleccione Ejecutar | Configuraciones.**

A continuación se muestra la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto.

**2 Elija la configuración deseada y pulse Modificar.**

- 3 En el cuadro de diálogo Propiedades de configuración de ejecución, abra la pestaña Depurar.



- 4 Para configurar cómo el depurador gestiona los archivos modificados, configure las siguientes opciones.

- Actualizar clases después de compilar

Actualiza automáticamente todos los archivos modificados cuando se compilan. No será advertido de que ha modificado el código fuente si se ha compilado. Si esta opción no está activada y sí lo está la opción Advertir de la modificación de archivos, aparece el cuadro de diálogo Archivos modificados, y le permite decidir cómo continuar. (Ésta es una función de JBuilder Enterprise.)

- Advertir de la modificación de archivos

Muestra el cuadro de diálogo Archivos modificados donde elige cómo continuar. Puede actualizar los archivos y continuar con la sesión de depuración actual reanudar la sesión de depuración si actualizar los archivos o iniciar una nueva sesión de depuración.

- 5 Para activar Paso inteligente, elija la opción Activar Paso inteligente o pulse el botón Paso inteligente de la barra de herramientas del depurador. Paso inteligente se encuentra activado por defecto.



- 6** Para configurar la activación y desactivación de Paso inteligente, defina las siguientes opciones: (La configuración de Paso inteligente es una función de JBuilder SE y Enterprise.)
- **Omitir métodos sintéticos**  
Omite métodos sintéticos cuando inspecciona las clases.
  - **Omitir constructores**  
Omite constructores al inspeccionar las clases.
  - **Omitir inicializadores estáticos**  
Omite inicializadores estáticos (clase) al inspeccionar las clases.
  - **Advertir si se detiene en una clase con la inspección desactivada**  
Muestra un mensaje de advertencia si hay un punto de interrupción en una clase que tenga la inspección desactivada. Consulte “[Puntos de interrupción y configuración de Inspección desactivada](#)” en la [página 8-45](#) para obtener más información.

Para obtener más información sobre las opciones de depuración remota, consulte [Capítulo 9, “Depuración remota”](#). (La depuración remota es una función de JBuilder Enterprise.)

## Definición de los intervalos de actualización

---

También es posible especificar la frecuencia de los intervalos que controlan cuándo se registran los cambios de estado de la consola/ proceso. Si los intervalos son pequeños, las respuestas del depurador/de ejecución de la salida y otros sucesos, como la ejecución paso a paso, serán más rápidos, pero JBuilder utilizará la mayor parte del tiempo del procesador.

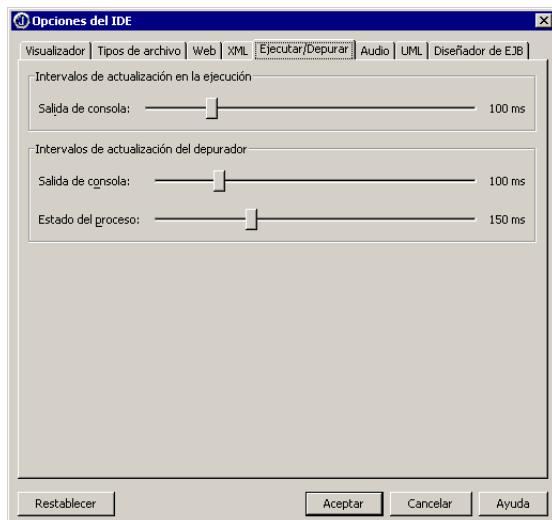
Por lo general, se pueden definir estas propiedades muy pequeñas, a menos que se estén ejecutando otras aplicaciones además de JBuilder, o que el programa que se esté depurando requiera mucho tiempo del procesador. Si es este el caso, debe alargar los intervalos.

Para cambiar los intervalos de actualización:

- 1** Seleccione Herramientas | Opciones del IDE.

Aparece el cuadro de diálogo Opciones del IDE.

- 2 Seleccione la pestaña Ejecutar/Depurar para mostrar la ficha homónima.



- 3 Para definir el intervalo de salida de consola de los procesos de ejecución, mueva la barra de desplazamiento de la Salida de consola en la parte superior del cuadro de diálogo.
- 4 Para definir el intervalo de salida de consola de los procesos de depuración, mueva la barra de desplazamiento de la Salida de consola en el centro del cuadro de diálogo.
- 5 Para definir el intervalo de las actualizaciones de estado del proceso de depuración, mueva la barra de desplazamiento de Estado del proceso.



# Depuración remota

Es una función de JBuilder Enterprise.

JBuilder incluye varias características que facilitan la depuración de aplicaciones distribuidas. En concreto, incluye apoyo para la depuración interprocesal y la depuración remota.

Estas características son complementarias a las funciones básicas de depuración de JBuilder. Si es la primera vez que utiliza JBuilder, en el [Capítulo 8, “Depuración de programas en Java”](#) encontrará información sobre el entorno de depuración de JBuilder.

La depuración remota consiste en depurar desde un ordenador el código que se ejecuta en otro. Esta posibilidad resulta ideal, por ejemplo, en los casos en que una aplicación experimenta un problema que sólo se produce en uno de los ordenadores de la red, y no en los demás. El depurador remoto de JBuilder también permite depurar entre distintos sistemas operativos.

En este capítulo, el “ordenador cliente” es el ordenador que está ejecutando JBuilder. Éste es el ordenador desde el que se efectúa la depuración. El “ordenador remoto” ejecuta la aplicación que se desea depurar.

Existen dos formas de efectuar la depuración remota. Se puede:

- Abrir un programa de un ordenador remoto desde un ordenador cliente y depurarlo con JBuilder. Para obtener más información, consulte [“Apertura y depuración de programas en un equipo remoto” en la página 9-2](#). En este caso se ejecuta el servidor de depuración remoto de JBuilder.
- Depurar por medio de JBuilder un programa que ya se está ejecutando en el ordenador remoto. Para obtener más información, consulte [“Depuración de un programa que se ejecuta en un ordenador remoto”](#)

[en la página 9-6](#). En este caso no es necesario ejecutar el servidor de depuración.

- Nota** Los ordenadores cliente y remoto deben tener instalado JDK 1.2 o superior (una versión de JDK que acepte el API de depuración JPDA). No es necesario que coincidan las versiones de JDK de los dos ordenadores. Durante la instalación de JBuilder, JDK 1.4 se instala automáticamente en el directorio `<jbuilder>/jdk1.4`.

También se puede depurar el código que se ejecuta en un proceso independiente, en el mismo ordenador en el que está instalado JBuilder. Para ello, inicie el proceso en el modo de depuración y cree un enlace con JBuilder. Para obtener más información, consulte “[Depuración del código local que se ejecuta en un proceso independiente](#)” en la página 9-10.

Además, es posible establecer puntos de interrupción interprocesales, ideales para la depuración de aplicaciones cliente/servidor. Para obtener más información, consulte “[Depuración con puntos de interrupción interprocesales](#)” en la página 9-10.

Para realizar depuraciones remotas de cualquier tipo es necesario utilizar configuraciones de depuración. Para obtener más información sobre las configuraciones de ejecución y depuración consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7 y “[Configuración de las opciones de depuración](#)” en la página 8-76.

## Apertura y depuración de programas en un equipo remoto

---

En este apartado se explica la forma de abrir un programa de un ordenador remoto desde un ordenador cliente, depurándolo desde aquí con JBuilder. Los pasos son los siguientes:

- 1 Instalar el servidor de depuración en el equipo remoto y ejecutarlo.
- 2 Compilar la aplicación en el ordenador remoto o copiar en él los archivos `.class`.
- 3 Utilizar JBuilder en el ordenador cliente para abrir y depurar la aplicación en el ordenador remoto.

- Importante** Los archivos de código fuente de la aplicación que se está depurando deben estar disponibles en el ordenador cliente. Los archivos `.class` compilados deben estar a disposición del ordenador remoto. Estos archivos deben coincidir. De lo contrario, podrían producirse resultados imprevisibles, por ejemplo errores, o que el depurador se detenga en una línea de código fuente incorrecta. Siempre que se modifica el código fuente se deben actualizar los archivos `.class` del ordenador remoto.

En primer lugar, instale el servidor de depuración en el equipo remoto y ejecútelo. Si JBuilder ya se encuentra instalado en el ordenador remoto, puede empezar por el cuarto paso.

- 1** Copie el archivo debugserver.jar (situado en el directorio <jbuilder>/remote) en el ordenador remoto. Anote el directorio donde lo ha copiado, pues lo necesitará en etapas posteriores.
- 2** Copie el script de shell del servidor de depuración, DebugServer (Unix) o el archivo por lotes, DebugServer.bat (Windows), en el mismo directorio del equipo remoto.
- 3** Compruebe que JDK 1.2.2 o superior se encuentra instalado en el ordenador remoto.
- 4** Vaya al directorio del ordenador remoto donde se han instalado los archivos del servidor de depuración. Ejecute DebugServer con el fin de personalizar las variables de entorno del servidor de depuración remoto.

En los sistemas Unix, utilice el siguiente comando:

```
./DebugServer <debugserver.jar_dir> <jdk_home_dir> [-port portnumber]
-[timeout milliseconds]
```

En el sistema Windows, utilice:

```
DebugServer <debugserver.jar_dir> <jdk_home_dir> [-port portnumber]
-[timeout milliseconds]
```

donde:

- debugserver.jar\_dir - Directorio del ordenador remoto donde se encuentra el archivo JAR del servidor de depuración. En los sistemas que utilizan Windows es necesario introducir la letra de la unidad.
- jdk\_home\_dir - El directorio raíz del ordenador remoto para la instalación de JDK. En los sistemas que utilizan Windows es necesario introducir la letra de la unidad.
- -port - Parámetro optativo que ejecuta el servidor de depuración en un puerto distinto del predeterminado, el 18699. Este valor sólo se debe modificar si se está utilizando el valor por defecto. Los números de puerto válidos están comprendidos entre el 1025 y el 65535. Este valor debe coincidir con el introducido en el campo Número de puerto en la ficha Depurar del cuadro de diálogo Propiedades de configuración de ejecución (en el ordenador cliente). Consulte el paso 6, más abajo.
- -timeout - Parámetro optativo que establece el número de milésimas de segundo durante las cuales se intenta conectar el ordenador remoto al cliente. El proceso se detiene cuando se alcanza este número. El valor por defecto es de 60.000 milésimas de segundo.

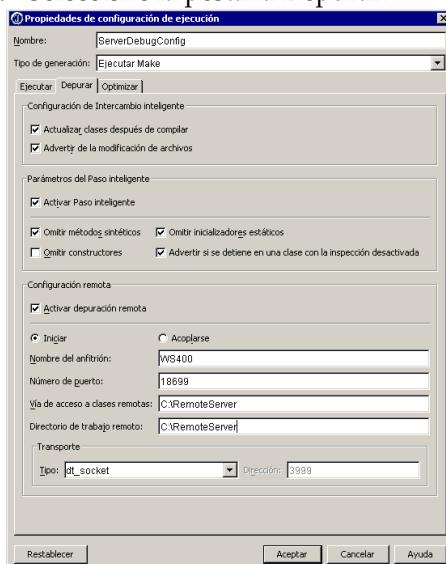
Ejemplo de este comando en un entorno Windows:

```
DebugServer d:\remote d:\jdk1.3 -port 1234 -timeout 20000
```

- 5** Pulse *Intro* para iniciar el servidor de depuración.

Las funciones de depuración remota de JBuilder en modo de lanzamiento estarán activas si el servidor de depuración se está ejecutando. Cuando se está ejecutando el servidor de depuración en el ordenador remoto es necesario compilar la aplicación y copiar en este ordenador los archivos .class. (También se puede compilar la aplicación de forma remota.) Después se utiliza JBuilder, que se ejecuta en el ordenador cliente, para abrir y depurar el programa del ordenador remoto.

- 1** Compile la aplicación. La aplicación se puede compilar utilizando JBuilder en el ordenador cliente, para después copiar los archivos .class en el ordenador remoto o transferirlos por FTP. También se puede compilar la aplicación directamente en el ordenador remoto llamando al compilador javac con la opción -g. (Esto le dice al compilador que tiene que añadir información de depuración al archivo compilado.)
- 2** Abra JBuilder en el ordenador cliente.
- 3** Abra el proyecto que va a depurar la aplicación.
- 4** Cree una configuración de depuración (puede formar parte de la configuración de ejecución o ser independiente):
  - a** Seleccione Ejecutar | Configuraciones. Para crear una nueva configuración, pulse el botón Nueva. Para modificar una configuración, selecciónela y pulse Modificar.
  - b** Si la configuración es nueva, escriba su nombre en el campo correspondiente.
  - c** Asigne el valor Ninguno a Tipo de generación.
  - d** Seleccione la pestaña Depurar.



- 5 Active la casilla de selección Activar depuración remota. Elija la opción Iniciar.
- 6 Rellene los siguientes campos:
  - Nombre del anfitrión - El nombre del ordenador remoto. El nombre por defecto es localhost. Para averiguar el nombre del anfitrión compruebe la configuración de red en el ordenador remoto.
  - Número de puerto - El número de puerto del ordenador remoto con el que el usuario se está comunicando. Utilice el número por defecto, 18699. Cámbielo únicamente si ya está en uso. Los valores válidos van del 1024 al 65535. Este valor debe coincidir con el parámetro -port utilizado para iniciar el servidor de depuración en el ordenador remoto. Consulte el cuarto paso del apartado anterior.
  - Vía de acceso a clases remotas - La vía de acceso a clases donde se encuentran los archivos .class compilados, de la aplicación que se está depurando, en el ordenador remoto. Este campo tiene el mismo funcionamiento que los campos de vía de acceso a clases: si las clases se encuentran en un paquete, se debe especificar el directorio raíz y no el subdirectorio que contiene las clases. En los sistemas Windows, especifique la unidad si ésta no es C:. Esta vía de acceso a clases remota sólo es aplicable a esta sesión de depuración concreta.
  - Directorio de trabajo remoto - El directorio de trabajo en el ordenador remoto. En los sistemas Windows, especifique la unidad si ésta no es C:. Este directorio de trabajo remoto sólo es aplicable a esta sesión de depuración concreta.

**Advertencia**

JDK 1.2.2 no acepta el directorio de trabajo. Si el ordenador remoto ejecuta JDK 1.2.2 y se introduce un directorio de trabajo remoto, en la vista de consola de salida, entrada y errores del depurador se mostrará una advertencia.

- Transporte - El tipo de transporte: dt\_shmem (transporte de memoria compartida, no disponible en los sistemas Unix) o dt\_socket (transporte de socket). Si desea más información sobre los métodos de transporte, consulte “JPDA: Connection and Invocation Details - Transports” en <http://java.sun.com/products/jpda/doc/conninv.html#Transports>.
- 7 Seleccione Aceptar dos veces para cerrar los cuadros de diálogo Propiedades de configuración de ejecución y Propiedades de proyecto.

- 8** Para dar comienzo a la sesión de depuración, elija una de las siguientes opciones:

Comando	Atajo de teclado	Descripción
Ejecutar   Depurar proyecto	Mayús + F9	Inicia el programa en el depurador utilizando la configuración por defecto o la seleccionada. Ejecuta el programa hasta que se completa o suspende la ejecución en la primera línea de código en la que se requiere que el usuario introduzca un dato o en un punto de interrupción.
Ejecutar   Omitir inspección	F8	Interrumpe la ejecución en la primera línea de código ejecutable.
Ejecutar   Inspeccionar	F7	Interrumpe la ejecución en la primera línea de código ejecutable.

Cuando se inicia el depurador, la aplicación que se desea depurar (según lo elegido en Classpath remota) se abre en el ordenador remoto. En JBuilder se muestra el depurador que se ejecuta en el ordenador cliente; no obstante, se están depurando los archivos .class que se ejecutan en el ordenador remoto.

**Nota**

Si esa aplicación ya se está ejecutando, el servidor de depuración lanzará una nueva instancia. (Si desea depurar una aplicación que ya se está ejecutando, consulte [“Depuración de un programa que se ejecuta en un ordenador remoto” en la página 9-6](#).)

- 9** Para cerrar la aplicación en el ordenador remoto, detenga el proceso en JBuilder. Para cerrar el servidor de depuración en el ordenador remoto, elija el comando Archivo | Salir.

## Depuración de un programa que se ejecuta en un ordenador remoto

En este apartado se explica la forma de conectarse a un programa que ya se está ejecutando en un ordenador remoto desde un ordenador cliente y depurarlo con JBuilder. Para ello es necesario:

- 1 Ejecutar la aplicación en el ordenador remoto utilizando las opciones de depuración de la máquina virtual (MV).
- 2 Pasar a y depurar la aplicación que se está ejecutando en el ordenador cliente por medio de JBuilder.

**Importante**

Los archivos de código fuente de la aplicación que se está depurando deben estar disponibles en el ordenador cliente. Los archivos .class compilados deben estar a disposición del ordenador remoto. Estos archivos deben coincidir. De lo contrario, podrían producirse resultados

imprevisibles, por ejemplo errores, o que el depurador se detenga en una línea de código fuente incorrecta. Siempre que se modifica el código fuente se deben actualizar los archivos .class del ordenador remoto.

Si desea seguir un tutorial que explica paso a paso la forma de enlazarse a un programa en ejecución, consulte [Capítulo 19, “Tutorial: Depuración remota”](#).

Para iniciar un programa en el ordenador remoto y enlazarse a él:

- 1** Compile la aplicación en el ordenador remoto. La aplicación también se puede compilar con JBuilder en el ordenador cliente, para después copiar los archivos .class en el ordenador remoto o transferirlos por FTP.
- 2** Ejecute la aplicación en el ordenador remoto con las siguientes opciones de MV.
  - Si JBuilder está instalado en el ordenador remoto, el programa se puede ejecutar desde JBuilder. Abra el proyecto y modifique la configuración de ejecución (Ejecutar | Configuraciones | Edición). Introduzca los siguientes parámetros en el campo Parámetros de la MV:

```
-Xdebug -Xnoagent -Djava.compiler=NONE -
Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=y
```

- Si JBuilder no está instalado en el ordenador remoto, el programa se debe ejecutar desde la línea de comandos. Añada las siguientes opciones de MV para la línea de comandos de Java:

```
-Xdebug -Xnoagent -Djava.compiler=NONE -
Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=y
```

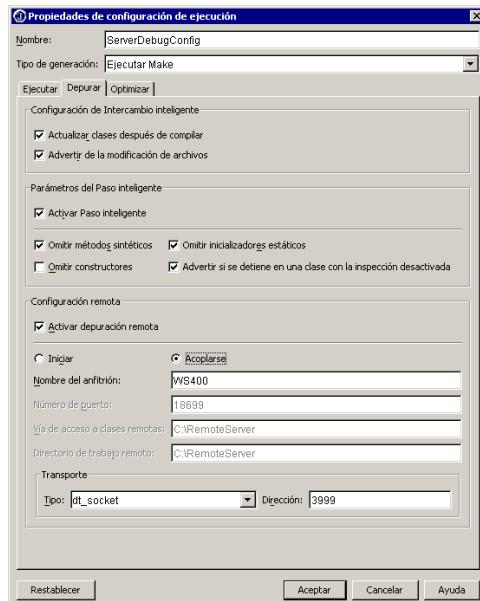
Los parámetros address y suspend son optativos. Se encuentran después del parámetro server, separados por una coma. No se pueden colocar espacios entre los parámetros.

- El parámetro address, basado en el transporte seleccionado, contiene el número y la dirección del puerto a través del cual se comunica el depurador con el ordenador remoto. Este parámetro simplifica la configuración, ya que elimina la necesidad de modificar continuamente el campo Dirección de la ficha Depurar del cuadro de diálogo Propiedades de ejecución. Si al tipo de transporte se le asigna el valor dt\_socket, el parámetro address contiene el número de puerto. Si se define como dt\_shmem, este parámetro contiene el nombre de dirección irrepetible. Si está utilizando XP, no utilice 5000 para el parámetro address. Esta dirección está reservada para Universal Plug & Play.
- El parámetro suspend indica si el programa se suspende inmediatamente cuando se inicia. Esta configuración se puede desactivar por medio de suspend=n. (Si se ha definido suspend=n y

no hay puntos de interrupción, cuando se inicia el programa se ejecuta hasta el final sin detenerse.)

**Nota** Para ejecutar la aplicación con JDK 1.2x o 1.3x utilice el ejecutable java del subdirectorio bin del directorio de instalación de JDK, no el archivo java.exe del directorio jre/bin. Esto permite a la MV de Java cargar el archivo de depurador (libjdw.so en Unix; jdwp.dll en Windows), necesario para la depuración. (No es aplicable a JDK 1.4x.)

- 3** Abra JBuilder en el ordenador cliente.
- 4** Abra el proyecto de la aplicación que se está ejecutando en el ordenador remoto.
- 5** Cree una configuración de depuración (puede formar parte de la configuración de ejecución o ser independiente):
  - a** Seleccione Ejecutar | Configuraciones. Para crear una nueva configuración, pulse el botón Nueva. Para modificar una configuración, selecciónela y pulse Modificar.
  - b** Si la configuración es nueva, escriba su nombre en el campo correspondiente.
  - c** Asigne el valor Ninguno a Tipo de generación.
  - d** Seleccione la pestaña Depurar.



- 6** Active la casilla de selección Activar depuración remota. Seleccione la opción Acoplarse.

**7 Rellene los siguientes campos:**

- Nombre del anfitrión - El nombre del ordenador remoto. El nombre por defecto es localhost. Para averiguar el nombre del anfitrión compruebe la configuración de red en el ordenador remoto.
- Transporte - Opciones del método de transporte:
  - Tipo: dt\_socket (transporte de socket) o dt\_shmem (transporte de memoria compartida, no disponible en los sistemas Unix). Si desea más información sobre los métodos de transporte, consulte “JPDA: Connection and Invocation Details - Transports” en <http://java.sun.com/products/jpda/doc/conninv.html#Transports>.
  - Dirección:
    - Si el tipo de transporte se define como dt\_socket, el parámetro contiene el número de puerto del ordenador remoto con el que se ha establecido la comunicación. Utilice el número por defecto, 3999. Cámbielo únicamente si ya está en uso. Este valor debe coincidir con el valor del parámetro address de la MV de Java que inicia el programa en el ordenador remoto. Consulte el segundo paso de este apartado.

**Importante**

Si está ejecutando Windows XP, no utilice 5000 como el número de puerto o dirección mediante el que el depurador se comunica con un ordenador remoto. XP reserva este número de puerto para el Universal Plug & Play.

- Si se selecciona dt\_shmem como tipo de transporte, asigne al parámetro address el nombre único del ordenador remoto con el que se ha establecido la comunicación. El valor por defecto es javadebug.

**8** Seleccione Aceptar dos veces para cerrar los cuadros de diálogo Propiedades de configuración de ejecución y Propiedades de proyecto.

**9** Elija Ejecutar | Omitir inspección o Ejecutar | Inspeccionar para iniciar el depurador.

 **10** Si el parámetro suspend de la MV del ordenador remoto está definido en y (consulte el segundo paso de este apartado), pulse el botón Reanudar el programa de la barra de herramientas Depurador para continuar con la depuración.

**11** Para poner fin a la ejecución de la aplicación, ciérrela en el ordenador remoto.

**12** Para desconectarse del ordenador remoto, detenga el proceso en JBuilder.

**Nota** Para ejecutar y depurar una aplicación en el ordenador remoto, consulte el apartado “[Apertura y depuración de programas en un equipo remoto](#)” en la página 9-2.

## Depuración del código local que se ejecuta en un proceso independiente

---

Para depurar el código que se ejecuta en un proceso independiente, en el mismo ordenador en el que está instalado JBuilder, siga las instrucciones anteriores, desde el segundo paso. Utilice los siguientes valores para las opciones de Conectar de la ficha Depurar del cuadro de diálogo Propiedades de configuración de ejecución.

- Nombre del anfitrión  
Configure la opción por defecto, localhost.
- Tipo de transporte  
dt\_socket (transporte de socket) o dt\_shmem (transporte de memoria compartida, no disponible en los sistemas Unix).
- Dirección de transporte  
Si el tipo de transporte es dt\_socket, asigne el valor 3999. Si es dt\_shmem, cámbielo a javadebug.

## Depuración con puntos de interrupción interprocesales

---

Un punto de interrupción interprocesal detiene el depurador cuando se inspecciona, en un proceso aparte, un método o el método especificado en la clase especificada. Esto permite inspeccionar un proceso servidor desde un proceso cliente, en lugar de tener que configurar puntos de interrupción en el cliente y en el servidor. Por lo general se configura un punto de interrupción de línea en el cliente y un punto de interrupción interprocesal en el servidor. Si desea seguir un tutorial que explica paso a paso la inspección interprocesal, consulte [Capítulo 19, “Tutorial: Depuración remota”](#).

Para activar un punto de interrupción interprocesal configurado en un proceso de servidor,

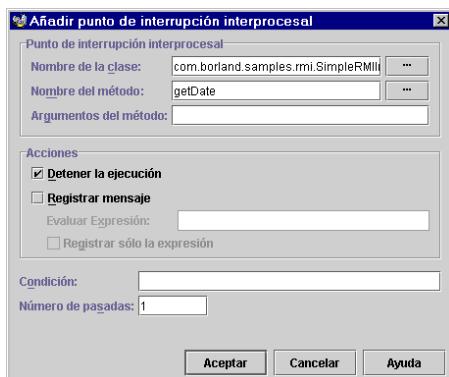
- 1 Inicie el proceso de servidor en el ordenador remoto en modo depuración. Consulte el Paso 2 de [“Depuración de un programa que se ejecuta en un ordenador remoto”](#) en la página 9-6.
- 2 En el ordenador cliente y desde JBuilder, conéctese con el servidor que ya está en ejecución en el ordenador remoto. Consulte los Pasos 4 – 10 de [“Depuración de un programa que se ejecuta en un ordenador remoto”](#) en la página 9-6.
- 3 Configure un punto de interrupción en el código del cliente y comience a depurar el cliente. Cuando llegue al punto de interrupción, inspeccione el código del servidor. No utilice Omitir inspección. La

omisión de inspección no parará en el punto de interrupción interprocesal.

Para configurar un punto de interrupción interprocesal, utilice el cuadro de diálogo Añadir punto de interrupción interprocesal. Para abrir el cuadro de diálogo Añadir punto de interrupción interprocesal, haga una de las siguientes cosas:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar | Añadir punto de interrupción interprocesal y seleccione Añadir punto de interrupción interprocesal.
- Durante la sesión de depuración, pulse la flecha de lista desplegable que se encuentra a la derecha del botón Añadir punto de interrupción () de la barra de herramientas del depurador y seleccione Añadir punto de interrupción interprocesal.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista Puntos de interrupción de datos y de código y seleccione Añadir punto de interrupción interprocesal.

Aparece el cuadro de diálogo Añadir punto de interrupción interprocesal.



Para configurar un punto de interrupción interprocesal:

- 1 En el campo Nombre de la clase, escriba el nombre de la clase del lado del servidor que contiene el método en el que desea se detenga el depurador. Pulse el botón de puntos suspensivos para buscar la clase.
- 2 En el campo Nombre del método, escriba el nombre del método en el que desea se detenga el depurador. Utilice el botón de puntos suspensivos para mostrar el cuadro de diálogo Seleccione un método, en el cual se puede acceder a la lista de métodos disponibles en la clase seleccionada. No es necesario escribir el nombre del método. Si no se especifica un nombre de método, el depurador se detiene en todas las llamadas a métodos en la clase especificada.

**Nota** No se puede seleccionar un método si la clase seleccionada contiene errores de sintaxis o de compilación.

- 3 En el campo Argumentos del método, introduzca una lista de argumentos del método separados por comas. El depurador se parará cuando el nombre del método y la lista de argumentos coincidan. Esto resulta útil con los métodos sobrecargados.
  - Si no se establece ningún argumento, el depurador se detiene en todos los métodos cuyo nombre coincide con el especificado.
  - Si se elige un nombre de método del cuadro de diálogo Seleccione un método, el campo Argumentos del método se rellena automáticamente.
- 4 Elija qué acciones realizará el depurador. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Para obtener más información, consulte “[Definición de las acciones de puntos de interrupción](#)” en la página 8-57.
- 5 En el campo Condición, si existe alguna, asígnela al punto de interrupción. Para obtener más información, consulte “[Creación de puntos de interrupción condicionales](#)” en la página 8-59.
- 6 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Para obtener más información, consulte “[Utilización de puntos de interrupción por número de pasadas](#)” en la página 8-60.
- 7 Pulse Aceptar para cerrar el cuadro de diálogo.
- 8 Fije un punto de interrupción de línea en el cliente, en el método que llama al punto de interrupción interprocesal.
- 9 Cuando se detenga en el punto de interrupción de línea, pulse el botón Inspeccionar de la barra de herramientas Depurador para inspeccionar el método correspondiente del servidor. (Si utiliza Omitir inspección, el depurador no se detendrá.)



# 10

## Creación de JavaBeans con BeansExpress

Es una función de JBuilder SE y Enterprise.

BeansExpress es la forma más rápida de crear JavaBeans. Se compone de un conjunto de asistentes, diseñadores visuales y ejemplos de código que permiten desarrollar JavaBeans rápida y fácilmente. BeansExpress también permite modificar JavaBeans ya existentes. Asimismo, es posible convertir clases de Java en JavaBeans.

### Definición de JavaBean

Los JavaBeans son colecciones de una o varias clases de Java, por lo general agrupadas en un archivo JAR (recopilatorio Java), que actúan como componentes autónomos y reutilizables. Los JavaBeans pueden ser componentes de interfaz de usuario o componentes no visuales, como módulos de datos o motores de cálculo.

En su forma más simple, los JavaBeans son clases `public` de Java que contienen un constructor sin parámetros. Normalmente, los JavaBeans disponen de propiedades, métodos y sucesos que respetan determinadas convenciones de nomenclatura (también denominadas pautas de diseño).

## ¿Por qué desarrollar JavaBeans?

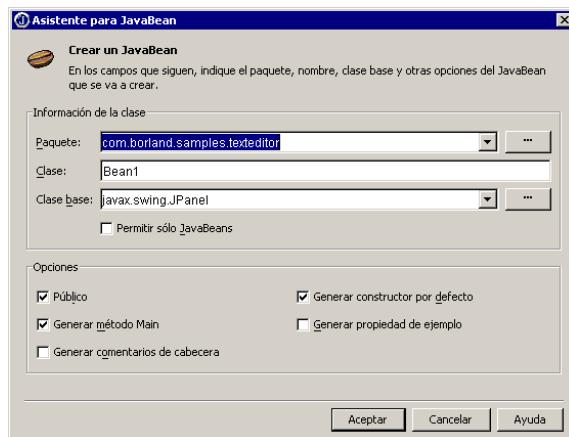
Al igual que otros tipos de componentes, los JavaBeans son fragmentos de código reutilizable que pueden actualizarse con una repercusión mínima en el proceso de prueba de los programas a los que se asocian. No obstante, los JavaBeans presentan ciertas ventajas sobre otros componentes:

- Se trata de componentes multiplataforma desarrollados íntegramente en Java.
- Es posible instalarlos en la paleta de componentes de JBuilder o en otras herramientas de desarrollo en Java y emplearlos en la construcción de programas.
- Pueden distribuirse en archivos .JAR.

## Generación de clases bean

Para iniciar la creación de un JavaBean mediante BeansExpress,

- 1 Seleccione Archivo | Nuevo proyecto y cree un proyecto mediante el Asistente para proyectos.
- 2 Para mostrar la galería de objetos, seleccione Archivo | Nuevo.
- 3 Seleccione la pestaña General y haga doble clic en el icono JavaBean para que se abra el Asistente para JavaBean.



- 4 En el primer campo de texto, indique el paquete al que ha de pertenecer el bean. Por defecto, se toma el nombre del proyecto actual.
- 5 En el segundo campo de texto, asigne un nombre al bean.
- 6 Seleccione la clase que desee ampliar, en el campo Clase base.

Puede optar por emplear la lista desplegable o hacer clic en el botón contiguo para acceder al visualizador de paquetes y especificar en él la clase de Java que deseé.

- 7 Elija las restantes opciones según sus preferencias. Ninguna de ellas es obligatoria:
  - a Marque la opción Permitir sólo JavaBeans si desea que JBuilder le avise cuando trate de ampliar una clase de Java que no pueda ser un JavaBean válido.
  - b Seleccione Público si desea que la clase sea `public`.
  - c Marque Generar método Main si desea que JBuilder coloque el método `main` en el bean, lo que permite ejecutarlo.
  - d Compruebe Generar comentarios de cabecera si desea comentarios de cabecera JavaDoc (Título, Descripción, Autor, etc.) añadido en la parte superior del archivo clase.
  - e Active la opción Generar constructor por defecto si desea que JBuilder cree un constructor sin parámetros.
  - f Marque la opción Generar propiedad de ejemplo si desea que JBuilder añada una propiedad `sample` al bean. Esta opción puede interesarle la primera vez que desarrolle un bean, con el fin de ver cómo JBuilder genera el código de una propiedad. Posteriormente, podrá eliminar esta propiedad o modificarla de modo que tenga alguna utilidad en el bean.
- 8 Pulse Aceptar para cerrar el Asistente para JavaBean.

JBuilder crea un JavaBean con el nombre especificado, lo añade al proyecto actual y muestra el código fuente que ha generado. Este es el código generado por JBuilder para los valores mostrados:

```
package myjavabean;

import java.awt.*;
import javax.swing.*;

public class BeanieBaby extends JPanel {
    BorderLayout borderLayout1 = new BorderLayout();

    public BeanieBaby() {
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void jbInit() throws Exception{
        this.setLayout (borderLayout1);
    }
}
```

```
}
```

Si examina el código generado por JBuilder, observará que:

- JBuilder ha asignado al bean el nombre especificado y ha ampliado la clase elegida, que está declarada como `public`.
- La clase cuenta con un constructor sin parámetros.

Aún en su estado más rudimentario, esta clase es un JavaBean válido.

## Diseño de la interfaz de usuario de un bean

---

No todos los JavaBeans disponen de interfaz de usuario, pero, si se desea que incluyan una, es posible emplear el diseñador de interfaces de usuario de JBuilder para crearla.

Para crear la interfaz de usuario de un bean:

- 1 En el panel de proyectos, seleccione el archivo de bean que JBuilder ha creado automáticamente.
- 2 Pulse sobre la pestaña Diseño para visualizar el diseñador de interfaces de usuario.
- 3 Mediante el diseñador de interfaces de usuario, elabore la interfaz de usuario del bean.

Para obtener más información sobre la creación de interfaces de usuario, consulte *Diseño de aplicaciones con JBuilder*.

## Adición de propiedades a un bean

---

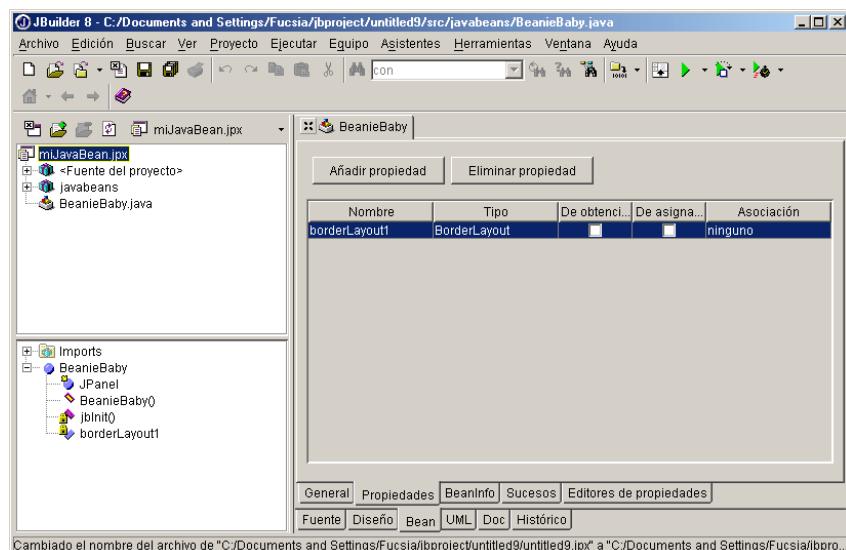
Las propiedades definen los atributos del bean. Por ejemplo, la propiedad `backgroundColor` describe el color de fondo.

Por lo general, las propiedades de JavaBeans disponen de un método de acceso de escritura y otro de acceso de lectura, también denominados método de obtención y método de asignación, respectivamente. El método de obtención devuelve el valor actual de la propiedad, mientras que el de asignación le asigna un nuevo valor.

Para incorporar una propiedad a un bean:

- 1 Seleccione el componente que desee en el panel de proyectos y haga clic en la pestaña Bean para visualizar los diseñadores BeansExpress.

- 2** Haga clic sobre la pestaña Propiedades para visualizar el diseñador de propiedades.



- 3** Pulse el botón Añadir propiedad. Aparece el cuadro de diálogo Nueva propiedad.

**4** Indique el nombre de la propiedad en el cuadro Nombre.

**5** Indique el tipo en el cuadro Tipo.

Puede escribir uno de los tipos de Java o recurrir al visualizador de paquetes para seleccionar uno, que también puede ser otro JavaBean.

- 6** Deje marcadas las casillas De obtención y De asignación si desea que JBuilder genere los métodos de lectura y escritura del valor de la propiedad.

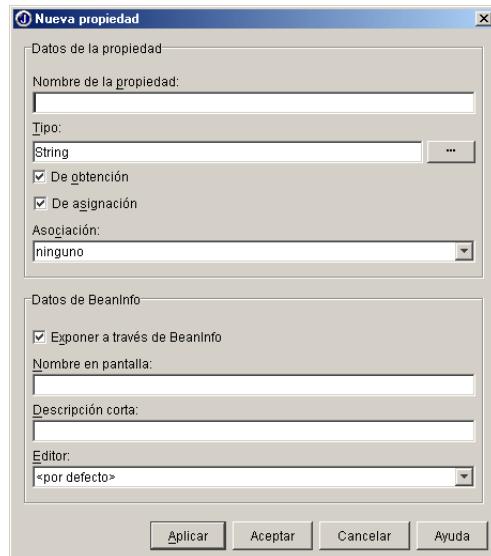
Si desea que una determinada propiedad sea de sólo lectura, desactive la casilla de selección De asignación.

- 7** Pulse Aceptar.

JBuilder genera el código necesario para la propiedad y añade esta propiedad a la rejilla del diseñador de propiedades. Puede observar que los métodos de acceso de lectura y escritura del bean se han añadido al árbol de componentes. Si hace clic en la pestaña Fuente podrá ver el código generado por JBuilder.

## Adición de propiedades a un bean

A continuación, se muestra el cuadro de diálogo Nueva propiedad con los campos obligatorios rellenos:



Los campos Nombre en pantalla y Descripción corta se llenan automáticamente con los valores por defecto. Este es el código fuente resultante:

```
package myjavabean;

import java.awt.*;
import javax.swing.JPanel;

public class BeanieBaby extends JPanel {
    BorderLayout borderLayout1 = new BorderLayout();
    private float price; // Se ha añadido un campo private para el precio

    public BeanieBaby() {
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void jbInit() throws Exception{
        this.setLayout (borderLayout1);
    }

    public void setPrice(float price) { // Se ha añadido un método para cambiar el
```

```

        // precio
        this.price = price;;
    }

    public float getPrice() {           // Se ha añadido un método para
        obtener el precio             // precio
        return price;
    }
}

```

JBuilder ha añadido un campo `price` a la clase `BeanieBaby`. El campo `price` contiene el valor de la propiedad. Por lo general, los campos de propiedades se declaran `privados`, de forma que sólo puedan acceder a ellos los métodos de obtención y asignación.

También se ha generado un método `setPrice()` que puede cambiar el valor del campo y un método `getPrice()` que puede leer su valor actual.

Una vez instalado el bean en la paleta de componentes de JBuilder, cuando el usuario lo coloque en el diseñador de interfaces de usuario, la propiedad `price` aparecerá en el Inspector y los usuarios podrán modificar su valor. El código necesario para leer y escribir el valor de `price` se ha incluido en el lugar oportuno.

## Modificación de propiedades

---

Una vez las propiedades son incorporadas a un bean, estas pueden modificarse en cualquier momento mediante el diseñador de propiedades.

Para modificar una propiedad:

- 1** Seleccione el bean mediante el panel de proyectos.
- 2** Pulse sobre la pestaña Bean para visualizar los diseñadores BeansExpress.
- 3** Haga clic en la pestaña Propiedades.
- 4** Seleccione cualquiera de los campos que aparecen en la rejilla del diseñador de propiedades y efectúe las modificaciones oportunas.

Puede, por ejemplo, cambiar el tipo de la propiedad introduciendo otro.

JBuilder refleja en el código fuente del bean los cambios realizados en el diseñador de propiedades. También existe la posibilidad de introducir cambios directamente en el código fuente y, siempre que se hayan realizado correctamente, se mostrarán en los diseñadores BeansExpress.

## Eliminación de propiedades

---

Para eliminar una propiedad de un bean:

- 1 En el panel de proyectos, seleccione el bean que contenga la propiedad deseada.
- 2 Pulse sobre la pestaña Bean para visualizar los diseñadores BeansExpress.
- 3 Haga clic en la pestaña Propiedades.
- 4 En la rejilla del diseñador de propiedades, seleccione la propiedad que desee eliminar.
- 5 Haga clic en Eliminar propiedad.

Tanto el campo de la propiedad como los métodos de acceso se eliminan del código fuente.

## Adición de propiedades monitorizables y restringidas

---

BeansExpress genera el código necesario para crear propiedades monitorizables y restringidas.

Para incorporar una propiedad monitorizable o restringida:

- 1 En el diseñador de propiedades, pulse el botón Añadir propiedad con el fin de mostrar el cuadro de diálogo Nueva propiedad.
- 2 Indique el nombre y el tipo de la propiedad.
- 3 Mediante la lista desplegable Asociación indique si la propiedad ha de ser monitorizable o restringidas.
- 4 Pulse Aceptar.

Si se trata de una propiedad monitorizable, JBuilder añade el correspondiente campo `private` y genera los métodos de lectura y escritura pertinentes. También crea una instancia de la clase `PropertyChangeSupport`. Por ejemplo, el código generado para una propiedad monitorizable denominada `allTiedUp` es:

```
public class BeanieBaby extends JPanel {
    ...
    private String allTiedUp;
    private transient PropertyChangeSupport propertyChangeListeners = new
        PropertyChangeSupport(this);
    ...
    public void setAllTiedUp(String allTiedUp) {
        String oldAllTiedUp = this.allTiedUp;
        this.allTiedUp = allTiedUp;
        propertyChangeListeners.firePropertyChange("allTiedUp", oldAllTiedUp,
```

```

        allTiedUp);
    }
    public String getAllTiedUp() {
        return allTiedUp;
    }
}

```

Observe que el método `setAllTiedUp()` incluye código para notificar los cambios del valor de la propiedad a los componentes de monitorización.

JBuilder también genera los métodos de registro a los que llaman los componentes de monitorización de sucesos que desean recibir notificación cuando cambia el valor de la propiedad `allTiedUp`:

```

public synchronized void removePropertyChangeListener(PropertyChangeListener l) {
    super.removePropertyChangeListener(l);
    propertyChangeListeners.removePropertyChangeListener(l);
}
public synchronized void addPropertyChangeListener(PropertyChangeListener l) {
    super.addPropertyChangeListener(l);
    propertyChangeListeners.addPropertyChangeListener(l);
}

```

El código generado por JBuilder para las propiedades restringidas es similar al anterior, con la diferencia de que los monitores tienen posibilidad de rechazar los cambios del valor de las propiedades.

## Creación de una clase BeanInfo

---

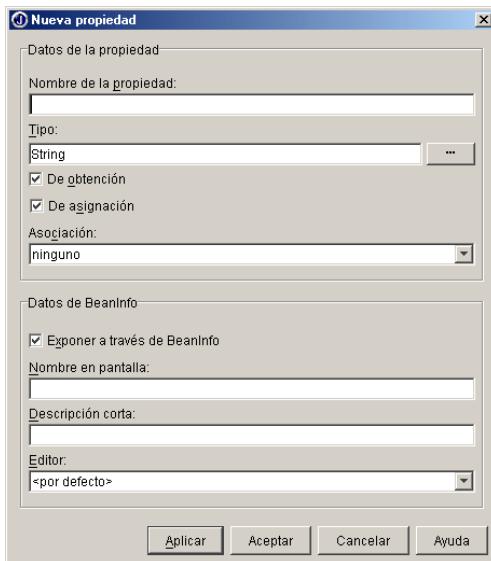
La clase BeanInfo ofrece la posibilidad de personalizar la presentación de los beans en las herramientas visuales como JBuilder. Por ejemplo, puede interesarle ocultar parte de las propiedades para que no aparezcan en el Inspector de JBuilder. Aunque desde el código fuente seguirá siendo posible acceder a esas propiedades, el usuario no podrá modificarlas en la fase de diseño.

Las clases BeanInfo son optativas. Con BeansExpress puede generar una clase BeanInfo para personalizar un bean. Si piensa utilizar una clase BeanInfo, ha de especificar información más detallada en las propiedades que añada al bean.

## Especificación de datos BeanInfo de una propiedad

---

Los datos BeanInfo de una propiedad pueden especificarse en el cuadro de diálogo Nueva propiedad:



Para ocultar una propiedad y que no aparezca en las herramientas de diseño visuales, como el Inspector de JBuilder, cerciórese de que la opción Exponer a través de BeanInfo no esté marcada.

Si desea que la propiedad se muestre con un nombre distinto, introduzca el que deseé en el campo Nombre en pantalla.

Si desea ofrecer una descripción corta de la propiedad, escríbala en el campo Descripción corta. El texto introducido aparece en el Inspector, en forma de ayuda inmediata para la propiedad.

Si ha creado un editor para modificar la propiedad, indique el nombre en el campo Editor. El campo Editor muestra los editores que pueden funcionar en el ámbito actual y tienen capacidad para ese tipo concreto de propiedad.

## Utilización del diseñador de BeanInfo

---

El diseñador de BeanInfo permite modificar los datos BeanInfo de una propiedad, elegir los iconos que han de representar al bean correspondiente en los creadores de aplicaciones como JBuilder y generar automáticamente la clase BeanInfo.

Pulse la pestaña BeanInfo del diseñador de BeansExpress para abrir el Diseñador de BeanInfo.



El diseñador de BeanInfo muestra todas las propiedades añadidas al bean. Si desea rectificar los datos BeanInfo introducidos para una propiedad, puede editar los que desee en la rejilla. El único dato que no puede cambiarse es el nombre de la propiedad.

Si desea elegir un ícono para representar al bean, indique las imágenes que desee en el panel Iconos.

Los paneles Iconos permiten elegir distintos iconos para entornos monocromos y en color, y también para diferentes resoluciones de pantalla. Rellene los cuadros según le convenga.

Si la superclase del bean dispone de una clase BeanInfo y desea que los datos de esta última aparezcan en la clase BeanInfo del bean, marque la casilla de selección Exponer BeanInfo de superclase. El código generado contendrá un método `getAdditionalBeanInfo()` que devolverá los datos BeanInfo de la superclase del bean.

Si desea crear una clase BeanInfo para el bean, pulse el botón Generar BeanInfo.

JBuilder creará automáticamente una clase BeanInfo. Lo verá aparecer en el panel de proyectos. Para examinar el código generado, seleccione la clase BeanInfo recién creada en el panel de proyectos y aparecerá el código fuente.

## Modificación de clases BeanInfo

Puede modificar las clases BeanInfo de los beans mediante BeansExpress.

- 1** Seleccione el bean en el panel de proyectos.
- 2** Pulse sobre la pestaña Bean para visualizar los diseñadores BeansExpress.
- 3** Pulse sobre la pestaña BeanInfo para visualizar el diseñador de BeanInfo.

4 Introduzca los cambios en la rejilla.

5 Pulse nuevamente el botón Generar BeanInfo.

Se le avisará de que está a punto de sobrescribir la clase BeanInfo anterior.  
Si elige Aceptar, la clase se sobreescibirá con los nuevos datos BeanInfo.

## Adición de sucesos a un bean

Los JavaBeans pueden:

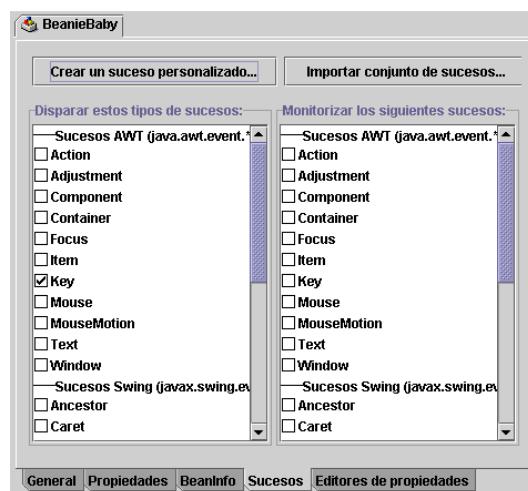
- Generar (o activar) sucesos enviando un objeto de suceso a un objeto monitor.
- Monitorizar sucesos y responder a ellos cuando se produzcan.

BeansExpress genera automáticamente código que permite realizar una o ambas de estas funciones.

## Activación de sucesos

Si desea poder enviar objetos de suceso a los componentes monitores de sucesos:

- 1 Seleccione el bean en el panel de proyectos.
- 2 Pulse sobre la pestaña Bean para visualizar los diseñadores BeansExpress.
- 3 Pulse sobre la pestaña Sucesos para mostrar el diseñador de sucesos.



- 4** En la ventana izquierda, seleccione los sucesos que desea el Bean pueda activar.

El diseñador de sucesos añade al bean los métodos de registro de sucesos. Los componentes que deseen recibir notificación de esos sucesos, llamarán a estos métodos. Por ejemplo, si ha seleccionado los sucesos Key, se añadirán los siguientes métodos al bean:

```
package myjavabean;

import java.io.*;
import java.beans.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;

public class BeanieBaby extends JPanel {
    BorderLayout borderLayout1 = new BorderLayout();
    private float price;
    private transient Vector keyListeners;

    public BeanieBaby() {
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        this.setLayout (borderLayout1);
    }

    public void setPrice(float price) {
        this.price = price;
    }

    public float getPrice() {
        return price;
    }

    public synchronized void removeKeyListener(KeyListener l) {
        super.removeKeyListener(l);
        if(keyListeners != null && keyListeners.contains(l)) {
            Vector v = (Vector) keyListeners.clone();
            v.removeElement(l);
            keyListeners = v;
        }
    }

    public synchronized void addKeyListener(KeyListener l) {
        super.addKeyListener(l);
        Vector v = keyListeners == null ? new Vector(2) : (Vector)
keyListeners.clone();
        if(!v.contains(l)) {
            v.addElement(l);
        }
    }
}
```

## Adición de sucesos a un bean

```
        keyListeners = v;
    }
}
...
}
```

Si un componente desea recibir notificación de los sucesos Key que tengan lugar en el bean, llamará al método `addKeyListener()` de este último y pasará a ser un elemento de `keyListeners`. Cada vez que se active un suceso Key en el bean, éste se notificará a todos los componentes de monitorización declarados en `keyListeners`.

La clase también genera métodos de activación de <suceso> que envían un suceso a todos los monitores registrados. Se genera uno de estos sucesos por cada método declarado en la interfaz Monitor. Por ejemplo, la interfaz `KeyListener` dispone de tres métodos: `keyTyped()`, `keyPressed()` y `keyReleased()`. El diseñador de sucesos añade estos tres métodos de activación de <suceso> a la clase bean:

```
protected void fireKeyPressed(KeyEvent e) {
    if(keyListeners != null) {
        Vector listeners = keyListeners;
        int count = listeners.size();
        for (int i = 0; i < count; i++) {
            ((KeyListener) listeners.elementAt(i)).keyPressed(e);
        }
    }
}

protected void fireKeyReleased(KeyEvent e) {
    if(keyListeners != null) {
        Vector listeners = keyListeners;
        int count = listeners.size();
        for (int i = 0; i < count; i++) {
            ((KeyListener) listeners.elementAt(i)).keyReleased(e);
        }
    }
}

protected void fireKeyTyped(KeyEvent e) {
    if(keyListeners != null) {
        Vector listeners = keyListeners;
        int count = listeners.size();
        for (int i = 0; i < count; i++) {
            ((KeyListener) listeners.elementAt(i)).keyTyped(e);
        }
    }
}
```

Una vez habilitado el bean para generar sucesos, éstos aparecerán en el Inspector de JBuilder cuando el usuario coloque el bean en el diseñador de interfaces de usuario.

## Monitorización de sucesos

---

Los beans también pueden convertirse en monitores de los sucesos que tengan lugar en otros componentes.

Si desea convertir un bean en monitor, seleccione en el diseñador de sucesos el tipo de suceso que deseé monitorizar.

En cuanto selecciona un tipo de suceso, el diseñador de sucesos añade la interfaz de monitor correspondiente al bean. Por ejemplo, si ha marcado `java.awt.event.KeyListener`, se añade la frase `implements KeyListener` a la declaración del bean y se implementan los métodos `KeyPressed()`, `KeyReleased()` y `KeyTyped()` con las secciones de código principal vacías.

A continuación, figura un bean con el código generado para implementar la interfaz `KeyListener`:

```
package myBeans;

import java.awt.*;
import javax.swing.JPanel;
import java.beans.*;
import java.awt.event.*;

public class JellyBean extends JPanel implements KeyListener { // implementa
                                                               // KeyListener
    BorderLayout borderLayout1 = new BorderLayout();

    public JellyBean() {

        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    void jbInit() throws Exception {
        this.setLayout (borderLayout1);
    }

    public void keyTyped(KeyEvent e) {           // Añade un nuevo método
    }

    public void keyPressed(KeyEvent e) {          // Añade un nuevo método
    }

    public void keyReleased(KeyEvent e) {         // Añade un nuevo método
    }
}
```

Si el bean se ha registrado como monitor en otro componente (llamando al método de registro de sucesos de este último), cuando se produce un

suceso del tipo elegido, el componente monitorizado llama a uno de los métodos del bean monitor. Por ejemplo, si se produce un suceso KeyPressed en el componente monitorizado, se llama al método KeyPressed() del componente monitor. Por tanto, si desea que el componente responda a este suceso, escriba el código oportuno en la sección principal del método KeyPressed(), por ejemplo.

## Creación de un conjunto de sucesos personalizado

---

Puede que, en algún momento, le interese crear un conjunto de sucesos personalizado que describa otros sucesos que puedan tener lugar en un bean determinado. Por ejemplo, si el bean implementa un cuadro de diálogo destinado a la introducción de contraseñas, le interesaría activar un suceso cada vez que un usuario introduzca correctamente su contraseña. Posiblemente, también interesaría que el bean active un suceso cuando se introduzca una contraseña incorrecta. BeansExpress permite crear un conjunto de sucesos personalizado para tratar estas situaciones.

Para crear un conjunto de sucesos personalizado:

- 1** Seleccione un bean en el panel del proyecto.
- 2** Pulse sobre la pestaña Bean para visualizar los diseñadores BeansExpress.
- 3** Pulse sobre la pestaña Sucesos.
- 4** Pulse el botón Crear un suceso personalizado.

Aparece el cuadro de diálogo Nuevo conjunto de sucesos.

- 5** Indique el nombre del conjunto de sucesos en el cuadro Nombre del conjunto de sucesos.

Los nombres del objeto de suceso y el monitor de sucesos se generan a partir del nombre del conjunto de sucesos; se muestran en el cuadro de diálogo.

- 6** Seleccione el elemento dataChanged y asígnale el nombre del primer suceso del conjunto.

- 7** Para añadir otros sucesos, pulse el botón Añadir suceso por cada uno de ellos y sustituya los nombres de los elementos añadidos por los correspondientes nombres de los sucesos:



- 8** Pulse Aceptar.

En el diseñador de sucesos, el nuevo conjunto de sucesos se añade a la lista de tipos de suceso que pueden activarse.

JBuilder genera la clase del objeto de suceso correspondiente al conjunto de sucesos:

```
package myBeans;

import java.util.*;

public class PasswordEvent extends EventObject {
    public PasswordEvent(Object source) {
        super(source);
    }
}
```

JBuilder también genera la interfaz de monitor para el conjunto de sucesos:

```
package myBeans;

import java.util.*;

public interface PasswordListener extends EventListener {
    public void passwordSuccessful(PasswordEvent e);
    public void passwordFailed(PasswordEvent e);
}
```

## Creación de editores de propiedades

---

Los editores de propiedades permiten cambiar los valores de éstas en la fase de diseño. En el Inspector de JBuilder pueden verse distintos tipos de editores de propiedades. Por ejemplo, con ciertas propiedades, al escribir un valor en el Inspector se cambia el valor de la propiedad. Este es el tipo más sencillo de editor de propiedades. En otros casos, aparece un menú de opciones (lista desplegable) que muestra todos los posibles valores de la propiedad. Los editores de propiedades de colores y fuentes son, en realidad, cuadros de diálogo que permiten establecer los valores correspondientes.

Cuando desarrolle sus propios JavaBeans, puede interesarle crear clases de propiedades y los correspondientes editores que puedan modificar sus valores. BeansExpress ofrece la posibilidad de crear editores que presenten listas de opciones.

Para crear un editor de propiedades:

- 1 Haga clic en la pestaña Editores de propiedades de BeansExpress.
- 2 Pulse el botón Crear un editor personalizado.

Aparecerá el cuadro de diálogo Nuevo editor de propiedades.

- 3 Indique el nombre del editor en el cuadro Nombre del editor. Asigne al editor el mismo nombre de la propiedad que va a editar y añádale la palabra Editor. Por ejemplo, si el nombre de la propiedad es platosPreferidos, llame al editor PlatosPreferidosEditor.
- 4 Seleccione el tipo de editor deseado en la lista desplegable Tipo de editor.

El aspecto del cuadro de diálogo Nueva propiedad cambia en función del tipo de editor seleccionado. En los cuatro apartados siguientes se explica cómo crear un editor de propiedades de cada uno de los cuatro tipos.

## Creación de un editor de lista de cadenas

---

Las listas de cadenas aparecen en el Inspector en forma de listas desplegables que contienen las cadenas especificadas. Cuando el usuario selecciona una entrada de la lista, la propiedad que se está modificando adquiere el valor seleccionado.

Para añadir elementos a un editor de listas de cadenas:

- 1 Seleccione Añadir entrada cada vez que desee incluir un nuevo elemento en la lista.
- 2 En cada entrada, introduzca la cadena que desee.

Este es el aspecto que presentaría el cuadro de diálogo Nuevo editor de propiedades:



- 3** Pulse Aceptar y habrá creado otra clase de editor de propiedades.

Para ver el código generado:

- 1** Seleccione la clase del editor de propiedades en el panel de proyectos.
- 2** Pulse sobre la pestaña Fuente.

## Creación de un editor de listas de etiquetas de cadena

---

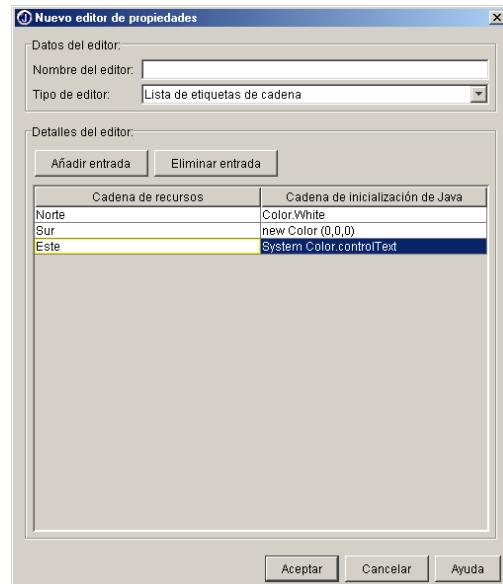
Los editores de propiedades que son editores de lista de etiquetas de cadena también presentan listas de cadenas en el Inspector. Cuando el usuario selecciona una de las entradas, como valor de la propiedad se toma la cadena de inicialización de Java especificada. Las cadenas de inicialización de Java son las que JBuilder emplea en el código fuente generado para las propiedades.

Para añadir elementos a un editor de listas de etiquetas de cadena:

- 1** Seleccione Añadir entrada cada vez que desee incluir un nuevo elemento en la lista.
- 2** En cada entrada, introduzca la cadena que desee presentar en pantalla y la cadena de inicialización asociada a ella.

Si desea incluir una cadena en la lista de cadenas de inicialización, enciérrala entre comillas (""), tal como haría si la estuviese introduciendo en el código fuente.

El cuadro de diálogo presentaría el siguiente aspecto:



- 3 Pulse Aceptar y habrá creado otra clase de editor de propiedades.

Para ver el código generado:

- 1 Seleccione la clase del editor de propiedades en el panel de proyectos.
- 2 Pulse sobre la pestaña Fuente.

## **Creación de un editor de listas de etiquetas de entero**

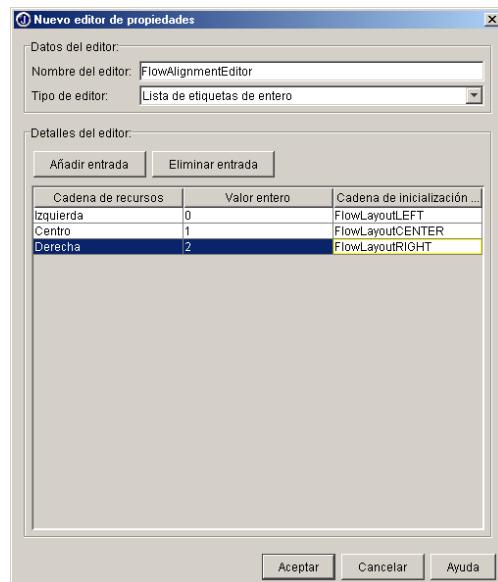
El editor de propiedades de listas de etiquetas de enteros permite editar las propiedades de los enteros. Este editor muestra al usuario una lista de cadenas en el Inspector. Cuando se selecciona un elemento, se usa la cadena de inicialización de Java para configurar la propiedad.

Para añadir elementos a un editor de listas de etiquetas de enteros:

- 1 Seleccione Añadir entrada cada vez que deseé incluir un nuevo elemento en la lista.
- 2 En cada entrada, introduzca la cadena que deseé presentar en pantalla y la cadena de inicialización asociada a ella.

Si desea incluir una cadena en la lista de cadenas de inicialización, enciérrala entre comillas (""), tal como haría si la estuviese introduciendo en el código fuente.

El cuadro de diálogo presentaría el siguiente aspecto:



- 3** Pulse Aceptar y habrá creado otra clase de editor de propiedades.

Para ver el código generado:

- 1** Seleccione la clase del editor de propiedades en el panel de proyectos.
- 2** Pulse sobre la pestaña Fuente.

## Creación de editores de propiedades basados en componentes personalizados

También puede emplear un componente personalizado para editar el valor de una propiedad. Si se selecciona esta opción, se genera un editor de propiedades básico que, para modificar la propiedad, se sirve del editor personalizado que se especifique.

Para especificar un editor de propiedades personalizado:

- 1** Escriba un nombre para la clase del editor que instancie su componente personalizado en el campo Nombre del editor del cuadro de diálogo Nueva propiedad.
- 2** Seleccione Componente editor personalizado en la lista desplegable.

- 3 Especifique el nombre del componente personalizado como el valor del campo Componente editor personalizado.
- 4 Marque la opción Admite paintValue() si desea que el editor personalizado se dibuje a sí mismo.

Para ver el código generado:

- 1 Seleccione la clase del editor de propiedades en el panel de proyectos.
- 2 Pulse sobre la pestaña Fuente.

## Admisión de serialización

---

Serializar un bean es guardar su estado en forma de secuencia de bytes que puede enviarse por una red o almacenarse en un archivo. BeansExpress puede añadir la admisión de serialización a las clases.

Para incorporar la admisión de serialización:

- 1 Seleccione el bean en el panel de proyectos.
- 2 Pulse sobre la pestaña Bean para visualizar los diseñadores BeansExpress.
- 3 Pulse sobre la pestaña General para visualizar la ficha General.
- 4 Marque la opción Admite serialización.

BeansExpress implementa la interfaz `Serializable` en la clase. Se añaden dos métodos a la clase: `readObject()` y `writeObject()`:

```
void writeObject(ObjectOutputStream oos) throws IOException {  
    oos.defaultWriteObject();  
}  
  
void readObject(ObjectInputStream ois) throws ClassNotFoundException,  
IOException {  
    ois.defaultReadObject();  
}
```

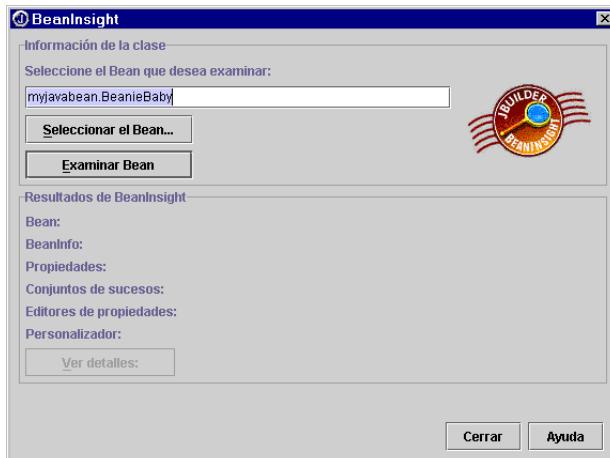
Debe escribir código en ambos para serializar y paralelizar el bean.

# Comprobación de la validez de un JavaBean

Una vez se finaliza un bean, puede emplearse BeanInsight para verificar que el componente recién creado es un JavaBean válido. Si se detectan errores en la clase, BeanInsight informa de ellos. Presenta todas las propiedades, personalizadores y editores de propiedades que encuentra en la clase. También muestra la procedencia de la información de las propiedades, es decir, si se obtiene mediante una clase BeanInfo o mediante introspección.

Para verificar que una clase Java es un JavaBean:

- 1 Seleccione Herramientas | BeanInsight para visualizar BeanInsight:



- 2 Escriba el nombre del componente que desea examinar o pulse el botón de puntos suspensivos para elegir uno. Si el bean ya está seleccionado en el panel de proyectos cuando aparece BeanInsight, su nombre se muestra en BeanInsight.
- 3 Pulse el botón Examinar Bean para iniciar el proceso de examen. BeanInsight ofrece un informe del resultado del examen en la sección Resultados de BeanInsight del asistente.
- 4 Si desea un informe completo del examen de BeanInsight, haga clic en el botón Ver detalles.
- 5 Haga clic en las distintas pestañas para examinar el informe detallado.

## Instalación de beans en la paleta de componentes

---

Una vez se ha desarrollado un componente JavaBean y se ha comprobado su validez, se puede instalar en la paleta de componentes mediante el cuadro de diálogo Propiedades de la paleta. Los archivos de clase del nuevo componente han de encontrarse en la vía de acceso a clases.

Para abrir el cuadro de diálogo Propiedades de la paleta, seleccione Herramientas | Configurar paleta o haga clic con el botón derecho del ratón sobre la paleta de componentes y seleccione Propiedades.

Si desea obtener más información sobre la instalación de componentes, pulse el botón de Ayuda del cuadro de diálogo Propiedades de la paleta, o bien, consulte “Cómo añadir un componente a la paleta de componentes” en *Diseño de aplicaciones con JBuilder*.

# 11

## Presentación de código con UML

Es una función de JBuilder Enterprise.

*UML (Unified Modeling Language, lenguaje unificado de modelado)* es una anotación estándar para el modelado de sistemas orientados a objetos. UML es, básicamente, un lenguaje que describe gráficamente un conjunto de elementos. Sus usos más complejos son los de especificar, presentar, construir y documentar sistemas de software y de otros tipos, así como modelos empresariales. Como los planos de construcción de los edificios, UML proporciona una representación gráfica del diseño de un sistema, que puede ser esencial para la comunicación entre los miembros del equipo y para garantizar la coherencia arquitectónica del sistema.

En la presentación del código, UML constituye una útil herramienta de examen, análisis del desarrollo de las aplicaciones y comunicación del diseño del software. JBuilder utiliza diagramas UML para presentar código y buscar clases y paquetes. Los diagramas UML pueden ayudarle a comprender rápidamente la estructura de un código desconocido, reconocer áreas especialmente complejas, e incrementar su productividad resolviendo problemas con mayor rapidez.

Si desea conocer más a fondo el lenguaje UML, visite estas sedes web:

- Object Management Group, en <http://www.omg.org/>
- Cetus UML links, en <http://www.cetus-links.org/oo.uml.html>
- UML Central, en [http://www.embarcadero.com/support/uml\\_central.htm](http://www.embarcadero.com/support/uml_central.htm)
- UML Resource Center, en <http://www.rational.com/uml/index.jsp>
- UML Zone en <http://www.devx.com/uml/>
- UML Dictionary, en <http://softdocwiz.com/UML.htm>

Si desea ver un tutorial sobre como utilizar el visualizador UML de JBuilder consulte el [Capítulo 20, “Tutorial: Visualización de código con el visualizador UML”](#).

## Java y UML

---

Dado que Java y UML son lenguajes orientados a objetos e independientes de la plataforma, se integran sin problemas. UML, una valiosa herramienta para la comprensión de Java y las complejas relaciones entre clases y paquetes, ayuda a los desarrolladores a captar el significado de las clases y el paquete completo en el que se encuentran. Sobre todo, puede ayudar a los desarrolladores en Java que se incorporan a un equipo, a familiarizarse rápidamente con la estructura y el diseño del sistema de software.

### Términos de Java y UML

---

Dado que UML está concebido para describir muchos entornos distintos, utiliza términos genéricos para referirse a las relaciones. En la tabla siguiente se definen los términos propios de Java y sus equivalentes en UML. En algunos casos, los términos son idénticos. En esta documentación se utilizan los correspondientes a Java.

**Tabla 11.1** Términos de Java y UML

Término Java	Definición Java	Términos de UML	Definición UML
Herencia	Mecanismo por el cual una clase o interfaz se define como especialización de otra más amplia. Por ejemplo, una subclase (hija, subordinada, descendiente, que amplía,...) hereda la estructura y el comportamiento en lo relativo a campos, métodos, etc., de su superclase (padre, principal, antecesora, ampliada,...). Las clases e interfaces herederas utilizan la palabra clave <code>extends</code> .	Generalización / especialización	Una relación entre un elemento especializado y otro generalizado, en la que el primero incorpora la estructura y el comportamiento del segundo.
Dependencia	Relación en la que los cambios realizados en un objeto independiente pueden influir sobre otro que depende de él.	Dependencia	Relación en la que las características semánticas de una entidad dependen de las de otra y están determinadas por ella.
Asociación	Dependencia especializada en la que se almacena una referencia a otra clase.	Relación (asociación)	Relación de estructura que describe los vínculos entre objetos.

**Tabla 11.1** Términos de Java y UML (continuación)

Término Java	Definición Java	Términos de UML	Definición UML
Interfaz	Grupo de constantes y declaraciones de métodos que definen la forma de una clase sin implementar los métodos. En la interfaz se establece lo que debe hacer una clase, pero no la forma en que debe hacerlo. Las clases e interfaces que implementan la interfaz utilizan la palabra clave <code>implements</code> .	Realización / interfaz	Conjunto de operaciones que especifican un servicio de una clase o componente. Definen el comportamiento de una abstracción sin implementarlo.
Método	Implementación de una operación definida por una interfaz o una clase.	Operación	Implementación de un servicio que puede solicitar un objeto y puede influir en su comportamiento. Normalmente, en los diagramas UML las operaciones se enumeran debajo de los atributos.
Campo	Variable de instancia o miembro de datos de un objeto.		
Propiedad	Información sobre el estado actual de un componente. Las propiedades de los componentes pueden considerarse atributos que tienen un nombre concreto y que puede leer ( <code>get</code> ) o definir ( <code>set</code> ) un usuario o un programa. En los diagramas UML existe una propiedad cuando el nombre de un campo coincide con el de un método, este último precedido por “ <code>is</code> ”, “ <code>set</code> ” o “ <code>get</code> ”. Por ejemplo, un campo llamado <code>parameterRow</code> es una propiedad si tiene un método llamado <code>setParameterRow()</code> .	Atributo	Propiedad de un clasificador, como una clase o una interfaz, que describe los valores que pueden adoptar las instancias de una propiedad.

## JBuilder y UML

En lugar de sustituir las herramientas de UML, JBuilder se centra en la presentación del código y en la adaptación de los diagramas UML al lenguaje Java. Las funciones de UML integradas en JBuilder permiten examinar visualmente paquetes y clases, con el fin de ayudar en el diseño, la comprensión y la solución de problemas en el proceso de desarrollo de aplicaciones.

JBuilder ofrece dos diagramas UML:

- Diagramas limitados a las dependencias de paquetes
- Diagramas combinados de clases

El visualizador UML de JBuilder proporciona funciones adicionales, como el perfeccionamiento (refactoring) del código, la personalización de la representación en UML y la presentación del Javadoc y el código fuente.

**Importante** En el código confuso, JBuilder excluye los campos privados y los miembros de los archivos de clase.

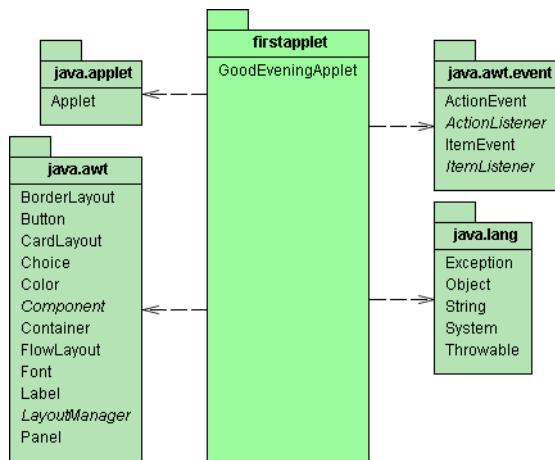
### Consulte

- “Personalización de diagramas UML” en la página 11-18
- Capítulo 12, “Perfeccionamiento de símbolos de código”

## Diagrama limitado a las dependencias de paquetes

Los *diagramas de paquetes* se centran en un paquete y muestran únicamente sus dependencias internas; no indican las dependencias de este paquete con otros. Las dependencias directas e inversas aparecen a la izquierda, a la derecha o a los dos lados del paquete central. Los paquetes con relaciones inversas se colocan en ambos lados, las mixtas en la parte central y las demás en la parte inferior. Los paquetes con dependencias muestran las clases dependientes dentro del paquete. Es posible desplazarse por ellas haciendo doble clic en el diagrama. El paquete central actual se muestra por defecto sobre un fondo verde brillante. Los demás paquetes se muestran por defecto sobre un fondo de un verde más oscuro.

**Figura 11.1** Diagrama de paquetes



Aunque en el diagrama UML se muestran únicamente el paquete actual y los importados, se pueden incluir referencias de las clases de las bibliotecas del proyecto. Con el fin de incluir las referencias de biblioteca, active la opción Incluir referencias de archivos de clase de las bibliotecas del proyecto de la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto).

## Consulte

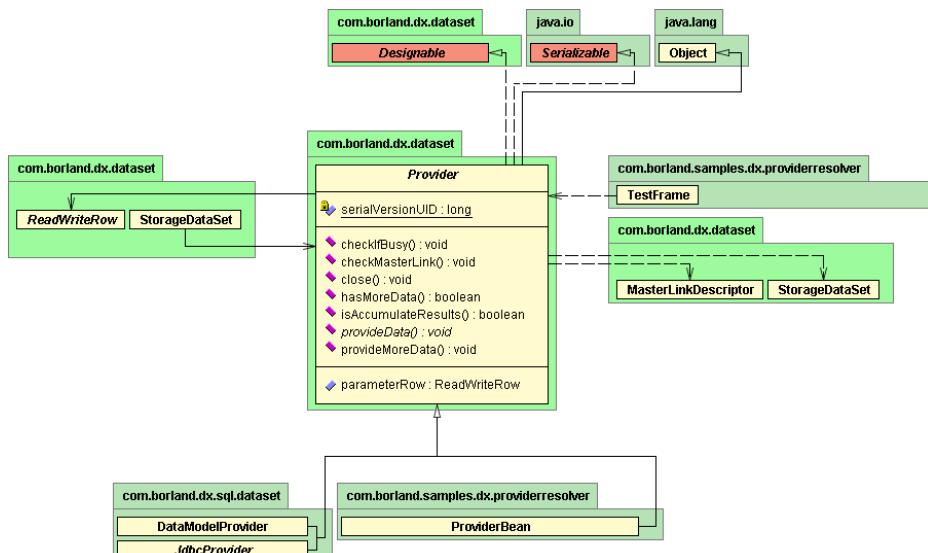
- “Presentación de diagramas de paquetes” en la página 11-12
- “Inclusión de referencias de bibliotecas de proyecto” en la página 11-20

## Diagrama combinado de clases

Los *diagramas combinados de clases* del archivo Java fuente o de clases abierto en el editor muestran la clase en el centro del diagrama, con las asociaciones a la izquierda y las dependencias a la derecha. Las clases ampliadas (superclases) y las interfaces ampliadas (padres o antecesores) aparecen en la parte superior, y las clases herederas y de implementación aparecen en la parte inferior. Las clases se agrupan con arreglo a un paquete.

Las asociaciones y dependencias agrupadas se ordenan como sigue: las asociaciones inversas aparecen en la parte superior izquierda de la clase central, y las dependencias inversas, en la parte superior derecha; las asociaciones y dependencias con relaciones mixtas aparecen en las partes centrales izquierda y derecha de la clase; todas las demás asociaciones y dependencias aparecen en las partes inferiores izquierda y derecha de la clase.

**Figura 11.2** Diagrama de clases



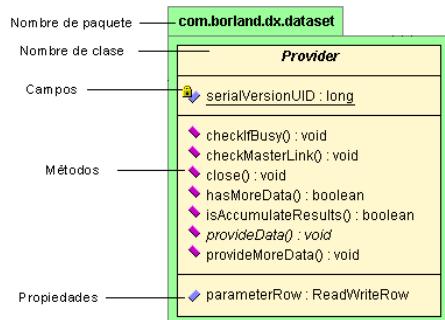
El diagrama UML de clases muestra la clase en el centro, en un rectángulo cuyo color de fondo por defecto es el amarillo. Rodeándola, aparece el paquete, con su nombre en una pestaña en la parte superior. La clase se

divide en varias secciones, separadas por líneas horizontales, en el orden siguiente:

- Nombre de la clase en la parte superior.
- Campos y propiedades\*.
- Métodos; funciones de obtención\* y definición\*.
- Propiedades\* en la parte inferior.

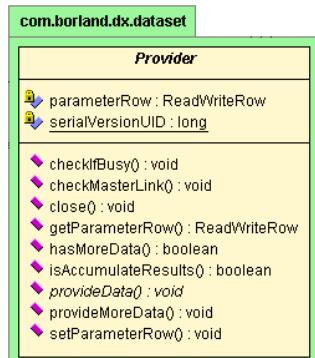
\*Por defecto, las propiedades se muestran en la sección inferior del diagrama de clases. La opción Mostrar propiedades por separado se configura en la ficha UML del cuadro de diálogo Opciones (Herramientas | Opciones del IDE). Si esta opción está desactivada, las propiedades se muestran en las secciones adecuadas, con campos y métodos. Consulte “[Configuración de las opciones del IDE](#)” en la página 11-21.

**Figura 11.3** Diagrama de clases con las propiedades por separado



**Nota** Los iconos indican si un campo, método o propiedad es privado, público, o protegido. Consulte “[Iconos de accesibilidad](#)” en la página 11-9.

**Figura 11.4** Diagrama de clases sin las propiedades por separado



Aunque en el diagrama UML se muestran únicamente el paquete actual y los importados, se pueden incluir referencias de las clases de las bibliotecas del proyecto. Con el fin de incluir las referencias de biblioteca, active la opción Incluir referencias de archivos de clase de las bibliotecas del proyecto de la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto).

### Consulte

- “Presentación de diagramas de clases” en la página 11-12
- “Inclusión de referencias de bibliotecas de proyecto” en la página 11-20

## Glosario de los diagramas UML de JBuilder

En la tabla siguiente se definen las carpetas del panel de estructura y los términos que aparecen en los diagramas. También se indica su representación en UML. Por ejemplo, las dependencias aparecen en la carpeta Dependencias del panel de estructura, y se representan en los diagramas UML por medio de una línea discontinua.

**Tabla 11.2** Definiciones de los diagramas UML

Término o elemento	Definición	Representación	Ejemplo de diagrama
Clases ampliadas	Clases de las que otra clase hereda los atributos (campos y propiedades) y métodos. También se denominan superclases, clases padre, clases base, clases principales y clases antecesoras.	Una línea continua con un gran triángulo que apunta de la subclase a la superclase. Aparece en la parte superior del diagrama UML.	↑
Clases	Estructuras que definen objetos. Las definiciones de clases definen campos y métodos.	Se muestran en un recuadro rectangular de fondo amarillo por defecto, con el nombre en la parte superior y los campos, métodos y propiedades por debajo.	
Clases abstractas	Clases de las que no se pueden crear instancias pero son antecesoras de otras clases.	Se muestran en cursiva.	<i>AWTEvent</i>
Clases herederas	Clases que amplían la superclase. También se denominan subclases, clases hijas, clases subordinadas y clases descendientes.	Una línea continua con un gran triángulo que apunta de la subclase a la superclase. Aparece en la parte inferior del diagrama UML.	↑

**Tabla 11.2** Definiciones de los diagramas UML (continuación)

Término o elemento	Definición	Representación	Ejemplo de diagrama
Clases de implementación	Clases que implementan la interfaz central.	Una línea discontinua con un gran triángulo que apunta de la clase de implementación a la interfaz heredada. Aparece en la parte inferior del diagrama UML.	
Interfaces ampliadas	Interfaces antecesoras heredadas por una descendiente.	Una línea continua con un gran triángulo que apunta de la subinterfaz a la interfaz antecesora. Aparece en la parte superior del diagrama UML.	
Interfaces	Grupos de constantes y declaraciones de métodos que definen la forma de una clase sin implementar los métodos. Las interfaces permiten establecer lo que debe hacer una clase, pero no la forma en que debe hacerlo.	Un rectángulo con el fondo naranja por defecto. El nombre de la interfaz se presenta en cursiva.	
Interfaces implementadas	Interfaces que implementa la clase central.	Una línea discontinua con un gran triángulo que apunta de la clase de implementación a la interfaz implementada. Aparece en la parte superior del diagrama UML.	
Dependencias directas e inversas	Relaciones en las que los cambios realizados en el objeto utilizado pueden influir sobre el que lo utiliza.	Una línea discontinua acabada en punta de flecha.	
Asociaciones directas e inversas	Dependencias especializadas en las que se almacena una referencia a otra clase.	Una línea continua acabada en punta de flecha.	
Paquetes	Conjuntos de clases relacionadas.	Un rectángulo con una pestaña en la parte superior y el nombre del paquete en esta pestaña o bajo ella. El paquete actual se muestra por defecto sobre un fondo verde brillante. Los demás paquetes se muestran por defecto sobre un fondo de un verde más oscuro.	

**Tabla 11.2** Definiciones de los diagramas UML (continuación)

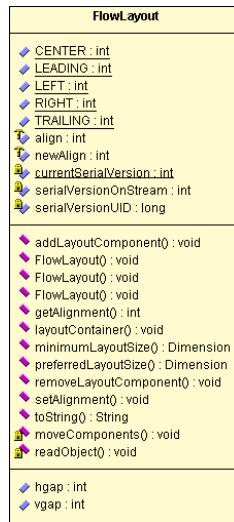
Término o elemento	Definición	Representación	Ejemplo de diagrama
Métodos	Operaciones definidas en una clase o una interfaz.	Se enumeran bajo los nombres y los campos, e incluyen el tipo de devolución.	 close(): void
Métodos abstractos	Métodos sin implementación.	Se muestran en cursiva.	 provideData(): void
Miembros / campos	Variables de instancia o miembros de datos de un objeto.	Se enumeran bajo el nombre de la clase, e incluyen el tipo de devolución.	 contentPane : JPanel
Propiedades	Las propiedades existen cuando el nombre de un método que coincide con el nombre de un campo va precedido de by "is", "get", or "set". Por ejemplo, un nombre de campo parameterRow con un método getParameterRow() constituye una propiedad.	Si se establece así en la ficha UML del cuadro de diálogo Opciones del IDE (Herramientas   Opciones del IDE), las propiedades se muestran por separado en la sección inferior del diagrama de clases.	 parameterRow : ReadWriteRow
Estático	Con ámbito de clase.	Los miembros, campos, variables y métodos estáticos se muestran subrayados en el diagrama UML.	 serialVersionUID : long

## Iconos de accesibilidad

En UML se utilizan iconos para representar la accesibilidad de las clases, como `public`, `private`, `protected` y `package`. En los diagramas se pueden utilizar los iconos de accesibilidad de JBuilder o los iconos UML estándar.

Los iconos de JBuilder son los que se utilizan en el código fuente, en el panel de estructura. Con el fin de utilizar los iconos de JBuilder en los diagramas UML, active la opción Utilizar iconos de accesibilidad de la ficha UML del cuadro de diálogo Opciones del IDE (Herramientas | Opciones del IDE). Esta opción se encuentra activada por defecto.

**Nota** Para conocer las definiciones de los iconos del panel de estructura, consulte “Iconos UML y panel de estructura de JBuilder en *Introducción a JBuilder*.

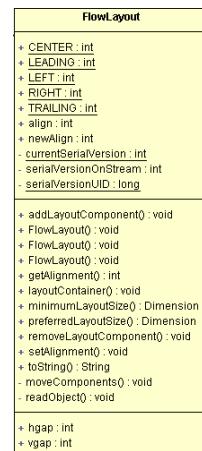
**Figura 11.5** Iconos de accesibilidad de JBuilder

En UML la accesibilidad de las clases se representa por medio de iconos más genéricos, definidos en la tabla siguiente.

**Tabla 11.3** Iconos de accesibilidad de UML

Icono UML	Descripción
+	public
-	private
#	protected

Con el fin de utilizar los iconos estándar de UML en los diagramas, desactive la opción Utilizar iconos de accesibilidad de la ficha UML del cuadro de diálogo Opciones del IDE (Herramientas | Opciones del IDE).



# Presentación de los diagramas UML

---

JBuilder proporciona un visualizador UML con el que se puede presentar el código con diagramas UML. El visualizador UML, que se abre desde la pestaña UML del panel de contenido, muestra los diagramas de clases y paquetes estándar de UML. Cuando selecciona la pestaña UML, JBuilder carga los archivos de clase para determinar sus relaciones que el visualizador UML utilizará para obtener la información de paquete y clase para los diagramas.

Para obtener un diagrama UML actualizado y exacto, siempre es mejor compilar antes de seleccionar la pestaña UML. El visualizador UML muestra los archivos fuente de Java de forma dinámica aunque no se hayan compilado, pero sólo si se encuentran en la vía de acceso a archivos fuente. En el visualizador de UML aparece un mensaje que indica que es posible que el diagrama UML no sea exacto. Sin embargo, si un archivo no está en la vía de acceso a archivos fuente, es necesario crear antes el archivo .class. Se muestra un mensaje en el que se solicita que se compile el proyecto con el fin de crear los archivos de clase para el diagrama UML. Si los archivos de clase son antiguos (por ejemplo, el archivo fuente se ha modificado pero no se ha vuelto a realizar la compilación), en el Visualizador de UML aparece un mensaje que indica que es posible que el diagrama UML no sea exacto.

El visualizador UML también acepta la representación en diagramas de dependencias inversas, de clases a JSP (páginas JavaServer). Por ejemplo, un bean generado por el Asistente para JSP conduce a la JSP que lo utiliza. No es necesario que se trate de un bean de JSP; puede ser cualquier clase utilizada por la página JSP.

JBuilder no incluye referencias de bibliotecas de proyecto en el diagrama UML, excepto si se activa la opción Incluir referencias de archivos de clase de las bibliotecas del proyecto de la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto). Consulte “[Inclusión de referencias de bibliotecas de proyecto](#)” en la página 11-20.

## El visualizador UML de JBuilder

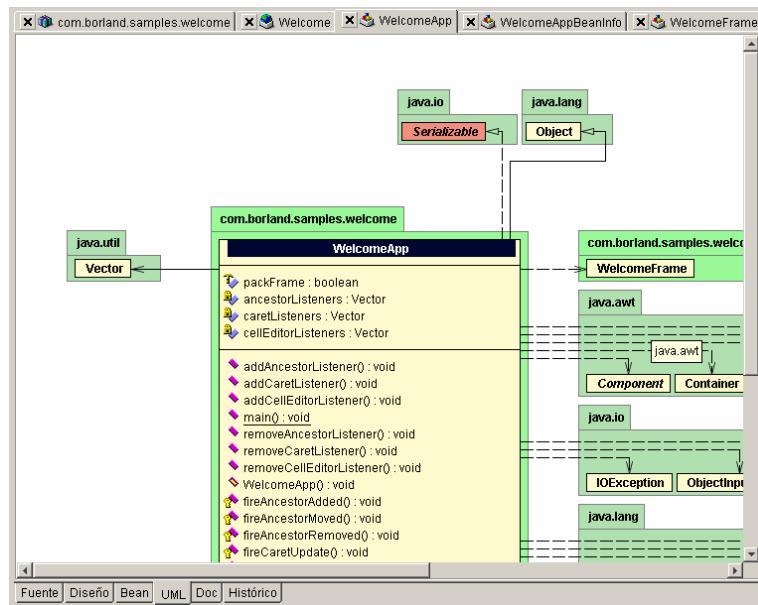
---

El visualizador UML de JBuilder proporciona diversas funciones para personalizar la presentación de los diagramas, desplazarse por ellos y por el código fuente, para mostrar las clases internas, el código fuente y el Javadoc, para crear e imprimir imágenes y para perfeccionar el código.

Para ver un diagrama UML en JBuilder, abra un paquete o una clase y pulse la pestaña UML del panel de contenido.

**Nota** Si el proyecto es voluminoso, la primera presentación del diagrama UML puede llevar bastante tiempo. Antes de generar los diagramas, JBuilder debe cargar las clases para determinar sus relaciones.

**Figura 11.6** Visualizador UML



## Presentación de diagramas de paquetes

Para ver un diagrama limitado a las dependencias de paquetes:

- 1 Elija Proyecto | Ejecutar Make del proyecto o Proyecto | Generar el proyecto para llevar a cabo la compilación.
- 2 Haga doble clic en el nodo del paquete, en el panel del proyecto, o haga clic con el botón derecho del ratón y elija Abrir.
- 3 Para ver el diagrama del paquete, abra la pestaña UML del panel de contenido.

**Nota** Si el nodo del paquete no está disponible, elija Proyecto | Propiedades de proyecto | General y habilite la opción Activar la localización y compilación de paquetes fuente.

## Presentación de diagramas de clases

Para ver un diagrama combinado de clases:

- 1 Elija Proyecto | Ejecutar Make del proyecto con el fin de realizar la compilación del proyecto o el archivo Java.

- 2** Haga doble clic en un archivo Java, en el panel del proyecto, o haga clic con el botón derecho del ratón y elija Abrir.
- 3** Abra la pestaña UML de la parte inferior del panel de contenido para ver el diagrama UML de clase.

## Presentación de las clases internas

---

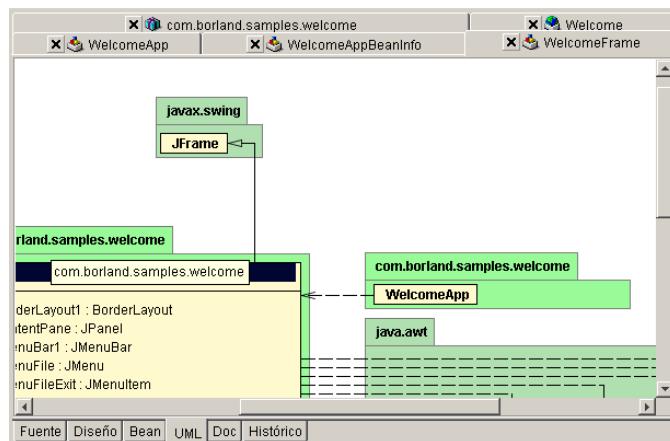
Una clase puede contener varias clases, incluidas las clases internas y las clases internas anónimas. Si ocurre esto, el visualizador UML presenta una interfaz de usuario con una clase por pestaña.

Sólo se muestra el diagrama de una clase interna anónima si el cursor del editor se coloca en ella o si se va a ella desde otro diagrama. Para colocar el cursor en el editor, abra la pestaña Fuente de un archivo fuente abierto, sitúe el cursor en el lugar deseado y abra la pestaña UML.

Las clases internas anónimas seleccionadas se guardan en memoria hasta que se cierra el archivo, de manera que puedan acumularse en forma de pestañas en el visualizador UML. Las dependencias de las clases internas anónimas se incluyen en las clases que las contienen.

El visualizador UML utiliza la posición del cursor en el editor para determinar la clase, el método o el campo seleccionado. No obstante, si no se cambia esta posición del cursor en las visitas posteriores al visualizador, se conserva la última selección. En los campos y los métodos, el cursor debe encontrarse entre el primer carácter y el último de la definición, y no al principio de la línea.

**Figura 11.7** Presentación de las clases internas



## Presentación del código fuente

---

En los diagramas de clases es posible desplazarse al código fuente y volver al diagrama UML. Haga doble clic en la clase central, en un método, en un campo o en una propiedad para ver su archivo de código fuente. El cursor del editor se coloca en el lugar adecuado. Cuando se sitúa el cursor en una clase, un método, un campo o una propiedad, en el editor, el elemento se resalta en el diagrama UML. En el editor, abra la pestaña UML para volver al diagrama UML.

El visualizador UML tiene una opción de presentación del código en el menú contextual: Ir a código fuente. Seleccione Ir a código fuente para presentar el código en el editor. Esto puede resultar útil para examinar el código fuente de otras clases e interfaces en los diagramas de clases y paquetes.

El visualizador UML también proporciona ayuda inmediata en la que se muestra la lista de argumentos de los métodos. Sitúe el ratón sobre un método para ver su ayuda inmediata. Si se coloca el puntero en el nombre de una clase aparece su nombre completo, que incluye el nombre del paquete.

## Visualización de Javadoc

---

Existen varias formas de acceder al Javadoc de paquetes, interfaces, clases, métodos y campos desde los diagramas UML.

- Seleccione un elemento en el diagrama UML, haga clic con el botón derecho del ratón y elija Ver Javadoc.
- Seleccione un elemento en el diagrama UML y pulse *F1*.
- Seleccione un elemento en el panel de estructura y pulse *F1*.

El Visualizador de ayuda de JBuilder muestra el Javadoc, que se genera a partir de los comentarios Javadoc del archivo de código fuente o a partir de la información disponible, como las firmas de los métodos.

**Nota** Si se ha ejecutado Javadoc con la herramienta **javadoc** o el Asistente para Javadoc, se incluye más información.

### Consulte

- “[Visualización de Javadoc](#)” en la página 14-22

## Uso del menú contextual

---

El visualizador UML tiene un menú contextual que permite acceder rápidamente a los comandos habituales. Para abrirlo, haga clic con el botón derecho del ratón en un elemento del visualizador UML. Si desea información sobre estos comandos y su función, consulte la documentación correspondiente:

- Buscar referencias: “[Obtención de información sobre un símbolo antes del perfeccionamiento](#)” en la página 12-8
- Renombrar “[Perfeccionamiento por cambio de nombre](#)” en la página 12-2
- Desplazar: “[Perfeccionamiento por desplazamiento](#)” en la página 12-4
- Cambiar parámetro: “[Cambio de parámetros de métodos](#)” en la página 12-24
- Guardar diagrama: “[Creación de imágenes de diagramas UML](#)” en la página 11-21
- Activar filtrado de clases: “[Filtrado de paquetes y clases](#)” en la página 11-19
- Ir a diagrama: “[Desplazamiento por diagramas](#)” en la página 11-16
- Ir a código fuente: “[Presentación del código fuente](#)” en la página 11-14
- Mostrar Javadoc: “[Visualización de Javadoc](#)” en la página 11-14

## Desplazamiento de la vista

---

Existen varias maneras de desplazar el diagrama UML dentro del visualizador UML:

- Hacer clic y arrastrarlo con el ratón
- Por medio de las teclas Arriba y Abajo
- Con los cursores
- Con las barras de desplazamiento

El comportamiento varía dependiendo del tipo de vista que aparezca. Con Ver | Ocultar todo se obtiene una vista completa y sólo muestra el panel de contenido. Con Ver | Mostrar todo aparecen los siguientes paneles por defecto: panel del proyecto, panel de contenido y panel de estructura.

## Vista completa

En la vista completa (Ver | Ocultar todo), puede utilizar el ratón para desplazar la vista hacia arriba y hacia abajo. Pulse sobre el fondo del diagrama, haga clic y arrastre el diagrama. Las teclas *RePag* y *AvPag*, así como los cursores arriba y abajo, también pueden desplazar la vista hacia

arriba y hacia abajo. La vista también se puede desplazar de forma manual por medio de las barras de desplazamiento.

## Vista parcial

En la vista parcial (Ver | Mostrar todo), puede arrastrar el diagrama en todas las direcciones. Pulse en el fondo del diagrama, haga clic y arrastre el diagrama en cualquier dirección. Con las teclas *RePag* y *AvPag* se puede mover la vista hacia arriba y hacia abajo. También se pueden usar los cuatro cursores para mover la vista en cualquier dirección. La vista también se puede desplazar de forma manual por medio de las barras de desplazamiento.

## Actualización de la vista

---

Si desea actualizar los diagramas UML después de realizar cambios en el proyecto, utilice uno de estos métodos:

- Genere el proyecto de nuevo (Proyecto | Generar el proyecto).
- Pulse el botón Actualizar de la barra de herramientas del panel del proyecto.



## Desplazamiento por diagramas

---

Haga doble clic en el nombre de una clase o un paquete en el diagrama a fin de ver su diagrama UML. Cuando se selecciona un elemento en el diagrama UML, cambia su color de fondo. Después de realizar una selección se pueden utilizar las teclas de flecha para subir y bajar por el diagrama. Si no hay ningún elemento seleccionado, las teclas *Re Pág* y *Av Pág* desplazan el diagrama arriba y abajo. Si desea ver diagramas UML mostrados previamente, pulse los botones Adelante y Atrás de la barra de herramientas principal del visualizador.

También es posible desplazarse seleccionando clases y paquetes en el panel de estructura. Para seleccionar una clase o un paquete, haga clic sobre él. Haga doble clic en una clase para ver su diagrama. Haga clic con el botón derecho en un paquete y elija Abrir para presentar su diagrama.

En el menú contextual del visualizador UML también hay una opción de presentación: Ir a diagrama. Haga clic con el botón derecho del ratón en el nombre de un paquete, una clase o una interfaz, en el diagrama UML, y elija Ir a diagrama.

### Consulte

- “UML y el panel de estructura” en la página 11-17

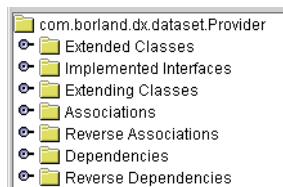
## UML y el panel de estructura

---

El panel de estructura, situado en la esquina inferior izquierda del diagrama UML, proporciona una vista en árbol de las relaciones que contienen las carpetas ampliables, por categorías, como Clases ampliadas y Dependencias. Si alguna de las categorías no está incluida en el diagrama, la carpeta correspondiente no aparece. Estas carpetas permiten desplazarse a otros diagramas, y pueden proporcionar información que no aparece en ellos, ya que reflejan las relaciones independientemente de la configuración de filtrado y cualquier otra restricción. Por ejemplo, aunque se hayan filtrado determinadas clases y paquetes siguen apareciendo en el panel de estructura. Si desea más información sobre el filtrado de diagramas UML, consulte “[Personalización de diagramas UML](#)” en la página 11-18. Si desea ampliar y contraer iconos de carpeta en el panel de estructura, haga doble clic en ellos o pulse el ícono comutador de ampliación.

Los paquetes y clases que no se muestran en el diagrama se muestran con un color más suave en el panel de estructura. No se muestran si se han filtrado o si son redundantes y se han eliminado para mayor claridad.

**Figura 11.8** Panel de estructura de los diagramas UML



El panel de estructura también se puede utilizar para seleccionar y dirigirse a clases y paquetes. Si se selecciona una clase, una interfaz o un paquete en el panel de estructura, queda seleccionado en el diagrama. Si se hace doble clic en una clase o un paquete en el panel de estructura se accede a su diagrama UML. Haga doble clic en un paquete y elija Abrir para presentar el diagrama UML de paquetes.

Si desea buscar rápidamente un paquete o una clase en el panel de estructura, pase el foco al árbol y empiece a escribir el nombre deseado. Si desea obtener más información, consulte el tema “Búsqueda en árboles” del capítulo “El entorno de JBuilder” de *Introducción a JBuilder*.

## Diagramas de paquetes

Las carpetas de los diagramas de paquetes pueden incluir cualquiera de los siguientes elementos o todos ellos:

- Dependencias
- Dependencias inversas

Para ver las definiciones de estos términos, consulte “[Glosario de los diagramas UML de JBuilder](#)” en la página 11-7.

Cuando se abre un paquete dependiente se muestran todas sus clases y las relaciones que tienen con el paquete central. Esto permite saber cuáles son las clases que originan la dependencia.

## Diagramas de clases

Las carpetas de los diagramas de clases pueden incluir cualquiera de los siguientes elementos o todos ellos:

- Clases ampliadas
- Interfaces ampliadas
- Interfaces implementadas
- Clases herederas
- Clases de implementación
- Asociaciones
- Asociaciones inversas
- Dependencias
- Dependencias inversas

Para ver las definiciones de estos términos, consulte “[Glosario de los diagramas UML de JBuilder](#)” en la página 11-7.

# Personalización de diagramas UML

---

Aunque no es posible manipular los diagramas UML desplazando o cambiando de tamaño sus elementos, es posible personalizar la presentación en los cuadros de diálogo Propiedades de proyecto y Opciones del IDE. Por ejemplo, se pueden filtrar proyecto por proyecto los elementos que aparecen en los diagramas, y también se pueden incluir referencias de bibliotecas de proyecto. También se puede personalizar la presentación general del diagrama UML mediante la configuración del orden, la fuente, los colores y otras opciones.

## Definición de las propiedades del proyecto

---

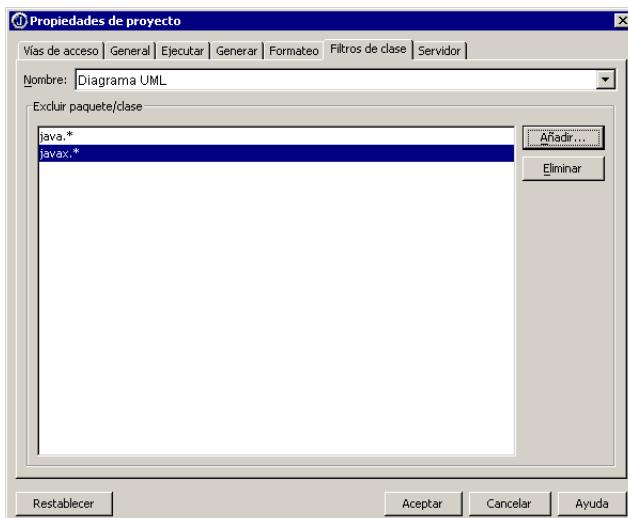
En el cuadro de diálogo Propiedades de proyecto se pueden configurar diversas propiedades de los diagramas UML.

- Filtrado: disponible en la ficha Filtros de clase
- Referencias de biblioteca: disponible en la ficha General
- Referencias del código generado: disponible en la ficha General

Para abrir el cuadro de diálogo Propiedades de proyecto, elija Proyecto | Propiedades de proyecto o haga clic con el botón derecho del ratón en el archivo del proyecto, en el panel del proyecto, y elija Propiedades.

## Filtrado de paquetes y clases

En la ficha Filtros de clase del cuadro de diálogo Propiedades de proyecto, puede excluir los paquetes y clases de los diagramas UML del proyecto. Seleccione Proyecto | Propiedades de proyecto y, a continuación, abra la pestaña Filtros de clase. Seleccione Diagrama UML en la lista desplegable Nombre. A continuación, pulse Añadir si desea añadir clases o paquetes a la lista de exclusiones. Las clases y paquetes de la lista se excluyen del diagrama UML.



Vuelva al diagrama UML y seleccione el botón Actualizar de la barra de herramientas del panel del proyecto. Observe que las clases y paquetes de la lista Excluir paquete/clase se excluyen del diagrama, pero todavía se puede acceder a ellas en el panel de estructura. Para desactivar provisionalmente el filtrado de paquetes y clases aplicado al diagrama UML, haga clic en él con el botón derecho del ratón y desactive la opción Activar filtrado de clases en el menú contextual del visualizador UML.

### Importante

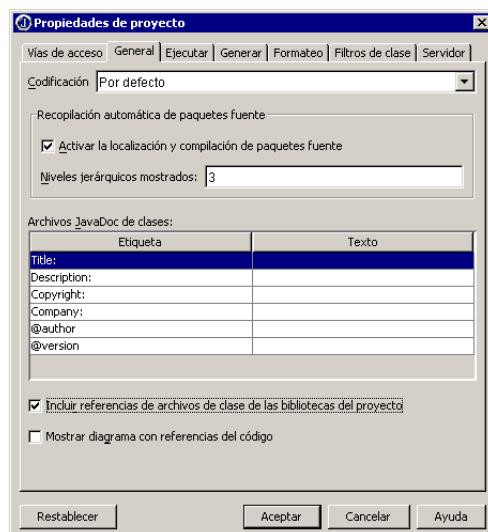
Si se han definido filtros en el cuadro de diálogo Propiedades de proyecto, se filtran todos los diagramas del proyecto. La desactivación del filtrado desde el menú contextual de un diagrama no se aplica a los demás. Si se desplaza a otro diagrama del proyecto, el filtrado sigue activado. Una vez que cierre el archivo o paquete, la configuración vuelve a la configuración a nivel del proyecto.

## Inclusión de referencias de bibliotecas de proyecto

Habitualmente, las bibliotecas proporcionan servicios a las aplicaciones que se construyen a partir de ellas, pero no saben nada acerca de sus usuarios. Para mostrar estas relaciones, necesita incluir referencias de las bibliotecas.

Con el fin de incluir las referencias de biblioteca en los diagramas UML, active la opción Incluir referencias de archivos de clase de las bibliotecas del proyecto de la ficha General del cuadro de diálogo Propiedades de proyecto. Por defecto, esta opción está desactivada y se excluyen las dependencias inversas entre las bibliotecas y los proyectos.

- Nota** Si el proyecto es muy voluminoso, esta opción puede aumentar significativamente el tiempo que tarda JBuilder en cargar las clases y crear el diagrama UML.



### Consulte

- “[Paso 4: Añadir referencias de bibliotecas](#)” en la página 20-10 en el Capítulo 20, “Tutorial: Visualización de código con el visualizador UML”.

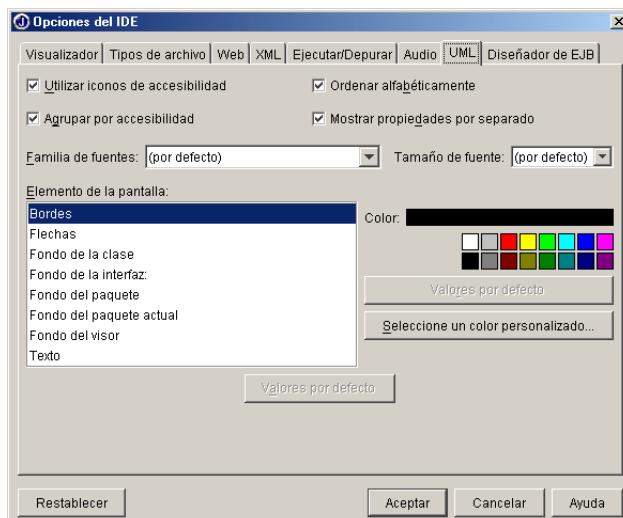
## Inclusión de referencias del código generado

En los diagramas UML también se pueden incluir referencias del código fuente generado, como archivos IIOP y stubs EJB. Para ello, seleccione la opción Mostrar diagrama con referencias del código en la ficha General de Propiedades de proyecto.

## Configuración de las opciones del IDE

La ficha UML del cuadro de diálogo Opciones del IDE (Herramientas | Opciones del IDE) proporciona opciones destinadas a la personalización general de los diagramas en el visualizador UML de JBuilder. Si desea acceder a la ficha UML, seleccione Herramientas | Opciones del IDE y haga clic en la pestaña UML.

Aquí se pueden modificar los iconos de accesibilidad, el orden, la agrupación, la presentación de propiedades, el tipo y tamaño de letra y el color de los distintos elementos de la pantalla de los diagramas UML. Si desea más información, pulse el botón Ayuda de la ficha UML.



### Consulte

- “Personalización del IDE de JBuilder” en *Introducción a JBuilder*

## Creación de imágenes de diagramas UML

El visualizador UML permite guardar diagramas UML como imágenes. Sin embargo, el tamaño de la imagen y el número de colores pueden constituir una limitación. En ese caso se muestra un mensaje de error. JBuilder acepta el formato de imagen PNG (Portable Network Graphics, gráficos portátiles de red).

Para guardar el diagrama UML como imagen, haga clic con el botón derecho del ratón en el visualizador UML y elija Guardar diagrama. Escriba un nombre de archivo en el cuadro de diálogo Guardar diagrama. Se le añade automáticamente la extensión .png.

## Impresión de diagramas UML

Si desea obtener una copia impresa del diagrama UML, pulse Imprimir en la barra de herramientas principal o elija el comando Imprimir (Archivo | Imprimir). El comando Configurar página (Archivo | Configurar página) permite definir el encabezado, los márgenes y la orientación. El diagrama se imprime a un tamaño ligeramente inferior al que se muestra en la pantalla. Si el diagrama no cabe en una sola página, se imprime en varias.

**Importante** Para que la opción de impresión esté disponible es necesario desplazar el foco al diagrama UML.

## Perfeccionamiento y Buscar referencias

Es posible acceder a las funciones de perfeccionamiento de JBuilder desde el visualizador UML. Existen varias formas de acceder al perfeccionamiento desde el visualizador UML:

- Haga clic con el botón derecho del ratón en el nombre de un paquete, una clase, un campo, un método o una propiedad, en el diagrama UML, y elija Cambiar nombre en el menú contextual.
- Seleccione en el diagrama UML el nombre de un paquete, una clase, un campo o un método, y pulse *Intro*. Escriba el nuevo nombre en el cuadro de diálogo Cambiar nombre.
- Haga clic con el botón derecho del ratón en el nombre de una clase, en el diagrama UML, y elija Mover en el menú contextual.
- Haga clic con el botón derecho del ratón en el nombre de un método, en el diagrama UML, y elija Cambiar parámetros en el menú contextual.

Antes de realizar el perfeccionamiento puede ser conveniente buscar todos los archivos fuente que utilizan un símbolo seleccionado. Para localizar todas las referencias a un símbolo, selecciónelo en un diagrama UML o en el editor. Haga clic con el botón derecho en el símbolo y elija Buscar referencias.

### Consulte

- [Capítulo 12, “Perfeccionamiento de símbolos de código”](#)
- [“Búsqueda de referencias a símbolos” en la página 12-9](#)

# 12

## Perfeccionamiento de símbolos de código

Es una función de JBuilder SE y Enterprise.

El código evoluciona según los cambios de la tecnología y las demandas del mercado. Es posible que resulte necesario modificar el código creado anteriormente con el fin de hacer posibles las ampliaciones, mejorar el rendimiento, adaptarlo a nuevas necesidades o, simplemente, depurar la base. El término *perfeccionamiento (refactoring)* designa la realización de alteraciones en el código fuente sin que cambie su comportamiento durante la ejecución a los ojos del usuario. También incluye las distintas tareas que constituyen el proceso de rediseño.

El perfeccionamiento se puede realizar a pequeña y gran escala, pero incluso los cambios mínimos pueden provocar errores. Con el fin de que el perfeccionamiento resulte eficaz debe ser correcto y completo. Un solo cambio puede desencadenar permutaciones en toda la base de código. Si el perfeccionamiento es correcto se gestiona todo el conjunto de permutaciones de forma razonable y coherente: ningún comportamiento debe cambiar si no es con vistas a la mejora del rendimiento o la capacidad de mantenimiento, y no se debe introducir ningún error.

### Tipos de perfeccionamiento

JBuilder proporciona muchos tipos de perfeccionamiento (refactoring):

- Optimizar importaciones.
- Perfeccionamiento por cambio de nombre.
- Perfeccionamiento por desplazamiento.
- Cambiar parámetros.
- Extraer método.
- Introducir variables.

- Insertar en sentencia try/catch.

Se accede a las opciones de perfeccionamiento desde el menú contextual del editor, desde el menú Editor, desde el menú contextual del panel de estructura y desde un menú contextual de los diagramas UML. Tenga en cuenta que no es posible realizar tarea de perfeccionamiento de un proyecto a otro.

**Nota** Si perfecciona un archivo EJB, recibirá la siguiente advertencia:

ADVERTENCIA: Está perfeccionado un archivo EJB. Esto puede hacer que tenga que cambiar el código fuente y el descriptor de distribución a mano. Le recomendamos que utilice el diseñador de EJB para la mayoría de los casos de perfeccionamiento.

Necesitará actualizar todos los archivos fuente relevantes para que acepten el perfeccionamiento. (El desarrollo de EJB es una función de JBuilder Enterprise.)

## Optimizar importaciones

---

Cuando se realiza el perfeccionamiento por optimización de las importaciones, las sentencias `import` se reescriben y se reorganizan según la configuración particular de las propiedades del proyecto. También se eliminan las sentencias `import` que ya no se utilicen en el código.

Si desea información sobre la forma de optimizar las importaciones en JBuilder, consulte “[Optimizar importaciones](#)” en la página 12-15.

## Perfeccionamiento por cambio de nombre

---

El perfeccionamiento (refactoring) por cambio de nombre consiste en aplicar un nombre nuevo a un paquete, una clase, un método, un campo, una variable local o una propiedad, asegurándose de que todas las referencias a este nombre se gestionan correctamente. Cuando se aplica este tipo de perfeccionamiento a un constructor, el nombre de la clase cambia.

El perfeccionamiento por cambio de nombre no es una simple tarea de buscar y reemplazar, ya que se deben tener en cuenta todas las referencias y se deben gestionar correctamente. También es necesario reconocer las pautas para gestionar correctamente los nombres sobrecargados. Por ejemplo, si se aplica el perfeccionamiento por cambio de nombre a un nombre de clase sobrecargado, el nuevo nombre se debe reflejar en la declaración de la clase, en todas sus instancias y en cualquier otra referencia a ella. Sin embargo, el nombre nuevo sólo se debe reflejar en la clase de destino, y no en las otras clases que comparten su nombre original o sus declaraciones, instancias, referencias, etc.

En los paquetes, el perfeccionamiento por cambio de nombre se aplica al paquete especificado. Se actualizan el paquete y las sentencias de importación de los archivos de clase. El paquete, los subpaquetes y los archivos fuente de clase se desplazan al nuevo directorio fuente, y el antiguo se borra.

En JBuilder es posible perfeccionar por cambio de nombre los siguientes símbolos de código.

**Tabla 12.1** Perfeccionamiento y símbolos de código

Símbolo de código	Descripción
Paquete	Cuando se aplica el perfeccionamiento por cambio de nombre a un paquete cambia su nombre, así como el de todo el subárbol de paquetes. No se puede utilizar un nombre de paquete que ya exista en el proyecto.
Clase, clase interna o interfaz	Cuando se aplica el perfeccionamiento por cambio de nombre a una clase pública externa cambia el nombre del archivo fuente. Si este nombre ya existe en el paquete actual no se permite el perfeccionamiento. Si la clase no es pública externa y hay otra clase con el nuevo nombre, el cambio no se realiza.
Método	Cuando se aplica el perfeccionamiento por cambio de nombre a un método cambia su nombre y el de todas sus referencias. El método se puede cambiar de nombre en todas las clases de las que ésta es heredera o en todas las clases de su jerarquía. Es posible crear un método de reenvío.
Campo	Cuando se aplica el perfeccionamiento por cambio de nombre a un campo cambia su nombre. El nombre nuevo no puede existir en la clase en la que se declaraba el original. Si hay conflictos de ámbito entre el nombre nuevo y el antiguo, se antepone al primero la palabra clave <code>this</code> . Si el nombre nuevo redefine un campo existente o es redefinido por él, en una clase o una subclase, se muestra una advertencia.
Variable local	Cuando se aplica el perfeccionamiento por cambio de nombre a una variable local, cambia su nombre. El nombre nuevo no puede existir en la clase en la que se declaraba el original. Si se produjera un conflicto con el nombre de una nueva variable, el nombre de la variable local se antepone al nombre del campo.
Propiedad	Cuando se aplica el perfeccionamiento por cambio de nombre a una propiedad, cambia su nombre, así como el de sus métodos de obtención y definición. El nombre nuevo no puede existir en la clase en la que se declaraba el original.

Si desea información sobre la forma de perfeccionar por cambio de nombre en JBuilder, consulte:

- “Perfeccionamiento por cambio de nombre de paquetes” en la página 12-18
- “Perfeccionamiento por cambio de nombre de clases” en la página 12-19

- “Perfeccionamiento por cambio de nombre de métodos” en la página 12-21
- “Perfeccionamiento por cambio de nombre de campos” en la página 12-23
- “Perfeccionamiento por cambio de nombre de variables locales” en la página 12-22
- “Perfeccionamiento por cambio de nombre de propiedades” en la página 12-24

## Perfeccionamiento por desplazamiento

---

En JBuilder es posible aplicar el perfeccionamiento por desplazamiento a las clases. Este tipo de perfeccionamiento (refactoring) consiste en desplazar la clase indicada a otro paquete. El perfeccionamiento por desplazamiento sólo se permite en las clases públicas de alto nivel. El paquete al que se desplaza la clase no puede contener un archivo fuente con el nuevo nombre. En el proceso de perfeccionamiento se actualizan la declaración y los usos de la clase.

Si desea información sobre la forma de perfeccionar por movimiento de clase en JBuilder, consulte “Perfeccionamiento por desplazamiento de clases” en la página 12-20.

## Cambiar parámetros

---

El cambio de parámetros permite añadir, cambiar de nombre, eliminar y reorganizar los parámetros de un método. Es posible modificar un parámetro recién añadido antes de cerrar el diálogo, pero no se pueden modificar los parámetros ya creados.

Para obtener más información sobre cómo cambiar parámetros de métodos, consulte “Cambio de parámetros de métodos” en la página 12-24.

## Extraer método

---

La extracción de métodos convierte un fragmento de código seleccionado en un método. JBuilder extrae el código del método actual, determina los parámetros necesarios, genera las variables locales que procedan y determina el tipo de devolución. Introduce una llamada al nuevo método en el lugar donde se encontraba el fragmento de código.

Para obtener más información sobre cómo extraer métodos, consulte “Extracción de métodos” en la página 12-26.

## Introducir variables

---

El perfeccionamiento por introducción de variables permite sustituir el resultado de una expresión compleja o una parte de ésta por un nombre de variable temporal que explica el propósito de la expresión o la subexpresión.

Si desea información sobre cómo introducir una variable en JBuilder, consulte “[Introducción de variables](#)” en la página 12-27.

## Insertar en sentencia try/catch

---

Este tipo de perfeccionamiento consiste en añadir una sentencia try/catch alrededor del bloque de código seleccionado. Detecta todas las excepciones activas del bloque y añade un bloque para cada una de ellas.

Si desea información sobre como añadir una sentencia try/catch en un bloque de código, consulte “[Perfeccionamiento con sentencia try/catch](#)” en la página 12-28.

# Herramientas de perfeccionamiento de JBuilder

---

Se accede a las opciones de perfeccionamiento de JBuilder desde los menús contextuales del editor y los diagramas UML. También hay comandos de perfeccionamiento en los menús Edición y Buscar. Si desea más información sobre el editor, consulte “El editor” en *Introducción a JBuilder*. Para obtener más información sobre UML, consulte el [Capítulo 11, “Presentación de código con UML”](#). (UML es una característica de JBuilder Enterprise.)

Antes de realizar el perfeccionamiento se pueden consultar, por categoría, todas las ubicaciones del proyecto actual en las que se hace referencia al símbolo seleccionado. También es posible desplazarse a la definición del símbolo. Si JBuilder no puede completar el perfeccionamiento, el IDE presenta mensajes de advertencia y error con los que se pueden investigar los motivos. Las advertencias no detienen el perfeccionamiento. Sin embargo, los errores sí. Por ejemplo, un proceso de perfeccionamiento se puede impedir si un archivo es de sólo lectura (aún no se ha extraído) o si ya existe el nombre del símbolo.

- Nota** Los perfeccionamientos de un solo archivo (por ejemplo, Extraer método e Introducir variable) no generan salida, a menos que se produzca un error o una advertencia.

Las herramientas de perfeccionamiento de JBuilder proporcionan gran cantidad de información, que incluye:

- Informe de limitaciones

JBuilder comprueba la existencia de condiciones que puedan provocar problemas en el perfeccionamiento. Por ejemplo, JBuilder determina si la información necesaria sobre las dependencias no está disponible o es antigua, si un archivo es de sólo lectura y si un archivo de clase no existe.

- Detección de referencias

JBuilder busca todos los archivos fuente que contienen dependencias. Se localiza la posición exacta de la fuente.

- Validación

JBuilder determina si el nombre nuevo es válido. Por ejemplo, es posible que ya exista o que su sintaxis sea inadecuada.

- Actualización del árbol de fuentes

Cuando se perfecciona por desplazamiento una clase y cuando se perfecciona por cambio de nombre un paquete, JBuilder desplaza los directorios o los archivos dentro del árbol de fuentes. También actualiza las sentencias de importación necesarias para las dependencias.

- Cambio de nombre de referencias

JBuilder asigna a las referencias el nombre nuevo.

## Configuración para la detección y el perfeccionamiento de referencias

---

Para buscar todas las referencias a un símbolo es necesario realizar la compilación con las referencias de bibliotecas de proyecto activadas. Esto se comprueba en la pestaña General del cuadro de diálogo Propiedades del proyecto. La opción Incluir referencias de archivos de clase de las bibliotecas del proyecto debe encontrarse activada. Esta opción carga todas las relaciones de biblioteca y permite que JBuilder detecte las referencias.

**Nota** No es necesario activar esta opción, ya que podría reducir la velocidad de compilación y perfeccionamiento. Sin embargo, si se desactiva, JBuilder no puede detectar todas las referencias necesarias para el comando Buscar referencias (si desea más información, consulte “[Búsqueda de referencias a símbolos](#)” en la página 12-9).

Además, el proyecto debe estar actualizado, esto es, la marca horaria de los archivos de clase debe coincidir con la de los archivos fuente. Para

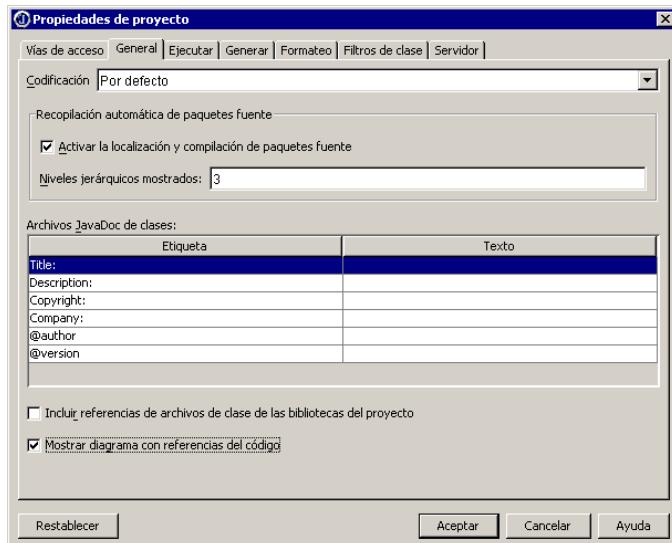
garantizar que el proyecto esté actualizado, compílelo por medio del comando Proyecto | Ejecutar Make del proyecto.

Para configurar JBuilder para la detección y el perfeccionamiento de referencias:

**1** Elija Proyecto | Propiedades de proyecto y se abrirá el cuadro de diálogo Propiedades de proyecto.

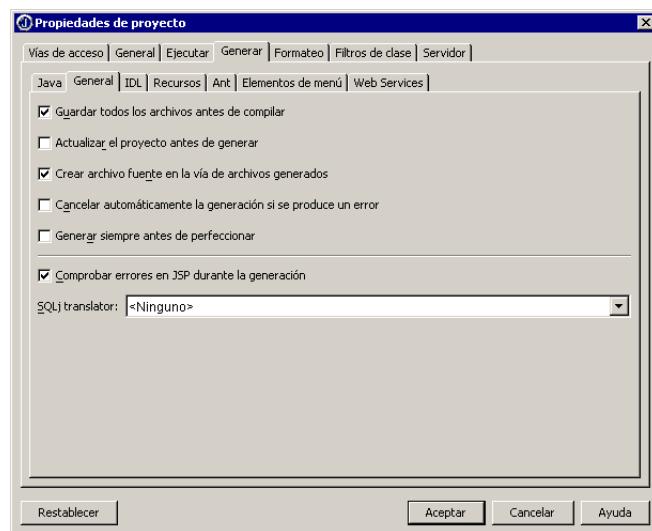
**2** Abra la pestaña General. Active la opción Incluir referencias de archivos de clase de las bibliotecas del proyecto.

La pestaña General del cuadro de diálogo Propiedades de proyecto tiene el siguiente aspecto:



**3** Para generar siempre el proyecto antes de realizar el perfeccionamiento, seleccione la pestaña Generar del cuadro de diálogo Propiedades de proyecto. A continuación, abra la pestaña General y seleccione la opción Generar siempre antes de perfeccionar. Esta opción está desactivada por defecto. Si selecciona esta opción, el perfeccionamiento es más lento, pero se detectan todos los casos. Si no la selecciona, el perfeccionamiento es más rápido, pero puede haber elementos específicos que no se detecten en el perfeccionamiento.

La pestaña General de la ficha Generar de Propiedades de proyecto tiene el siguiente aspecto:



4 Pulse Aceptar para cerrar el cuadro de diálogo.

- 5 Elija Archivo | Guardar todo o pulse el botón Guardar todo de la barra de herramientas.
- 6 Elija Proyecto | Ejecutar Make del proyecto para compilar todo el proyecto y cargar todas las referencias.

## Obtención de información sobre un símbolo antes del perfeccionamiento

JBuilder proporciona varias formas de obtener información sobre un símbolo antes de realizar el perfeccionamiento. Permite buscar su definición y todas sus referencias, esto es, todos los archivos fuente que lo utilizan.

### Búsqueda de definiciones de símbolos

El comando Buscar definición del menú Buscar y del menú contextual permite localizar la definición de los símbolos. Para buscar la definición de un símbolo:

- 1 Compile el proyecto.
- 2 Seleccione el símbolo en el editor.

**3** Haga clic con el botón derecho en el símbolo y elija Buscar definición.

El archivo fuente donde se define el símbolo se abre en el editor.

- Si el símbolo es una instancia de una clase, el cursor se desplaza hasta la definición de la instancia.
- Si el símbolo es un método, la clase en la que se define se abre en el editor, con el cursor en el principio de la firma del método.
- Si el símbolo es una variable que se define en la clase abierta, el cursor se desplaza a la definición. Si la variable es pública y se define en otra clase, ésta se abre en el editor, con el cursor en la definición.

**Importante** Para que sea posible buscar definiciones, el proyecto debe estar compilado. La clase que incluye la definición debe encontrarse en la vía de acceso `import` del mismo paquete que el símbolo.

## Búsqueda de referencias a símbolos

Antes de realizar el perfeccionamiento puede ser conveniente buscar todos los archivos fuente que utilizan un símbolo seleccionado. Para localizar todas las referencias a un símbolo:

- 1** Compile el proyecto.
- 2** Seleccione el símbolo en el editor o en el panel de estructura.
- 3** Haga clic con el botón derecho en el símbolo y elija Buscar referencias, o elija Buscar | Buscar referencias.

Las referencias se muestran en la pestaña Buscar del panel de mensajes siguiendo el orden en el que se han ido encontrando. Las referencias a clases y métodos se ordenan por categoría. Las referencias a campos y variables locales se ordenan por nombre de archivo. No es posible buscar las referencias de paquetes ni propiedades.

**Nota** Si busca información sobre un símbolo y JBuilder no abre un archivo fuente en el editor ni muestra la pestaña Resultado de la búsqueda, es posible que el proyecto no esté compilado con referencias de las bibliotecas de proyecto. Para obtener más información, consulte “[Configuración para la detección y el perfeccionamiento de referencias](#)” en la página 12-6.

En la tabla siguiente se detallan, con arreglo a los símbolos de código, las categorías de referencia que pueden aparecer en la pestaña Buscar.

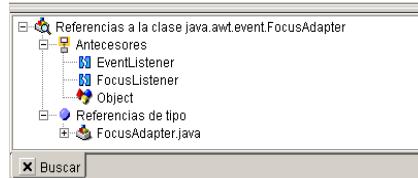
**Tabla 12.2** Detalles de Buscar referencias

Símbolo de código	Categoría de referencia
Clase, clase interna o interfaz	Antecesores: Clases de las que ésta es heredera directa. Descendientes: Clases que descienden de ésta directamente. Referencias de tipo: Clases que declaran o crean una instancia del tipo de objeto de la clase. Referencias de tipo descendientes: Clases que descienden o utilizan descendientes del tipo de objeto de la clase. Referencias de miembros: Miembros de esta clase. Referencias de miembros descendientes: Miembros de clases que descienden de ésta.
Método o constructor	Declaraciones: Lugares donde se declara este método. Usos directos: Lugares donde se llama a este método de clases de las que se crea una instancia directa. Usos indirectos - Lugares en la superclase y en la clase descendiente que, indirectamente, llaman a este método a través de un antecesor o descendiente.
Campo y variable local	Escrituras: Lugares donde se escribe el campo o la variable local. Lecturas: Lugares donde se lee el campo o la variable local.

### Referencias a clases

Si han aparecido referencias a una clase, haga doble clic en una categoría de referencias, en la pestaña Buscar, para ampliarla. Se enumeran los archivos fuente que contienen referencias a la clase. Pulse uno de estos archivos y pulse la referencia para abrirla directamente en el editor.

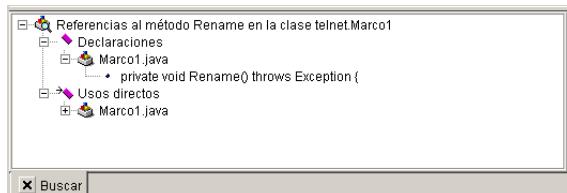
**Figura 12.1** Referencias a clases en la pestaña Buscar Resultados



## Referencias a métodos

Si han aparecido referencias a un método, haga doble clic en una categoría de referencias para ampliarla. Se enumeran los archivos fuente que contienen referencias al método. Pulse uno de estos archivos y pulse la referencia para abrirlo directamente en el editor.

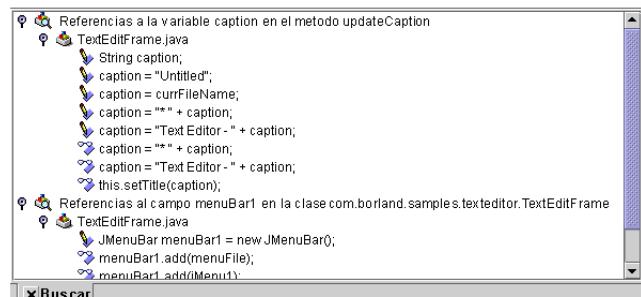
**Figura 12.2** Referencias a métodos en la pestaña Buscar Resultados



## Referencias a campos y variables locales

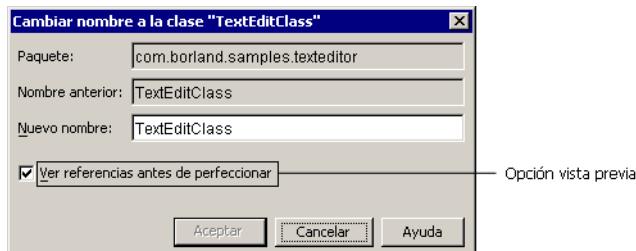
Si han aparecido referencias a un campo o una variable local, haga doble clic en un nombre de archivo para mostrar las escrituras y lecturas de este símbolo. Pulse la referencia de lectura o escritura para colocar el cursor en ella, en el editor.

**Figura 12.3** Referencias a campos y variables locales en la pestaña Resultado de la búsqueda



## Presentación de los cambios antes del perfeccionamiento

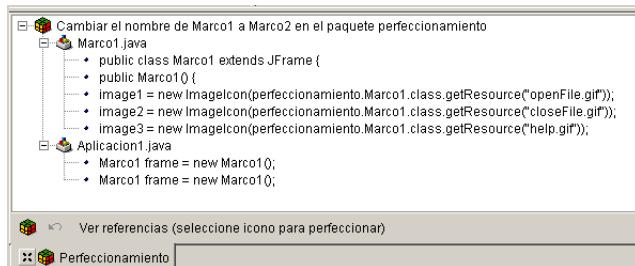
JBuilder permite ver los cambios que se producirán antes de confirmar algunos tipos de perfeccionamiento. Puede resultar conveniente observar las modificaciones cuando se empiezan a utilizar las herramientas de perfeccionamiento, con el fin de comprender qué cambios se van a realizar. A continuación se muestra la opción de vista previa, Ver referencias antes de perfeccionar.

**Figura 12.4** Cuadro de diálogo Cambiar nombre a la clase

Cuando se elige la opción de vista previa y se pulsa Aceptar en el cuadro de diálogo, los cambios que se van a realizar se muestran en la pestaña Perfeccionamiento del panel de mensajes. Los cambios en potencia, esto es, las líneas que cambiarán si se realiza el perfeccionamiento, se muestran según el nombre de archivo, por orden de detección. Amplíe el nodo de archivo y pulse la referencia para dirigirse a ella en el archivo fuente.

**Nota** Algunas tareas de perfeccionamiento no cuentan con la opción de vista previa.

Antes de perfeccionar, la pestaña Perfeccionamiento se asemejará a la figura siguiente:

**Figura 12.5** La pestaña Perfeccionamiento antes de la acción

La barra de herramientas de la pestaña Perfeccionamiento contiene un botón llamado Perfeccionar. También muestra una X abierta para indicar que el perfeccionamiento no se ha finalizado. Para finalizarlo y confirmar los cambios, pulse el botón Perfeccionamiento. La barra de estado de la pestaña Perfeccionamiento muestra un mensaje que informa sobre el progreso.

**Nota** Si modifica alguno de los archivos seleccionados antes de completar el perfeccionamiento, JBuilder no permitirá el perfeccionamiento, ya que los archivos saldrían desfasados. La barra de estado de la pestaña Perfeccionamiento muestra el siguiente mensaje: "Cambios en archivos; no se puede perfeccionar".

En la tabla siguiente se detalla el tipo de información que se muestra.

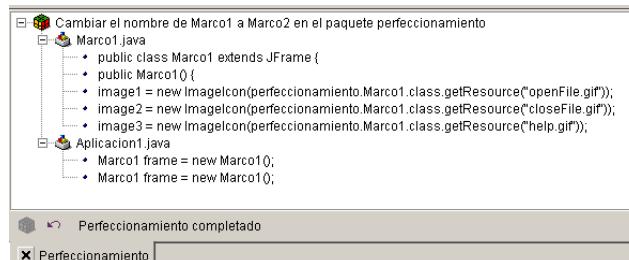
**Tabla 12.3** Detalles de perfeccionamiento

Símbolo de código	Tipo de perfeccionamiento	Información presentada
Paquete	Cambiar nombre	Archivos fuente que contienen una referencia a clase que va a cambiar.
Clase, clase interna o interfaz	Cambiar nombre	Posición de las líneas del archivo fuente actual donde se declara la clase; incluye constructores. También enumera los lugares del código fuente donde se utiliza la clase.
Clase	Mover	Lugares del código fuente donde se declara o se importa el paquete actual de la clase. Indica si se añade o se borra un paquete de la lista de importaciones. (Se añade la sentencia <code>import</code> a todas las dependencias que tenga la clase en el paquete desde el que se mueve.)
Método	Cambiar nombre	Lugares del código fuente donde se declara y se utiliza el método. Indica si se crea un método de reenvío.
Método	Cambiar parámetros	Lugares del código fuente donde se declara y llama al método.
Campo y variable local	Cambiar nombre	Lugares del código fuente donde se declara el símbolo y se realiza la llamada.
Propiedad	Cambiar nombre	Lugares del código fuente donde se declara la propiedad y donde se declaran y llaman los métodos de obtención y definición que la acompañan.

Cuando termina el perfeccionamiento, en la barra de estado aparece el mensaje “Perfeccionamiento completado”. El botón Perfeccionar se muestra atenuado.

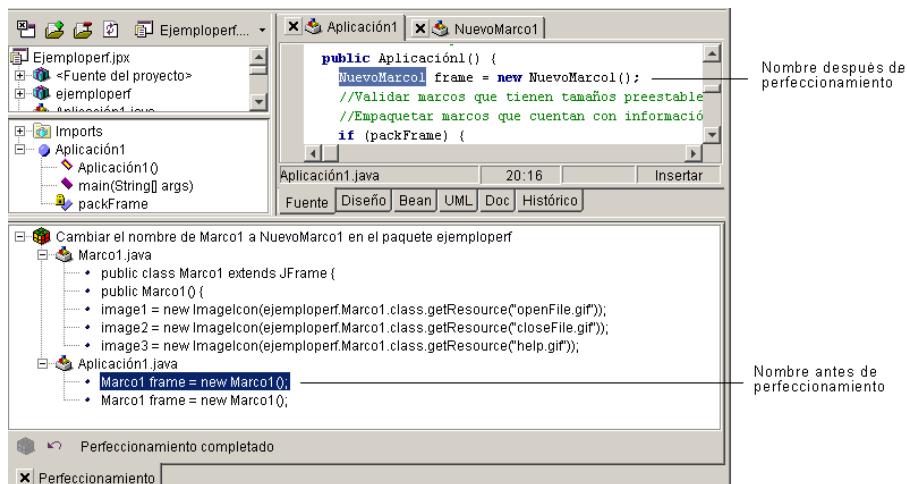
Después de perfeccionar, la pestaña Perfeccionamiento se asemejará a la siguiente figura.

**Figura 12.6** La pestaña Perfeccionamiento después de la acción



Después de perfeccionar, se elimina el botón Perfeccionar de la pestaña y el símbolo en forma de cruz abierta se transforma en una X. Sin embargo, los contenidos de la pestaña Perfeccionamiento no cambian. Aún se muestran las líneas originales del código fuente, de forma que se pueden comparar los cambios realizados. Pulse una línea del código fuente original para dirigirse al cambio.

**Figura 12.7** El archivo fuente y la pestaña Perfeccionamiento después de la acción



## Ejecución del perfeccionamiento

Para realizar un proceso de perfeccionamiento en JBuilder, empiece por seleccionar el símbolo o el bloque de código que desea perfeccionar. A continuación, haga clic con el botón derecho del ratón o abra el menú Edición para elegir el tipo de perfeccionamiento. En la mayoría de los casos, JBuilder proporciona un cuadro de diálogo en el que se puede escribir un nombre nuevo y decidir si se desea observar previamente el perfeccionamiento. En otros casos, como el de rodear un bloque con `try/catch`, JBuilder realiza automáticamente el perfeccionamiento.

Para obtener más información, consulte los siguientes temas:

- “Optimizar importaciones” en la página 12-15
- “Perfeccionamiento por cambio de nombre de paquetes” en la página 12-18
- “Perfeccionamiento por cambio de nombre de clases” en la página 12-19
- “Perfeccionamiento por desplazamiento de clases” en la página 12-20

- “Perfeccionamiento por cambio de nombre de métodos” en la página 12-21
- “Perfeccionamiento por cambio de nombre de variables locales” en la página 12-22
- “Perfeccionamiento por cambio de nombre de campos” en la página 12-23
- “Perfeccionamiento por cambio de nombre de propiedades” en la página 12-24
- “Cambio de parámetros de métodos” en la página 12-24
- “Extracción de métodos” en la página 12-26
- “Introducción de variables” en la página 12-27
- “Perfeccionamiento con sentencia try/catch” en la página 12-28

## Optimizar importaciones

---

El comando Optimizar importaciones reescribe y reorganiza las sentencias `import` según la configuración de las propiedades del proyecto. También elimina las sentencias `import` que ya no se utilizan. El orden de importación se puede personalizar en la pestaña Importaciones de la ficha Formato del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto). Se puede acceder a Optimizar importaciones desde el editor.

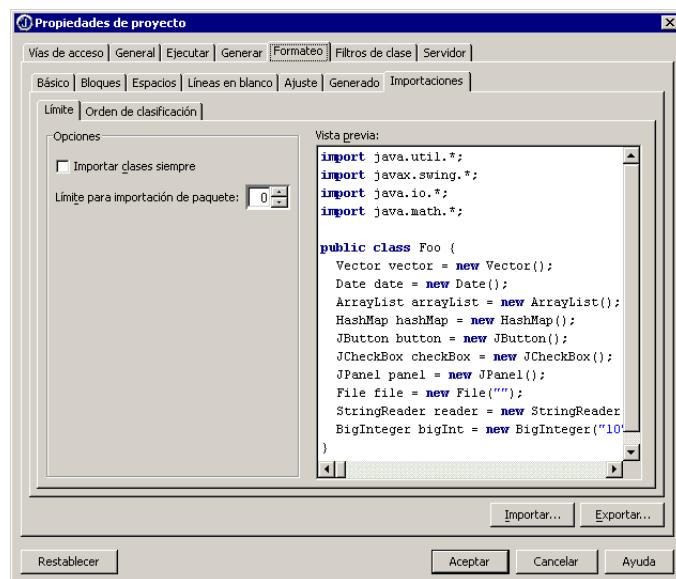
Para personalizar el estilo de la importación de paquetes, abra el cuadro de diálogo Propiedades de proyecto. Elija uno de los siguientes métodos:

- Pulse el botón derecho sobre el archivo del proyecto en el panel del proyecto y seleccione Propiedades o
- Seleccione Proyecto | Propiedades de proyecto.

Para establecer límites y opciones de clasificación para las importaciones:

- 1 Seleccione la pestaña Formateo y, a continuación, la ficha Importaciones. Abra la pestaña Límite para establecer el límite de

importación de paquetes. La pestaña Límite tiene la siguiente apariencia:

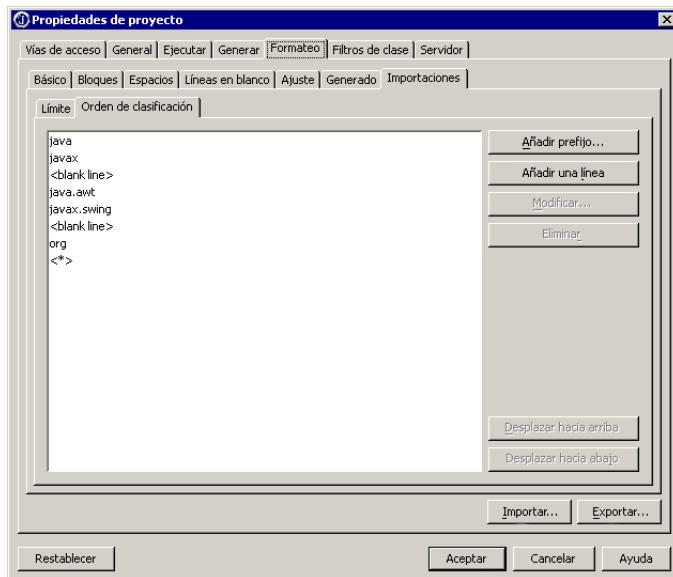


- 2 La opción Importar clases siempre determina si se añaden al código las sentencias de importación de paquetes. Elija esta opción si no desea añadir sentencias de importación de paquetes a su código. En su lugar, las clases individuales se importan directamente. Cuando se utiliza esta opción se pasa por alto la configuración de Límite para importación de paquete.
- 3 En Límite para importación de paquete se establece el número de clases de un paquete que se deben importar para que la acción se reescriba en una sentencia de paquetes importados.

Hasta este límite, las clases se importan utilizando sentencias de importación individuales. Cuando se sobrepasa este límite, se importa el paquete completo. Por ejemplo, si se elige 3 en este campo y se utilizan cuatro clases de un paquete o más, se importa el paquete entero.

El cuadro Vista previa muestra los resultados de las diferentes configuraciones de límites de importación.

- 4** Seleccione la ficha Orden de clasificación para decidir cómo se clasifican las importaciones. La ficha Orden de clasificación tiene este aspecto:



- 5** Para añadir una importación que empiece con un prefijo concreto, pulse el botón Añadir prefijo. Escriba el prefijo en el cuadro de diálogo Añadir prefijo.
- 6** Para insertar una línea de ruptura adicional entre sentencias de importación o grupos de sentencias de importación, seleccione el paquete bajo el que desea insertarla y haga clic en el botón Añadir una línea en blanco.
- 7** Para cambiar una sentencia de importación de paquete, selecciónela y haga clic en el botón Modificar.
- 8** Si lo que desea es eliminar una importación de la lista, selecciónela y, a continuación, haga clic en Eliminar.
- 9** Para mover una línea de importación o en blanco dentro de la lista, haga clic sobre Desplazar hacia arriba o Desplazar hacia abajo.
- 10** Para importar un formato predefinido, pulse el botón Importar. Aparece el cuadro de diálogo Importar configuración de formateo de código. Elija el formato que desee importar y pulse Aceptar.
- 11** Para exportar y guardar el formato, pulse el botón Exportar. Aparece el cuadro de diálogo Exportar la configuración de formateo de código. Escriba el nombre del archivo en el que desee guardar la configuración y pulse Aceptar. El tipo de archivo debe ser .codestyle. Los archivos de configuración de formato se guardan en el directorio <.jbuilder>.

- Nota** Se conservan los comentarios de la sección import del código.
- Sugerencia** Si desea personalizar el orden de las sentencias de importación de *todos* los proyectos nuevos, elija Proyecto | Propiedades por defecto para proyectos y realice las modificaciones en la ficha Estilo de importación del cuadro de diálogo Propiedades por defecto para proyectos.

## Optimización de las importaciones

Para optimizar las importaciones:

- 1 Elija Proyecto | Ejecutar Make del proyecto con el fin de realizar la compilación.
- 2 Seleccione Edición | Optimizar importaciones o haga clic en el editor con el botón derecho del ratón y elija Optimizar importaciones. También puede utilizar el método abreviado *Ctrl+I*. Adicionalmente, se puede seleccionar el símbolo en el panel de estructura, haciendo clic con el botón derecho y seleccionando Optimizar importaciones.

- Sugerencia** Para deshacer la optimización de importaciones elija Edición | Deshacer.

Para optimizar las importaciones desde el panel del proyecto:

- 1 Haga clic con el botón derecho del ratón sobre el paquete del panel del proyecto.
- 2 Seleccione Formatear paquete.
- 3 Active la opción Optimizar importaciones del cuadro de diálogo Formatear código y, a continuación, pulse Aceptar.

## Perfeccionamiento por cambio de nombre de paquetes

---

El perfeccionamiento por cambio de nombre de un paquete se puede realizar desde el editor, desde el panel de estructura y desde un diagrama UML de clase o paquete. El perfeccionamiento por cambio de nombre de un paquete traslada el nombre de un paquete y de todo el subconjunto de paquetes al nuevo nombre del paquete raíz. También se desplazan el paquete y todos los nombres de clases al nuevo nombre y directorio fuente. La estructura del directorio fuente para ese paquete se borra.

Para perfeccionar por cambio de nombre un paquete:

- 1 Haga clic en el nombre del paquete en un diagrama UML de clase o paquete, en el panel de estructura o en el editor.
- 2 Seleccione Cambiar nombre.

Se abre el cuadro de diálogo Cambiar el nombre al paquete “nombre del paquete”.



- 3** Escriba un nombre para el paquete en el campo correspondiente.
- 4** Active la opción Ver referencias antes de perfeccionar si desea examinar los cambios antes de realizar el perfeccionamiento. De lo contrario, pulse Aceptar para completar el proceso. (Si utiliza la vista previa, pulse el botón Perfeccionar de la barra de herramientas para completar el perfeccionamiento.)



Se impide el perfeccionamiento del nombre del paquete si ya existe o no es válido.

## Perfeccionamiento por cambio de nombre de clases

El perfeccionamiento por cambio de nombre de una clase, una clase interna o una interfaz se puede realizar desde el editor, el panel de estructura y desde un diagrama UML de clase. Si se perfecciona por cambio de nombre una clase pública externa, cambia el nombre de todas las declaraciones de todos los usos de la clase y el archivo fuente. Si selecciona un constructor, el perfeccionamiento por cambio de nombre modifica el nombre de la clase.

Para perfeccionar por cambio de nombre una clase, una clase interna o una interfaz:

- 1** Abra el archivo de clase al que desee cambiar el nombre en el editor o como un diagrama UML.
- 2** En el editor, diagrama UML o panel de estructura, pulse con el botón derecho sobre la clase, clase interna o interfaz a la que desee cambiar el nombre.
- 3** Seleccione Cambiar nombre.

Aparece el cuadro de diálogo Cambiar nombre a la clase “nombre”.



- 4 Escriba un nombre para la clase en el campo correspondiente.
- 5 Active la opción Ver referencias antes de perfeccionar si desea examinar los cambios antes de realizar el perfeccionamiento. De lo contrario, pulse Aceptar para completar el proceso. (Si utiliza la vista previa, pulse el botón Perfeccionar de la barra de herramientas para completar el perfeccionamiento.)



Se impide el perfeccionamiento si el identificador de la clase no es válido. Si la clase no es pública externa y hay otra clase pública no externa con el nuevo nombre, el cambio no se realiza.

## Perfeccionamiento por desplazamiento de clases

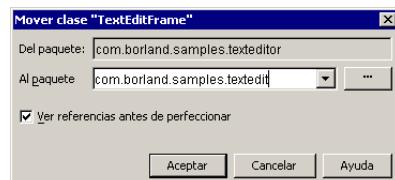
---

Las clases se pueden desplazar a otros paquetes desde el editor, el panel de estructura y desde un diagrama UML de clase. Cuando se perfecciona por desplazamiento una clase, ésta se desplaza a otro paquete, siempre que no contenga un archivo fuente con el mismo nombre. Se actualizan las sentencias de paquete e importación en el archivo fuente de la clase y en todas las clases que contengan una referencia. (Se añade la sentencia `import` a todas las dependencias que tenga la clase en el paquete desde el que se mueve.) Debe ser la clase pública de nivel superior.

Para desplazar una clase a otro paquete:

- 1 Abra el archivo de clase que deseé desplazar en el editor o como un diagrama UML.
- 2 En el editor, diagrama UML o panel de estructura, pulse con el botón derecho del ratón el nombre de la clase.
- 3 Elija Mover clase.

Aparece el cuadro de diálogo Mover clase "nombre".



- 4 Introduzca el nombre del paquete al que se desplaza la clase en el campo Al paquete.
- 5 Active la opción Ver referencias antes de perfeccionar si desea examinar los cambios antes de realizar el perfeccionamiento. De lo contrario, pulse Aceptar para completar el proceso. (Si utiliza la vista previa, pulse el botón Perfeccionar para completar el perfeccionamiento.)



La clase no se traslada si su identificador no es válido o si el nombre del archivo fuente ya existe en el nuevo paquete. En caso necesario, JBuilder añade una sentencia `import` para el antiguo nombre de paquete.

- Nota** Si una clase se desplaza a un paquete que no existe, JBuilder lo crea, lo añade al proyecto, crea el nuevo directorio fuente y coloca la clase en él. También actualiza los nombres de paquetes y las sentencias de importación. Además, si el paquete ya no contiene clases, JBuilder lo elimina del proyecto y borra su directorio fuente.

## Perfeccionamiento por cambio de nombre de métodos

---

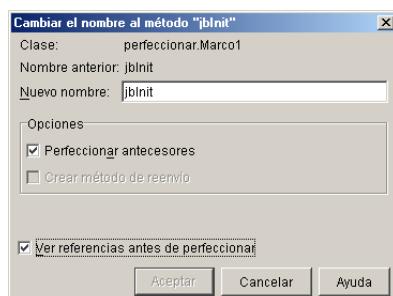
El perfeccionamiento por cambio de nombre de un método se puede realizar desde el editor, el panel de estructura y desde un diagrama UML. El perfeccionamiento por cambio de nombre de un método cambia el nombre del método, de todas las declaraciones de ese método y todos sus usos. Se puede cambiar el nombre del método desde la clase seleccionada en la jerarquía o en toda la jerarquía. Se puede crear un método de reenvío, que pasa la llamada del método al nuevo método. Esto permite que la API pública permanezca intacta.

- Nota** El cambio de nombre no afecta a los métodos sobrecargados, esto es, los que tienen el mismo nombre y una firma distinta.

Para perfeccionar por cambio de nombre un método:

- 1 Abra el archivo fuente que contenga el método al que desea cambiar el nombre en el editor, o bien, ábralo como un diagrama de clase UML.
- 2 En el editor, diagrama UML o panel de estructura, pulse con el botón derecho del ratón el método al que desea cambiar el nombre.
- 3 Seleccione Cambiar nombre.

Aparece el cuadro de diálogo Cambiar el nombre al método “nombre”.



- 4 Escriba un nombre para el método en el campo correspondiente.
- 5 La opción Perfeccionar antecesores (activada por defecto) cambia el nombre de los métodos en las clases de las que la actual es heredera.

- 6** Desactive Perfeccionar antecesores si sólo desea cambiar el nombre al método en esta clase y en sus descendientes. Si lo desea, active la opción Crear método de reenvío.
- 7** Active la opción Ver referencias antes de perfeccionar si desea examinar los cambios antes de realizar el perfeccionamiento. De lo contrario, pulse Aceptar para completar el proceso. (Si utiliza la vista previa, pulse el botón Perfeccionar de la barra de herramientas para completar el perfeccionamiento.)



Si este nombre ya existe en el archivo donde se declara, no se permite el perfeccionamiento. Si el nombre existe en otros archivos de la línea directa de herencia se muestra una advertencia. Al utilizar la opción Perfeccionar antecesores también puede aparecer una advertencia si el método existe pero no se encuentra en la vía de acceso a fuentes que se puede modificar. Por ejemplo, si el método existe en una biblioteca no es posible perfeccionarlo, ya que las bibliotecas son de sólo lectura.

## Perfeccionamiento por cambio de nombre de variables locales

---

El perfeccionamiento por cambio de nombre de una variable local sólo se puede realizar desde el editor. Cuando se aplica el perfeccionamiento por cambio de nombre a una variable local se modifican su declaración y sus usos. Tenga en cuenta que un parámetro de método se considera una variable local.

Para perfeccionar por cambio de nombre una variable local:

- 1** Haga clic con el botón derecho del ratón en la variable cuyo nombre desea cambiar.
- 2** Seleccione Cambiar nombre.

Se abre el cuadro de diálogo Cambiar el nombre a la variable "nombre".



- 3** Escriba un nombre para la variable en el campo correspondiente.
- 4** Active la opción Ver referencias antes de perfeccionar si desea examinar los cambios antes de realizar el perfeccionamiento. De lo contrario, pulse Aceptar para completar el proceso. (Si utiliza la vista



previa, pulse el botón Perfeccionar de la barra de herramientas para completar el perfeccionamiento.)

Si este nombre ya existe en la clase donde se declara la variable original, no se permite el perfeccionamiento.

## Perfeccionamiento por cambio de nombre de campos

El perfeccionamiento por cambio de nombre de un campo se puede realizar desde el editor, el panel de estructura y desde un diagrama UML de clase. Cuando se aplica el perfeccionamiento por cambio de nombre a un campo se modifican su declaración y sus usos.

Para perfeccionar por cambio de nombre un campo:

- 1** Abra el archivo fuente que contenga el campo al que desea cambiar el nombre en el editor, o bien, ábralo como un diagrama de clase UML.
- 2** En el editor, diagrama UML o panel de estructura, pulse con el botón derecho del ratón el campo al que desea cambiar el nombre.
- 3** Seleccione Cambiar nombre.

Se abre el cuadro de diálogo Cambiar el nombre al campo “nombre”.



- 4** Escriba un nombre para el campo en el campo correspondiente.
- 5** Active la opción Ver referencias antes de perfeccionar si desea examinar los cambios antes de realizar el perfeccionamiento. De lo contrario, pulse Aceptar para completar el proceso. (Si utiliza la vista previa, pulse el botón Perfeccionar de la barra de herramientas para completar el perfeccionamiento.)



Si este nombre ya existe en la clase donde se declara el campo, no se permite el perfeccionamiento. Si hay conflictos de ámbito entre el nombre nuevo y el antiguo, se antepone al primero la palabra clave `this`. Si el nombre nuevo redefine un campo existente o es redefinido por él, en una clase o una subclase, se muestra una advertencia.

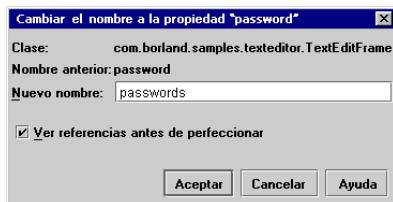
## Perfeccionamiento por cambio de nombre de propiedades

**Es una función de JBuilder Enterprise.**

El perfeccionamiento por cambio de nombre de una propiedad sólo se puede realizar desde un diagrama UML de clase. Cuando se aplica el perfeccionamiento por cambio de nombre a una propiedad, cambian todas sus declaraciones, así como sus métodos de obtención y definición.

Para perfeccionar por cambio de nombre una propiedad:

- 1 Haga clic con el botón derecho del ratón en la propiedad cuyo nombre desea cambiar.
- 2 Seleccione Cambiar nombre. Se abre el cuadro de diálogo Cambiar el nombre de la propiedad “nombredelapropiedad”.



- 3 Escriba un nombre para la propiedad en el campo correspondiente.
- 4 Active la opción Ver referencias antes de perfeccionar si desea examinar los cambios antes de realizar el perfeccionamiento. De lo contrario, pulse Aceptar para completar el proceso. (Si utiliza la vista previa, pulse el botón Perfeccionar de la barra de herramientas para completar el perfeccionamiento.)



Si este nombre ya existe en la clase donde se declara la propiedad original, no se permite el perfeccionamiento.

## Cambio de parámetros de métodos

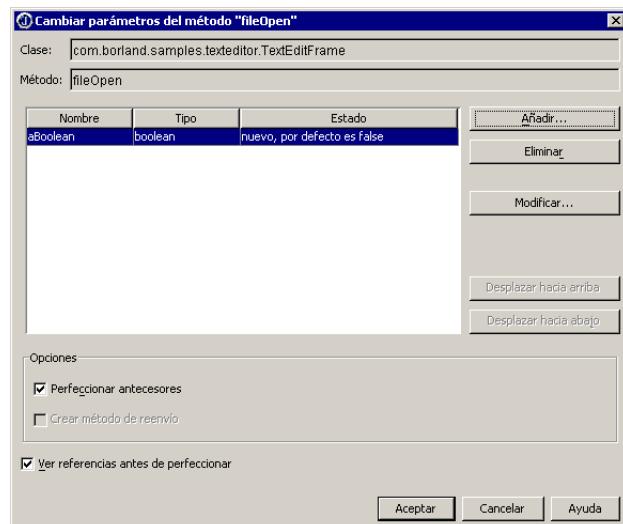
Los parámetros de un método se pueden añadir, borrar y reorganizar desde el editor, desde el panel de estructura y desde un diagrama UML. Es posible modificar un parámetro recién añadido antes de cerrar el cuadro de diálogo Cambiar parámetros, pero no se pueden modificar los parámetros ya creados.

Para cambiar los parámetros de un método:

- 1 Haga clic con el botón derecho del ratón en la firma del método cuyos parámetros desea cambiar.

**2 Selecione Cambiar parámetros de “nombredelmétodo”.**

Se muestra el cuadro de diálogo Cambiar parámetros. En la lista se muestran los parámetros existentes.



- La columna Nombre muestra el nombre del parámetro.
- La columna Tipo muestra el tipo de Java.
- La columna Estado indica si el parámetro es nuevo o ya existente y si está en uso en su código. En los parámetros nuevos, muestra el valor por defecto.

**3 Si desea añadir un parámetro, pulse el botón Añadir. Se abre el cuadro de diálogo Añadir parámetro, donde se elige el tipo de parámetro y se le asignan un nombre y un valor por defecto.**



- 4 Para modificar el parámetro recientemente añadido, seleccione el parámetro y pulse Modificar. Se abre el cuadro de diálogo Modificar parámetro, donde se pueden cambiar el nombre, el tipo y el valor por defecto.
- 5 Si desea eliminar el parámetro, selecciónelo y pulse el botón Eliminar. No puede eliminar parámetros ya existentes que se encuentren en uso.
- 6 Para cambiar el orden de los parámetros de método, utilice los botones Desplazar hacia arriba y Desplazar hacia abajo.

- 7 La opción Perfeccionar antecesores (activada por defecto) perfecciona los métodos en las clases de las que la actual es heredera. Desactive Perfeccionar antecesores si desea perfeccionar el método sólo en esta clase y en sus descendientes. Si lo desea, active la opción Crear método de reenvío.
- 8 Active la opción Ver referencias antes de perfeccionar si desea examinar los cambios antes de realizar el perfeccionamiento. De lo contrario, pulse Aceptar para completar el proceso. (Si utiliza la vista previa, pulse el botón Perfeccionar de la barra de herramientas para completar el perfeccionamiento.)



Si la firma del nuevo método ya existe en el archivo donde se declara, no se permite el perfeccionamiento. Si la firma existe en otros archivos de la línea directa de herencia, se muestra una advertencia. Si se utiliza la opción Perfeccionar antecesores también puede aparecer una advertencia si el mismo método existe pero no se encuentra en la vía de acceso a fuentes que se puede modificar. Por ejemplo, si el método existe en una biblioteca no es posible perfeccionarlo, ya que las bibliotecas son de sólo lectura.

Tenga en cuenta que el perfeccionamiento se impide si el nombre o el tipo del nuevo parámetro no es un identificador de Java válido.

## Extracción de métodos

El perfeccionamiento por extracción de métodos permite convertir un fragmento de código seleccionado en un método. Se puede acceder a este perfeccionamiento desde el editor.

Para extraer un método:

- 1 Seleccione en el editor el bloque de código que desea convertir en método.
- 2 Haga clic con el botón derecho y seleccione Extraer método. Se abre el cuadro de diálogo Extraer método.



- 3 Escriba un nuevo nombre para el método en el campo correspondiente. Este nombre debe explicar la finalidad del método.
- 4 Pulse Aceptar para completar el proceso de perfeccionamiento.

**5 Utilice Modificar | Deshacer para deshacer el perfeccionamiento.**

JBuilder extrae el código del método actual, determina los parámetros necesarios, genera las variables locales que procedan y determina el tipo de devolución. Introduce una llamada al nuevo método en el lugar donde se encontraba el fragmento de código. JBuilder no le permitirá perfeccionar si más de una variable se encuentra escrita en el bloque o se lee después del bloque.

- Nota** Si no se selecciona el conjunto de sentencias en su totalidad, JBuilder intentará ampliar la selección hasta llegar a la expresión o sentencia incluyente más cercana.

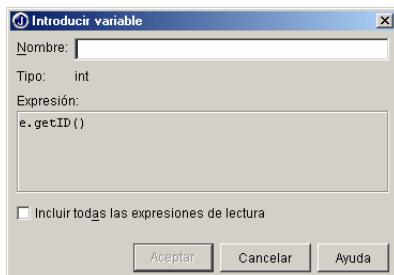
## Introducción de variables

---

El perfeccionamiento por introducción de variables permite sustituir el resultado de una expresión compleja o una parte de ésta por un nombre de variable temporal. Este nombre debe explicar la finalidad de la expresión o la subexpresión. También se conoce como "variable explicativa".

Para introducir una variable:

- 1** Seleccione la expresión compleja que desea sustituir por una variable temporal.
- 2** Haga clic con el botón derecho y seleccione Introducir variable. Se abre el cuadro de diálogo Introducir variable.



- 3** Escriba un nombre para la nueva variable en el campo correspondiente.
- 4** Active la opción Incluir todas las expresiones de lectura para reemplazar todas las lecturas de esa expresión. Si esta opción está desactivada, sólo es reemplazada la selección actual.
- 5** Pulse Aceptar para cerrar el cuadro de diálogo. El perfeccionamiento se completa automáticamente.
- 6** Utilice Modificar | Deshacer para deshacer el perfeccionamiento.

Se genera una variable temporal final con el nombre elegido y se inicializa en el lugar adecuado. La variable recién generada sustituye a la expresión original.

## Perfeccionamiento con sentencia try/catch

---

Consiste en añadir una sentencia `try/catch` alrededor de un bloque de código seleccionado. JBuilder detecta todas las expresiones activas del bloque y añade un bloque para cada una de ellas. Se puede acceder a este tipo de perfeccionamiento desde el editor.

Para englobar un bloque en una sentencia `try/catch`:

- 1 Seleccione el bloque de código en el editor.
- 2 Haga clic con el botón derecho del ratón y elija Insertar en sentencia `try/catch`.

El código queda englobado en una sentencia `try/catch`: Si no se ha seleccionado un bloque de sentencias válido, en la pestaña Perfeccionamiento aparece un error y no se realiza la acción. Utilice Modificar | Deshacer para deshacer el perfeccionamiento.

## Cómo deshacer un perfeccionamiento

---

Cuando termina la acción de perfeccionamiento se puede deshacer pulsando el botón Deshacer de la barra de herramientas



Perfeccionamiento. Deshaga la operación inmediatamente, antes de realizar más cambios en los archivos. Todos los cambios se deshacen. La acción de perfeccionamiento se puede repetir mediante el botón Perfeccionar de la barra de herramientas. La opción Deshacer está activa mientras la pestaña Perfeccionamiento permanece abierta.



Si utiliza un perfeccionamiento que no muestra ninguna salida en la pestaña Perfeccionamiento, puede deshacer los cambios con Modificar | Deshacer. Los perfeccionamientos que no muestran ninguna salida son:

- Optimizar importaciones.
- Extraer método.
- Introducir variables.
- Insertar en sentencia `try/catch`.

## Almacenamiento de perfeccionamientos

---

Cuando termine el perfeccionamiento, guarde los archivos del proyecto mediante el comando Archivo | Guardar todo. Si utiliza un sistema de control de versiones, confirme o incorpore los cambios inmediatamente.

Si se intenta cerrar el proyecto sin guardar, JBuilder abre el cuadro de diálogo Guardar archivos modificados, donde se pueden seleccionar los archivos que se desea guardar. Si los archivos no se guardan, el código fuente recupera el estado anterior al perfeccionamiento.

**Importante**

El perfeccionamiento puede provocar cambios en archivos que no se hayan abierto en el editor durante los preparativos. JBuilder guarda automáticamente estos cambios y modifica los archivos, para evitar que surjan incoherencias en el código.



# 13

## Test de módulos

El test de módulos una función de JBuilder Enterprise.

El test de módulos consiste en escribir pruebas para partes pequeñas de código, definidas a discreción del desarrollador, como por ejemplo un método, y a continuación ejecutarlas y analizar sus resultados. Cuando un desarrollador realiza tests de módulos como parte del proceso de desarrollo debe escribir varias pruebas pequeñas, que se pueden repetir, y ejecutarlas con regularidad. Las ventajas son la creación de un código más fiable y la detección temprana de *regresiones* cuando se modifica el código. Las regresiones son errores que aparecen en un código que funcionaba anteriormente. Muchas metodologías recomiendan la ejecución de pruebas de módulos como parte del proceso de generación de proyectos. Si el resultado de los tests de módulos es negativo, se considera que el proceso de generación es erróneo.

### JUnit

JUnit es un entorno de código abierto para el test de módulos, creado por Erich Gamma y Kent Beck. JUnit proporciona diversas herramientas para el test de módulos, entre las que se encuentran dos clases, `junit.framework.TestCase` y `junit.framework.TestSuite`, que se utilizan como base para escribir comprobaciones parciales. JUnit proporciona también tres ejecutores de tests distintos: `TextUI`, `SwingUI` y `AwtUI`. Dos de ellos, `TextUI` y `SwingUI`, están disponibles en el IDE de JBuilder. Para obtener información acerca de JUnit, consulte <http://www.junit.org>. También hay documentación disponible sobre JUnit en el directorio `<jbuilder>/thirdparty/<junit>/doc`.

JBuilder integra el entorno de test de módulos de JUnit. Esto significa que se pueden crear y ejecutar pruebas de JUnit desde el IDE de JBuilder. Además de las eficaces funciones de comprobación de módulos de JUnit, JBuilder incluye asistentes para la creación de tests, conjuntos de tests y

montajes para tests, así como un ejecutor llamado JBTestRunner que combina elementos de texto e interfaz gráfica en la salida y se integra directamente en el IDE de JBuilder.

### Consulte

- “[Detección de tests](#)” en la página 13-3
- “[Creación de tests y conjuntos de tests JUnit](#)” en la página 13-5
- “[Ejecución de tests](#)” en la página 13-15

## Cactus

---

Cactus extiende JUnit para suministrar test de módulos del código Java de la parte del servidor. Esto lo hace redirigiendo la prueba a un proxy en el lado del servidor. Para obtener más información sobre Cactus, viste <http://jakarta.apache.org/cactus/index.html>. La documentación de Cactus también se encuentra disponible en el directorio <jbuilder>/thirdparty/<jakarta-cactus>/doc.

JBuilder ofrece varias funciones que facilitan el test de Cactus. El Asistente para la configuración de Cactus permite configurar el proyecto para que admita el test Cactus. El Asistente para cliente de prueba EJB puede generar un test Cactus para su Enterprise JavaBean (EJB). La integración de Cactus de JBuilder en el entorno permite ejecutar los tests Cactus dentro del IDE de JBuilder si hay disponible un servidor y un proyecto debidamente configurados.

### Consulte

- “[Utilización de Cactus](#)” en la página 13-12
- “[Asistente para la configuración de Cactus](#)” en la página 13-12
- “[Ejecución y comprobación de un enterprise bean](#)” en la Guía del desarrollador de Enterprise JavaBeans

## Funciones de test de módulos de JBuilder

---

Las funciones de comprobación de módulos de JBuilder integran Junit y Cactus en el IDE de JBuilder y proporcionan herramientas para la creación de comprobaciones de módulos y su organización en conjuntos, así como para la ejecución, el análisis y la depuración de las pruebas. JBuilder proporciona un conjunto de montajes para tests con el objeto de realizar tareas comunes a varios tests. JBTestRunner de JBuilder ofrece una forma de ejecución de tests que combina la salida por texto e interfaz gráfica. JBuilder incluye las siguientes funciones de test de módulos:

- Asistente para tests.
- Asistente para conjuntos de tests.
- Asistente para cliente de prueba EJB.
- Montaje JDBC.
- Montaje JNDI.
- Montajes de comparación.
- Asistente para montajes personalizados.
- Comprobación de Cactus.
- Ejecución de tests.
- JBTestRunner.
- Compatibilidad con JUnit TextUI.
- Compatibilidad con JUnit SwingUI.
- Test del filtro del seguimiento de la pila.
- Depuración de tests.
- Recopilador de tests de JUnit.

## Detección de tests

---

Por defecto, JBuilder identifica automáticamente como tests las clases que son ampliaciones de `junit.framework.TestCase` o `junit.framework.TestSuite`. Si una clase se identifica como test y existe una configuración para la ejecución adecuada, cuando se pulsa con el botón derecho del ratón el nombre del archivo fuente en el panel del proyecto o la pestaña en la que figura su nombre, si este archivo fuente está abierto en el editor, aparece un menú contextual que contiene las opciones Ejecutar test y Depurar test. La opción Optimizar test está también disponible si Borland Optimizeit está instalado adecuadamente.

El recopilador de test de JUnit proporciona un método alternativo de identificación de tests.

## Recopilador de tests de JUnit

---

El recopilador de test de JUnit es una función de JBuilder que proporciona una interfaz gráfica de usuario (GUI) para la clase `PackageTestSuite`. Esto resulta útil para la detección de tests, para que no necesite mantener una lista de todas las clases de tests. JUnit Test Collector está disponible en el cuadro de diálogo Propiedades de configuración de ejecución para el tipo

Test de configuración de ejecución. Se activa seleccionando el botón circular Paquete de la ficha Ejecutar de este cuadro de diálogo.

Para añadir una configuración de ejecución tipo Test que utilice el recopilador de tests de JUnit:

- 1 Seleccione Proyecto | Propiedades de proyecto.
- 2 Abra la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto
- 3 Pulse el botón Nuevo.
- 4 Seleccione la ficha Ejecutar del cuadro de diálogo Propiedades de configuración de ejecución.
- 5 Asigne Test a Tipo.
- 6 Pulse el botón circular Paquete. Esto activa el recopilador de tests de JUnit y desactiva el modo de detección de tests por defecto.
- 7 Especifique si desea o no que el analizador de tests incluya los subpaquetes activando o desactivando Incluir subpaquetes.
- 8 Especifique las cadenas con las que empiezan o terminan los nombres de clase de tests en los campos El nombre comienza con y El nombre finaliza con. Al hacerlo, se restringen los tests que encuentra el analizador de tests a aquellos que contengan dichas cadenas. Este paso es optativo.
- 9 Pulse Aceptar para guardar la configuración de ejecución y cierre el cuadro de diálogo Propiedades de configuración de ejecución.
- 10 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

Los tests que coincidan con el filtro que se estableció en el cuadro de diálogo Propiedades de configuración de ejecución ahora se identifican correctamente como tests. Esto significa que las opciones Ejecutar test y Depurar test aparecen en el menú contextual si pulsa con el botón derecho del ratón uno de los tests que coinciden en el panel del proyecto. La opción Optimizar test también está disponible si Borland Optimizeit está instalado adecuadamente.

### Consulte

- “Definición de las configuraciones para la ejecución” en la página 7-7

# Creación de tests y conjuntos de tests JUnit

---

Un test es una instancia de `junit.framework.TestCase`. Los tests contienen uno o más métodos que ejecutan una o más partes de una clase de la aplicación que se comprueba. También contiene los métodos `setUp()` y `tearDown()`. El método `setUp()` se utiliza para llevar a cabo la configuración necesaria antes de ejecutar cada método de test. El método `tearDown()` se utiliza para limpiar y liberar recursos después de haber ejecutado cada método de test. Cuando se ejecutan tests de JUnit se crea una instancia de la clase de test para cada método. Los métodos `setUp()` y `tearDown()` se ejecutan una vez por instancia. Por ejemplo, en un test llamado `MyTestCase` que contiene los métodos `testMethod1()` y `testMethod2()`, el orden de ejecución es el siguiente:

- 1 El ejecutor de tests crea dos instancias de `MyTestCase`.
- 2 Se llama al método `setUp()`.
- 3 Se llama al método `testMethod1()`.
- 4 Se llama al método `tearDown()`.
- 5 Se llama al método `setUp()`.
- 6 Se llama al método `testMethod2()`.
- 7 Se llama al método `tearDown()`.

Tanto un test como un conjunto de tests amplían `TestCase`. La diferencia entre los tests y los conjuntos es que los primeros contienen métodos de test individuales, mientras que los segundos se utilizan para organizar conjuntos en un grupo lógico y ejecutarlos juntos. Los conjuntos pueden contener llamadas a cualquier número de tests o a otros conjuntos de tests.

Uno de los principales objetivos del test de módulos es el de crear tests que se puedan repetir; de esta forma siempre devuelven el mismo resultado si el software comprobado funciona correctamente. Si uno de los métodos que se comprueba contiene errores, el resultado del test es negativo. Si se ejecutan tests de módulos cada vez que se realizan cambios en el software se verifica con más facilidad que no se han introducido errores ni regresiones.

El desarrollador decide el número de tests. En algunos casos la práctica habitual consiste en escribir un test para cada método público del código, pero no es necesario llegar a semejante grado de exhaustividad con el fin de prevenir las regresiones. Al principio puede resultar conveniente concentrarse en la escritura de pruebas para las partes más críticas del software o las que presentan más probabilidades de fallo.

Si el código es correcto, los métodos de test devuelven un resultado esperado. De lo contrario, proporcionan información útil para la localización del origen del fallo. El Asistente para tests crea estructuras de

métodos de tests; el desarrollador decide qué resultados son significativos y aporta la implementación.

### Consulte

- [Capítulo 21, “Tutorial: Creación y ejecución de tests y conjuntos de tests”](#)

## El Asistente para tests

---

El Asistente para tests se emplea para crear clases de test que amplían `TestCase` y contienen definiciones de métodos con las secciones principales vacías, que se deben llenar con arreglo a la clase que se desea comprobar. Si desea abrir el Asistente para tests, elija Archivo | Nuevo en el menú, abra la pestaña Test de la galería de objetos, elija Test y pulse Aceptar. El Asistente para tests crea pruebas en el directorio fuente del proyecto especificado en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto. Si desea ver o modificar el directorio fuente de tests, elija Proyecto | Propiedades de proyecto, abra la ficha Vías de acceso y pulse la pestaña Fuente.

El Asistente para tests permite elegir la clase y los métodos que se deben comprobar, utilizar montajes para tests predefinidos y crear una configuración de ejecución para el test. Si desea más información sobre la interfaz del Asistente para tests pulse el botón Ayuda.

### Consulte

- [Capítulo 21, “Tutorial: Creación y ejecución de tests y conjuntos de tests”](#)
- [“Montajes para tests predefinidos” en la página 13-8](#)

## Adición de código a los tests

---

El Asistente para tests crea la estructura de los tests, pero el desarrollador debe introducir el código. El Asistente para tests señala las zonas del código que se deben rellenar con comentarios `@todo` de Javadoc. Estos comentarios se muestran en el panel de estructura, en el nodo "Por Hacer" del árbol. Para completar los tests se debe añadir código a todos los métodos. En muchas ocasiones también es necesario definir valores distintos de null para determinadas variables, que el asistente también etiqueta con comentarios `@todo`.

A continuación se presenta un ejemplo de un método de test sencillo:

```
public void testSum() {  
    assertEquals( 2, sum(1,1) );  
}
```

Los métodos de test deben ser `public` y `void`, y no deben llevar argumentos.

Cuando se añade código de test a los métodos es necesario tener la forma de averiguar si el resultado del test ha sido positivo o negativo. Es posible crear un método de test que examine varias condiciones y devuelva un fallo si no se cumplen. La forma más normal de conseguirlo consiste en llamar a uno de los métodos de orden en `junit.framework.Assert`, por ejemplo:

- `assertEquals()`: ordena que los argumentos que se le pasan tengan el valor `equal`.
- `assertTrue()`: ordena que el valor de la expresión booleana que se le pasa sea `true`.
- `assertNotNull()`: ordena que el argumento que se le pasa tenga un valor distinto de `null`.

En `junit.framework.Assert` hay varias versiones sobrecargadas de estos métodos. Las diversas firmas de los métodos toman argumentos de distintos tipos, lo que los hace más flexibles. Si la condición no se cumple, estos métodos desencadenan un fallo del test, del que informa el ejecutor de tests. Si un método de test concluye sin desencadenar ningún fallo, el ejecutor de tests comunica que el resultado es correcto. Los métodos de orden se pueden llamar directamente desde el test, porque `TestCase` es una subclase de `Assert`.

**Sugerencia** Si desea ver los métodos de `junit.framework.Assert`, abra un test en el editor; en el panel de estructura haga doble clic en `TestCase`, la clase antecesora, y a continuación haga doble clic en `Assert`, su clase antecesora.

También es posible escribir un método de test que lance una excepción. A continuación, se propone un ejemplo.

```
public void testException() throws Exception {
    throw new Exception("ouch!");
}
```

Cuando un test arroja una excepción que no gestiona, el ejecutor de tests informa que el método ha fallado.

Si desea más información sobre la escritura de tests con JUnit, consulte el artículo de Kent Beck y Erich Gamma, "JUnit Test Infected: Programmers Love Writing Tests", en el sitio web de JUnit.

## El Asistente para conjuntos de tests

---

El Asistente para conjuntos de tests se utiliza para crear conjuntos de tests que se ejecutan por lotes. Si desea abrir el Asistente para conjuntos de tests, elija Archivo | Nuevo en el menú, abra la pestaña Tests de la galería de objetos y elija Conjunto de tests.

En este asistente se pueden elegir los tests que se deben incluir en el conjunto, así como crear una configuración de ejecución. Si desea más información sobre la interfaz del Asistente para conjuntos de tests, pulse el botón Ayuda.

#### Consulte

- [Capítulo 21, “Tutorial: Creación y ejecución de tests y conjuntos de tests”](#)

## El Asistente para clientes de prueba EJB

---

El Asistente para clientes de prueba EJB, disponible en la ficha Enterprise de la galería de objetos, permite crear tres tipos diferentes de clientes de prueba para probar sus Enterprise JavaBeans (EJB). De estos tipos, dos, cliente de prueba JUnit y cliente de prueba Cactus JUnit, están diseñados para el test de módulos.

**Sugerencia** Aunque es posible crear un test que pruebe EJB mediante el Asistente para tests, es mejor utilizar el Asistente para clientes de prueba EJB al probar EJB. Esto se debe a que el Asistente para clientes de prueba EJB genera más código específico de EJB.

#### Consulte

- “Ejecución y comprobación de un enterprise bean” en la *Guía del desarrollador de Enterprise JavaBeans*

## Montajes para tests predefinidos

---

Los montajes para tests son clases de utilidades que se pueden emplear en los tests con el fin de realizar tareas rutinarias de creación de entornos. Un ejemplo es la gestión de las conexiones de base de datos con datos utilizados con fines de comprobación.

El Asistente para tests de JBuilder puede instalar automáticamente los montajes para tests si cuentan con un constructor que tome el argumento `Object` y contenga los métodos `setUp()` y `tearDown()`. He aquí un ejemplo básico de montaje válido:

```
public class CustomFixture1 {  
  
    public CustomFixture1(Object obj) {  
        // aquí va el código  
    }  
  
    public void setUp() {  
        // aquí va el código  
    }  
}
```

```

    }

    public void tearDown() {
        // aquí va el código
    }

}

```

Con el fin de instalar un montaje de este tipo en un test nuevo, selecciónelo en el tercer paso del Asistente para tests. El test crea una instancia del montaje, y se llama a sus métodos `setUp()` y `tearDown()`.

JBuilder proporciona estos tres montajes predefinidos para la realización de tareas comunes:

- Montaje JDBC
- Montaje JNDI
- Montaje de comparación

También es posible crear montajes para tests personalizados con el Asistente para montajes de tests.

## **Montaje JDBC**

---

El montaje JDBC, `com.borland.jbuilder.unittest.JdbcFixture`, se puede utilizar en las pruebas para gestionar conexiones JDBC. Los métodos del test pueden utilizar el método `getConnection()` con el fin de obtener una conexión JDBC. Para indicar una conexión JDBC, utilice los métodos `setUrl()` y `setDriver()`. El método `runSqlFile()` se utiliza para ejecutar archivos de script SQL.

La forma más sencilla de crear un montaje JDBC consiste en utilizar el Asistente para montajes para JDBC. El Asistente para montajes para JDBC crea una clase que amplía `JdbcFixture`. La ampliación de `JdbcFixture` permite especificar la conexión JDBC necesaria para utilizar y proporcionar otras funciones, en caso necesario. He aquí un resumen de los métodos más utilizados en `JdbcFixture`:

- `dumpResultSet()` vuelve en `Writer` los valores de un conjunto de resultados. Toma como parámetros `java.sql.ResultSet` y `java.io.Writer`.
- `getConnection()` devuelve un objeto `java.sql.Connection` que define la conexión JDBC.
- `runSqlBuffer()` ejecuta una sentencia SQL contenida en `StringBuffer`.
- `runSqlFile()` lee un script SQL de un archivo y lo ejecuta. Toma como parámetros una cadena `String` que indica la posición del archivo y un valor booleano.
- `setDriver()` configura la propiedad `Driver` de la conexión JDBC. Toma una cadena `String` como parámetro.

## Montajes para tests predefinidos

- `setUrl()` configura la propiedad `URL` de la conexión JDBC. Toma una cadena `String` como parámetro.
- `setUsername()` establece el nombre de usuario para el acceso a la conexión JDBC. Toma una cadena `String` como parámetro.
- `setPassword()` establece la contraseña para el acceso a la conexión JDBC. Toma una cadena `String` como parámetro.

**Sugerencia** Los montajes para JDBC creados por medio del asistente amplían `JdbcFixture`. La estructura de la clase heredada se puede presentar en el panel de estructura, haciendo doble clic sobre el nodo de la clase heredada en el panel de estructura con el montaje JDBC abierto en el editor. Otro método consiste en hacer clic con el botón derecho del ratón en el nombre de la clase antecesora y seleccionar Buscar definición en el menú contextual.

Si desea más información sobre la interfaz del Asistente para montajes para JDBC, pulse el botón Ayuda.

### Consulte

- [Capítulo 22, “Tutorial: Utilización de montajes para tests”](#)

## Montaje JNDI

---

Los montajes JNDI son clases que facilitan la realización de consultas JNDI. Se pueden crear con un asistente. El Asistente para montajes para JNDI se encuentra en la ficha Test de la galería de objetos. Si desea más información sobre la interfaz del Asistente para montajes para JNDI, pulse el botón Ayuda.

## Montaje para comparación

---

Los montajes para comparación se utilizan para registrar el resultado de un test y compararlo con el de test anteriores y posteriores. Los montajes para comparación son clases que amplían `com.borland.jbuilder.unittest.TestRecorder`. La clase `TestRecorder` es una ampliación de `java.io.Writer`, por lo que los montajes para comparación se pueden utilizar siempre que se requiera un `Writer`. Los montajes para comparación se pueden generar por medio del asistente que se encuentra en la ficha Test de la galería de objetos.

`TestRecorder` contiene cuatro constantes que establecen el modo de registro:

- `UPDATE` - El montaje de comparación compara la nueva salida con un archivo de salida existente, o lo crea si no existe y graba la salida en él.

- **COMPARE** - El montaje de comparación siempre compara la nueva salida con la ya existente.
- **RECORD** - El montaje de comparación graba todas las salidas, sobrescribiendo cualquiera ya existente en el archivo de salida.
- **OFF** - El montaje de comparación está desactivado.

Tenga en cuenta que si se añaden o se borran cadenas de un test o un conjunto tras el registro de los resultados, es necesario reiniciar el archivo de datos. Si las pruebas han cambiado, utilice **RECORD** en lugar de **UPDATE** o borre el archivo de datos. Se trata de un archivo binario que se encuentra en el mismo directorio que los archivos de código fuente de prueba. Lleva el mismo nombre que el test.

He aquí un resumen de los métodos más utilizados en los montajes para comparación:

- `print()` imprime una cadena que se le pasa como parámetro.
- `println()` imprime una cadena que se le pasa como parámetro, con un salto de línea.
- `compareObject()` llama al método `equals()` de un objeto y compara un objeto que se le pasa con otro registrado anteriormente por medio de `recordObject()`.
- `recordObject()` registra un objeto que más adelante se comparará con otro por medio de `compareObject()`.

Si desea más información sobre la interfaz del Asistente para montajes para comparación, pulse el botón Ayuda.

### Consulte

- [Capítulo 22, “Tutorial: Utilización de montajes para tests”](#)

## Creación de un montaje para tests personalizado

---

También es posible escribir montajes personalizados con el fin de realizar tareas habituales en los tests. Los tests podrán compartir así el montaje personalizado. El Asistente para montajes personalizados resulta útil para generar la estructura de montajes para tests o crear un englobador para el código ya escrito. La estructura de los montajes para tests personalizados incluye los métodos `setUp()` y `tearDown()`. El Asistente para montajes personalizados se encuentra en la ficha Test de la galería de objetos. Si desea más información sobre la interfaz del Asistente para montajes personalizados, pulse el botón Ayuda.

## Utilización de Cactus

---

Cactus extiende JUnit para suministrar test de módulos del código Java de la parte del servidor. Resulta útil comprobar los Enterprise JavaBeans (EJB) junto con las aplicaciones web. JBuilder proporciona funciones que facilitan la comprobación de Cactus.

- Asistente para la configuración de Cactus - Configura el proyecto para que utilice Cactus y se pueden ejecutar los tests Cactus en el IDE de JBuilder.
- Asistente para clientes de prueba EJB - Ayuda a crear el cliente de prueba Cactus para sus EJB.

El objetivo principal de Cactus de JBuilder es facilitar la comprobación de EJB con Cactus. La comprobación de los EJB con Cactus se detalla en “Ejecución y comprobación de un enterprise bean” en la *Guía del desarrollador de Enterprise JavaBeans*.

También puede utilizar Cactus para comprobar otros tipos de código Java en el servidor. Incluso si su objetivo no es comprobar sus EJB, puede utilizar el Asistente para la configuración de Cactus para configurar el proyecto para la comprobación de Cactus y facilitar la distribución adecuada de los archivos necesarios.

### Asistente para la configuración de Cactus

---

El Asistente para la configuración de Cactus configura el proyecto para que pueda utilizar Cactus. Esto hace que sea posible ejecutar pruebas con Cactus desde el IDE de JBuilder. El asistente se abre mediante la opción de menú Asistentes | Configuración de Cactus.

Para configurar su proyecto para Cactus:

- 1 Seleccione Asistentes | Configuración de Cactus. Se abre el Asistente para configuración de Cactus.
- 2 Seleccione la WebApp a la que el asistente capacitará para realizar pruebas con Cactus. Puede utilizar la WebApp por defecto, una WebApp ya creada o pulsar el botón Nuevo para abrir el Asistente para aplicaciones web y crear una.
- 3 Seleccione las opciones de registro para los archivos de registro de Cactus. Especifique las ubicaciones para los archivos de registro de cliente y servidor de Cactus, o bien, desactive la opción Activar registro si no desea utilizar archivos de registro.
- 4 Pulse Siguiente.
- 5 Seleccione los recopilatorios que se van a distribuir en el servidor y redistribuir antes de cada prueba. De este modo, los recopilatorios se

mantienen sincronizados con el proyecto. Si cualquiera de los recopilatorios se muestra en rojo con un signo de exclamación antes del nombre, significa que el archivo físico ya no existe. Esto se debe probablemente a que el recopilatorio aún no se ha generado. No hay ningún problema en seleccionar alguno de estos recopilatorios, siempre que recuerde generar el recopilatorio antes de ejecutar las pruebas con Cactus.

- 6** Seleccione una configuración de ejecución de tipo Servidor. Puede crear una mediante el botón Nuevo.
- 7** Seleccione una configuración de ejecución de tipo Test. Puede crear una mediante el botón Nuevo.
- 8** Pulse el botón Finalizar. El proyecto ha quedado configurado para su uso con Cactus.

#### Consulte

- “Configuración del proyecto para la comprobación de EJB con Cactus” en la *Guía del desarrollador de Enterprise JavaBeans*

## Creación de un test Cactus para los Enterprise JavaBean

---

Puede que desee utilizar Cactus para comprobar sus Enterprise JavaBeans (EJB). JBuilder incorpora el Asistente para clientes de prueba EJB, que puede generar tres tipos diferentes de clientes de prueba EJB. Uno de ellos es un cliente de prueba Cactus. Para abrir el Asistente para clientes de prueba EJB:

- 1** Seleccione Archivo | Nuevo.
- 2** Seleccione la ficha Enterprise de la galería de objetos.
- 3** Seleccione Cliente de prueba EJB y pulse Aceptar.

**Nota** El Asistente para clientes de prueba EJB no está disponible si el proyecto no se ha configurado correctamente para utilizar un servidor compatible con los servicios EJB.

La comprobación de EJB está fuera del ámbito de este capítulo. Este tema se cubre en “Ejecución y comprobación de un enterprise bean” en la *Guía del desarrollador de Enterprise JavaBeans*.

#### Consulte

- “Ejecución y comprobación de un enterprise bean” en la *Guía del desarrollador de Enterprise JavaBeans*
- “Configuración del servidor de aplicaciones de destino” en la *Guía del desarrollador de Enterprise JavaBeans*

## Ejecución de test Cactus

---

La ejecución de test Cactus es más complicada que la ejecución de otro tipo de tests de módulos, ya que es necesario comprobar que cuenta con un servidor configurado correctamente, los descriptores de distribución adecuados y las configuraciones de ejecución Test y Servidor adecuadas. Además, la diferencia principal entre la ejecución de test Cactus y otro tipo de tests JUnit es que en el caso de un test Cactus, es necesario que inicie primero el servidor.

Una vez que la configuración es correcta y ya se ha iniciado el servidor, la ejecución de tests Cactus dentro del IDE de JBuilder es parecida a la ejecución de otros tests JUnit. Si su proyecto está configurado correctamente, todo lo que necesita para ejecutar los tests Cactus es:

- 1 Inicie el servidor con la configuración de ejecución Servidor.
- 2 Haga clic con el botón derecho del ratón en el archivo de prueba Cactus, del panel del proyecto.
- 3 En el menú contextual, seleccione Ejecutar test utilizando <configuración de prueba>. El test se ejecuta en el ejecutor de tests especificado en la configuración de ejecución Test.

Todo lo relacionado con la configuración del servidor y los descriptores de distribución necesarios están fuera del ámbito de este capítulo. Se detallan en la *Guía del desarrollador de Enterprise JavaBeans* y en la *Guía del desarrollador de aplicaciones web*.

### Consulte

- “Ejecución y comprobación de un enterprise bean” en la *Guía del desarrollador de Enterprise JavaBeans*
- “Configuración del servidor de aplicaciones de destino” en la *Guía del desarrollador de Enterprise JavaBeans*
- “Las WebApps y los archivos WAR” en la *Guía del desarrollador de aplicaciones Web*
- “[Ejecución de tests](#)” en la página 13-15

# Ejecución de tests

---

Existen tres ejecutores de tests distintos destinados a la ejecución de tests. El ejecutor de tests por defecto de JBuilder se llama JBTestRunner. Si lo prefiere, puede utilizar los ejecutores de tests de JUnit, TextUI y SwingUI. El ejecutor de test que desee utilizar se especifica en la configuración de ejecución tipo Test. Para seleccionar un ejecutor de tests:

- 1 Elija Proyecto | Propiedades de proyecto en el menú.
- 2 Seleccione la ficha Ejecutar.
- 3 Seleccione una configuración de ejecución de tipo Test existente y pulse Modificar, o bien, pulse Nuevo si todavía no existe ninguna configuración de ejecución Test. Se muestra el cuadro de diálogo Propiedades de configuración de ejecución.
- 4 Escriba el nombre de la configuración de ejecución, si es necesario.
- 5 Seleccione la ficha Ejecutar del cuadro de diálogo Propiedades de configuración de ejecución.
- 6 Asigne el valor Test al tipo de configuración de ejecución.
- 7 Seleccione un elemento de la lista desplegable Ejecutores de tests.
- 8 Haga clic en Aceptar para cerrar el cuadro de diálogo.
- 9 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

En los apartados siguientes se describe la ejecución de tests con los distintos ejecutores de tests disponibles:

## Consulte

- “[Definición de las configuraciones para la ejecución](#)” en la página 7-7

## JBTestRunner

---

JBTestRunner proporciona una combinación de salida de texto e indicaciones por interfaz gráfica para informar sobre el estado de los tests. JBTestRunner muestra la jerarquía actual de los tests y los conjuntos, así como de los métodos de test que contienen. Para desplazarse a un método de test basta con hacer clic sobre él en el árbol. El cursor del editor se coloca en el método de test. JBTestRunner es el ejecutor de tests por defecto de JBuilder.

Para ejecutar un test, haga clic con el botón derecho del ratón en un test o un conjunto, en el panel del proyecto, y elija Ejecutar test en el menú contextual. Si no se ha cambiado el ejecutor de tests por defecto en el cuadro de diálogo Propiedades de proyecto, se utiliza JBTestRunner. Si ha

cambiado el ejecutor de tests y desea volver a JBTestRunner, siga las instrucciones para la selección de ejecutores de tests que se facilitan en “[Ejecución de tests](#)” en la página 13-15.

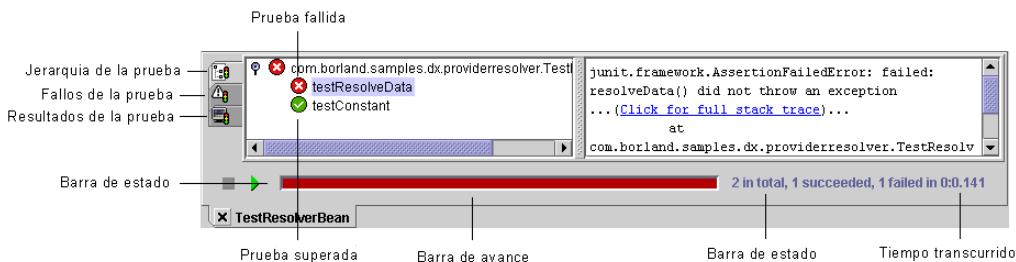
Cuando se ejecutan pruebas, el resultado se muestra en la ficha JBTestRunner del panel de mensajes. En esta ficha hay tres vistas: Fallos del test, Jerarquía del test y Resultados del test.

Cuando se ejecutan los tests, JBTestRunner muestra una barra de progreso que indica el porcentaje de tests realizados. Esta barra es de color verde excepto si falla algún test, en cuyo caso adquiere el color rojo.

JBTestRunner también muestra en el árbol de jerarquía de comparación iconos con una marca de verificación verde cuando el resultado es positivo, e iconos con una X roja cuando se presentan fallos o errores. Si ocurre un fallo o un error, JBTestRunner muestra una pila en la parte derecha de la ficha Fallos del test o Jerarquía del test, si el nodo de este fallo o error se encuentra seleccionado a la izquierda. Si se pulsa una línea de la pila, el punto en el que ha fallado una orden se resalta en el editor.

A medida que se ejecutan los tests, la barra de estado de JBTestRunner indica el número de tests efectuados, así como el número de éxitos, fallos y errores. También muestra el tiempo transcurrido desde el inicio de las pruebas, incluida la carga. Este tiempo se actualiza cada vez que se completa un test.

Si se hace clic en un nodo del árbol Fallos del test o Jerarquía del test, aparece un menú contextual que contiene las opciones Ejecutar selección y Depurar selección. Estas opciones permiten ejecutar y depurar tests específicas cuando se investiga un fallo.



## Jerarquía del test

 La vista Jerarquía del test aparece por defecto, excepto si el test ha fallado. Esta vista muestra un árbol con los tests, así como los conjuntos y los métodos de test. Junto a los nodos del árbol hay iconos que indican el estado del test correspondiente. Una marca de verificación verde  indica que el test ha sido correcto. Una X de color rojo  indica que el test ha fallado. Esta vista se actualiza de forma dinámica a medida que se ejecutan los tests. Cuando se pulsa un nodo de este árbol, su resultado aparece en el panel derecho de la vista de mensajes, y en el editor se

resalta la línea de código que ha provocado el fallo, en caso de un test con fallos, o la primera línea, en caso de test correctos.

## Fallos del test



La vista Fallos del test aparece cuando se pulsa la pestaña central de la parte izquierda de la ficha JBTestRunner. Se muestra por defecto la vista Fallos del test si el test falla. En el panel de la izquierda aparece una línea por cada test con fallos. Cuando se pulsa la línea correspondiente a un fallo, éste se resalta en el editor, y aparece más información sobre él en el panel de la derecha. Si no ha fallado ningún test, esta vista estará vacía.

## Resultados del test



La vista Resultados de del test aparece cuando se pulsa la pestaña inferior de la parte izquierda de la ficha JBTestRunner. Esta vista muestra la salida generada por el test, que incluye las excepciones.

## JUnit TextUI

---

TextUI de JUnit es un ejecutor de tests sencillo que presenta la salida en texto. JUnit se ha integrado en JBuilder de forma que cuando se realizan tests con TextUI por medio del IDE de JBuilder, basta con pulsar una línea de salida que indique el fallo de un test, en la vista de mensajes, para que se abra el editor de JBuilder con la línea que ha provocado el fallo resaltada.

## JUnit SwingUI

---

SwingUI de JUnit es un ejecutor de tests que proporciona una interfaz gráfica en la que se indica el estado de los tests y mensajes de texto en los que se indican los fallos. Aunque es posible ejecutar tests con SwingUI desde el IDE de JBuilder, no es posible pulsar una línea de texto que indique un fallo y pasar directamente a esta línea en el editor, como ocurre con JBTestRunner y con JUnit TextUI. La ventaja de SwingUI consiste en que se puede revisar la jerarquía del test y volver a ejecutar los métodos uno a uno.

## Configuraciones de ejecución

---

Las configuraciones de ejecución contienen los parámetros de MV que se deben utilizar, y en el caso de los tests, el ejecutor de tests que se utiliza. Con el fin de definir las propiedades de una configuración de ejecución para realizar tests, elija Proyecto | Propiedades de proyecto, abra la ficha Ejecutar, pulse el botón Nuevo, Copiar o Modificar y haga clic en la pestaña Test. Aquí se pueden establecer los parámetros de máquina

virtual que se utilizan para la ejecución de los tests, así como seleccionar un ejecutor de tests. Además de la configuración de ejecución por defecto se pueden definir configuraciones adicionales en la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto, así como en Ejecutar | Configuraciones. Los Asistentes para tests y conjuntos de tests también permiten definir configuraciones de ejecución.

**Sugerencia** También es posible ejecutar tests con la configuración para la ejecución de aplicaciones; para ello, llame al método `main()` del ejecutor de tests TextUI. Esto puede resultar útil si se desea escribir un script que llame al ejecutor de tests con argumentos de línea de comandos.

### Consulte

- “[Definición de las configuraciones para la ejecución](#)” en la página 7-7

## Definición del filtro de seguimiento de la pila de tests

---

Un filtro de seguimiento de pila de tests permite especificar los paquetes y clases para excluir de los seguimientos de pila al ejecutar los tests de módulos mediante JBTestRunner. Las líneas del seguimiento de la pila de los paquetes y clases excluidos no aparecen. Esto le permite concentrarse en la información de seguimiento de la pila que le resulte útil.

Para especificar un filtro de seguimiento de la pila de la unidad en test:

- 1 Seleccione Proyecto | Propiedades de proyecto.
- 2 Seleccione la ficha Filtros de clase del cuadro de diálogo Propiedades de proyecto.
- 3 Seleccione Seguimiento de la pila de la unidad en test en la lista desplegable.
- 4 Utilice los botones Añadir y Eliminar para especificar los paquetes y las clases que se van a excluir.
- 5 Pulse Aceptar.

El filtro de seguimiento de la pila de la unidad en test excluye por defecto los siguientes paquetes y clases:

- `junit.framework.*`
- `java.lang.reflect.Method`
- `com.borland.jbuilder.unittest.JBTestRunner`
- `sun.reflect.NativeMethodAccessorImpl`
- `sun.reflect.DelegatingMethodAccessorImpl`

Utilice los botones Añadir y Eliminar de la ficha Filtros de clase del cuadro de diálogo Propiedades de proyecto para modificar esta lista.

## Depuración de tests

---

La depuración de tests de módulos es parecida a la depuración de otro código mediante el depurador de JBuilder. La única diferencia radica en que al depurar un test, aparecen las pestañas Jerarquía del test y Fallos del test de JBTestRunner además de la interfaz del usuario normal del depurador. Para depurar un test, haga clic con el botón derecho del ratón en cualquier test del panel del proyecto, y elija Depurar test en el menú contextual.

- Sugerencia** Cuando se ejecutan tests con JBTestRunner o TextUI y se pulsa un error del Resultados del test, la línea de código que ha provocado el fallo se muestra resaltada en el editor. Si se vuelve a hacer clic en el margen izquierdo de la ventana del editor, junto a la línea de código resaltada, se coloca un punto de interrupción en ella. De esta forma se puede realizar la depuración cómodamente hasta llegar al punto de interrupción.
- Sugerencia** Es posible definir un punto de interrupción en la excepción lanzada en el fallo de un test. Para ello, elija Ejecutar | Añadir punto de interrupción | Añadir punto de interrupción por excepción y escriba junit.framework.AssertionFailedError en Nombre de la clase.

### Consulte

- “JBTestRunner” en la página 13-15
- Capítulo 8, “Depuración de programas en Java”



# 14

## Creación de Javadoc a partir de archivos fuente de la API

Es una función de  
JBuilder SE y Enterprise.

Javadoc es una herramienta creada por Sun Microsystems para generar documentación de la API en archivos con formato HTML. La documentación en HTML se crea a partir de comentarios a nivel de clases y método que se introducen en los archivos fuente API. Los comentarios deben tener un formato acorde con la norma Javadoc. Si desea información más completa acerca de la herramienta Javadoc, puede dirigirse a la página principal de esta herramienta en la página web de Sun, <http://www.java.sun.com/j2se/javadoc/>

JBuilder incorpora varias funciones para la creación de Javadoc. El asistente crea un nodo de documentación con propiedades para la ejecución del Javadoc. Este nodo aparece en el panel del proyecto. Se puede crear Javadoc siempre que se cree un proyecto, utilizando las propiedades adecuadas.

JBuilder también incluye las siguientes funciones relacionadas con Javadoc:

- Una plantilla de comentarios que se rellena con los parámetros basados en la clase, la interfaz, el método, el campo o la firma del constructor
- Una plantilla para añadir etiquetas `@todo`.
- Información sobre los conflictos en los comentarios Javadoc.
- Un visualizador para los nuevos documentos Javadoc.
- Generación de Javadoc sobre la marcha.
- Recopilación de documentación con el Creador de recopilatorios.

## Adición de comentarios Javadoc a los archivos fuente API

Se pueden añadir comentarios Javadoc de clase e interfaz y así como comentarios de método, constructor y campo. La herramienta Javadoc extrae los comentarios Javadoc de los archivos fuente, y los coloca en los archivos de documentación con formato HTML.

Un comentario Javadoc empieza con un símbolo de comienzo de comentario (/\*\*) y termina con un símbolo de final de comentario (\*/). Cada comentario consta de una descripción seguida de una o más etiquetas. Si lo desea, puede utilizar el formato HTML en los comentarios Javadoc. Al escribir sus comentarios, siga los siguientes pasos:

- Alinee el símbolo de comienzo de comentario (/\*\*) de tal manera que coincida con el código al que hace referencia.
- Comience las siguientes líneas del comentario con \* (asterisco). Alinee también estas líneas.
- Escriba el texto con la descripción en la línea siguiente al símbolo de comienzo de comentario (\*\*).
- Introduzca un espacio en blanco delante del texto o de la etiqueta.
- Introduzca una línea de comentario en blanco entre el texto y la lista de etiquetas.

A continuación se muestra un ejemplo de comentario Javadoc para un método:

```
/**  
 * Sets this check box's label to the string argument.  
 *  
 * @param label a string to set as the new label, or null for no label.  
 */
```

En el archivo HTML generado, este comentario aparece de la siguiente manera:

Sets this check box's label to the string argument.

### Parameters:

label - a string to set as the new label, or null for no label.

Observe cómo Javadoc convierte la etiqueta @param en un título. También añade el guión que separa el nombre del parámetro de su descripción. Además, muestra el nombre del parámetro utilizando una fuente para códigos.

Al escribir la descripción del comentario, escriba la primera frase a modo de resumen. Debería ser una descripción concisa y completa del elemento API. La herramienta Javadoc copia la primera frase del comentario en la clase, interfaz o tabla resumen miembro.

**Nota** La herramienta Javadoc hereda los comentarios para métodos que implementan o que redefinen otros métodos. En estos casos, no es necesario duplicar estos comentarios.

Si desea información adicional acerca de la creación de comentarios Javadoc, consulte "How to Write Doc Comments for the Javadoc Tool" en <http://www.java.sun.com/j2se/javadoc/writingdoccomments/index.html>.

### Consulte

- ["Creación automática de etiquetas Javadoc"](#) en la página 14-7
- ["Javadoc Tags"](#) (equipos Windows) en <http://java.sun.com/j2se/1,4/docs/tooldocs/win32/javadoc.html#javadoctags>
- ["Javadoc Tags"](#) (equipos Solaris) en <http://java.sun.com/j2se/1,4/docs/tooldocs/solaris/javadoc.html#javadoctags>

## Colocación de los comentarios Javadoc

---

Puede añadir comentarios Javadoc para clases, interfaces, métodos, campos y constructores. Coloque los comentarios de clase e interfaz en la parte superior del archivo, después de las sentencias `import` y justo antes de la sentencia de declaración `class` o `interface`. Por ejemplo, el comentario de clase para `com.borland.internetbeans.PageProducer.java` es:

```
package com.borland.internetbeans;

import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.servlet.ServletContext;
import javax.servlet.http.*;

/**
 * Generates markup text from a template file, replacing
 * identified spans with dynamic content from IX
 * components.
 */
public class PageProducer implements Binder, Renderable, Cloneable,
Serializable {
...
}
```

Puede introducir comentarios de clase en Archivos Javadoc de clases de la ficha General del cuadro de diálogo Propiedades de proyecto. Los comentarios que introduzca aquí se añaden a todas las clases o interfaces que cree con un asistente de JBuilder.

Los comentarios para métodos, campos y constructores se colocan inmediatamente antes de la firma del método en el archivo fuente de clases. Por ejemplo, si su código fuente cuenta con los métodos siguientes:

```
public void addValues(Double valueOneDouble, Double valueTwoDouble) {
    double valueOneDoubleResult = valueOneDouble.doubleValue();
    double valueTwoDoubleResult = valueTwoDouble.doubleValue();
    addResult = (valueOneDoubleResult + valueTwoDoubleResult);
    addStringResult = Double.toString(addResult);
    addResultDisplay.setText(addStringResult);
}

public void subtractValues(Double valueOneDouble, Double valueTwoDouble) {
    double valueOneDoubleResult = valueOneDouble.doubleValue();
    double valueTwoDoubleResult = valueTwoDouble.doubleValue();
    subtractResult = (valueOneDoubleResult - valueTwoDoubleResult);
    subtractStringResult = Double.toString(subtractResult);
    subtractresultDisplay.setText(subtractStringResult);
}
```

Tiene que añadir los comentarios Javadoc justo antes de la declaración de método. Los métodos resultantes y los comentarios Javadoc tendrán la apariencia que se muestra a continuación. Los comentarios Javadoc aparecen en negrita.

```
/**
 * Adds Value One and Value Two and displays result.
 *
 * @param valueOneDouble The first value.
 * @param valueTwoDouble The second value.
 */
public void addValues(Double valueOneDouble, Double valueTwoDouble) {
    double valueOneDoubleResult = valueOneDouble.doubleValue();
    double valueTwoDoubleResult = valueTwoDouble.doubleValue();
    addResult = (valueOneDoubleResult + valueTwoDoubleResult);
    addStringResult = Double.toString(addResult);
    addResultDisplay.setText(addStringResult);
}

/**
 * Subtracts Value One and Value Two and displays result.
 *
 * @param valueOneDouble The minuend.
 * @param valueTwoDouble The subtrahend.
 */
public void subtractValues(Double valueOneDouble, Double valueTwoDouble) {
    double valueOneDoubleResult = valueOneDouble.doubleValue();
    double valueTwoDoubleResult = valueTwoDouble.doubleValue();
    subtractResult = (valueOneDoubleResult - valueTwoDoubleResult);
    subtractStringResult = Double.toString(subtractResult);
    subtractresultDisplay.setText(subtractStringResult);
}
```

## Etiquetas Javadoc

En los comentarios Javadoc se pueden utilizar las siguientes etiquetas. Algunas de ellas, como `@param` y `@return` se incluyen automáticamente en la ejecución de Javadoc que se inicia en JBuilder. El asistente utilizado para crear Javadoc permite seleccionar otras etiquetas en la ficha Especificar opciones doclet de línea de comandos. También puede introducir otras etiquetas en el campo Opciones adicionales de la misma ficha, con lo que se obliga al asistente a incluir esas etiquetas de comentarios en los archivos creados.

En la siguiente tabla se recogen y se describen las etiquetas Javadoc. Indica el doclet de Javadoc para el que se van a procesar las etiquetas. Por ejemplo, no todas las etiquetas las va a procesar el JDK 1.1. También indica a qué parte del archivo de clases se puede aplicar la etiqueta. Para obtener más información sobre las etiquetas individuales, consulte.

- **Usuarios de Windows - “Javadoc Tags”** en <http://java.sun.com/j2se/1,4/docs/tooldocs/win32/javadoc.html#javadoctags>
- **Usuarios de Solaris - “Javadoc Tags”** en <http://java.sun.com/j2se/1,4/docs/tooldocs/solaris/javadoc.html#javadoctags>

**Tabla 14.1** Etiquetas Javadoc

Etiqueta	Descripción	Doclet JDK 1.1	Docklet estándar	Tipo de etiqueta
<code>@author nombre</code>	Añade a los documentos creados una entrada Author con el <i>nombre</i> especificado si la opción <code>@author</code> se ha seleccionado en la ficha Especificar opciones doclet de línea de comandos del asistente utilizado para crear Javadoc.	X	X	Aspectos generales, paquete, clase, interfaz
<code>{@docRoot}</code>	Es la vía de acceso relativa desde cualquier ficha al directorio raíz o destino de los documentos generados. Este asistente no define automáticamente esta opción; es necesario introducirla en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos.		X	Aspectos generales, paquete, clase, interfaz, campo
<code>@version número-de-versión</code>	Añade a los documentos creados un subtítulo Version con el <i>número de versión</i> especificado si la opción <code>@version</code> se utiliza en la ficha Especificar opciones doclet de línea de comandos del asistente.	X	X	Aspectos generales, paquete, clase, interfaz
<code>@param descripción-del-nombre-del-parámetro</code>	Añade un parámetro al subtítulo Parameters. Incluido automáticamente en documentos generados.	X	X	Método Constructor
<code>@return descripción</code>	Añade un subtítulo Returns con el texto de la <i>descripción</i> . Incluido automáticamente en documentos generados.	X	X	Método Constructor

Tabla 14.1 Etiquetas Javadoc (continuación)

Etiqueta	Descripción	Doclet JDK 1.1	Docklet estándar	Tipo de etiqueta
<code>@deprecated texto-desaconsejado</code>	Añade un comentario que indica que se ha desaconsejado la API y no debe volver a utilizarse, aunque todavía funcione. Esta opción se puede configurar en la ficha Especificar opciones doclet de línea de comandos del asistente.	X	X	Paquete, clase, interfaz, campo, constructor, método
<code>@exception descripción-del-nombre-de-clase</code>	Es un sinónimo de <code>@throws</code> . Incluido automáticamente en documentos generados.	X	X	Método Constructor
<code>@throws descripción-del-nombre-de-clase</code>	Es un sinónimo de <code>@exception</code> . Añade un subtítulo <code>Throws</code> a los documentos generados con la excepción <i>nombre de clase</i> que se puede lanzar. Incluido automáticamente en documentos generados.		X	Método Constructor
<code>@see referencia</code>	Añade un subtítulo <code>See Also</code> a los documentos generados. Incluido automáticamente en documentos generados.	X	X	Aspectos generales, paquete, clase, interfaz, campo, constructor, método
<code>@since texto-since</code>	Añade un título <code>Since</code> con el <i>texto</i> especificado a los documentos generados. Este asistente no define automáticamente esta opción; es necesario introducirla en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos.	X	X	Aspectos generales, paquete, clase, interfaz, campo, constructor, método
<code>@serial descripción-del-campo</code>	Describe un campo serializable por defecto. Este asistente no define automáticamente esta opción; es necesario introducirla en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos.		X	Campo
<code>@serialField nombre-del-campo tipo-de-campo descripción-del-campo</code>	Documenta un componente <code>ObjectStreamField</code> de un miembro de la clase serializable <code>serialPersistentFields</code> . Este asistente no define automáticamente esta opción; es necesario introducirla en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos.		X	Campo

**Tabla 14.1** Etiquetas Javadoc (continuación)

Etiqueta	Descripción	Doclet JDK 1.1	Docklet estándar	Tipo de etiqueta
<code>@serialData descripción-de-datos</code>	Documenta el tipo y el orden de los datos de forma serializada. Este asistente no define automáticamente esta opción; es necesario introducirla en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos.		X	Método Constructor
<code>{@link paquete.clase#etiquet a del miembro}</code>	Inserta un enlace en línea con la <i>etiqueta</i> como texto visible. Incluido automáticamente en documentos generados.		X	Aspectos generales, paquete, clase, interfaz, campo, constructor, método

## Creación automática de etiquetas Javadoc

JBuilder puede crear algunas de estas etiquetas de forma automática. Si el cursor está en una línea anterior a una declaración de clase o interfaz, al escribir el símbolo de comienzo de comentario (/\*\*) y pulsar *Intro* inserta la siguiente plantilla en el código:

```
/**  
 * <p>Título: </p>  
 * <p>Descripción: </p>  
 * <p>Copyright: Copyright (c) 2001</p>  
 * <p>Empresa: </p>  
 * @author  
 * @versión 1.0  
 */
```

Puede llenar los campos como deseé.

**Nota** Para un proyecto nuevo, esta plantilla (para clases e interfaces) se puede llenar en la ficha Especificar configuración general del proyecto del Asistente para proyectos a la vez que se crea el proyecto. Estos valores se pueden modificar en cualquier momento en la ficha General del cuadro de diálogo Propiedades de proyecto.

Si el cursor se encuentra delante de un método, campo o firma del constructor, al escribir /\*\* se inserta la plantilla que se indica a continuación. Tenga en cuenta que solamente aparecen en la plantilla ampliada de comentario las etiquetas utilizadas en la firma.

```
/**  
 *  
 * @param  
 * @throws
```

```
* @returns
*/
```

JBuilder completa la etiqueta rellenando el nombre del parámetro o la excepción. Por ejemplo, en el caso de la siguiente firma de método:

```
public void addValues(Double valueOneDouble, Double valueTwoDouble)
```

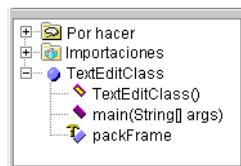
al escribir `/**` se crea el siguiente comentario Javadoc:

```
/**
 *
 * @param valueOneDouble
 * @param valueTwoDouble
 */
```

## Etiquetas @todo Javadoc

Las etiquetas `@todo` de Javadoc resultan útiles para recordar que es necesario hacer algo en una zona del código. Estas etiquetas se colocan dentro de los comentarios Javadoc. Estas etiquetas `@todo` aparecen en el panel de estructura de JBuilder en la carpeta `Por Hacer`.

**Figura 14.1** Carpeta Por Hacer del panel de estructura



Si desea añadir etiquetas `@todo` a su código:

- 1 Escriba `todo` en el nivel adecuado de sangrado del editor.
- 2 A continuación pulse `Ctrl+J` para ampliar la plantilla en el código.

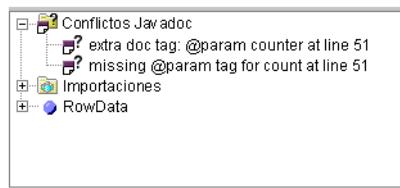
```
/** @todo */
```

Algunos asistentes de JBuilder generan etiquetas `@todo` para recordar que es necesario añadir código al stub generado

## Conflictos en comentarios Javadoc

Los conflictos Javadoc tienen lugar cuando las etiquetas de un comentario Javadoc no coinciden con la firma del método, o bien, si no se ofrece ningún argumento en etiquetas como `@param`. Por ejemplo, si la firma del método contiene dos parámetros y el comentario solamente uno, se produce un conflicto.

En JBuilder, los conflictos Javadoc se registran en la parte superior del panel de estructura, en la carpeta `Conflictos Javadoc`. Amplíe la carpeta y pulse en el conflicto si desea ir a la firma del método donde tuvo lugar el conflicto.

**Figura 14.2** Conflictos Javadoc en el panel de estructura

**Nota** No se informa de los conflictos Javadoc hasta que se hayan solventado todos los errores de sintaxis de la carpeta Errores.

## Generación del nodo de documentación

---

El Asistente para Javadoc de JBuilder genera un nodo de documentación en el panel del proyecto. En este nodo se almacenan las propiedades para la ejecución de Javadoc. Entre las propiedades se incluyen el formato de la documentación Javadoc, qué paquetes se documentan y qué archivos de salida se generan para esos paquetes. Si desea cambiar las propiedades Javadoc después de crear el nodo, pulse con el botón derecho del ratón y, a continuación, seleccione Propiedades.

Durante la creación del nodo, tiene la opción de crear también archivos Javadoc cada vez que se genera un proyecto. También puede crear Javadoc solamente cuando lo necesite si pulsa con el botón derecho del ratón sobre el nodo y selecciona Ejecutar Make.

Para abrir el Asistente para Javadoc, seleccione Archivo | Nuevo. En la ficha Generar de la galería de objetos, haga doble clic sobre el icono Javadoc. También puede seleccionar Asistentes | Javadoc.

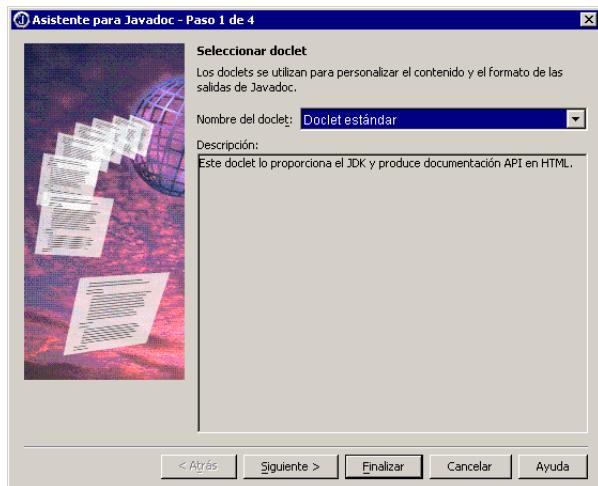
El asistente consta de cuatro pasos. Las opciones del asistente incluyen:

- El formato de los archivos generados por Javadoc.
- El nombre del nodo de documentación que crea el asistente.
- El directorio de salida.
- Cuándo ejecutar Javadoc.
- Los paquetes y el ámbito documentados.
- Qué archivos de salida se generan y qué etiquetas se procesan.

## Selección del formato de la documentación

---

En el primer paso del asistente se elige el formato de la documentación Javadoc. Estos archivos están controlados por un doclet, una clase Java que especifica los contenidos y el formato de los archivos de salida HTML.

**Figura 14.3** Paso de selección de doclet

Para seleccionar el formato de los archivos de salida:

- 1 Si desea crear archivos de salida Javadoc con formato JDK 1.1, seleccione la opción Doclet JDK 1.1 de la lista desplegable Nombre del doclet. Este doclet crea la documentación con formato HTML, pero no incluye el nivel adicional de detalles que proporciona la opción Doclet estándar. Esta opción corresponde a la etiqueta Javadoc **-1.1**. Tenga en cuenta que este doclet no se encuentra disponible si se ejecuta Javadoc utilizando JDK 1.4.
- 2 Si desea crear archivos de salida con formato JDK 1.4, seleccione la opción Doclet estándar en la lista desplegable. Este doclet crea la documentación API con formato HTML, e incluye más características que la opción que permite crear archivos JDK 1.1, entre las que se encuentran:
  - Tablas de campos y métodos para una clase o interfaz.
  - Descripciones de nivel de paquetes.
  - Listas de campos y métodos heredados.
  - Listas de clases internas.
  - Un índice por letra.
  - Comentarios sobre la etiqueta `@use`.
  - Barra de desplazamiento ampliable.

Si desea un ejemplo de archivos de salida estándar, seleccione Ayuda | Referencia de Java en la barra del menú principal de JBuilder. Aparece entonces la documentación de referencia de la API de JDK de Sun. Utiliza el doclet JDK 1.4 para dar formato a la documentación.

Los dos doclets utilizan convenciones de nomenclatura HTML y estructuras de directorio diferentes. Cuando aparece Javadoc, la ficha Documentación busca los archivos formateados mediante la opción Doclet

estándar. Si no encuentra este tipo de archivo, JBuilder busca a continuación archivos formateados mediante el doclet JDK 1.1. Para obtener más información, consulte “[Visualización de Javadoc](#)” en la [página 14-22](#).

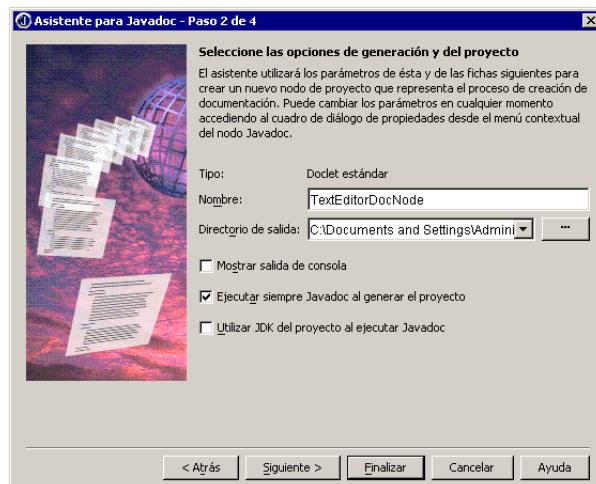
**Nota** La opción Nombre del doclet se corresponde con la opción **-doclet** de Javadoc. El asistente para Javadoc configura de forma explícita la opción **-docletpath**.

## Selección de opciones para generar documentación

En el segundo paso del asistente, se seleccionan:

- El nombre del nodo de documentación que genera el asistente.
- El directorio de salida.
- Cómo se presenta la documentación Javadoc.
- Las opciones de generación de Javadoc.

**Figura 14.4** Paso de selección de opciones de generación y del proyecto



Para seleccionar las opciones de generación y del proyecto:

- 1 Escriba el nombre del nodo de documentación en el campo Nombre. Este nombre aparece en el panel del proyecto. Por defecto, aparece el tipo de doclet que se seleccionó en el paso anterior. Puede cambiarlo por cualquier otro nombre descriptivo.
- 2 Escriba el directorio de salida de la documentación en el campo Directorio de salida. Esta es la vía de salida para los archivos HTML. El asistente utiliza la primera vía de acceso al directorio configurada en la pestaña Documentación de la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto). Si no ha configurado una vía de acceso a la documentación, el asistente

le sugiere un directorio `doc` en el directorio del proyecto y, a continuación, la crea.

Utilice el botón de puntos suspensivos para desplazarse hasta un nuevo directorio. Si no hay ninguno, JBuilder se encarga de crearlo. También puede seleccionar en la lista desplegable una vía de acceso que se haya utilizado anteriormente. Estas vías de acceso corresponderían a otras vías de acceso a la documentación configuradas en su proyecto.

Un solo proyecto puede tener varias vías de acceso Javadoc, con lo que, por ejemplo, paquetes diferentes pueden utilizar distintas opciones Javadoc. Dos proyectos no deberían compartir la misma vía de acceso. Cada nodo tiene su propia vía de acceso.

Esta opción corresponde a la opción Javadoc `-d`. El asistente configura las opciones Javadoc `-sourcepath`, `-classpath`, y `-bootclasspath`, basándose en las configuraciones del proyecto.

**Nota** Por motivos de mantenimiento, la documentación Javadoc deben mantenerse en su propio directorio, en lugar de colocarlos en los directorios de salida o fuente. Consulte “[Mantenimiento de Javadoc](#)” en [la página 14-25](#) para obtener más información.

**3** Seleccione la opción Mostrar salida de consola para que aparezcan las advertencias Javadoc en el panel de mensajes mientras se generan los archivos HTML. Esta opción se corresponde con la opción Javadoc `-verbose`.

**Nota** La generación de Javadoc se detiene si se produce algún error. Los errores aparecen en el panel de mensajes, esté o no activada la opción Mostrar salida de consola.

**4** Seleccione la opción Ejecutar siempre Javadoc al generar el proyecto para generar archivos Javadoc cada vez que cree un proyecto. Puede desactivar esta opción al desarrollar el proyecto, ya que así se reduce de forma significativa el tiempo de compilación.

Si no elige esta opción, puede crear los archivos Javadoc en cualquier momento; solamente tiene que pulsar dos veces con el botón derecho en el nodo de documentación del panel del proyecto y seleccionar Ejecutar Make.

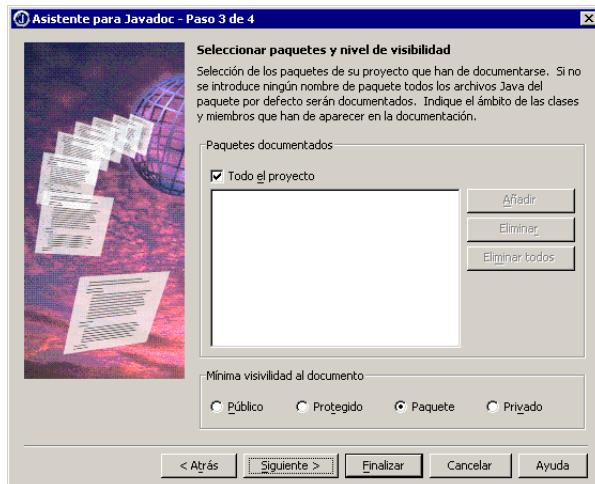
**5** Seleccione la opción Utilizar JDK del proyecto al ejecutar Javadoc para utilizar la versión del JDK que se especifica en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto. De lo contrario, Javadoc se ejecutaría utilizando el JDK que hospeda a JBuilder.

**Nota** El Asistente para Javadoc utiliza el valor de la opción Codificación de la ficha General del cuadro de diálogo Propiedades de proyecto.

## Selección de paquetes que se han de documentar

En este paso del asistente se seleccionan los paquetes que se han de documentar y qué ámbito de clases y miembros han de aparecer en la documentación.

**Figura 14.5** Paso de selección de paquetes y nivel de visibilidad



Para seleccionar el paquete y el nivel de visibilidad:

- 1 Marque la opción Todo el proyecto para que se documenten todos los paquetes del proyecto. Esta opción se activa por defecto, para que se documenten todos los paquetes. Desactívela si desea seleccionar paquetes que se han de documentar de forma individualizada.
- Nota** Por defecto, siempre se documentan los archivos fuente del paquete.
- 2 Desactive la opción Todo el proyecto si desea que los paquetes se documenten de forma individualizada. Los paquetes del proyecto aparecen en la lista Paquetes que se han de documentar.
    - Seleccione el botón Añadir para poder añadir los paquetes a la lista. Aparece entonces el cuadro de diálogo Seleccionar paquetes que se han de documentar, en el que puede elegir de forma individualizada los paquetes para el proyecto.
    - Seleccione un paquete y, a continuación, pulse el botón Eliminar si desea eliminar un solo paquete de la lista.
    - Seleccione el botón Eliminar todos si lo que desea es eliminar todos los paquetes de la lista.
  - 3 Seleccione una de las opciones Mínima visibilidad al documento si desea elegir el ámbito de clases y miembros que se van a incluir en la documentación:

- Pública: incluye en la documentación solamente clases y miembros públicos. Esta opción se corresponde con la opción Javadoc **-public**.
- Protegida: incluye en la documentación clases y miembros protegidos y públicos. Esta opción se corresponde con la opción Javadoc **-protected**.
- De paquete: incluye clases y miembros protegidos, públicos y de paquete. Esta opción se corresponde con la opción Javadoc **-package**.
- Privada: incluye en la documentación todas las clases y miembros, excepto aquellos con visibilidad privada. Esta opción se corresponde con la opción Javadoc **-private**.

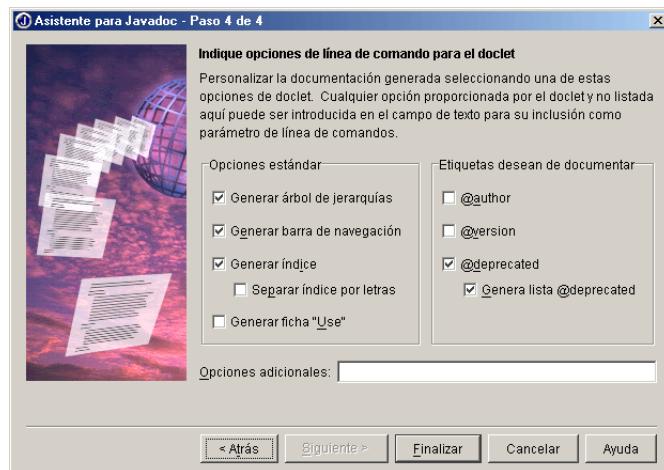
**Nota** Si un archivo fuente no tiene elementos con comentarios Javadoc que cumplan los requisitos mínimos de visibilidad, la herramienta Javadoc no crea un archivo HTML para esa clase.

## Especificación de opciones de línea de comandos para el doclet

---

En el último paso del asistente se determina qué archivos de salida se generan y qué etiquetas se procesan. Todas las selecciones de esta ficha son optativas, ninguna es necesaria para que se genere Javadoc.

**Figura 14.6** Paso de especificación de opciones doclet de línea de comandos



Para especificar opciones de línea de comandos:

- 1 Seleccione la opción Generar árbol de jerarquías para crear el árbol de jerarquías para todas las clases de todos los paquetes. Un árbol de jerarquías es una lista de las jerarquías de todos los paquetes, clases e interfaces de la documentación. En el caso del doclet estándar, el árbol de jerarquías se crea a nivel de paquetes. El árbol de jerarquía se guarda en el archivo `overview-tree.html`, en el directorio raíz de la vía de acceso a la documentación.
- 2 Elija la opción Generar barra de navegación para crear una barra de navegación en la parte superior de cada archivo de salida HTML. Esta barra incluye enlaces a la clase y al paquete anterior y posterior, los aspectos generales de todos los paquetes de la documentación, el archivo del árbol, el índice y el tema de ayuda generado por Javadoc.
- 3 Seleccione la opción Generar índice si desea crear una entrada de índice para cada método, campo, paquete, clase e interfaz. El índice se guarda en el archivo `index-all.html`, en el directorio raíz de la vía de acceso a la documentación. Si se desactiva esta opción, se correspondería con la opción Javadoc **-noindex**.

Para la documentación Javadoc en JDK 1.4, se pueden crear enlaces a las entradas de índice por letras. Seleccione la opción Separar índice por letras. Esta opción se corresponde con la opción Javadoc **-splitindex**. Esta opción no se tiene en cuenta en el caso del doclet JDK 1.1, ya que este doclet siempre crea un índice independiente para cada letra.

- 4 Seleccione la opción “Uso” de la ficha Generar para generar una página “Uso” para cada paquete y otra para cada clase e interfaz. El archivo Use del paquete tiene el nombre `package-use.html`; el archivo Use de la clase tiene el nombre `class-use/classname.html`. En esta ficha se describe qué paquetes, clases, métodos, constructores y campos utilizan cada parte de la clase, interfaz o paquete. Esta opción no se tiene en cuenta para el tipo de doclet JDK 1.1.
- 5 Seleccione la opción `@author` si desea generar la documentación para las etiquetas `@author` del código fuente. Esta opción permite añadir el nombre del autor a los archivos generados Javadoc. En una misma etiqueta se pueden incluir uno o varios nombres.
- 6 Elija la opción `@version` para crear la documentación para las etiquetas `@version` del código fuente. Esta opción permite añadir el número de versión del código a los archivos generados Javadoc. Se puede aplicar tanto a una clase como a un elemento dentro de ella.
- 7 Seleccione la opción `@deprecated` si desea crear la documentación para las etiquetas `@deprecated` de su código fuente. Esta opción permite añadir un comentario que indica que el elemento API que se especifica se eliminará en una versión posterior de la API.

- 8 Elija la opción Generar lista @deprecated si desea crear una lista de elementos @deprecated. Si esta opción está desactivada, se correspondería con la opción Javadoc **-nodeprecatedlist**.
- 9 Puede introducir más opciones en el campo Opciones adicionales. Estas opciones se añaden a la línea de comandos anterior a la lista de paquetes o archivos. Las opciones que configure en este campo sustituyen a las opciones configuradas en el asistente. Observe que si especifica la opción **-locale**, siempre aparece como la primera opción de la línea de comandos.
- 10 Haga clic en Finalizar para crear el nodo de documentación.

En la siguiente tabla se enumeran las opciones que no se pueden configurar en el asistente. Estas opciones se pueden configurar en el campo Opciones adicionales. La tabla indica para qué doclet Javadoc se van a procesar las opciones. Por ejemplo, no todas las opciones las procesa el doclet JDK 1.1. Si desea más información acerca de las opciones Javadoc, consulte:

- Usuarios de Windows: "Javadoc options" en <http://java.sun.com/j2se/1.4/docs/tooldocs/win32/javadoc.html#javadoptions>
- Usuarios de Solaris: "Javadoc options" en <http://java.sun.com/j2se/1.4/docs/tooldocs/solaris/javadoc.html#javadoptions>

**Tabla 14.2** Opciones que no se configuran en el asistente

Opción	Descripción	Doclet JDK 1.1	Docklet estándar
<b>-overview</b> <i>vía de acceso\nnombre del archivo</i>	Especifica el archivo que contiene el texto para la documentación de aspectos generales.	X	X
<b>-help</b>	Muestra la ayuda Javadoc, que contiene una relación de las opciones Javadoc y de línea de comandos.	X	X
<b>-J</b> <i>flag</i>	Pasa el <i>flag</i> directamente al JDK de tiempo de ejecución que ejecuta Javadoc. No incluya un espacio entre la <b>J</b> y el <i>flag</i> . Utilice esta opción para aumentar la memoria de Javadoc, por ejemplo, <b>-J-Xms64m</b> . (El flag <b>-Xms</b> establece el tamaño de la memoria inicial. Puede utilizarlo junto con el flag <b>-Xmx</b> si desea aumentar la memoria disponible.)	X	X
<b>-locale</b> <i>idioma_país_variante</i>	Especifica la lengua que utiliza Javadoc al crear la documentación. Javadoc selecciona los archivos de recursos de la lengua elegida para los mensajes tales como las cadenas de la barra de navegación, los títulos de las listas y tablas, los contenidos de los archivos de ayuda y los comentarios de la hoja de estilos. También especifica la clasificación de las listas alfabéticas.	X	X
<b>-doctitle</b> <i>title</i>	Define el título que se tiene que colocar en la parte superior del archivo resumen de aspectos generales, por debajo de la barra de navegación superior.		X
<b>-windowtitle</b> <i>título</i>	Especifica qué título hay que colocar en la etiqueta <b>&lt;title&gt;</b> .		X

**Tabla 14.2** Opciones que no se configuran en el asistente (continuación)

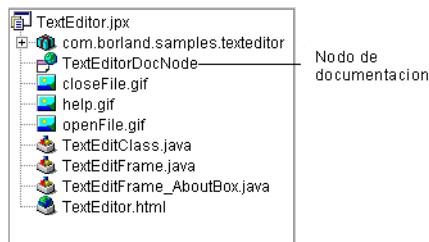
Opción	Descripción	Doclet JDK 1.1	Docklet estándar
<b>-header</b> <i>encabezado</i>	Define el texto de encabezado que hay que colocar en la parte superior de cada archivo HTML generado, a la derecha de la barra de navegación superior.		X
<b>-footer</b> <i>pie de página</i>	Especifica qué texto de pie de página se coloca en la parte inferior de cada archivo HTML generado, a la derecha de la barra de navegación inferior.		X
<b>-bottom</b> <i>texto</i>	Define el texto que se coloca en la parte inferior de cada archivo HTML generado, debajo de la barra de navegación inferior.		X
<b>-link</b> <i>URL de la documentación externa</i>	Crea enlaces con la documentación Javadoc existente para las clases externas referenciadas.		X
<b>-linkoffline</b> <i>ubicación de la lista de paquetes en la URL de documentación externa</i>	Crea enlaces con la documentación Javadoc existente para las clases externas referenciadas, en el caso de que el archivo de lista de paquetes no exista en la ubicación <i>extdocURL</i> . (Consulte <b>-link</b> .)		X
<b>-group</b> <i>cabecera de grupo modelo de paquete:modelo de paquete:...</i>	Separa los paquetes de la lista de aspectos generales en los grupos especificados.		X
<b>-nosince</b>	No incluye comentarios en las etiquetas <code>@since</code> .		
<b>-nohelp</b>	No incluye el enlace Ayuda de las barras de navegación superior e inferior.		X
<b>-helpfile</b> <i>vía de acceso\nombre del archivo</i>	Especifica la vía de acceso a un archivo de ayuda sustitutivo para el enlace Ayuda de la barra de navegación.		X
<b>-stylesheetfile</b> <i>vía de acceso\nombre del archivo</i>	Define la vía de acceso a un archivo de hoja de estilos sustitutivo.		X
<b>-serialwarn</b>	Genera advertencias del compilador para las etiquetas <code>@serial</code> que falten.		X
<b>-charset</b> <i>nombre</i>	Define el conjunto de caracteres HTML para la documentación generada.		X
<b>-doencoding</b> <i>nombre</i>	Especifica la codificación de la documentación generada.		X

**Importante** Si en el campo Opciones adicionales introduce opciones que ya se han configurado en el asistente, estas opciones sustituyen a las anteriores.

## Creación de archivos de salida

Una vez que haya configurado todas las opciones en el asistente y que haya pulsado el botón Finalizar, se crea el nodo de documentación en el panel del proyecto.

**Figura 14.7** Nodo de documentación en el panel del proyecto



Los archivos HTML no están disponibles hasta que no se generan. Para generar los archivos HTML:

- El proyecto se genera si se ha configurado la opción Ejecutar siempre Javadoc al crear el proyecto de la ficha Especificar opciones de proyecto y generación del asistente.
- Pulse con el botón derecho del ratón en el nodo de documentación y seleccione Ejecutar Make. Esta opción sólo crea Javadoc, no crea el proyecto. Para borrar archivos HTML en el directorio configurado, seleccione Limpiar. Seleccione Generar de nuevo si desea limpiar y, después, generar.

La información aparece en la ficha Compilador del panel de mensajes, si se ha activado la opción Mostrar salida de consola de la ficha Seleccione las opciones de generación y del proyecto en el asistente.

Javadoc genera un conjunto de archivos con formato HTML, basados en las propiedades seleccionadas y que se colocan en el directorio de salida seleccionado, que normalmente es el directorio `doc`. Los archivos de salida para la opción Doclet estándar incluyen:

- Un archivo `classname.html` para cada clase o interfaz del proyecto, que contiene la documentación para la clase o interfaz.
- Un archivo `package-summary.html` para cada paquete del proyecto, que enumera las clases y paquetes, y que ofrece una descripción general.
- Un archivo `overview-summary.html` para el conjunto completo de paquetes, que es la página principal de la documentación. Este archivo solamente se crea si su proyecto contiene dos o más paquetes y ha utilizado la opción **-overview** del campo Opciones adicionales, en la

ficha Especificar opciones doclet de línea de comandos del asistente utilizado para crear Javadoc.

- Un archivo `overview-tree.html` para la jerarquía de clases del conjunto entero de paquetes.
- Un archivo `package-tree.html` para la jerarquía de clases de cada paquete.
- Un archivo `package-use.html` para cada paquete, clase e interfaz, que determina qué paquetes, clases, métodos, constructores y campos utiliza cada parte del paquete, clase o interfaz en cuestión. Para crear este archivo, tiene que activar la opción `@use` de la ficha Especificar opciones doclet de línea de comandos del asistente.
- Un archivo `deprecated-list.html` para todos los nombres desaconsejados. Para crear este archivo, tiene que activar la opción `@deprecated` de la ficha Especificar opciones doclet de línea de comandos del asistente.
- Un archivo `serialized-form.html` con la información acerca de las clases serializables y exteriorizables. Para que se genere este archivo, hay que introducir las etiquetas `@serial`, `@serialField` y `@serialData` en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos del asistente.
- Un archivo `index-*.html` que recoge todos los nombres de clases, interfaces, constructores, campos y métodos por orden alfabético. Para crear este archivo, debe activar la opción Crear índice de la ficha Especificar opciones doclet de línea de comandos del asistente.

Javadoc también crea archivos de apoyo, que incluyen:

- Un archivo `help-doc.html`, que describe cómo se utiliza la barra de navegación.
- Un archivo `index.html`, que genera los marcos HTML.
- Un número de archivos `*-frame.html`, que recogen los paquetes, clases e interfaces que se utilizan para mostrar los marcos HTML.
- Un archivo `package-list`, que es un archivo de texto utilizado por las opciones `-link`. No se puede acceder a él a través de ningún enlace.
- Un archivo `stylesheet.css` que controla la apariencia de los archivos HTML, y que se basa en el doclet seleccionado en la ficha Seleccionar doclet del asistente.

Si desea más información acerca de los archivos generados, consulte “Generated Files” en:

- **Usuarios de Windows:** <http://java.sun.com/j2se/1,4/docs/tooldocs/win32/javadoc.html#generatedfiles>

- **Usuarios de Solaris:** <http://java.sun.com/j2se/1,4/docs/tooldocs/solaris/javadoc.html#generatedfiles>

## Creación de archivos adicionales

---

Javadoc genera de forma automática numerosos archivos de salida para los archivos fuente .java . También se pueden crear archivos adicionales de salida, como archivos descriptivos de paquetes o archivos de comentarios de aspectos generales, a partir de archivos auxiliares.

### Archivos de paquetes

Cada paquete puede tener su propio comentario de documentación en su propio archivo fuente. Javadoc fusiona este archivo de comentarios en la ficha resumen de paquetes que genera para cada paquete del proyecto. Este archivo debe tener el nombre `package.html`, y debe estar colocado en el directorio de paquetes del árbol fuente, junto con los archivos de clase del paquete. Javadoc buscará automáticamente este nombre de archivo en esa ubicación.

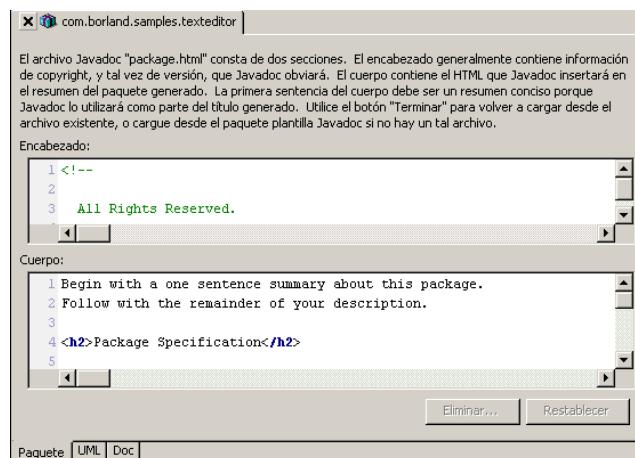
El archivo `package.html` debe contener un solo comentario de documentación en formato HTML. No incluya las etiquetas Javadoc de comienzo y final de comentario (`/**` y `*/`). No ponga ningún título ni ningún otro texto entre la etiqueta `<body>` y la primera frase. La primera frase debe ser un resumen.

JBuilder ofrece un editor de archivos de paquetes que permite crear, modificar o eliminar fácilmente el archivo `package.html` para los paquetes individuales del proyecto. El editor de archivos de paquetes coloca el archivo en la ubicación correcta para que Javadoc lo pueda procesar.

Para utilizar este editor:

- 1 Abra el proyecto. En el panel del proyecto, seleccione el paquete para el que deseé crear un archivo de paquetes.

- 2** Haga doble clic en el paquete. El editor de archivos de paquetes aparece en la pestaña Paquete del panel de contenido.



- 3** Escriba el texto de encabezado del archivo package.html en la sección Encabezado del editor. Esta sección normalmente contiene la información de la versión y el copyright. Javadoc no procesa texto en esta sección, sino que lo deja en una etiqueta de comentario en el archivo package.html.
- 4** Escriba el texto del cuerpo en la sección Cuerpo. La primera frase debe ser una breve frase resumen del paquete. A continuación, escriba una descripción completa del paquete. Puede utilizar la plantilla para introducir enlaces a otros paquetes y/o a la documentación relacionada.

Al generar los archivos de salida, la frase resumen aparece en la parte superior del archivo de paquetes. El resto de la descripción del paquete sigue a la lista Resumen de clase. Tenga en cuenta que puede crear archivos package.html individuales para cada uno de los paquetes del proyecto.

Si desea información adicional acerca de los archivos de paquete, consulte "Package Comment Files" en:

- **Usuarios de Windows:** <http://java.sun.com/j2se/1,4/docs/tooldocs/win32/javadoc.html#packagecomment>
- **Usuarios de Solaris:** <http://java.sun.com/j2se/1,4/docs/tooldocs/solaris/javadoc.html#packagecomment>

## Archivos de comentarios de aspectos generales

Cada proyecto puede tener su propio archivo de comentarios de aspectos generales en su propio archivo fuente. Javadoc fusiona este archivo de comentarios en la ficha de aspectos generales que crea para cada uno de

los paquetes del proyecto. Puede ponerle al archivo el nombre que desee, normalmente es `overview.html`. También lo puede colocar donde quiera, aunque normalmente se coloca en la parte superior del árbol de fuentes.

El archivo `overview.html` debe contener un solo comentario de documentación, en formato HTML. No incluya las etiquetas Javadoc de comienzo y final de comentario (`/** y */`). No ponga ningún título ni ningún otro texto entre la etiqueta `<body>` y la primera frase. La primera frase debe ser un resumen.

Si desea información adicional acerca del archivo de comentarios de aspectos generales, consulte “Overview Comment Files” en:

- **Usuarios de Windows:** <http://java.sun.com/j2se/1,4/docs/tooldocs/win32/javadoc.html#overviewcomment>
- **Usuarios de Solaris:** <http://java.sun.com/j2se/1,4/docs/tooldocs/solaris/javadoc.html#overviewcomment>

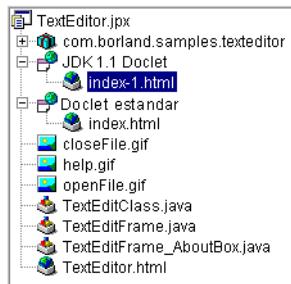
## Visualización de Javadoc

---

Existen varias formas de visualizar Javadoc una vez que se ha generado.

Si desea visualizar el Javadoc de todo el proyecto, amplíe el nodo de documentación y haga doble clic en `index.html` (para la información en la que se ha utilizado el Doclet estándar), o bien, `index-1.html` (para la información en la que se ha utilizado el Doclet JDK 1.1).

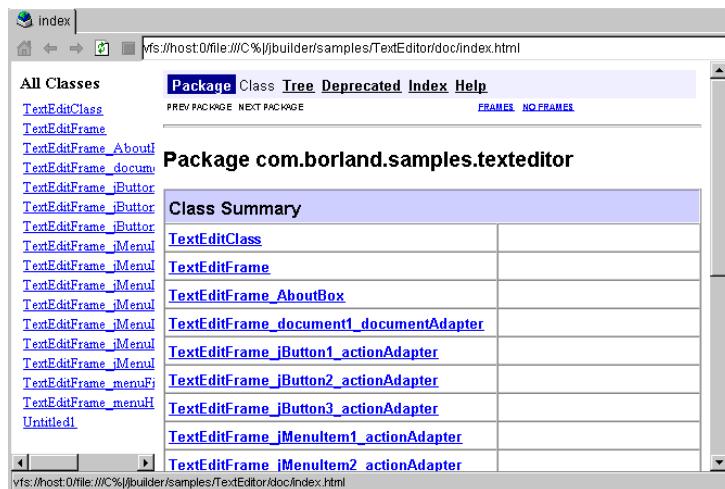
**Figura 14.8** Nodos de documentación ampliados



Se abre el archivo de índice en el panel Ver del Visualizador de aplicaciones.

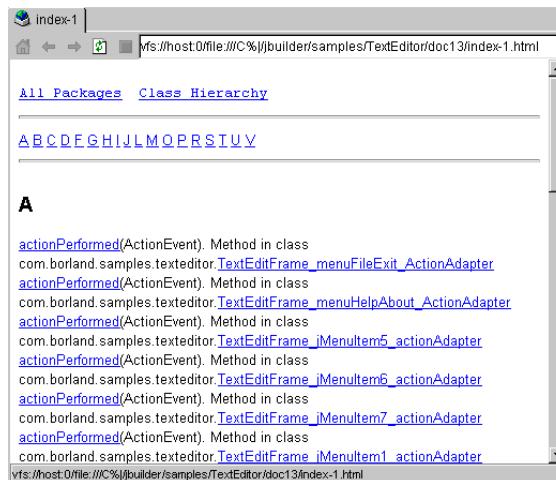
- Para la documentación en Doclet estándar, los paquetes y clases aparecen en el marco izquierdo. El marco derecho muestra las tablas resumen.

**Figura 14.9** Salida de archivos de índices desde el Doclet estándar



- Para la documentación en Doclet JDK 1.1, aparece un índice alfabético.

**Figura 14.10** Salidas de archivos de índices desde el Doclet JDK 1.1



También puede visualizar el Javadoc de un solo archivo, seleccionando ese archivo en el panel del proyecto y, a continuación, la pestaña Documentación. Para ver el Javadoc de un solo archivo desde un diagrama de clase UML, pulse con el botón derecho del ratón en el

nombre de la clase del diagrama y seleccione Ver Javadoc. Aparece entonces el archivo HTML en el Visualizador de ayuda.

En cualquiera de estas formas de visualizar Javadoc, puede utilizar la barra de navegación superior o inferior y los enlaces en las tablas resumen para desplazarse por la documentación.

## Apariencia de Javadoc en JBuilder

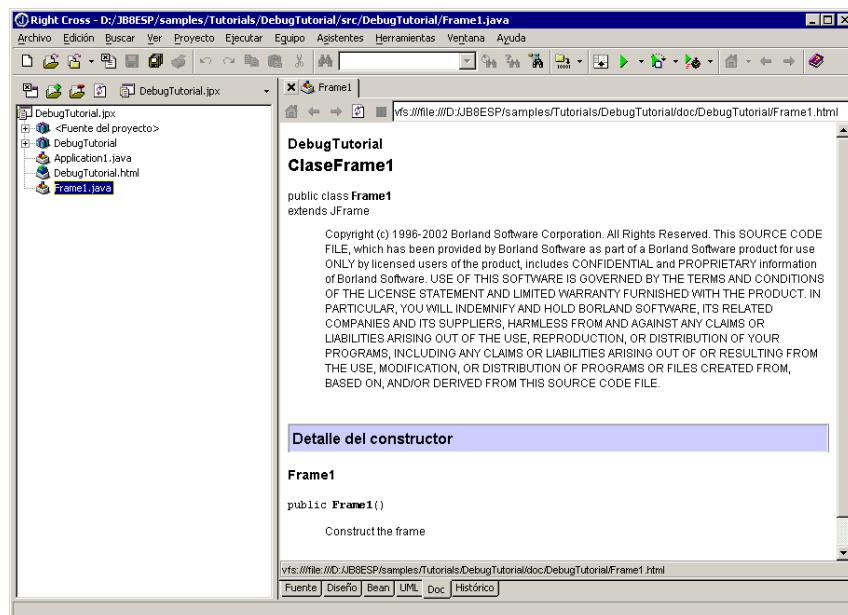
---

Si selecciona la pestaña Documentación para mostrar Javadoc de un archivo fuente abierto en el editor, JBuilder busca en primer lugar archivos con formato HTML o archivos en JDK 1.1, utilizando la vía de acceso a la documentación que se define en la pestaña Documentación de la ficha Vías de acceso de Propiedades de proyecto (Proyecto | Propiedades de proyecto).

- Si solamente existen archivos de salida con formato JDK 1.1, son éstos los que aparecen.
- Si existen archivos tanto con formato JDK 1.1 como con formato estándar, aparece el archivo del doclet estándar.

Si no hay archivos de ninguno de estos dos tipos, JBuilder muestra el Javadoc “sobre la marcha”, es decir, muestra el Javadoc que se crea directamente a partir de comentarios en el archivo fuente API. Esto permite actualizar Javadoc de tal manera que siempre se muestre para los archivos fuente, incluso si todavía no ha creado Javadoc. En esta vista no hay enlaces disponibles.

Figura 14.11 Información Javadoc sobre la marcha



## Mantenimiento de Javadoc

La ventaja principal de crear Javadoc en su propio directorio de salida, como un directorio `doc`, es que así puede mantenerlo fácilmente. Para eliminar los archivos HTML del directorio de salida de la documentación, pulse el nodo de la documentación con el botón derecho del ratón y seleccione Limpiar. Se eliminan los archivos HTML y los archivos CSS. Sin embargo, no se borra la estructura de directorios. Si selecciona Generar de nuevo, en primer lugar se limpiará el directorio y, a continuación, se generarán de nuevo los archivos HTML.

## Cambio de propiedades para el nodo de documentación

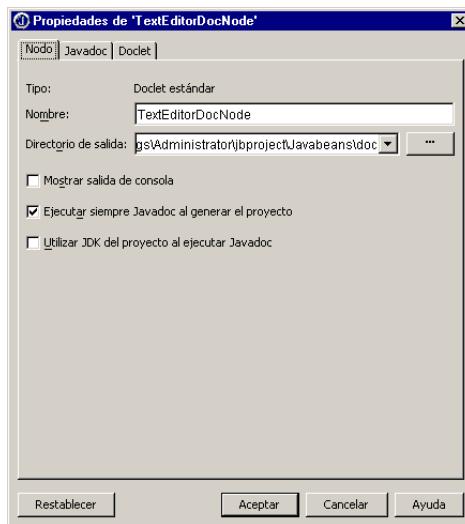
Una vez que se ha creado el nodo de la documentación, puede cambiar las propiedades del nodo, incluidos el nombre del nodo, el directorio de salida y el momento de creación de Javadoc. Para cambiar las propiedades, pulse con el botón derecho del ratón en el nodo de documentación del panel del proyecto y seleccione Propiedades. En el cuadro de diálogo Propiedades, seleccione la pestaña Nodo para cambiar las propiedades del nodo. Seleccione la pestaña Javadoc si desea cambiar los paquetes que tienen que documentarse. Si desea cambiar las opciones de doclet, seleccione la pestaña Doclet.

## Cambio de las propiedades del nodo

Si desea cambiar las propiedades del nodo, utilice la pestaña Nodo del cuadro de diálogo Propiedades. Es posible cambiar las siguientes opciones:

- Nombre del nodo.
- Directorio de salida.
- Apariencia de la consola de salida.
- Cuándo crear Javadoc.
- Qué JDK utilizar (el del proyecto o uno alojado en JBuilder).

La ficha Nodo tendrá este aspecto:



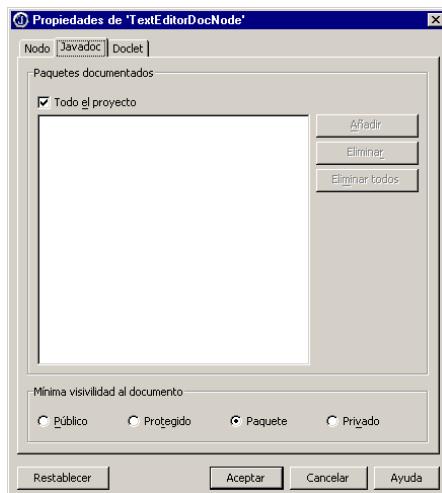
Para obtener más información, consulte “[Selección de opciones para generar documentación](#)” en la página 14-11.

## Modificación de propiedades Javadoc

Utilice la pestaña Javadoc del cuadro de diálogo Propiedades para modificar los paquetes que se han de documentar. Es posible cambiar las siguientes opciones:

- Paquetes que se han de documentar.
- Visibilidad más baja documentado.

La ficha Javadoc tendrá este aspecto:



Para obtener más información, consulte “[Selección de paquetes que se han de documentar](#)” en la página 14-13.

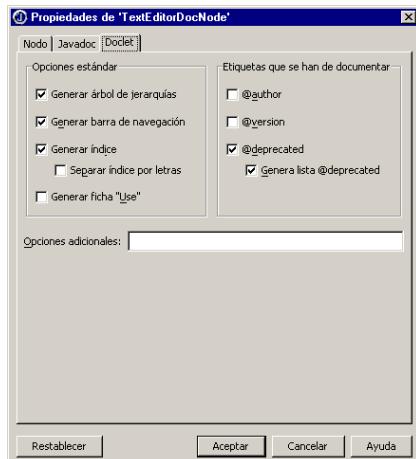
## Modificación de propiedades doclet

---

Utilice la pestaña Doclet del cuadro de diálogo Propiedades si desea modificar las opciones y etiquetas documentados. Puede elegir:

- Si se genera el archivo del árbol de jerarquías.
- Si aparece la barra de navegación en la parte superior de cada archivo generado.
- Si se crea un índice.
- Las etiquetas que se han documentado.

La ficha Doclet tendrá este aspecto:



Para obtener más información, consulte “[Especificación de opciones de línea de comandos para el doclet](#)” en la página 14-14.

## Creación de un archivo recopilatorio de documentación

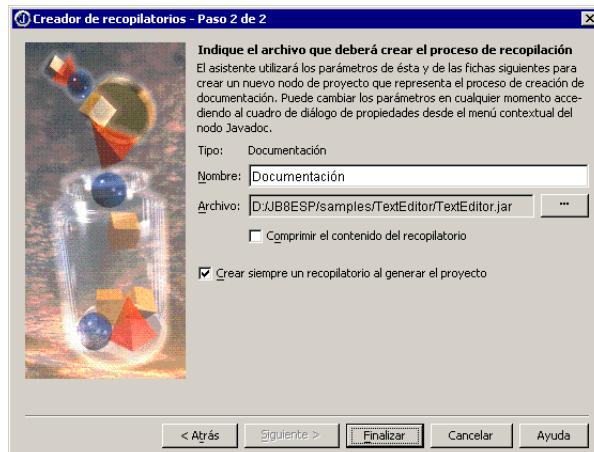
Después de la ejecución definitiva de Javadoc, puede utilizar el Creador de recopilatorios para crear un archivo de documentación JAR. En este tipo de archivo JAR se encuentran todos los archivos de los directorios de la vía de acceso a la documentación y, normalmente, suele ser el directorio `doc`. Para crear un recopilario de documentación:

- 1 Seleccione Asistentes | Creador de recopilatorios.
- 2 En la ficha Seleccione un tipo de recopilatorio del Creador de recopilatorios, seleccione Documentación en la lista desplegable Tipo de recopilatorio. El Creador de recopilatorios tiene la siguiente apariencia:

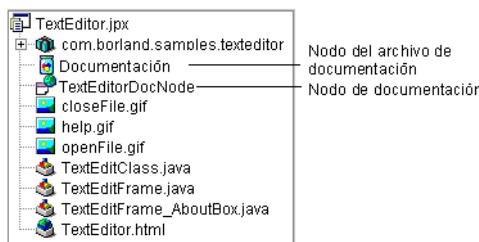


- 3 Pulse Siguiente para pasar al siguiente paso.
- 4 Escriba un nombre en el campo Nombre para el nodo de documentación. Este nodo aparece en el panel del proyecto al pulsar Finalizar.
- 5 Introduzca el nombre del archivo JAR en el campo Archivo: Este archivo debe tener la extensión .jar . Por defecto, se encuentra en el directorio raíz del proyecto.
- 6 Si desea comprimir el recopilatorio, seleccione Comprimir el contenido del recopilatorio. Esta opción se utiliza para reducir al mínimo el tamaño del archivo JAR.
- 7 Seleccione Crear siempre un recopilatorio al generar el proyecto para que se cree un archivo JAR cada vez que seleccione Ejecutar Make o Generar.

La ficha del Creador de recopilatorios debe tener la siguiente apariencia:



- 8 Seleccione Finalizar para cerrar el asistente y crear el nodo de recopilatorio en el panel del proyecto. El panel del proyecto presentará este aspecto:



- 9** Seleccione Proyecto | Ejecutar Make del proyecto para ejecutarlo y crear el archivo JAR.

El archivo JAR de documentación se coloca en el directorio especificado en el Creador de recopilatorios.

También tiene la posibilidad de crear un recopilatorio fuente para los archivos fuente del proyecto, si selecciona Tipo de recopilatorio fuente en la ficha Seleccione un tipo de recopilatorio del Creador de recopilatorios.

Para obtener más información sobre el Creador de recopilatorios, consulte “[Distribución con el Creador de recopilatorios](#)” en la página 15-18

## Creación de un doclet personalizado

---

Puede crear un doclet personalizado si amplía el Asistente para Javadoc con la API de OpenTools. Si desea un ejemplo de doclet personalizado, abra el archivo Doclet.jpx del proyecto en la carpeta samples/OpenToolsAPI/wizards/doclet de su instalación de JBuilder. Un doclet personalizado no necesita generar archivos HTML. Por ejemplo, las etiquetas personalizadas se pueden utilizar junto con la habilidad de la herramienta Javadoc para analizar archivos fuente. El doclet puede crear entonces archivos XML o archivos Java adicionales. El doclet personalizado debe situarse en el directorio <jbuilder>/doclet.

Si desea más información acerca de la API de OpenTools, consulte “JBuilder OpenTools basics” en *Developing OpenTools*.

# 15

## Distribución de programas en Java

El Creador de recopilatorios y el Creador de ejecutables nativos son funciones de JBuilder SE y Enterprise

La distribución de programas Java consiste en reunir los diferentes archivos de clase de Java, los archivos de imágenes y demás archivos necesarios para el programa, para copiarlos en una dirección de un ordenador cliente o servidor con el fin de que se puedan ejecutar. Los archivos se pueden entregar por separado, o todos juntos en un recopilatorio comprimido o sin comprimir.

**Nota**

Todas las referencias a “programas” Java de este documento se refieren a aplicaciones, applets, Java Beans o Enterprise Java Beans de Java.

Las ediciones SE y Enterprise de JBuilder cuentan con el Creador de recopilatorios, que le puede ayudar a distribuir el programa. Además, el Creador de ejecutables nativos agrupa los ejecutables nativos en un archivo JAR de aplicación, con el fin de acelerar la distribución. Los archivos JAR también se pueden crear en la línea de comandos con la ayuda de la herramienta `jar` de Sun, que se incluye en el JDK. El Creador de recopilatorios de Jbuilder simplifica la distribución reuniendo automáticamente las clases, los recursos y bibliotecas que necesita su programa y distribuyendo los archivos en otro archivo ZIP o JAR comprimido o sin comprimir. También crea el descriptor del archivo JAR. El Constructor de recopilatorios de JBuilder crea también un nodo de archivos del proyecto, que permite un fácil acceso al archivo recopilatorio y al archivo descriptor. El archivo recopilatorio puede crearse o modificarse en cualquier fase del desarrollo. También se puede ver el contenido del archivo recopilatorio y del archivo descriptor. Consulte “[Distribución con el Creador de recopilatorios](#)” en la página 15-18 para obtener más información.

El primer paso para la distribución de cualquier programa es identificar qué archivos, del proyecto y de biblioteca, se van a incluir en el archivo recopilatorio. Esto ayudará a determinar las clases, recursos y dependencias necesarias. La inclusión de todas las clases, recursos y dependencias genera un archivo recopilatorio muy voluminoso. Sin embargo, no es necesario proporcionar al usuario final otros archivos, ya que el archivo de revisiones contiene todo lo necesario para ejecutar el programa. Si se excluyen clases, recursos o dependencias parcial o totalmente, será necesario proporcionarlos por separado al usuario final.

La distribución es un tema complejo y avanzado que se debe estudiar con el fin de comprender todas sus implicaciones. El Creador de recopilatorios de JBuilder reduce esta complejidad y ayuda a crear un archivo de revisiones con todos los requisitos necesarios para la distribución.

### Consulte

- “Trail: Jar Files” en el Java Tutorial de <http://java.sun.com/docs/books/tutorial/jar/index.html>
- “Using JAR Files: The Basics” en <http://java.sun.com/docs/books/tutorial/jar/basics/index.html>.
- Paso 16 del tutorial de JBuilder “Creación de un editor de texto de Java”, en *Diseño de aplicaciones con JBuilder*

**Nota** En este documento se da por supuesto que ya entiende la diferencia entre un applet (un programa que se ejecuta dentro de otro contexto, normalmente un navegador web) y una aplicación (una aplicación autónoma que contiene un método `main()`). Si desea obtener información sobre las applets, navegadores y su compatibilidad con JDK, consulte el apartado “Los navegadores” en “Las applets” de la *Guía del desarrollador de aplicaciones Web*.

## Distribución de archivos recopilatorios de Java (JAR)

---

Los programas Java pueden consistir en muchos archivos .class, además de varios archivos de recursos, propiedades y documentación. Un programa voluminoso puede tener cientos o incluso miles de estos archivos. Cuando los programas están completos y listos para su distribución, es necesario contar con una herramienta cómoda para agrupar los archivos de clase y de otro tipo en un solo conjunto de distribución.

Los archivos se pueden distribuir por separado, o unidos en un archivo recopilatorio para facilitar la entrega. También se puede entregar un programa voluminoso en varios archivos recopilatorios que representen bibliotecas y programas principales. Los archivos recopilatorios comprimidos tienen la ventaja de que el tiempo de descarga de las applets

es más corto, y los archivos ocupan menos espacio en el servidor o el sistema de destino. Su desventaja es que son un poco más lentos durante la ejecución.

La forma más eficaz de entregar y distribuir programas de Java consiste en utilizar archivos JAR comprimidos. Los archivos JAR también contienen un archivo descriptor, y en ocasiones, archivos de firma, tal y como se define en Manifest Specification. El archivo descriptor posibilita las funciones más avanzadas de los archivos JAR, como el sellado de paquetes, las versiones de paquetes y la firma electrónica.

Los archivos JAR (.jar) son fundamentalmente archivos ZIP con una extensión distinta y con determinadas reglas sobre la estructura interna de directorio. JavaSoft utilizaba el formato de archivo ZIP de PKWARE como base del formato de archivos JAR.

**Nota** Sólo los navegadores con JDK 1.1 o posteriores aceptan los archivos JAR. Si está desarrollando un applet para un navegador JDK 1.0.2 debe utilizar un archivo comprimido ZIP.

Además de los archivos de clase y recurso (colocados en una estructura de directorio adecuada al paquete), los archivos JAR deben contener un archivo descriptor y en ocasiones, archivos de firma de clase.

Aunque es posible colocar cualquier elemento en un recopilatorio, la MV de Java sólo busca archivos de clase.

El archivo HTML desde el que se carga el applet no procede de este recopilatorio. Es un archivo distinto del servidor. No obstante, la especificación JavaBeans indica que los archivos HTML de documentación de un bean se pueden colocar en un recopilatorio.

## Conceptos básicos acerca del archivo descriptor

---

El archivo descriptor de un archivo JAR es un archivo de texto que incluye información acerca de algunas o todas las clases contenidas en el archivo JAR en cuestión. En Java 2, también contiene información sobre cuál de las clases del archivo JAR es la ejecutable.

El archivo descriptor de todo archivo JAR debe llamarse `manifest.mf` y debe encontrarse en el directorio `/meta-inf` del archivo JAR.

El archivo descriptor por defecto generado por el Creador de recopilatorios, disponible en las ediciones JBuilder SE y Enterprise, incluye los dos encabezados siguientes al principio del archivo:

- `Manifest-Version: 1.0`

Informa de que las entradas del descriptor toman la forma de pares “cabecera:valor” y que se trata de la versión 1.0 de la especificación del descriptor.

## Distribución de archivos recopilatorios de Java (JAR)

- Main-Class: nombre-clase

Este encabezado se utiliza para los tipos de archivo de aplicación. Indica qué clase ejecuta la aplicación: la que contiene el método `public static void main(String[] args)`.



La cabecera Main-Class permite ejecutar aplicaciones desde el archivo JAR por medio de la opción `-jar` de las herramientas de Java, que inicia la aplicación de Java desde una línea de comandos.

Hay otros encabezados que se pueden añadir al archivo descriptor, que proporcionan al archivo JAR funciones adicionales que permiten su uso con varias finalidades.

### Consulte

- “Understanding the manifest” en <http://java.sun.com/docs/books/tutorial/jar/basics/manifest.html>
- “Special purpose manifest headers” en <http://java.sun.com/docs/books/tutorial/jar/basics/manifest.html#special-purpose>
- “JAR Manifest” en <http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html#JAR Manifest>
- “JAR Manifest - Main Attributes” en [http://java.sun.com/products/js2e/1.4/docs/tooldocs/tools.html#basic](http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html>Main Attributes</a></li><li>• “Command-line tools” en <a href=)
- “JAR Guide” en <http://java.sun.com/j2se/1.4/docs/guide/jar/jarGuide.html>

## Estrategias de distribución

---

Existen dos estrategias básicas para distribuir su aplicación:

- Entregar las bibliotecas de libre distribución con el archivo JAR e incluirlas en `CLASSPATH` durante la ejecución, en vez de colocar las clases necesarias de estas bibliotecas dentro del archivo JAR. Ésta es la manera más fácil de efectuar una distribución y la que crea el archivo JAR más pequeño. Ésta es la mejor elección si se entregan varias aplicaciones o applets al mismo lugar y se desea que comparten las bibliotecas.

Si desea información sobre los elementos de libre distribución y los de uso exclusivo, según las condiciones de la licencia de JBuilder, consulte los archivos <`jbuilder>/license.html` y <`jbuilder>/redist/deploy.html`.

- Cree un archivo JAR con la ayuda de la herramienta **jar** de Sun, que se encuentra en el JDK o en el Creador de recopilatorios de JBuilder, disponible para las ediciones SE y Enterprise. El Creador de recopilatorios ofrece muchas opciones para reunir las clases, recursos y bibliotecas que el programa necesita. Las opciones elegidas dependen de los requisitos de la distribución (incluidas las consideraciones de espacio), de si el programa es un applet o una aplicación autónoma y de la instalación que hagan los usuarios del programa. Consulte “[Distribución con el Creador de recopilatorios](#)” en la página 15-18.

El proceso de distribución de JBuilder se puede resumir en los siguientes pasos básicos:

- 1 Cree y compile el código en JBuilder.
- 2 Cree un archivo JAR con la ayuda de la herramienta **Jar** de Sun o del Creador de recopilatorios de JBuilder.
- 3 Cree un procedimiento de instalación.
- 4 Entregue el archivo JAR, todos los archivos JAR de libre distribución que sean necesarios y los archivos de instalación.

### Consulte

- La herramienta **jar** de Sun en <http://java.sun.com/products/js2e/1.4/docs/tooldocs/tools.html#basic>
- “[Distribución rápida](#)” en la página 15-11
- “[Distribución con el Creador de recopilatorios](#)” en la página 15-18

## Utilización de la herramienta de recopilación de Java del JDK

---

Sun facilita la creación y modificación de archivos JAR desde la línea de comandos por medio de la herramienta de recopilación de Java (herramienta **jar**), que se entrega en el Kit de desarrollo de Java. Se llama a la herramienta **jar** con el comando **jar**, mediante el formato básico que se detalla a continuación:

```
jar cf jar-file input-file(s)
```

Si desea más información sobre la creación y modificación de archivos JAR, consulte “[Actualización del contenido de un archivo JAR](#)” en la [página 15-7](#).

### Consulte

- “Using JAR Files: The Basics” en <http://java.sun.com/docs/books/tutorial/jar/basics/index.html>.

## Ejecución de programas desde un archivo JAR

---

Es posible ejecutar un programa recopilado en un archivo JAR desde la línea de comandos. Añada el archivo JAR a CLASSPATH por ejemplo, CLASSPATH=user/username/jbproject/myapp/myjar.jar, o a la opción de línea de comandos -cp o -classpath para java.exe. Asigne a la clase el nombre completo del paquete.

```
java -classpath user/username/jbproject/myapp/myjar.jar mypackage.myclassname
```

En la versión 1.2 y posteriores del software JDK, puede añadir **-jar** al comando **java** para indicar al intérprete que la aplicación está empaquetada en formato de archivo JAR. La MV de Java obtiene la información de la Main-Class: del archivo descriptor sobre la clase que se debe ejecutar.

```
java -jar jar-file
```

Por ejemplo:

```
java -jar user/username/jbproject/myapp/myjar.jar
```

Por supuesto, si ejecuta la aplicación desde el mismo directorio que el archivo JAR, sólo tiene que escribir lo siguiente:

```
java -jar myjar.jar
```

### Consulte

- “Running JAR-packaged software” en <http://java.sun.com/docs/books/tutorial/jar/basics/run.html>

- “Modifying a manifest file” en <http://java.sun.com/docs/books/tutorial/jar/basics/mod.html>
- “Updating a JAR file” en <http://java.sun.com/docs/books/tutorial/jar/basics/update.html>

## Presentación del contenido de un archivo recopilatorio

---

Si desea ver una lista del contenido de un archivo JAR con la herramienta **jar**, utilice el comando:

```
jar -tf jar-file
```

También funciona con archivos ZIP:

```
jar -tf archivo-zip
```

**Nota** También puede ver el contenido del recopilatorio en JBuilder. Añada un archivo JAR al proyecto, haga doble clic en el archivo JAR en el panel del proyecto, expanda los nodos en el panel de estructura y haga doble clic en un archivo para abrirlo en el editor. Los archivos JAR son de sólo lectura en el editor.

La utilidad JAR tiene otros muchos usos. Si desea información sobre ellos, teclee **jar** para obtener ayuda en la línea de comandos.

La mayoría de las herramientas de archivo ZIP de PKWARE permiten examinar o incluso modificar archivos JAR. Si la herramienta requiere la extensión ZIP, se puede cambiar el nombre del archivo provisionalmente, de JAR a ZIP.

**Nota** Las versiones de WinZip anteriores a la 6.3 tienen un error que les impide extraer y ver el contenido de los archivos JAR válidos. Este problema está resuelto en las versiones posteriores.

## Actualización del contenido de un archivo JAR

---

Java 2 cuenta con la opción **u** para **jar.exe**. Esta opción se puede utilizar para actualizar el contenido de un archivo JAR añadiéndole archivos. Para utilizar este comando, escriba:

```
jar uf archivo-jar archivo(s)-entrada
```

donde

**u** = actualizar un archivo JAR

**f** = el archivo JAR que se va a actualizar está en la línea de comandos

**archivo jar** = nombre del archivo JAR que se va a actualizar

**archivos de entrada** = lista de archivos que se van a añadir, separados por espacios

Se sustituyen los archivos que se encuentren dentro del recopilatorio y tengan el mismo nombre de vía de acceso que los archivos añadidos.

También se puede utilizar la opción **u** en combinación con **m** para actualizar el archivo descriptor de un archivo JAR:

```
jar umf archivodescriptor archivo-jar
```

donde

**m** = actualizar el archivo descriptor del archivo JAR

**archivo descriptor** = nombre del archivo descriptor cuyo contenido se desea añadir al archivo descriptor del archivo JAR

### Consulte

- “Modifying a manifest file” en <http://java.sun.com/docs/books/tutorial/jar/basics/mod.html>
- “Updating a JAR file” en <http://java.sun.com/docs/books/tutorial/jar/basics/update.html>
- “How classes are found” en <http://java.sun.com/j2se/1.4/docs/tooldocs/findingclasses.html>
- “JAR Files Trail” en el Java Tutorial de <http://java.sun.com/docs/books/tutorial/jar/>

## Aspectos relativos a la distribución

---

Es importante tener claras las siguientes cuestiones para determinar cuál es la estrategia de distribución idónea:

- ¿Está todo lo necesario en la vía de acceso a clases?
- ¿El programa depende de funciones JDK 1.1 o Java 2 (JDK 1.2 y superior)?
- ¿Tiene el usuario bibliotecas Java distintas del JDK instaladas en su ordenador?
- ¿Se trata de un applet o de una aplicación?
- ¿Existen en el servidor limitaciones de espacio en disco o de tiempo de descarga de archivos?

### ¿Está todo lo necesario en la vía de acceso a clases?

---

Los problemas de distribución se deben, sobre todo, a que el archivo JAR o la vía de acceso a clases no contiene todo lo necesario. Si no se han añadido las clases necesarias de una biblioteca determinada al archivo JAR, se debe comprobar que las bibliotecas de libre distribución se encuentran especificadas en la opción **-classpath** de la línea de comandos de Java o en la variable de entorno **CLASSPATH**. Es recomendable utilizar la

opción **-classpath**, ya que las vías de acceso a clases se pueden definir independientemente para una aplicación sin que esto influya sobre las otras, y las demás aplicaciones no pueden modificar su valor.

**Sugerencia** JBuilder indica la vía de acceso a clases cuando se ejecuta desde el IDE. Si lo refleja en la línea de comandos, el programa funcionará.

La forma en que se define la variable de entorno **CLASSPATH** depende del sistema operativo que se esté utilizando.

### Consulte

- “Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos” en la página B-2
- “Setting the class path” en <http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html>

---

## ¿El programa depende de funciones JDK 1.1 o Java 2 (JDK 1.2 y superior)?

---

Si está desarrollando un applet, puede tener ciertas complicaciones si algunos usuarios utilizan navegadores de Internet que no estén actualizados para aceptar aplicaciones escritas con características de JDK 1.1.x o posteriores, como Swing. Si desea información adicional, consulte “Las applets” en la *Guía del desarrollador de aplicaciones Web*.

Los navegadores que cumplen las especificaciones de JDK 1.0.2 no aceptan recopilatorios JAR; por tanto, si ha escrito un applet que cumpla las normas de JDK 1.0.2 y desea distribuirlo, cree un recopilatorio en formato ZIP con el Creador de recopilatorios.

---

## ¿Tiene el usuario bibliotecas Java instaladas en su ordenador?

---

Si su programa utiliza componentes que dependen de bibliotecas que no son JDK, necesita proporcionárselas al usuario en el archivo JAR. Los archivos de libre distribución de JBuilder se encuentran en el directorio `<jbuilder>/redist`. Todos los archivos de libre distribución del JDK se encuentran en los directorios `<jbuilder>/<jdk>/lib/` y `<jbuilder>/<jdk>/jre/lib/`. `<jdk>` representa el nombre del directorio de JDK.

**Nota** Una de las ventajas de utilizar el Creador de recopilatorios para crear un recopilatorio, disponible en las ediciones SE y Enterprise, es que se pueden incluir esas bibliotecas seleccionando la opción Incluir clases necesarias y recursos conocidos en el paso 4 del asistente. Esta opción asegura que las clases requeridas, así como todos los recursos (incluidos los de las bibliotecas de terceros), están incluidos en el archivo JAR del

programa. El Creador de recopilatorios nunca incluye JDK en el archivo recopilatorio. Presupone que las clases JDK ya existen en el ordenador de destino en forma de JDK instalado, entorno de ejecución Java o Plug-in Java, o que se proporcionarán en la instalación. Consulte “[Distribución con el Creador de recopilatorios](#)” en la página 15-18.

Si puede presuponer que todos los usuarios tienen ya estos recopilatorios en su entorno, ya sea porque los hayan instalado o porque se les hayan suministrado en algún tipo de proceso de instalación, no es necesario que las aplicaciones y applets que les suministre contengan estos paquetes.

Si no está seguro de que el usuario cuenta con estas bibliotecas, debe proporcionárselas. Esto es especialmente importante en el caso de distribución de applets. Cuando se distribuye un applet es necesario copiar en el servidor tanto estas bibliotecas como los otros recursos que se puedan necesitar.

**Importante** En JDK 1.1.x, las clases Swing/JFC **no** se entregan como parte del JDK. Si escribe programas que utilizan cualquiera de los JDK, debe descargar y entregar swingall.jar, que contiene estos archivos.

### Consulte

- “[Redistribución de las clases que se suministran con JBuilder](#)” en la página 15-16

## ¿Se trata de un applet o de una aplicación?

---

Las aplicaciones no se distribuyen de la misma forma que las applets. En lo relativo a la distribución, ésta es la principal diferencia:

- En el caso de las aplicaciones, el usuario debe utilizar java.exe (Entorno de ejecución de Java) para ejecutar las clases o los archivos JAR que se le han suministrado, ya sea directamente desde el servidor o después de haberlas instalado en el ordenador. Si el usuario no tiene los archivos JRE necesarios, se deben incluir en el conjunto de distribución.
- En el caso de las applets, se presupone que el usuario utiliza un navegador de Internet o un visualizador de applet, y que es necesario que haya una página HTML que contenga la etiqueta <APPLET> que hace referencia a las clases que se quieren ejecutar. En este caso, se debe comprobar que el navegador del usuario acepta la versión del JDK utilizado para desarrollar las applets, y en caso contrario, proporcionarle el Plug-in de Java, de Sun, para el navegador. Si desea más información, consulte “[Las applets](#)” en la *Guía del desarrollador de aplicaciones Web*.

**Importante** Si tiene intención de utilizar el Plug-in de Java, de Sun, lea antes toda la documentación sobre los asuntos relacionados. Hay varias versiones del Plug-in de Java y del Convertidor HTML. Ya que las versiones de

plug-in son incompatibles entre sí, sólo se puede tener instalada una versión al mismo tiempo. Sólo es recomendable utilizar el Plug-in Java en entornos controlados, por ejemplo en una intranet, en los que se sabe qué versión de navegador se utiliza y qué JDK acepta.

Si desea más información, consulte el Plug-in Java, que se encuentra en <http://java.sun.com/products/plugin/>.

## Tiempo de descarga de archivos

---

Una de las primeras preguntas a las que debe contestar un desarrollador es si desea guardar el programa en un recopilatorio. Los factores que influyen en mayor medida sobre esta decisión son el tiempo de descarga y el tamaño del programa, especialmente para las applets.

Una de las ventajas de utilizar el Creador de recopilatorios disponible en las ediciones SE y Enterprise para crear un recopilatorio es que sólo se incluyen los archivos necesarios. El Creador de recopilatorios identifica todas las clases que necesita el proyecto y las reúne en un archivo recopilatorio. Esto aumenta la eficacia del tiempo de recuperación de archivos, utilización del disco y consumo de sockets de red, cuando la aplicación o applet se inicia desde un navegador.

Mientras se familiariza con las clases Java y sus dependencias, es posible que deba crear archivos JAR más voluminosos para asegurarse de que está incluido todo lo necesario. A medida que adquiera más experiencia podrá reducir el tamaño de los archivos JAR, ya que le bastará con incluir en ellos y en el proyecto las clases y dependencias necesarias.

## Distribución rápida

---

Los pasos de distribución varían dependiendo de lo que se vaya a distribuir. Estos pasos describen el modo de distribuir aplicaciones, applets y JavaBeans. Para obtener más información sobre el Creador de recopilatorios, disponible en JBuilder SE y Enterprise, consulte “[Distribución con el Creador de recopilatorios](#)” en la página 15-18.

## Aplicaciones

---

-  1 Añada al proyecto todos los archivos de recurso y clases de carga dinámica que necesita la aplicación, por medio del botón Añadir archivos/paquetes de la barra de herramientas del panel del proyecto.

Optativo: Utilice el Asistente para cadenas de recursos, disponible en las ediciones SE y Enterprise, para desplazar las cadenas a un conjunto de recursos. Esto es optativo, pero facilita la internacionalización de las aplicaciones.

- 2** Compile la aplicación.
- 3** Utilice el Asistente para Javadoc de JBuilder, disponible en JBuilder SE y Enterprise, y otro medio para crear la documentación adecuada para los clientes desarrolladores.
- 4** Cree un archivo JAR mediante la herramienta **jar** de Sun, el Creador de recopilatorios o el Creador de ejecutables nativos de JBuilder.
- 5** Copie el archivo JAR en el servidor de destino o en los directorios de instalación.
- 6** Prepare un procedimiento de instalación que cree las carpetas y subcarpetas necesarias en el ordenador del usuario final y coloque los archivos en estas carpetas. Este procedimiento también debe modificar la variable de entorno **CLASSPATH** de la forma necesaria en el entorno del usuario final, para especificar la **CLASSPATH** (vía de acceso a clases) correcta para localizar las clases de Java.

**Nota** Si está creando un ejecutable o un tipo de recopilatorio JAR ejecutable con el Creador de recopilatorios o con el Creador de ejecutables nativos, puede personalizar el procedimiento de instalación modificando el archivo de configuración que inicia el ejecutable.

**Nota** Si es necesario que los usuarios dispongan de determinadas clases de Java o recopilatorios instalados en el ordenador, es posible que deba proporcionarles una forma cómoda de descargar estos archivos y añadirlos a la vía de acceso a clases local.

- 7** Indique a los usuarios cómo iniciar la aplicación (por ejemplo, haciendo doble clic en un ícono que ha suministrado, o ejecutando un script de shell en la ventana del terminal).

### Consulte

- Paso 16 del tutorial de JBuilder “Creación de un editor de texto de Java”, en *Diseño de aplicaciones con JBuilder*

## Applets

---

Dado que la aceptación de JDK es variable, la creación y la distribución de applets puede ser bastante complicada. Los siguientes pasos se pueden utilizar como directrices generales. Si desea más información sobre las applets y los navegadores, consulte “Las applets” en *Guía del desarrollador de aplicaciones Web*

- 1** Añada al proyecto todos los archivos de recurso y clases de carga dinámica que necesita la aplicación, por medio del botón Añadir archivos/paquetes de la barra de herramientas del panel del proyecto.



Optativo: Utilice el Asistente para cadenas de recursos, disponible en las ediciones SE y Enterprise, para desplazar las cadenas a un conjunto de recursos. Esto es optativo, pero facilita la internacionalización de aplicaciones.

- 2** Compile el applet.
- 3** Cree un archivo JAR o ZIP mediante la herramienta **jar** de Sun, una utilidad ZIP o el Creador de recopilatorios de JBuilder.
- 4** Añada el atributo `archive` con el nombre del archivo JAR dentro de las etiquetas `<applet>` en el archivo HTML del applet. Si tiene varios archivos JAR, inclúyalos separados por comas: `archive="archivo1.jar, archivo2.jar, archivo3.jar"`. Algunos navegadores anteriores no admiten listas múltiples y solamente aceptan archivos ZIP.
- 5** Compruebe la precisión de los valores `codebase` y `code`.

El atributo `codebase` indica la ubicación de las clases en relación con la ubicación del archivo HTML. Si el atributo `codebase` está definido en un valor de "", ".", o "./", las clases deben encontrarse en el mismo directorio que el archivo HTML. Si las clases pertenecen a un paquete, deben encontrarse en un subdirectorío del directorio del archivo HTML que coincida con la estructura del paquete. Cuando las clases se encuentran en el mismo directorio que el archivo HTML, el atributo `codebase` es optativo, ya que es el predeterminado. Si las clases se encuentran en un directorio diferente que el archivo HTML, el atributo `codebase` es necesario y debe especificar la ubicación del directorio de los archivos de la clase en relación con el directorio del archivo HTML.

El valor `code` debe ser el nombre de clase completo del applet: nombre de paquete + nombre de clase.

- 6** Inicie el navegador web y abra su archivo local HTML para la prueba inicial.
- 7** Copie el recopilatorio o el archivo de distribución configurado en el servidor de destino.
- 8** Compruebe que todas las mayúsculas y minúsculas de los nombres de clases, paquetes, archivos de revisiones y directorios de la etiqueta `<applet>` coinciden exactamente con los nombres del servidor.
- 9** Elimine la vía de acceso a clases para que no queden bibliotecas ni JDK en su vía. Debe suponer que los usuarios no cuentan con ninguna de estas en sus vías de acceso a clases.

- 10** Lleve a cabo una prueba por medio del servidor web: dirija el navegador a la URL que representa el archivo HTML del servidor de destino.

**Nota** Asegúrese de copiar el archivo HTML adecuado en la carpeta de destino y de colocar el archivo JAR en la misma carpeta. Los proyectos de JBuilder pueden contener varios archivos HTML. Si hay más de un archivo HTML, elija el que contenga la pestaña <applet>.

Optativo: Si es necesario que los usuarios dispongan de determinadas clases de Java o recopilatorios instalados en el ordenador, es posible que deba proporcionarles una forma cómoda de descargar estos archivos y añadirlos a la vía de acceso a clases local.

**Importante** Los navegadores que cumplen las especificaciones de JDK 1.0.2 no aceptan archivos comprimidos JAR. Asegúrese de que crea en su lugar un archivo ZIP.

### Consulte

- El Paso 7 del tutorial de JBuilder, “Creación de un applet”, en *Introducción a JBuilder*

## JavaBeans

---

- 1** Añada al proyecto todos los archivos de recurso y clases de carga dinámica que necesita la aplicación, por medio del botón Añadir archivos/paquetes de la barra de herramientas principal del panel del proyecto.

Optativo: Utilice el Asistente para cadenas de recursos, disponible en JBuilder SE y Enterprise, para desplazar las cadenas a un conjunto de recursos. Esto es optativo, pero facilita la internacionalización de los beans.

- 2** Compile los beans.
- 3** Utilice el Asistente para Javadoc de JBuilder, disponible en JBuilder SE y Enterprise, para crear la documentación apropiada para los beans y otro medio para crear la documentación adecuada para los clientes desarrolladores.
- 4** Cree un archivo JAR con la ayuda de la herramienta **jar** de Sun, una herramienta ZIP o el Creador de recopilatorios de JBuilder. La documentación se puede incluir en el recopilatorio si se trata de una versión de desarrollo, u omitirla si es una versión redistribuible. Si desea incluir la documentación en el recopilatorio, asegúrese de que se trata de un nodo de proyecto añadiéndolo al proyecto antes de la distribución.
- 5** Suministre a sus clientes los archivos .jar.

**Nota** Si el bean necesita alguna biblioteca de JBuilder, consulte “[Redistribución de las clases que se suministran con JBuilder](#)” en la página 15-16.

## Sugerencias de distribución

---

Consejos básicos para que el proceso de distribución resulte satisfactorio:

- Incluya las imágenes en un subdirectorio (generalmente denominado `images`) que esté asociado a las clases que las utilizan. Haga lo mismo con todos los demás archivos de recursos.
- Utilice sólo vías de acceso relativas (por ejemplo, `images/logo.gif`) para acceder a los archivos de imagen o de otro tipo que utilice la aplicación o el applet.
- Utilice paquetes, por pequeño que sea el applet o la aplicación.

## Configuración del entorno de trabajo

---

La clave para combinar el desarrollo y la distribución consiste en gestionar la colocación de los archivos. Normalmente, lo mejor es conservar los archivos de código fuente destinados a la distribución en un nivel distinto del dedicado a las herramientas de desarrollo. Esto ayuda a prevenir la eliminación o modificación accidental de elementos insustituibles.

El proceso de distribución le puede resultar más fácil si configura un entorno de trabajo que refleje las condiciones reales que tendrá el applet o la aplicación una vez distribuida, con lo que el desarrollo se acerca un poco más a la distribución instantánea. Comience el proyecto con el Asistente de JBuilder porque se encarga de colocar todo en su lugar. Una vez que el proyecto ya está creado y ejecutándose, puede crear un archivo JAR con la ayuda de la herramienta `jar` de Sun o del Creador de recopilatorios de JBuilder, disponible en las ediciones SE y Enterprise. Después de crear el recopilatorio, puede probar el programa en la línea de comandos. Asegúrese de comprobar sus applets con todos y cada uno de los navegadores que necesita utilizar.

**Sugerencia** Si se está creando un applet, se puede utilizar una copia de la página HTML que se encuentra en el proyecto local, en vez de emplear una página experimental más sencilla. Esto aumenta el realismo de la prueba externa. Cuando esté satisfecho, envíe todo el directorio a la página Web.

## Distribución en Internet

---

Si distribuye el programa en una dirección remota por medio de FTP (como un proveedor de Internet), el procedimiento básico de la distribución sigue siendo el mismo. No obstante, debe utilizar una

utilidad de FTP para transferir los archivos, siguiendo las instrucciones del proveedor del espacio web.

**Importante** No olvide transferir los archivos comprimidos y de clase como archivos binarios. Si la transferencia a la página de Internet se efectúa de forma incorrecta se provocan excepciones `java.lang.ClassFormatError`.

A menudo, la estructura de directorios de la página, vista desde FTP, no es la misma que la de la URL a la que acceden los usuarios. Pregunte a su proveedor de Internet dónde está el directorio raíz de su página y cómo puede transferir archivos hasta ella por medio de FTP. La mayoría de los programas de FTP, tanto comerciales como de shareware, permiten crear directorios además de copiar archivos, por lo que todos los pasos detallados anteriormente deberían poder realizarse, aunque con distintos mecanismos de transferencia de archivos.

Si desea información adicional acerca de la distribución en aplicaciones Web, consulte "Distribución de su aplicación web" en la *Guía del desarrollador de aplicaciones Web*.

## Distribución de aplicaciones distribuidas

---

Cuando se distribuyen aplicaciones distribuidas, el Creador de recopilatorios coloca los stubs y esqueletos en un archivo JAR. Se debe instalar el ORB en cada máquina que ejecute un programa CORBA servidor, cliente o de nivel medio. Si utiliza el ORB VisiBroker, consulte el apartado sobre distribución en la documentación de Borland Enterprise Server o en la del servidor de aplicaciones.

## Redistribución de las clases que se suministran con JBuilder

---

Los archivos JAR de libre distribución de JBuilder se encuentran en el directorio `<jbuilder>/redist/`. Los archivos recopilatorios de libre distribución del JDK se encuentran en el directorio `<jbuilder>/<jdk>/lib/`.

Si crea su recopilatorio con el Creador de recopilatorios de JBuilder, disponible en las ediciones SE y Enterprise, se detectan todos los archivos de recursos, clases y bibliotecas que necesite. El Creador de recopilatorios nunca incluye JDK en el archivo recopilatorio. Presupone que las clases JDK ya existen en el ordenador de destino en forma de JDK instalado, entorno de ejecución Java o Plug-in Java, o que se proporcionarán en la instalación.

**Importante** No obstante, en Java 1.1.1, las clases Swing/JFC no forman parte del núcleo del JDK y el Creador de recopilatorios no las detecta. Si escribe programas que utilizan estos JDK, debe descargar y entregar `swingall.jar`.

Cuando se distribuyen applets no es necesario entregar las clases JRE de JDK, porque el navegador las facilita durante la ejecución. No obstante, es necesario asegurarse de que la versión del JDK que utiliza el applet coincide con la versión que utiliza el navegador. En una intranet, puede utilizar el plug-in Java de Sun para que le proporcione el JDK actual.

Si desea más información sobre problemas con los navegadores y las compatibilidades con JDK, consulte “Problemas con los navegadores” en “Las applets” de *Guía del desarrollador de aplicaciones Web*.

**Nota** Normalmente, cuando se ejecuta un applet, en `CLASSPATH` sólo están las clases de Java. Si hay una biblioteca de terceros en `CLASSPATH` en forma de clases o como recopilatorio, y algunas de estas clases también se distribuyen en el applet o en el archivo JAR, es preferible la copia de `CLASSPATH`, ya que el cargador de clases System puede cargarla. El primer elemento de la lista es suficiente para la MV, que no toma en consideración el segundo.

Si desea información sobre los elementos de libre distribución y los de uso exclusivo del desarrollador, según las condiciones de la licencia de JBuilder, examine los archivos `<jbuilder>/license.txt` y `<jbuilder>/redist/deploy.txt`. `<jbuilder>` representa el nombre del directorio de JBuilder.

### Consulte

- Java Runtime Environment – “Notes for Developers” en <http://java.sun.com/j2se/1.4/runtime.html>
- “The JAR Trail” en el Java Tutorial de <http://java.sun.com/docs/books/tutorial/jar/>
- “JAR Guide” en <http://java.sun.com/j2se/1.4/docs/guide/jar/jarGuide.html>

## Información adicional de distribución

---

Puede encontrar información adicional en las siguientes URL:

- Java Runtime Environment – “Notes for Developers” en <http://java.sun.com/j2se/1.4/runtime.html>
- “Trail: Writing Applets” en el Java Tutorial, que discute las consideraciones básicas de las applets como la seguridad, en <http://java.sun.com/docs/books/tutorial/applet/index.html>
- “Trail: Security in Java 2 SDK 1.2” en el Java Tutorial, que discute las API de seguridad general en <http://java.sun.com/docs/books/tutorial/security1.2/index.html>
- “The JAR Trail” en el Java Tutorial de <http://java.sun.com/docs/books/tutorial/jar/>

- “Understanding the manifest” en <http://java.sun.com/docs/books/tutorial/jar/basics/manifest.html>
- “JAR Guide” en <http://java.sun.com/j2se/1.4/docs/guide/jar/jarGuide.html>
- Guías y tutoriales para desarrolladores de Sun en <http://developer.java.sun.com/developer/onlineTraining/index.html>
- “Writing advanced applications”, que resume los problemas y soluciones relacionados con la coexistencia de varias versiones de la plataforma Java en el sistema, en <http://developer.java.sun.com/developer/onlineTraining/Programming/JDCBook/version.html>
- “The Extension Mechanism Trail” del Java Tutorial en <http://java.sun.com/docs/books/tutorial/ext/index.html>. Explicación de las nuevas clases de ampliación de Java 1.2 que ponen las API personalizadas a disposición de todas las aplicaciones que se ejecutan en la plataforma Java. El mecanismo de ampliación permite al entorno de ejecución detectar y cargar las clases de ampliación sin necesidad de mencionar las clases de ampliación en la vía de acceso a clases.

## Distribución con el Creador de recopilatorios

---

Es una función de JBuilder SE y Enterprise.

El Creador de recopilatorios agiliza el proceso de creación de la distribución. Busca en el proyecto clases y recursos y da la oportunidad de personalizar el contenido del archivo JAR antes de recopilar los archivos en un archivo JAR con el archivo descriptor adecuado.

### El Creador de recopilatorios y los recursos

---

El Creador de recopilatorios reconoce automáticamente determinados tipos de archivo como recursos, tal y como se especifica en la pestaña Recursos de la ficha Generar de Propiedades de proyecto. JBuilder, al compilar, copia recursos tales como archivos de imagen, sonido y propiedades de la vía de acceso a fuentes en la vía de salida. La vía de salida, que se encuentra en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto, contiene los archivos .class que JBuilder crea al compilar un programa. Consulte “[Cómo construye JBuilder las vías de acceso](#)” en la página 4-10 para obtener más información acerca de las vías de acceso.

**Nota** JBuilder no puede determinar los recursos necesarios basándose en el código Java. Esos recursos deben estar en el proyecto.

Si desea ver una lista de la configuración por defecto según las extensiones de archivo, consulte la pestaña Recursos de la ficha Generar. Puede cambiar la configuración por defecto de JBuilder y especificar los tipos de

archivos individuales o de extensiones de archivos que desea que se copien en la vía de salida al compilar. Consulte “[Copia selectiva de los recursos](#)” en la página 6-29.

Si en el proyecto hay tipos de archivos que JBuilder no reconoce, puede añadirlos como archivos de recursos genéricos y, a continuación, especificar que se copien en la vía de salida. Para obtener más información, consulte “[Adición de tipos de archivos no reconocidos como archivos de recursos genéricos](#)” en la página 6-31.

## Selección de tipos de recopilatorios

El primer paso del Creador de recopilatorios permite seleccionar el tipo de recopilatorio que se desea crear: Applet JAR, Applet ZIP, Aplicación, Básico, Documentación, JAR ejecutable y Ejecutable nativo, además de otros tipos. Según la opción elegida, se definen diferentes valores por defecto y se encuentran disponibles diferentes opciones en los pasos del asistente. Si desea obtener más información sobre los tipos de recopilatorios, seleccione el botón de ayuda del asistente.



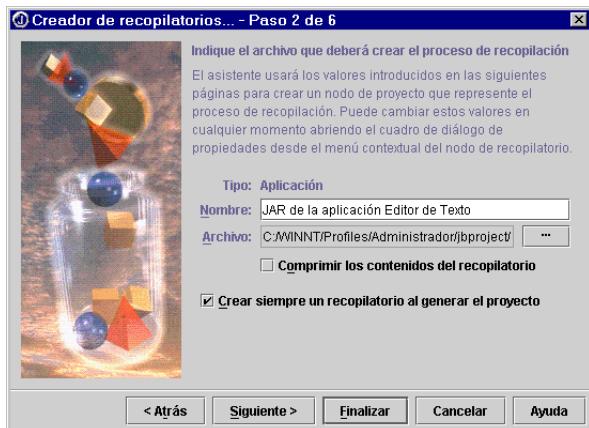
Para empezar a crear un recopilatorio:

- 1 Genere el proyecto (Proyecto | Ejecutar Make del proyecto).
- 2 Elija Asistentes | Creador de recopilatorios para abrir el Creador de recopilatorios. El Creador de recopilatorios está disponible en la ficha Generar de la galería de objetos (Archivo | Nuevo | Generar).
- 3 Seleccione un tipo de recopilatorio en la lista desplegable Tipo de recopilatorio. Si desea obtener más información sobre los tipos de recopilatorios, seleccione el botón de ayuda del asistente.
- 4 Haga clic en Siguiente para avanzar al paso siguiente del asistente.

## Definición del archivo que se ha de crear

El nombre de este paso y las opciones disponibles cambian según el tipo de recopilatorio seleccionado. Si se selecciona el tipo de recopilatorio Aplicación Web Start o Applet Web Start, una de las funciones de JBuilder Enterprise, el nombre del paso pasará a ser Seleccione aplicación web. Si se selecciona el tipo de recopilatorio JAR ejecutable, el nombre del paso es Especifique el recopilatorio que se va a utilizar.

En el Paso 2 del Creador de recopilatorios se especifica un nombre para el archivo y el nodo del recopilatorio, se selecciona la compresión del recopilatorio y se elige su frecuencia de creación. En el caso de los recopilatorios de tipo documentación o fuente, este es el último paso. Si el tipo de recopilatorio es Aplicación Web Start o Applet Web Start, es necesario que seleccione también una aplicación web o WebApp definida en el proyecto. Para poder acceder al recopilatorio Web Start, el JAR debe estar en un directorio de aplicación web. Si el tipo de recopilatorio es JAR ejecutable, es necesario que seleccione el recopilatorio fuente para el ejecutable que desee crear.



Aunque este paso cambia ligeramente según el tipo de recopilatorio que se seleccione, para la mayoría de los tipos de recopilatorios puede seguir estos pasos básicos:

- 1 Acepte el nombre del nodo de recopilatorio en el campo Nombre o escriba uno nuevo. El nodo de recopilatorio se muestra en el panel del proyecto tras finalizar el asistente, pero el recopilatorio no se crea realmente hasta que se ejecute el make o se vuelva a generar el nodo de recopilatorio. Puede hacer clic con el botón derecho del ratón en el nodo y crearlo o volver a generarlo en cualquier momento, así como restablecer sus propiedades. Para obtener más información sobre el nodo de recopilatorio, consulte “[Los nodos de recopilatorios](#)” en la página 15-35.

- 2** Introduzca el nombre, con la vía de acceso completa, del recopilatorio que debe generar el Creador de recopilatorios. El botón puntos suspensivos (...) permite buscar un directorio distinto. En el caso del tipo de recopilatorio JAR ejecutable, seleccione un recopilatorio fuente existente para el ejecutable que desea crear.

**Nota**

Sólo los navegadores JDK 1.1 o posteriores aceptan los archivos JAR. Si está desarrollando un applet para un navegador JDK 1.0.2 debe utilizar un archivo comprimido ZIP.

- 3** Tipos de recopilatorios Aplicación Web Start o Applet Web Start: Seleccione una aplicación web (WebApp) definida en el proyecto.
- 4** Active o desactive la opción Comprimir el contenido del recopilatorio. Normalmente, se suele dejar esta opción desactivada, de modo que el recopilatorio no se comprime y el tiempo necesario para cargarlo se reduce. Si el recopilatorio es un applet, esta opción aparece seleccionada por defecto. Debido a que la compresión hace que los archivos recopilatorios se vuelvan más pequeños, esto posibilita que los applets se descarguen en Internet o en las intranets de forma más rápida.
- 5** Active o desactive la opción Crear siempre un recopilatorio al generar el proyecto. Esta opción determina la frecuencia con que se crea el archivo recopilatorio. Esta opción se encuentra activada por defecto para todos los tipos de recopilatorio. Si está activada, el archivo recopilatorio se volverá a crear cada vez que se cree (make) o genere (build) el proyecto.
- 6** Pulse Siguiente para pasar al siguiente paso o bien, pulse Finalizar si este es el último paso. Si este es el último paso, genere el recopilatorio tal y como se describe en “[Generación de archivos recopilatorios](#)” en la [página 15-34](#).

Para obtener más información sobre este paso, pulse el botón Ayuda del asistente.

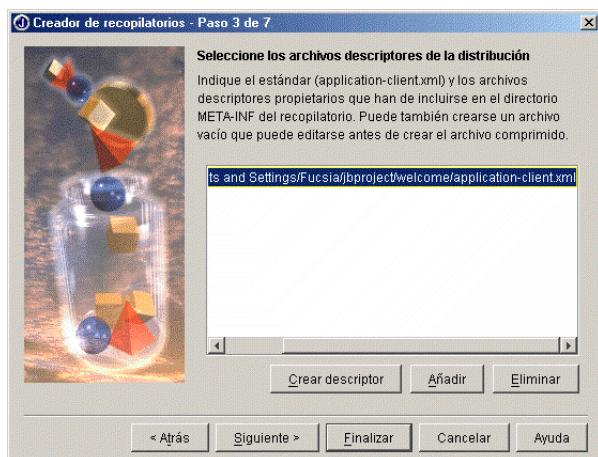
## Selección de los archivos descriptores de la distribución

---

El tipo de recopilatorio  
Cliente de aplicación  
J2EE es una función de  
JBUILDER Enterprise

Si se selecciona el tipo de recopilatorio Cliente de aplicación J2EE, en este paso del Creador de recopilatorios se eligen los archivos estándar del descriptor de distribución (`application-client.xml`) y los archivos específicos que se incluyen en el directorio `META-INF` del recopilatorio. El

Cliente de aplicación J2EE debe disponer de un archivo `application-client.xml` y puede tener otros archivos dependientes del servidor.



Para indicar los archivos del descriptor de distribución para un recopilatorio Cliente de aplicación J2EE:

- 1 Seleccione el botón Crear descriptor y acepte el nombre de archivo por defecto, `application-client.xml`, o bien, escriba otro nombre de archivo para crear un archivo vacío del descriptor de distribución. Así puede modificar este archivo antes de crear el recopilatorio.
- 2 Seleccione el botón Añadir para buscar los archivos del descriptor de distribución y añadirlos a su recopilatorio.
- 3 Haga clic en Siguiente para avanzar al paso siguiente del asistente.

Para obtener más información sobre este paso, pulse el botón Ayuda del asistente.

## Especificar las partes del proyecto que se van a recopilar

En este paso del Creador de recopilatorios se eligen las partes del proyecto que se van a incluir en el recopilatorio. También puede escoger archivos o clases adicionales.



Para indicar qué partes del proyecto desea incluir:

**1 Selecione una de estas opciones para las clases:**

- Sólo las especificadas
- Especificadas y dependientes
- Todas

Para tener un control completo sobre las clases incluidas en el recopilatorio, seleccione Sólo las especificadas. Si selecciona Las especificadas y sus dependencias, las clases que añade con el botón Añadir clases se añaden al recopilatorio así como todas las clases de la vía de salida de las clases añadidas dependen.

**Importante**

Si elige Sólo las especificadas o Especificadas y dependientes, debe añadir las clases o paquetes con el botón Añadir clases.

**2 Selecione una de estas opciones para los recursos. Si selecciona Sólo los especificados, debe añadir los recursos mediante el botón Añadir archivos.**

- Sólo los especificados
- Todos

Por ejemplo, si desea incluir todas las clases y recursos del proyecto en el recopilatorio, debe seleccionar Todas, tanto para las clases y como para los recursos. Si no desea incluir en el recopilatorio todas las dependencias, sino sólo los recursos especificados, seleccione Sólo las especificadas para las clases y los recursos y, a continuación, añada las

clases y recursos que deseé mediante los botones Añadir clases y Añadir archivos. Para añadir las clases y los archivos, las clases deben encontrarse en la vía de salida del proyecto, y los archivos en la vía de acceso a archivos fuente. Si desea obtener más información sobre estas opciones, pulse el botón de ayuda del asistente.

- 3 Seleccione el botón Añadir clases si en la categoría Clases seleccionó Sólo las especificadas o Las especificadas y sus dependientes. A continuación, seleccione las clases o paquetes que desea añadir a su recopilatorio. Las clases deben encontrarse en la vía de salida del proyecto. Si selecciona Las especificadas y sus dependientes, el Creador de recopilatorios examina los archivos de clase añadidos para encontrar dependencias de clase adicionales e incluye las dependencias en el recopilatorio.
- 4 Seleccione el botón Añadir archivos si ha elegido Sólo las especificadas en la categoría Recursos. A continuación, seleccione los archivos que desea añadir a su recopilatorio. Los archivos deben estar en la vía de acceso a fuentes del proyecto. Utilice esta opción para añadir archivos de distintos tipos al recopilatorio, tales como recursos (.gif,.jpg y sonido), archivos de propiedades y documentación recopilada (.html, readme.txt).

**Nota**

El cuadro de diálogo Añadir archivos no permite examinar el interior de los archivos recopilatorios. Si un archivo o paquete que se necesita se encuentra en un archivo recopilatorio, se debe extraer a la carpeta fuente y después añadirlo pulsando el botón Añadir archivos.

- 5 Haga clic en Siguiente para avanzar al paso siguiente del asistente.

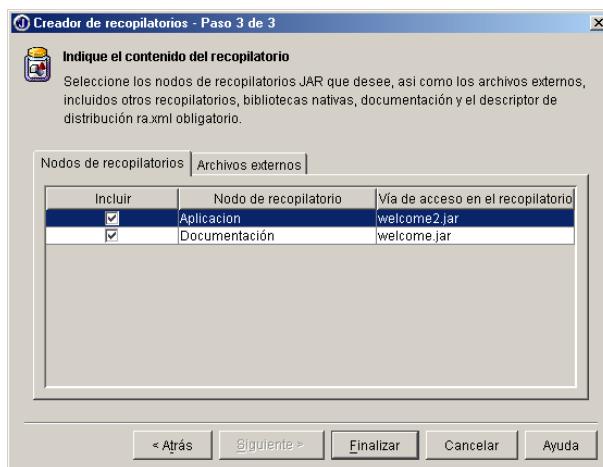
## Definición del contenido para un recopilatorio RAR (Adaptador de recursos)

---

El tipo de recopilatorio  
Adaptador de recursos  
(RAR) es una función de  
JBuilder Enterprise

Si selecciona un tipo de recopilatorio RAR (Adaptador de recursos), en este paso del Creador de recopilatorios ha de elegir los nodos de recopilatorios JAR del proyecto, además de los archivos externos, que desea añadir al recopilatorio. Los archivos externos pueden incluir otros

recopilatorios, más bibliotecas y documentación, y el descriptor de distribución `ra.xml` necesario. El archivo `ra.xml` ya debe estar creado.



Para definir el contenido para un recopilatorio RAR (Adaptador de recursos):

- 1 En la ficha Nodos de recopilatorios, marque todos los nodos de recopilatorios JAR que desee incluir en el recopilatorio.
- 2 Seleccione todos los archivos que desee añadir al recopilatorio, tales como otros recopilatorios, bibliotecas, documentación y el descriptor de distribución `ra.xml` necesario. Seleccione el botón Añadir para buscar los archivos.
- 3 Pulse Finalizar para cerrar el asistente.
- 4 Genere el recopilatorio según se explica en “[Generación de archivos recopilatorios](#)” en la página 15-34.

## Determinar las dependencias entre bibliotecas

En este paso del Creador de recopilatorios se determina cómo tratar las dependencias entre bibliotecas. Las bibliotecas utilizadas en el proyecto se muestran en una lista y es posible elegir una estrategia de distribución individual para cada una.

**Nota** El Creador de recopilatorios nunca incluye JDK en el archivo recopilatorio. Presupone que las clases JDK ya existen en el ordenador de destino en forma de JDK instalado, entorno de ejecución Java o Plug-in Java, o que se proporcionarán en la instalación.

**Importante** En JDK 1.1.x, las clases Swing/JFC **no** se entregan como parte del JDK. Si escribe programas que utilizan cualquiera de los JDK, debe descargar y entregar swingall.jar, que contiene estos archivos.



**Nota** Si distribuye clases del paquete DataStore (`com.borland.datastore`) o el paquete VisiBroker, aparecerá un mensaje que recuerda al programador que la distribución de estos paquetes requiere una licencia de distribución independiente. Si ya tiene la licencia necesaria y no desea volver a ver esta advertencia en el proyecto actual, active “No advertir nuevamente en este proyecto”.

Para definir las dependencias entre bibliotecas:

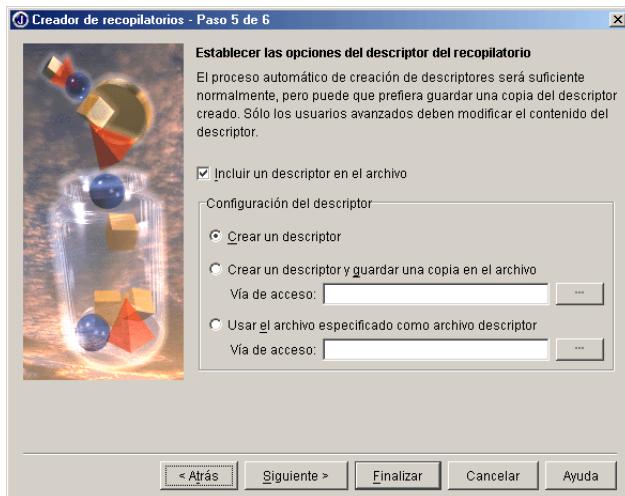
- 1 Seleccione una biblioteca de la lista.
- 2 Elija una de las siguientes opciones:
  - Nunca incluir clases ni recursos
  - Incluir clases necesarias y recursos conocidos
  - Incluir clases necesarias y todos los recursos
  - Incluir siempre todas las clases y recursos

Normalmente, la opción Incluir clases necesarias y todos los recursos es una buena elección para la distribución de bibliotecas. Si desea obtener más información sobre estas opciones, pulse el botón de ayuda del asistente.

- 3 Seleccione otra biblioteca de la lista y una opción de biblioteca.
- 4 Haga clic en Siguiente para avanzar al paso siguiente del asistente.

## Configurar las opciones del archivo descriptor del recopilatorio

En este paso del Creador de recopilatorios, se selecciona cómo crear el archivo descriptor. En la mayoría de los casos, la opción por defecto, Crear un descriptor, resulta suficiente. Para obtener información acerca del descriptor, consulte el tema “[Conceptos básicos acerca del archivo descriptor](#)” en la página 15-3.

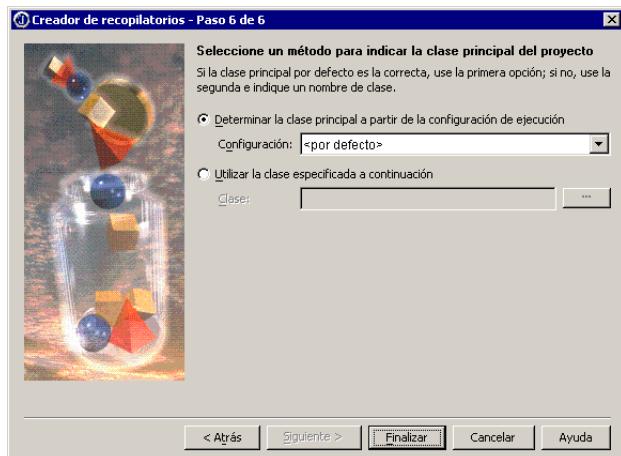


Para configurar las opciones del archivo descriptor del recopilatorio:

- 1 Acepte la opción por defecto, Incluir un descriptor en el recopilatorio, si desea que se incluya un archivo descriptor.
- 2 Seleccione una de estas opciones si desea que se incluya un archivo descriptor:
  - Crear un descriptor
  - Crear un descriptor y guardar una copia en el archivo
  - Usar el archivo especificado como archivo descriptor
- 3 Si desea obtener más información sobre estas opciones, pulse el botón de ayuda del asistente.
- 4 Pulse Siguiente para continuar o Finalizar si éste es el último paso del asistente. Si este es el último paso, genere el recopilatorio tal y como se describe en “[Generación de archivos recopilatorios](#)” en la página 15-34.

## Seleccionar un método para indicar la clase principal de la aplicación

Este paso le permite definir la clase principal de la aplicación. La clase principal es la encargada de ejecutar la aplicación. Contiene el método `public static void main(String[] args)`.



Para configurar la clase principal del recopilatorio:

**1** Elija una de las siguientes opciones:

- Determinar la clase principal para la configuración de ejecución

Esta opción determina la clase principal de la elección realizada en la lista desplegable Configuración. La lista desplegable incluye todas las configuraciones de ejecución del tipo Aplicación del proyecto. Seleccione una configuración de ejecución del proyecto o bien, seleccione <Por defecto>. <Por defecto> utiliza la configuración de ejecución del proyecto que está marcada por defecto. Si no existe ninguna o la que hay no es una configuración de ejecución de Aplicación, se utiliza la primera configuración de ejecución de Aplicación de la lista de configuraciones de ejecución del proyecto. Para obtener más información sobre las configuraciones de ejecución, consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7.

**Nota**

Si esta creando un ejecutable nativo o un tipo de recopilatorio JAR ejecutable y especifica en este paso una configuración de ejecución, la clase principal, los parámetros de la aplicación y los parámetros de la MV de esta configuración de ejecución se incluyen en el archivo de configuración utilizado para lanzar el ejecutable. Puede personalizar el archivo de configuración en el último paso del asistente.

- Utilizar la clase especificada a continuación

Esta opción determina la clase principal para la clase especificada. Pulse el botón de puntos suspensivos y busque y seleccione una clase.

- 2** Pulse Siguiente para continuar o Finalizar si éste es el último paso del asistente. Si este es el último paso, genere el recopilatorio tal y como se describe en “[Generación de archivos recopilatorios](#)” en la página 15-34.

Para obtener más información sobre este paso, pulse el botón Ayuda del asistente.

**Advertencia** Si no se especifica una clase principal, lo siguiente **no** se ejecuta:

- Lanzamiento de ejecutables nativos
- Utilización de `java -jar <jarname>` desde la línea de comandos
- Doble clic en un recopilatorio JAR

## Determinar los archivos ejecutables que se van a crear

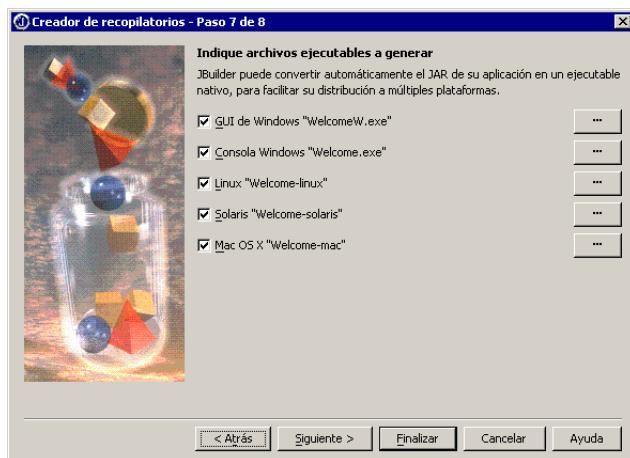
---

El Creador de recopilatorios puede agrupar automáticamente el archivo JAR de la aplicación con englobadores de ejecutables nativos para facilitar su distribución a varias plataformas.

Esta paso se encuentra disponible si ha seleccionado Ejecutable nativo o JAR ejecutable como el tipo de recopilatorio en el Creador de recopilatorios, y también si utiliza el Creador de ejecutables nativos (Asistentes | Creador de ejecutables nativos).

El archivo ejecutable contiene el programa de inicio, el archivo de configuración para el programa de inicio y el archivo JAR con las clases Java y los recursos. El Creador de recopilatorios configura el comentario JAR en el archivo de configuración, y añade el programa de inicio al principio del archivo. El Creador de recopilatorios permite personalizar el archivo de configuración del programa de inicio en el último paso del asistente. Tenga en cuenta que, dado que el JDK **no** se incluye en el archivo JAR, éste debe estar en el ordenador del usuario para que la ejecución sea posible.

- Importante** Debe especificar una clase principal en el paso anterior o, de lo contrario, no podrá ejecutarlo.



Para crear un archivo ejecutable:

- 1 Seleccione los ejecutables que desea crear. Pulse el botón de puntos suspensivos para cambiar el nombre y/o guardar el archivo en una ubicación diferente.
- 2 Seleccione Siguiente para establecer las opciones de la configuración de ejecución del ejecutable.

Para obtener más información sobre este paso, pulse el botón Ayuda del asistente.

## Ejecución de ejecutables

Tenga en cuenta que el archivo JAR **no** incluye el JDK, por lo que éste debe estar situado en el ordenador del usuario para que el ejecutable funcione. Los ejecutables específicos de la plataforma buscan el JDK instalado en la siguiente ubicación:

- Windows: Registro.
- Linux/Solaris variable de entorno de JAVA\_HOME y en la vía de acceso del usuario.
- Mac OS X ubicación predeterminada para el JDK.

- Nota** Puede redefinir este comportamiento por defecto si especifica la ubicación del JDK en un archivo de configuración personalizado. El archivo ejecutable aparece entonces en la ubicación especificada. Si desea obtener más información sobre los archivos de configuración, consulte el siguiente paso.

Si crea el ejecutable en la plataforma Windows y desea trasladarlo a otra plataforma, puede que necesite cambiar los permisos para convertirlo en ejecutable.

Si selecciona la opción Mac OS X, se crea una aplicación que solamente se puede iniciar desde la línea de comandos. Para crear una aplicación que se pueda iniciar desde el Buscador, los usuarios de Mac necesitan crear una agrupación de aplicaciones. Consulte la documentación para el desarrollador de Mac OS X de Apple para obtener más información acerca de las agrupaciones y paquetes de aplicaciones.

## Definir las opciones de la configuración de ejecución

JBuilder crea automáticamente una configuración ejecutable para iniciar el recopilatorio ejecutable, basándose en la configuración de ejecución especificada en el paso Selección de un método para indicar la clase principal de la aplicación. Si selecciona una configuración de ejecución en ese paso del asistente, el Creador de recopilatorios incluye la clase principal, los parámetros de la aplicación y los parámetros de la MV del archivo de configuración que inicia el ejecutable. Para obtener más información sobre las configuraciones de ejecución, consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7.

Si desea personalizar la configuración del ejecutable, puede modificar la configuración que crea JBuilder o crear su propia configuración. Para obtener más información sobre la creación de archivos de configuración, consulte el [Apéndice A, “Creación de archivos de configuración para ejecutables nativos.”](#)



Para crear una configuración ejecutable para el recopilatorio:

**1** Elija una de las siguientes opciones:

- Crear configuración ejecutable
- Crear configuración ejecutable y guardar una copia en el archivo especificado
- Redefinir la configuración ejecutable con el archivo especificado

**Nota**

Si elige guardar una copia o redefinir la configuración, el archivo de configuración se añade al proyecto.

**2** Pulse Finalizar para cerrar el asistente.

**3** Genere el recopilatorio según se explica en “[Generación de archivos recopilatorios](#)” en la página 15-34.

Para obtener más información sobre este paso, pulse el botón Ayuda del asistente.

## Creación de ejecutables con el Creador de ejecutables nativos

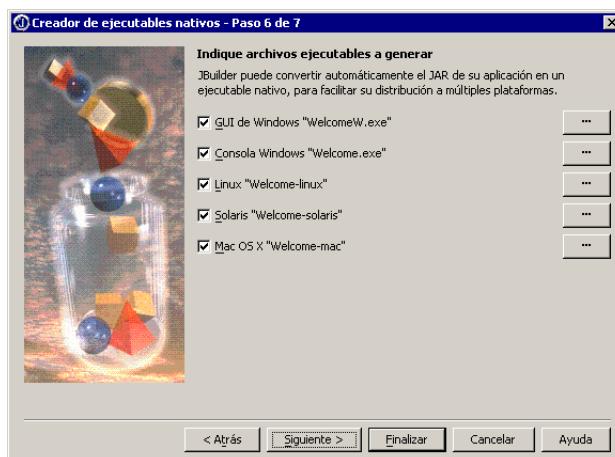
---

Es una función de JBuilder SE y Enterprise.

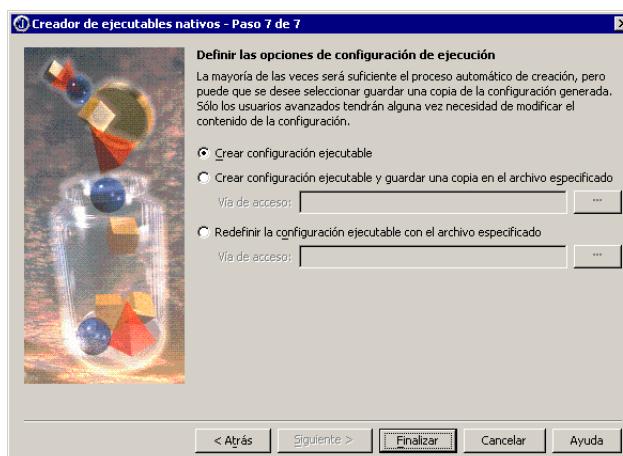
El Creador de ejecutables nativos agrupa automáticamente un archivo JAR de aplicación con los englobadores de ejecutables nativos para Windows, Linux, Solaris y Mac OS X. Tenga en cuenta que el JDK **no** se incluye en el archivo JAR, por lo que debe instalarse en el ordenador del usuario para poder ejecutar el ejecutable. El Creador de ejecutables nativos, disponible en el menú Asistentes y en la ficha Generar de la galería de objetos, es una forma de acceso abreviado a este tipo de recopilatorio del Creador de ejecutables. Consulte “[Distribución con el Creador de recopilatorios](#)” en la página 15-18 si desea más detalles sobre el Creador de recopilatorios.

JBuilder genera los ejecutables seleccionados y los guarda con el nombre del proyecto y la extensión adecuada en el directorio raíz del directorio del proyecto actual. Seleccione el botón de puntos suspensivos junto a su selección, si desea modificar el nombre por defecto del ejecutable y/o

guardarlo en una ubicación diferente. Quite la marca de los ejecutables que no desee que cree JBuilder.



El Creador de ejecutables nativos también proporciona opciones para determinar la clase principal para las configuraciones de ejecución y para crear archivos de configuración personalizados para iniciar el ejecutable.



El Creador de ejecutables nativos incluye estos pasos, que son los mismos que los de los tipos JAR ejecutable y Ejecutable nativo del Creador de recopilatorios:

- Indique el archivo que deberá crear el proceso de recopilación.
- Indique qué partes de este proyecto deben recopilarse.
- Indique cómo traficar las dependencias de las bibliotecas.
- Establecer las opciones del descriptor del recopilatorio.
- Seleccione un método para indicar la clase principal del proyecto.

- Indique archivos ejecutables a generar.
- Definir las opciones de la configuración de ejecución.

**Importante** Los ejecutables nativos **tienen que** tener una clase principal especificada para poder ejecutarse.

Una vez que ha terminado de utilizar el asistente, pulse con el botón derecho del ratón en el nodo de archivos ejecutables nativos del panel del proyecto y, a continuación, seleccione Ejecutar Make para crear los ejecutables y el archivo JAR. Amplíe el nodo para ver el archivo JAR creado y los ejecutables. Si desea modificar las propiedades de este nodo, pulse con el botón derecho del ratón y seleccione Propiedades.

### Consulte

- [Apéndice A, “Creación de archivos de configuración para ejecutables nativos”](#)
- [“Los nodos de recopilatorios” en la página 15-35](#)

## Generación de archivos recopilatorios

---

Al salir del Creador de recopilatorios y del Creador de ejecutables nativos, aparece automáticamente un nodo de recopilatorios en el panel del proyecto. Sin embargo, el archivo recopilatorio no se crea hasta que no se ha generado el nodo de recopilatorios.

El momento de la generación del nodo de recopilatorios viene determinado por la opción elegida en el paso Especifique el archivo que se va a crear del Creador de recopilatorios y del Creador de ejecutables nativos: la opción Crear siempre un recopilatorio al generar el proyecto.

- Si esta opción está activada, el archivo de revisiones se genera cada vez que se elige Proyecto | Ejecutar Make de proyecto o Proyecto | Generar el proyecto.
- Si esta opción está desactivada, puede crear el archivo de revisiones haciendo clic con el botón derecho en el nodo del recopilatorio en el panel del proyecto y eligiendo Ejecutar Make o Generar.

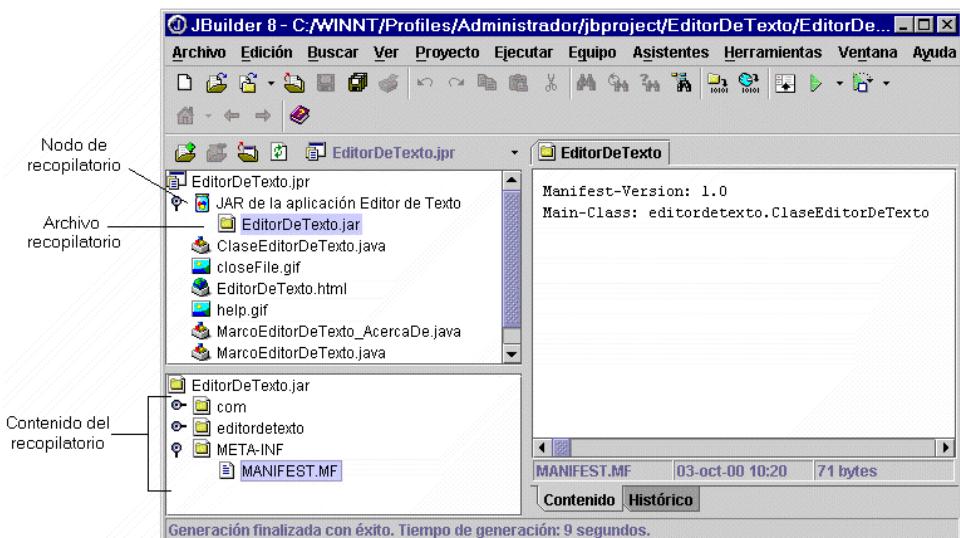
# Los nodos de recopilatorios

Con la ayuda del Creador de recopilatorios o del Creador de ejecutables nativos puede crear varios archivos recopilatorios con opciones diferentes para probar varias distribuciones. Primero, utilice el Creador de recopilatorios para incluir diferentes clases, dependencias y recursos en diferentes combinaciones. Luego podrá comparar el contenido de cada archivo de revisiones y descubrir la mejor estrategia para lograr sus objetivos de tamaño, tiempo de descarga e instalación.

El archivo recopilatorio puede crearse o modificarse en cualquier fase del desarrollo. También se puede ver el contenido del archivo recopilatorio y del archivo descriptor.

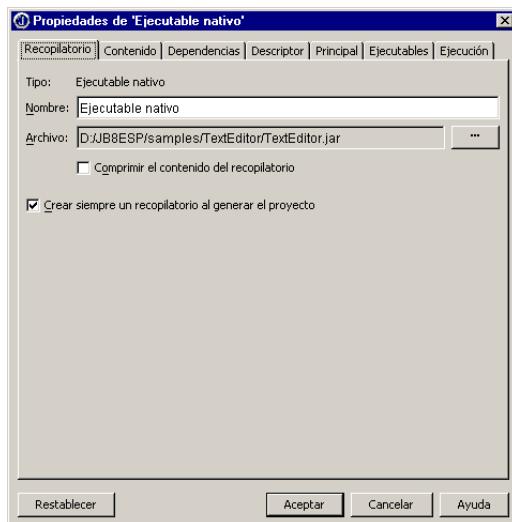
## Presentación del archivo recopilatorio y el archivo descriptor

Para ver el archivo de revisiones y el archivo descriptor, expanda el nodo de recopilatorios en el panel del proyecto. Haga doble clic en el archivo de revisiones en el panel del proyecto para mostrar su contenido en el panel de la estructura y el archivo descriptor en el panel de contenido. Haga doble clic en otros archivos del panel de estructura para abrirlos como archivos de sólo lectura en el editor.



## Modificación de las propiedades de los nodos de recopilatorios

Durante el desarrollo puede restablecer las propiedades del nodo de archivos para cambiar el contenido del archivo de revisiones resultante. Para cambiar las propiedades, haga clic con el botón derecho en el nodo de recopilatorios y elija Propiedades. El cuadro de diálogo Propiedades muestra las fichas correspondientes a los pasos del Creador de recopilatorios y del Creador de ejecutables nativos.



## Eliminación, borrado y asignación de nombres a recopilatorios

Después de crear varias versiones de recopilatorios para su proyecto, puede que desee eliminar, borrar o cambiarle el nombre al que no le interesa. Existen varios modos de llevar esto a cabo, según lo que desee hacer.

Si se elimina el nodo de recopilatorio no se suprime el archivo JAR, pero se elimina a ambos del proyecto. Si lo desea, siempre puede añadir de nuevo a su proyecto un recopilatorio que haya eliminado.

Si desea eliminar del proyecto el nodo de recopilatorios y su contenido, haga lo siguiente:

- Haga clic con el botón derecho en el nodo de recopilatorio del panel del proyecto y a continuación seleccione la opción Eliminar del proyecto.
- Seleccione el nodo de recopilatorio del panel del proyecto y haga clic en el botón Eliminar del proyecto de la barra de herramientas de ese panel.

- Seleccione el nodo de recopilatorios del panel del proyecto y, a continuación, seleccione Proyecto | Eliminar del proyecto.

También se puede eliminar el archivo recopilatorio del proyecto. Esto resulta útil si desea restablecer las propiedades del nodo de recopilatorio o crear un archivo recopilatorio nuevo para el proyecto. Pero tenga en cuenta que si ha seleccionado la opción Crear siempre un recopilatorio al generar el proyecto, el archivo recopilatorio se crea de nuevo al generar el nuevo proyecto. Para ver si esta opción está activada, haga clic con el botón derecho del ratón en el nodo de recopilatorio y a continuación seleccione Propiedades. Esta opción de la ficha Recopilatorio está activada por defecto. Después de eliminar el archivo recopilatorio, restablezca las propiedades del recopilatorio, haga clic con el botón derecho en el nodo de recopilatorios y a continuación seleccione Ejecutar Make para recrear el recopilatorio con la nueva configuración.

Para borrar el archivo recopilatorio, realice una de las siguientes acciones:

- Pulse el botón derecho sobre el nodo de recopilatorios del panel del proyecto y seleccione Limpiar.
- Expanda el nodo del recopilatorio, haga clic con el botón derecho del ratón sobre el archivo JAR, y escoja Borrar <filename.jar>

Por último, puede cambiar el nombre de los nodos de recopilatorios y de los archivos.

Para cambiar el nombre de los archivos y nodos de recopilatorios:

- Pulse con el botón derecho del ratón en el nodo de recopilatorios o en el archivo recopilatorio del panel del proyecto, y seleccione Cambiar nombre a.
- Seleccione el nodo de recopilatorios o el archivo recopilatorio en el panel del proyecto y, a continuación, Proyecto | Cambiar nombre a.



# 16

## Internacionalización de programas con JBuilder

Es una función de JBuilder SE y Enterprise.

En este capítulo se tratan los aspectos relacionados con el diseño de aplicaciones de Java destinadas a cumplir las necesidades del público mundial. No hay motivos para limitar el uso de las applets y aplicaciones a un país determinado, cuando, con un poco más de esfuerzo, se podrían utilizar en todo el mundo. Las funciones especiales de JBuilder facilitan el aprovechamiento de la capacidad de internacionalización de Java, que permite personalizar las aplicaciones para los países e idiomas deseados sin necesidad de efectuar trabajosos cambios en el código.

Este capítulo trata sobre funciones especiales de JBuilder, y no se ha escrito con el ánimo de examinar a fondo las funciones de internacionalización de Java. Por tanto, si desea más información antes de empezar, consulte la documentación de Java. Antes de emprender la explicación sobre las funciones de “[Funciones de internacionalización de JBuilder](#)” en la página 16-3, repase el siguiente apartado, que trata sobre los términos específicos de la internacionalización.

### Términos y definiciones de internacionalización

- **Internacionalización (i18n)**

La internacionalización es el proceso de diseño o conversión de programas para su uso en más de una versión idiomática. A causa de su longitud, en inglés esta palabra se abrevia a menudo como “i18n”, 18 representa el número de letras que hay entre la ‘i’ y la ‘n’ de “internacionalización”.

- **Versión localizada**

"Versión localizada" define un conjunto de convenciones culturales específicas para la presentación, el formato y la ordenación de datos. En Java, las versiones localizadas se especifican por medio de un objeto `Locale`, que es simplemente un contenedor para cadenas que identifican un idioma y un país determinados.

- **Inclusión en recursos**

La inclusión en recursos es la parte del proceso de internacionalización que consiste en aislar los recursos específicos de una versión localizada del código fuente en módulos que se pueden añadir o extraer de la aplicación de forma independiente. Algunos ejemplos de recursos específicos de versiones localizadas son el texto que se muestra al usuario, las reglas empresariales y la lógica de la aplicación. Java suministra una serie de clases `ResourceBundle` para la inclusión en recursos de cadenas y objetos en programas Java.

- **Localización (l10n)**

La localización es la adaptación de los recursos de un programa a un país y un idioma determinados. Observe que la internacionalización generaliza los programas para todos los países e idiomas, mientras que la localización los especializa para zonas concretas. A causa de su longitud, esta palabra se abrevia a menudo en inglés como "l10n". El '10' representa el número de letras que hay entre la 'l' y la 'n' de "localización".

- **Codificación nativa**

La codificación nativa, también conocida como página de códigos y como conjunto de caracteres, determina la redefinición de valores numéricos a caracteres simbólicos dentro de cada sistema operativo. Como las codificaciones nativas varían según los distintos sistemas operativos (y en ocasiones dentro del mismo sistema operativo), un archivo que contenga caracteres de un sistema puede presentar caracteres completamente distintos en otro sistema que utilice una codificación nativa distinta.

- **Unicode**

Unicode es una norma de codificación de caracteres que mantiene [The Unicode Consortium](#). Esta norma define una correspondencia de caracteres para la mayoría de los idiomas del mundo. Se puede especificar cualquier carácter de Unicode en el código fuente de Java por medio de su secuencia de escape de Unicode, \uNNNN, donde NNNN es el valor hexadecimal del carácter en el conjunto de caracteres Unicode. Los caracteres y cadenas se procesan siempre en la máquina virtual de Java como valores Unicode de 16 bits.

# Funciones de internacionalización de JBuilder

---

Entre las funciones de JBuilder hay varias destinadas a facilitar la internacionalización de las applets y aplicaciones Java. En los siguientes apartados se tratan estas funciones:

- Aplicación de ejemplo multilingüe
- Eliminación de cadenas no modificables (hard-coded) incluidas en el código por medio del Asistente para extracción de recursos
- Funciones de internacionalización de dbSwing
- Componentes que identifican la versión localizada
- Los componentes de JBuilder muestran todos los caracteres Unicode
- Funciones de internacionalización del diseñador de interfaz de usuario
- Unicode en el Depurador IDE
- Elección de una codificación nativa para el compilador

## Aplicación de ejemplo multilingüe

---

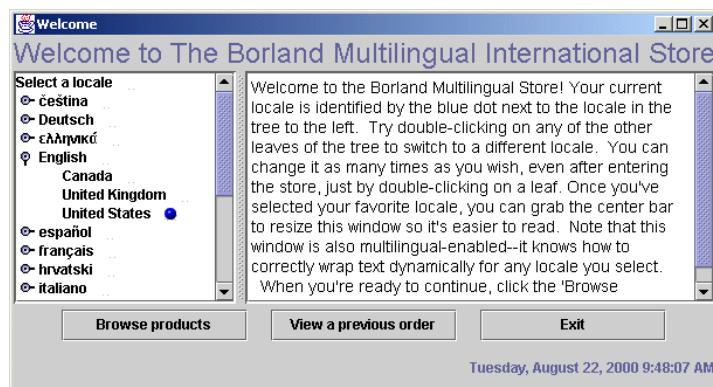
JBuilder incluye una amplia aplicación multilingüe de anotación de pedidos, que demuestra detalladamente muchos de los conceptos importantes de la internacionalización. Este ejemplo muestra también muchas otras funciones importantes de JBuilder, como es la construcción de aplicaciones con los componentes de JBuilder, la creación de JavaBeans internacionalizados y el uso de la arquitectura DataExpress.

El proyecto `IntlDemo.jpr` se encuentra en el directorio `samples/dbswing/MultiLingual` de la instalación de JBuilder. Encontrará más información en el archivo de documentación `IntlDemo.html` y en el código fuente del ejemplo suministrado. El ejemplo `IntlDemo` acepta e incluye traducciones para 15 versiones localizadas distintas.

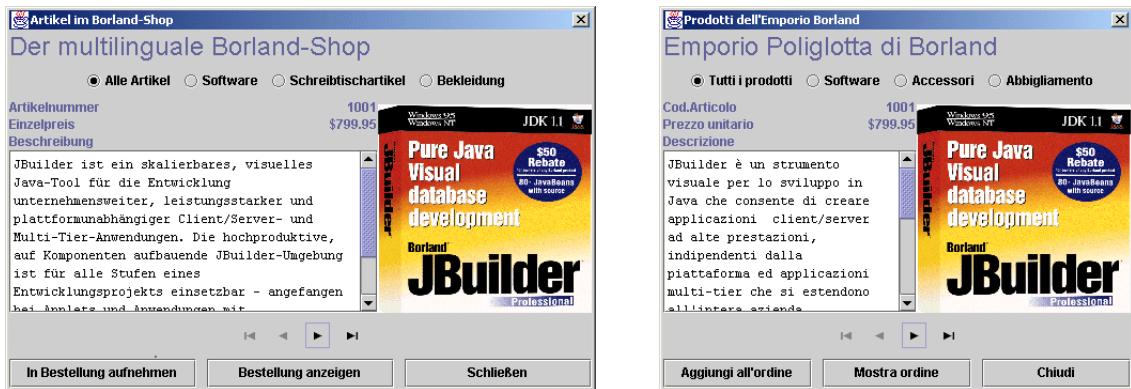
El JavaBean `LocaleChooser` del Multilingual International Store de Borland, permite cambiar el idioma de la aplicación en tiempo de ejecución. Con

## Aplicación de ejemplo multilingüe

ello se adapta automáticamente la interfaz gráfica de usuario al idioma y las convenciones de la versión localizada elegida.



ProductFrame permite a los usuarios ver imágenes de productos de Borland Store y descripciones escritas en su propio idioma. Observe que el tamaño de los botones y etiquetas se ajusta automáticamente para las traducciones al italiano y al alemán que se muestran a continuación.



OrderFrame presenta la dirección del cliente y el importe del pedido en el formato adecuado para la versión localizada del usuario. A continuación se muestra OrderFrame en alemán:



## Eliminación de cadenas no modificables (hard-coded) incluidas en el código

Un error de diseño frecuente que impide que las aplicaciones y applets se localicen con facilidad es la presencia en el código fuente de cadenas no modificables (hard-coded) que se muestran en la interfaz gráfica de usuario de la aplicación o el applet.

Aunque es posible incluir estas cadenas en los recursos después de haber finalizado y probado el código fuente, es mejor realizar esta tarea como parte del proceso de diseño de la interfaz gráfica de usuario.

La inclusión en recursos de la interfaz gráfica de usuario a medida que se escribe el código tiene dos ventajas:

- No es necesario volver y examinar todas las cadenas no modificables (hard-coded) del código fuente para comprobar cuáles se deben incluir en los recursos. Además de ser un proceso muy trabajoso, en ocasiones al desarrollador (o a otra persona que esté menos familiarizada con el código) le resulta difícil determinar qué cadenas se deben incluir en los recursos.
- Incluir las cadenas en recursos durante una fase temprana puede ayudar a descubrir los diseños de interfaz gráfica de usuario no

internacionalizados al principio del proceso de desarrollo, con lo que se evita el esfuerzo de tener que volver a escribirlo posteriormente.

JBuilder proporciona dos herramientas para beneficiarse de este método con muy poco esfuerzo: El Asistente para extracción de recursos y el cuadro de diálogo Propiedad localizable.

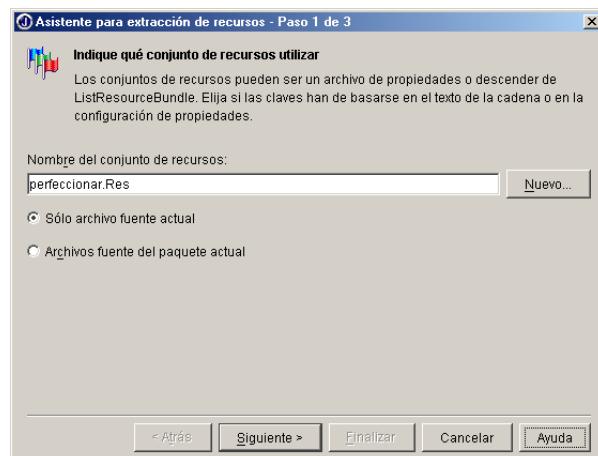
## Utilización del Asistente para extracción de recursos

El Asistente para extracción de recursos examina el código fuente y permite extraer las cadenas no modificables (hard-coded) y los caracteres aislados, como las teclas aceleradoras, y llevarlas a clases ResourceBundle de Java. Este asistente funciona con todos los archivos Java, no sólo con el código fuente generado con JBuilder.

Los ResourceBundle (conjuntos de recursos) son archivos especializados que contienen una serie de cadenas traducibles. (También pueden contener otros tipos de datos, aunque es menos frecuente.) Una clave de recurso única identifica cada cadena traducible del ResourceBundle. La cadena incrustada en el código de la aplicación es sustituida por una referencia al conjunto de recursos y a la clave del recurso. Esta separación entre la lógica de la aplicación y los elementos traducibles se conoce como *extracción de recursos*. A continuación, estos archivos de recursos separados se envían a los traductores.

Para colocar las cadenas en una clase ResourceBundle,

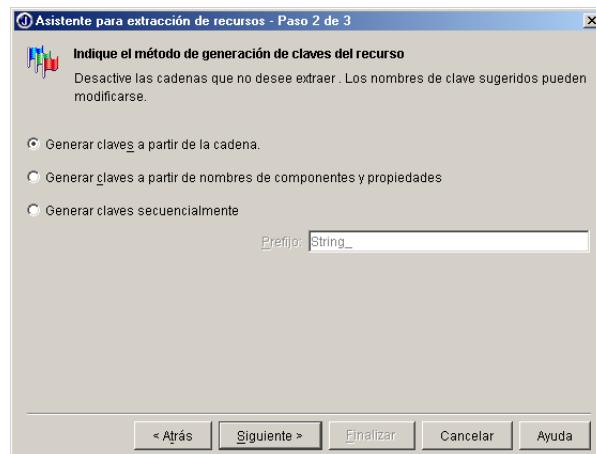
- 1 En el panel del proyecto, haga doble clic en el archivo de código fuente que desea examinar; de esta forma se abre en el editor.
- 2 Seleccione Asistentes | Recursos de cadenas para abrir el Asistente para extracción de recursos:



- 3** Indique el nombre del conjunto de recursos que utiliza. JBuilder propone un nombre por defecto que se puede aceptar o cambiar. Por defecto, el objeto ResourceBundle que se crea es del tipo ListResourceBundle. Si desea crear un conjunto de recursos PropertyResourceBundle pulse Nuevo, elija PropertyResourceBundle en la lista desplegable Tipo y pulse Aceptar en el cuadro de diálogo que aparezca.

Los PropertyResourceBundles son archivos de texto con extensión .properties y tienen la misma ubicación que los archivos de clase del código fuente. Los ListResourceBundles se proporcionan como archivos fuente de Java. Tanto los ListResourceBundles nuevos como los modificados necesitan volver a compilarse para su distribución ya que están implementados como código fuente Java. No es necesario recompilar los PropertyResourceBundles cuando se modifican o añaden traducciones a la aplicación. Los ListResourceBundles proporcionan un rendimiento mucho mayor que los PropertyResourceBundles.

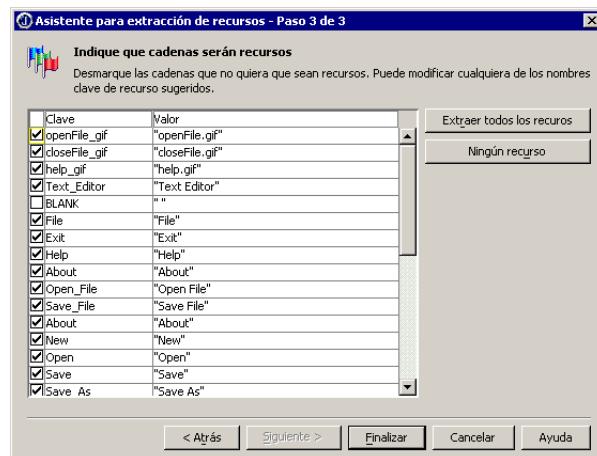
- 4** Si desea reestructurar únicamente el código del archivo que se encuentra en el editor, elija Sólo archivo fuente actual. Si desea extraer el código de todos los archivos del paquete en el que se encuentra el archivo abierto, elija la opción Archivos fuente del paquete actual y pulse Siguiente.



- 5** Indique de qué forma desea que se generen las claves. Las cadenas se identifican por medio de claves. Por ejemplo, si se desea extraer el código de la cadena "Open File" y se activa la opción Generar clave a partir de la cadena, la clave es Open\_File. La misma cadena se puede convertir en jButton1\_ToolTipText si se elige Generar claves a partir de los nombres de componentes y propiedades. Si se elige la opción Generar claves secuencialmente, la clave podría ser String\_10 si es la décima cadena del archivo. Si se elige la tercera opción también se

puede añadir un prefijo al nombre de la clave. El prefijo por defecto es String\_.

- 6 Pulse Siguiente para abrir la última ficha del Asistente para extracción de recursos:



Para ordenar la columna Clave o Valor, haga clic en la cabecera correspondiente. Si se vuelve a pulsar la cabecera, la columna se ordena en sentido inverso.

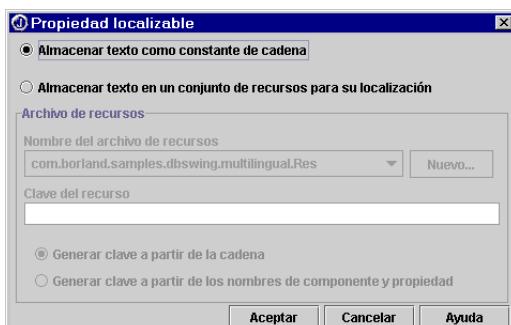
Si se hace clic en una cadena que aparece en el Asistente para extracción de recursos, la línea en la que aparece se resalta en el código fuente. Esta función permite examinar el contexto de la cadena en el código.

- 7 Desactive las cadenas para las que no desee extraer recursos.  
8 Pulse el botón Finalizar.

## Utilización del cuadro de diálogo Propiedad localizable

El cuadro de diálogo Propiedad localizable permite definir cadenas visibles mientras crea o personaliza componentes en la interfaz de usuario. Basta con hacer clic con el botón derecho del ratón en cualquier propiedad de texto, como la etiqueta de un control ButtonControl y

seleccionar el comando Conjunto de recursos para presentar el cuadro de diálogo Propiedad localizable.



Este cuadro de diálogo presenta opciones parecidas a las del Asistente para extracción de recursos, pero sólo incluye las que afectan a la propiedad seleccionada. Active la opción Almacenar texto en conjunto de recursos para localización y elija la forma de generar las claves, si no lo ha hecho aún o desea efectuar algún cambio. Como resulta tan rápido y cómodo efectuar la extracción de estos recursos desde el Inspector, se puede convertir con facilidad en parte integrante de la personalización de los componentes de la aplicación.

## Funciones de internacionalización de dbSwing

dbSwing es una  
característica de JBuilder  
Enterprise.

La arquitectura dbSwing incluye varias decisiones de diseño que facilitan la internacionalización de aplicaciones y applets:

- Las funciones desaconsejadas (deprecated), no internacionalizadas, que se utilizan en JDK 1.0 se evitan en dbSwing.
- Todos los mensajes en dbSwing se guardan en clases ResourceBundle, con lo que se permite a las aplicaciones que se construyen con estos componentes mostrar el texto en el idioma correspondiente a la versión localizada del usuario final.
- Todos los componentes de dbSwing gestionan aspectos internacionales, como el formato de datos y el sistema de ordenación acordes con la versión localizada.

La mayoría de los componentes dbSwing de JBuilder incluye la propiedad textWithMnemonic. Esta propiedad acepta un carácter acelerador que se especifica en la cadena utilizada para mostrar el texto del componente. El diseño Swing (de donde derivan muchos componentes dbSwing) consiste en guardar el texto y los caracteres mnemotécnicos en propiedades independientes. Esto dificulta la localización ya que los traductores suelen disponer de poco contexto en el que basar la traducción de las cadenas. Los componentes dbSwing guardan el carácter mnemotécnico incrustado

en el texto, el traductor puede escoger el mnemotécnico correcto. Si esta función se utiliza adecuadamente, puede proporcionar contexto durante la traducción.

El componente `IntlSwingSupport` de `JBuilder` admite la internacionalización de Swing en doce versiones distintas. Cuando se instancia `IntlSwingSupport` actualiza automáticamente los recursos internos localizables de Swing en función del idioma actual. `IntlSwingSupport` sólo debe ser instanciado una vez en una aplicación y al inicio de la aplicación, antes de que se presenten los componentes Swing.

Para inicializar `IntlSwingSupport` en un idioma distinto del establecido por defecto (por ejemplo, en una aplicación multilingüe), defina la propiedad locale del componente `IntlSwingSupport` para el país destino. Por ejemplo:

```
new IntlSwingSupport();  
int response = JOptionPane.showConfirmDialog(frame, localizedMessageString,  
localizedTitleString, JOptionPane.OK_CANCEL_OPTION);
```

**Nota** `IntlSwingSupport` sirve para suministrar apoyo internacional a algunos componentes Swing, pero no para los de dbSwing. Todos los componentes dbSwing ya están completamente internacionalizados.

En JDK 1.2, los únicos componentes Swing con cadenas de texto visibles y traducibles son `JFileChooser`, `JColorChooser` y `JOptionPane`. Para obtener más información sobre las versiones internacionales, consulte la documentación de Sun acerca de la clase `Locale`.

## Utilización de componentes que identifican el país

---

Además de estar completamente incluidos en los recursos, muchos componentes de `JBuilder` cuentan también con un práctico comportamiento acorde con la versión localizada. Por ejemplo, los datos de cadena que se cargan en la columna `Column` de una `JdbTable` por medio de componentes DataExpress para conjuntos de datos `DataSet` se ordenan automáticamente según el sistema de ordenación por defecto de la versión localizada de tiempo de ejecución del usuario. Del mismo modo, la fecha, la hora y los valores numéricos adquieren automáticamente el formato acorde con la versión localizada del usuario.

Por defecto, los objetos heredan la versión localizada de sus contenedores. Por tanto, la configuración localizada de un Conjunto de datos es la que utilizan por defecto las columnas de ese conjunto de datos. También existe la posibilidad de elegir una configuración localizada distinta para cada objeto `Column` del `DataSet`. Esto puede resultar útil si, por ejemplo, cada columna contiene datos que se deben ordenar con arreglo a las normas de una versión localizada distinta. Si desea más información sobre las formas de ordenación acordes con la versión localizada, consulte la documentación de la API de JDK sobre la clase `Collator`.

Si desea más información sobre el formato de tipos de datos acorde con la versión localizada en Java, consulte la documentación de la API de JDK sobre las clases `DateFormat`, `NumberFormat` y `MessageFormat`.

## Los componentes de JBuilder muestran todos los caracteres Unicode

---

Las arquitecturas de componentes que dependen únicamente de los controles visuales homólogos nativos para la presentación de caracteres sólo pueden mostrar el conjunto de caracteres que acepta el homólogo nativo. Dado que los componentes de JBuilder utilizan Java para mostrar caracteres en vez de homólogos nativos, pueden presentar cualquier carácter de Unicode del que se tenga una fuente instalada, independientemente de que ese carácter exista o no en la página de códigos por defecto del sistema operativo.

Para mostrar los caracteres Unicode de una nueva fuente:

- 1 Instale la fuente deseada en el sistema operativo.
- 2 Modifique el archivo `font.properties` de JDK de su versión localizada e indique que la fuente correspondiente a ese carácter está ahora disponible.

Si desea instrucciones sobre la forma de conseguirlo, consulte el tema “Adding Fonts” de la documentación sobre la internacionalización de JDK.

## Funciones de internacionalización del diseñador de interfaz de usuario

---

El diseñador de interfaces de usuario de JBuilder es una valiosa herramienta para la creación y verificación de interfaces gráficas de usuario internacionalizadas. Cuando se añaden a la interfaz gráfica de usuario elementos de texto que se pueden traducir, es posible colocarlos inmediatamente en conjuntos de recursos. El Inspector lee automáticamente las cadenas de los conjuntos de recursos y vuelve a escribirlas. Además, una vez incluido en los recursos todo el texto de la interfaz gráfica de usuario y recibido el grupo de recursos localizado del traductor, se puede utilizar el diseñador para construir y comprobar rápidamente la interfaz de usuario internacionalizada.

El Inspector presenta descripciones breves, acordes con la versión localizada, sobre las propiedades de cada JavaBean, tal y como se describe en la sección de internacionalización de la especificación JavaBeans.

El Inspector permite utilizar secuencias de escape de caracteres Unicode para indicar caracteres que no se pueden introducir directamente desde el teclado con la configuración de país del sistema operativo. Cuando se desea insertar un carácter de Unicode dentro de una propiedad de cadena que se está editando, basta con introducir el valor hexadecimal de la secuencia de escape de Unicode correspondiente a ese carácter, entre corchetes angulares. Por ejemplo, para introducir el carácter japonés de la palabra "montaña" en la etiqueta de un botón, hay que escribir "<5C71>". Si tiene fuentes japonesas instaladas en el sistema y el archivo `font.properties` de JDK está configurado correctamente, el carácter se muestra como etiqueta del botón, y la secuencia de escape de Unicode "\u5C71" aparece en el código fuente.

El diseñador de interfaces de usuario proporciona un excelente apoyo a los gestores dinámicos de diseño, un importante requisito para el diseño de interfaces gráficas del usuario internacionalizadas. Es difícil construir una sola interfaz gráfica de usuario que pueda aceptar muchos idiomas, pero la compatibilidad del diseñador visual con los gestores dinámicos de diseño AWT de Java facilita enormemente esta tarea. Cuando se diseña una interfaz gráfica de usuario con intención de localizarla a más de un idioma, es muy importante *utilizar siempre un gestor dinámico de diseño*.

Observe, por ejemplo, el siguiente cuadro de diálogo, que contiene los botones Aceptar, Cancelar y Ayuda:

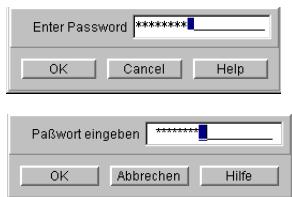


La presentación es la adecuada para las etiquetas en inglés, pero cuando se traducen al alemán, el texto de las etiquetas es demasiado largo para ajustarse al tamaño fijo del botón. Éste es un problema que se da con frecuencia cuando se intenta localizar una IU no internacionalizada.



La solución consiste en utilizar uno o más gestores dinámicos de diseño AWT para permitir que el tamaño de los botones se ajuste al de su etiqueta. Éstas son las versiones inglesa y alemana, internacionalizadas, del mismo cuadro de diálogo, escrito con un panel con GridLayout dinámico

para los botones e incrustados dentro de un cuadro de diálogo BorderLayout.



Para obtener más información sobre la creación de diseños dinámicos mediante el diseñador de interfaces de usuario, consulte el apartado "Utilización de gestores de diseño en *Diseño de aplicaciones con JBuilder*".

La aplicación de ejemplo internacional multilingüe también muestra algunas técnicas avanzadas para actualizar el diseño de Marcos de una aplicación durante la ejecución.

## Unicode en el Depurador IDE

---

El depurador de JBuilder permite ver y editar caracteres de Unicode, incluso en el caso de que el sistema operativo no los acepte. Cuando se examinan los valores del panel de punto de observación del depurador, es necesario ampliar el valor del árbol que se desea inspeccionar hasta ver su tipo Java primitivo. Por defecto, el depurador intenta presentar el carácter de Unicode, dando por supuesto que el sistema operativo lo acepta.

Para ver el equivalente Unicode de un carácter, haga clic con el botón derecho en su valor y seleccione la opción Mostrar valor hexadecimal para ver la secuencia de escape de Unicode correspondiente al carácter. También se puede cambiar el valor seleccionando Modificar valor e introduciendo otra secuencia de escape de Unicode en el cuadro de diálogo Cambiar valor de los datos.

## Elección de una codificación nativa para el compilador

---

Los compiladores JBuilder y **javac** pueden compilar código fuente codificado con codificaciones nativas (también denominadas páginas de códigos locales), que es el formato de almacenamiento que utilizan la mayoría de los editores, como el editor de JBuilder.

El IDE y el compilador aceptan todas las codificaciones nativas de JDK. Todos los compiladores de JBuilder seleccionan automáticamente la codificación nativa correspondiente al país configurado en el sistema operativo. También se puede elegir otra codificación de JDK para compilar archivos de código fuente escritos con una codificación nativa distinta.

Puede especificar un nombre nativo de codificación, para controlar cómo interpreta el compilador los caracteres que no pertenecen al conjunto de caracteres ASCII. Esto se puede hacer en todo el proyecto o con la opción de codificación del compilador, desde la línea de comandos. Si no define un valor en esta opción, se utiliza el convertidor por defecto de codificación nativa de la plataforma.

## Definición de la opción de codificación

---

Para definir la opción de codificación desde el IDE:

- 1 Elija Proyecto | Propiedades de proyecto para abrir el cuadro de diálogo homónimo.
- 2 Haga clic en la pestaña General.
- 3 Seleccione un nombre de codificación en la lista desplegable Codificación.
- 4 Pulse Aceptar.

Para definir la opción de codificación en la línea de comandos:

- 1 Utilice la opción **-encoding** de **bcj** seguida del nombre de codificación.
- 2 Utilice la opción **-encoding** de **bmj** seguida del nombre de codificación.

## Codificaciones nativas admitidas

---

Existen dos nombres de codificaciones que tienen un significado especial:

- **null**

Especifica que no debe realizarse ninguna conversión de codificación nativa. Cada byte del archivo se convierte a Unicode situándolo en el byte inferior del carácter Unicode. El byte superior del carácter Unicode se pone a cero.

- **default**

Equivale a no especificar ninguna opción de codificación. Con esta opción, se utiliza la codificación por defecto del entorno del usuario.

Para obtener una descripción de cada codificación, consulte la JDK Internationalization Specification (Especificación de internacionalización del JDK): Character Set Conversion: (Conversión de conjuntos de caracteres): Supported Encodings (Codificaciones admitidas) en <http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html>. Las descripciones que vienen a continuación aportan más información sobre este tema:

- **Unicode**

Unicode, con bytes más o menos significativos al principio, según indique la marca de orden de bytes.

- **UnicodeBig**

Unicode con bytes más significativos al principio.

- **UnicodeLittle**

Unicode con bytes menos significativos al principio.

## Adición y redefinición de codificaciones

---

Para añadir codificaciones a la lista desplegable Codificación de la ficha Generar del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto):

- 1 Abra el archivo user.properties de la carpeta <.jbuilder>.

- 2 Añada codificaciones, como en el ejemplo siguiente:

```
compiler.java;encodings.add.1=ISO8859_2
compiler.java;encodings.add.2=ISO8859_3
```

- 3 Guarde y cierre el archivo.

- 4 Reinicie JBuilder.

Para sustituir las codificaciones de la lista desplegable Codificación:

- 1 Abra el archivo user.properties de la carpeta <.jbuilder>.

- 2 Redefina las codificaciones, como en el ejemplo siguiente:

```
compiler.java;encodings.override.1=ISO8859_2
compiler.java;encodings.override.2=ISO8859_3
```

- 3 Guarde y cierre el archivo.

- 4 Reinicie JBuilder.

## Otros asuntos relacionados con las codificaciones nativas

---

Los entornos no basados en Unicode representan los caracteres utilizando distintos sistemas de codificación. En el mundo de los PC se denominan páginas de códigos; Java las conoce como codificaciones nativas. Cuando se trasladan datos de un sistema de codificación a otro, es necesario realizar una conversión. Dado que cada sistema puede contar con un conjunto diferente de caracteres extendidos, se necesita una conversión para impedir que se pierda información.

Si lo prefiere, puede emplear la utilidad **native2ascii** de Java para convertir los caracteres de codificación nativa a secuencias de escape de

Unicode (por ejemplo, \uNNNN). Los archivos convertidos se pueden compilar en cualquier sistema, sin necesidad de efectuar más conversiones ni especificar una codificación.

La mayoría de los editores de textos, como el editor de JBuilder, almacenan el texto con la codificación nativa. Por ejemplo, en la versión japonesa de Windows se utiliza el formato Shift-JIS, mientras que en la versión estadounidense se utiliza la página de códigos 1252 de Windows. A partir de JDK 1.1, **javac** también puede compilar código fuente "con codificación nativa". La codificación puede especificarse utilizando el modificador "encoding". Si no se utiliza el modificador "encoding", el compilador utiliza la codificación que corresponde al entorno del usuario.

Al contrario que en el caso de Unicode, el código fuente escrito con una codificación nativa no puede llevarse directamente a sistemas que utilizan otras codificaciones. Por ejemplo, si el código fuente se ha codificado con Shift-JIS (una codificación japonesa) y utiliza el compilador en un entorno de Windows en español, debe especificar la codificación Shift-JIS para que el compilador pueda leer correctamente el código fuente.

## Formato Unicode de 16 bits

---

Unicode es un sistema universal de representación de caracteres con números de 16 bits. El conjunto de caracteres Unicode de 16 bits puede implementarse directamente o puede representarse de forma indirecta dentro del conjunto de caracteres ASCII de 7 bits, utilizando el carácter de escape \u seguido de cuatro dígitos hexadecimales.

Cuando todos los sistemas operativos principales admitan directamente Unicode, se reemplazará el sistema establecido actualmente, que requiere una conversión entre las distintas codificaciones nativas que tengan valores de caracteres conflictivos. Java es uno de los primeros entornos que utilizan la norma Unicode, que es el conjunto de caracteres interno del entorno Java.

### Utilización de Unicode por medio de ASCII y '\u'

---

Actualmente, la mayoría de los editores de texto de Windows, incluido el editor de JBuilder, almacenan y procesan el texto como caracteres de 7 u 8 bytes, en lugar de caracteres de 16 bits de Unicode. El conjunto de caracteres ASCII utiliza una codificación de 7 bits que contiene las 26 letras del alfabeto inglés y algunos símbolos. Casi todas las codificaciones nativas tienen ASCII como un subconjunto y lo representan de la misma forma: los primeros 127 caracteres de la codificación componen el conjunto de caracteres ASCII. El conjunto de caracteres ASCII puede considerarse un subconjunto de Unicode.

Para permitir a los usuarios especificar caracteres de Unicode en su código fuente sin utilizar un editor preparado para Unicode, la especificación de Java permite utilizar el carácter escape \u de Unicode en los archivos ASCII. Esta utilización permite representar caracteres extendidos con una combinación de caracteres ASCII. Esta forma de representar Unicode utiliza 6 caracteres para representar un solo carácter no perteneciente a ASCII. Para introducir un carácter ASCII normal, basta con pulsar en el teclado la tecla que corresponde al carácter; para introducir uno de otro conjunto de caracteres, se escribe la secuencia de escape de Unicode que representa al carácter.

En esta representación de 7 bits de Unicode, los caracteres no pertenecientes al conjunto de caracteres ASCII se representan con la forma \uNNNN, donde NNNN son los 4 dígitos hexadecimales del carácter Unicode. Por ejemplo, el carácter Unicode "f latina minúscula con gancho", una letra 'f' cursiva, que se representa en Unicode con el número hexadecimal 0192, puede introducirse escribiendo "\u0192".

Unicode, tanto en el formato de 16 bits como el de 7 bits, está en formato universal; el código fuente en Unicode se puede trasladar directamente a todas las plataformas, en todos los idiomas.

## JBuilder en el mundo

---

JBuilder está disponible en varios idiomas, entre los que se encuentran el inglés, el alemán, el francés, el español y el japonés. Las versiones localizadas generalmente incluyen traducciones de la documentación, la IU y los componentes. Se pueden encargar versiones localizadas de JBuilder en las delegaciones de ventas de Borland de los siguientes países. Para encontrar enlaces a las sedes web internacionales de Borland, diríjase a Borland Worldwide en <http://www.borland.com/bww/>.

## Asistencia internacional en línea

---

Visite el grupo de noticias sobre aplicaciones plurilingües en la sede Web de Borland en <news://forums.borland.com/borland.public.jbuilder.multi-lingual-apps>. Este grupo de noticias se dedica a asuntos de internacionalización y localización de JBuilder, y está moderado activamente por nuestros ingenieros de servicio técnico, así como por los ingenieros de Investigación y desarrollo y Control de calidad del grupo de internacionalización de JBuilder.



## Tutorial: Tutorial de compilación, ejecución y depuración

Este tutorial es una función de JBuilder SE y Enterprise

En este tutorial aprenderemos, paso a paso, a encontrar y resolver errores de sintaxis, de compilación y de ejecución en un archivo de ejemplo incluido en JBuilder.

- Los errores de sintaxis afloran antes de la compilación. Los errores de sintaxis corresponden a código que no cumple los requisitos sintácticos del lenguaje Java.
- Los errores de compilación son los generados por el compilador: la sintaxis puede ser correcta, pero el compilador no puede compilar el código porque faltan variables o clases o porque hay sentencias incompletas. La verdadera causa del error podría encontrarse en una o más líneas antes o después del número de línea especificado en el mensaje de error.
- Los errores en tiempo de ejecución suceden cuando su programa puede ser compilado satisfactoriamente pero genera excepciones en tiempo de ejecución o se bloquea cuando se le ejecuta. Las sentencias del programa son sintácticamente correctas, pero provocan errores cuando se ejecutan.

El tutorial emplea el proyecto de ejemplo incluido en la carpeta <jbuilder>/samples/Tutorials/DebugTutorial. Se trata de una sencilla calculadora. Se han introducido errores en el programa con el fin de que no se compile ni se ejecute correctamente. Para que llegue a ejecutarse es preciso seguir este tutorial, encontrando y solucionando los errores.

## Paso 1: Abrir el proyecto de ejemplo

**Importante** Es posible que los números de línea mencionados en este tutorial no coincidan con los que se muestran en JBuilder.

Si desea información más detallada sobre la compilación, ejecución y depuración, consulte los siguientes apartados:

- [Capítulo 6, “Generación de programas en Java”](#)
- [Capítulo 5, “Compilación de programas en Java”](#)
- [Capítulo 7, “Ejecución de programas en Java”](#)
- [Capítulo 8, “Depuración de programas en Java”](#)

El apartado Opciones de accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder para personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte “[Convenciones de la documentación](#)” en la página 1-5.

## Paso 1: Abrir el proyecto de ejemplo

---

En este paso, abriremos el archivo de proyectos y uno de los archivos del proyecto. Veremos que hay errores de sintaxis en uno de los archivos.

Para abrir el proyecto de ejemplo,

- 1 Seleccione Archivo | Abrir proyecto. Se abre el cuadro de diálogo Abrir proyecto.
- 2 Desplácese a la carpeta samples/Tutorials/DebugTutorial de la instalación de JBuilder.
- 3 Haga doble clic sobre DebugTutorial.jpx. El proyecto se abre en el panel de proyectos. Los archivos del proyecto se enumeran en el panel de proyectos. Este proyecto se compone de tres archivos:
  - Application1.java - Archivo ejecutable que contiene el método main().
  - DebugTutorial.html - Archivo HTML que ofrece una descripción general del proyecto.
  - Frame1.java - Archivo que contiene el marco, los componentes y los métodos del programa.

- 4 Haga doble clic sobre Frame1.java. De este modo se abre el archivo en el editor y en el panel de estructura aparece su estructura.

Fíjese en la carpeta Errores del panel de estructura. En el Paso 2 del tutorial encontraremos y solucionaremos estos errores.

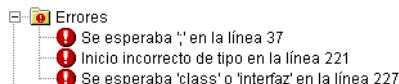
## Paso 2: Solucionar errores de sintaxis

Los errores de sintaxis son situaciones en las que no se cumplen los requisitos sintácticos del lenguaje Java. JBuilder detecta estos errores antes de la compilación. Aparecen en una lista en la carpeta Errores del panel de estructura. Si intenta compilar el programa sin haber solucionado estos errores de sintaxis, JBuilder mostrará los errores en el panel de mensajes. El programa no puede compilarse hasta que se hayan solucionado estos errores.

En este paso, encontraremos los errores de sintaxis del programa de ejemplo y los solucionaremos. Si desea información más detallada sobre los mensajes de error de JBuilder, consulte el tema “Mensajes de error del compilador”.

Para encontrar y solucionar errores de sintaxis:

- 1 Amplíe la carpeta Errores en el panel de estructura.



Aparecen tres errores. El primer error indica que al final de la línea de código falta un punto y coma.

- 2 Haga clic sobre el primer error del panel de estructura. JBuilder desplaza el cursor a la línea de código correspondiente, en el editor. Si se hace clic en el mensaje de error, JBuilder resalta la línea del código correspondiente. Si hace doble clic, el cursor se desplaza a la columna en la que se produjo el error.

### Sugerencia

La barra de estado del panel de contenido muestra el número de línea y de columna, además del modo inserción.



- 3 Inserte un punto y coma al final de la línea. Se ha solucionado el error, y éste desaparece del panel de estructura.
- 4 Haga clic sobre el siguiente error del panel de estructura. JBuilder desplaza el cursor a la línea de código correspondiente, en el editor. Este error es un poco más complicado de descifrar. El mensaje indica que se esperaba la presencia de un identificador de tipo en este punto del programa.

Observe que la línea de código empieza con la palabra clave `else` y que la línea siguiente se compone solamente de una llave de cierre. Si ha leído las líneas de código anteriores, habrá visto el principio de una sentencia `if`. En Java, las sentencias `if` deben incluir llaves de apertura y cierre. Sin embargo, si observa la línea que contiene la sentencia `if`, verá que falta la llave de apertura.

- 5 Añada la llave de apertura al final de la línea. La línea de código completa será la siguiente:

```
if (valueOneOddEven) {
```

Observe cómo la característica de coincidencia de llaves del editor muestra la llave de cierre coincidente.

Los otros dos errores de sintaxis desaparecen del panel de estructura. Ahora aparecen errores diferentes en la carpeta Errores. En el siguiente paso se solucionan estos errores.

- 6 Haga clic en el botón Guardar todo en la barra de herramientas.



Algunas veces hay que actuar como un detective para corregir los errores de sintaxis. A menudo, al solucionar un error de sintaxis se solucionan varios de los que aparecen en el panel de estructura. En nuestro caso, por ejemplo, el tercer error de sintaxis era: se espera 'clase' o 'interfaz' en la línea 227 Debido a que la llave de cierre no tiene una llave de apertura correspondiente, JBuilder espera encontrar una declaración de clase tras el cierre del método actual. Sin embargo, al insertar la llave de apertura, JBuilder detectó que esta llave tenía su pareja y que la siguiente línea de código no contenía ningún error.

**Sugerencia** Puede encontrar las llaves correspondientes moviendo el cursor hasta ellas. La llave correspondiente aparece resaltada.

En el siguiente paso, podrá encontrar y corregir errores que impedirían que este programa se compilara.

## Paso 3: Solucionar errores de compilación

---

En este paso del tutorial, podrá buscar y solucionar errores que podrían impedir la compilación de su programa. Estos errores se muestran en la carpeta Errores del panel de estructura.

Para encontrar y solucionar errores de compilación:

- 1 Haga clic sobre el primer error de la carpeta Errores:

No se ha encontrado el constructor Double() en la clase java.lang.Double en la línea 38

JBuilder coloca el cursor en la línea de código correspondiente:

```
Double valueOneDouble = new Double();
```

El mensaje de error indica que la clase Java `java.lang.Double` no contiene un constructor sin parámetros. La sentencia resaltada intenta crear un objeto `Double` sin parámetro. Si presta atención a los constructores de la clase `java.lang.Double`, verá que todos ellos requieren un parámetro.

Además, si se fija en unas líneas más adelante, verá que el objeto Double, valueTwoDouble, está construido con un valor inicial de 0.0.

#### Sugerencia

Coloque el cursor entre los paréntesis y pulse *Ctrl+Mayús+Espacio* para ver ParameterInsight, la ventana emergente de JBuilder que muestra el tipo de parámetro necesario. También puede hacer clic con el botón derecho sobre el método Double() y seleccionar Buscar definición para abrir el código fuente en el editor.

- Escriba 0.0 entre los paréntesis. Ahora, la sentencia tiene la siguiente forma:

```
Double valueOneDouble = new Double(0.0);
```

#### Sugerencia

La barra de estado del panel de contenido muestra la palabra Modified, indicando así que ha realizado cambios en el archivo.



- Haga clic en el botón Guardar todo en la barra de herramientas.

- Haga clic sobre el siguiente error del panel de estructura:

No se ha encontrado la variable subtractresultDisplay en la clase DebugTutorial.Frame1 en la línea 243

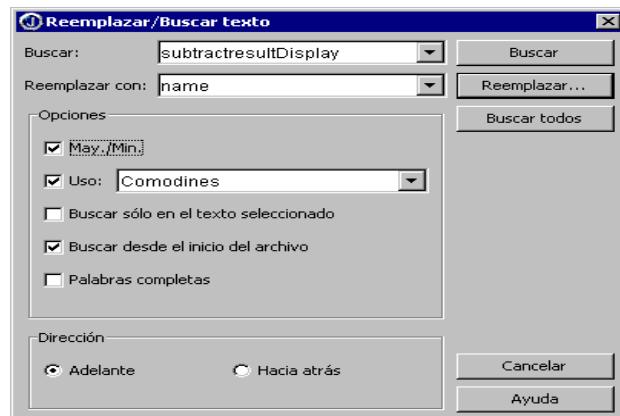
Este error indica que no se ha definido la variable subtractresultDisplay.

- Seleccione Buscar | Buscar para abrir el cuadro de diálogo Buscar / Reemplazar texto.

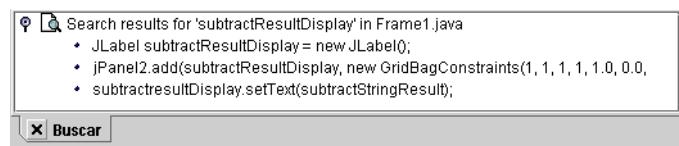
#### Sugerencia

Si el comando Buscar aparece atenuado, haga clic en el editor y vuelva a seleccionar Buscar | Buscar.

- Escriba subtractresultDisplay en el campo Buscar. Asegúrese de que la opción May./Min. está desactivada. Si desea comenzar la búsqueda desde el principio, haga clic sobre la opción Buscar desde el inicio del archivo.



- 7 Haga clic en Buscar todos. Los resultados aparecen en la pestaña Buscar del panel de mensajes.



Observe que dos de las tres referencias a esta etiqueta son `subtractResultDisplay`; hay una R mayúscula en la palabra `Result`. La distinción entre mayúsculas y minúsculas es fundamental en Java: `subtractresultDisplay` no es lo mismo que `subtractResultDisplay`.

- 8 Haga doble clic sobre la referencia incorrecta en la pestaña Buscar para desplazar el cursor hasta la referencia dentro del editor.
- 9 Cambie `subtractresultDisplay` por `subtractResultDisplay`.
- 10 Haga clic en la X de la pestaña Buscar para cerrarla. (También puede hacer clic en la pestaña con el botón derecho y elegir Eliminar la pestaña “Buscar”.)

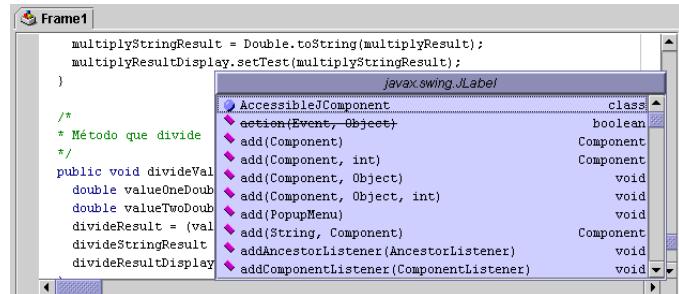
Use CodeInsight para solucionar los demás errores:

- 1 Haga clic sobre los restantes errores de la carpeta. Este error indica que no existe un método `setTest()` en `javax.swing.JLabel`.

Method `setTest(java.lang.String)` not found in class `javax.swing.JLabel` at line 254

JBuilder coloca el cursor en la línea de código correspondiente.

- 2 Sitúe el cursor detrás del punto (.) y pulse *Ctrl+Espacio*. De este modo se abre la ventana emergente CodeInsight que muestra las funciones miembro disponibles.



#### Nota

Si no aparece la ventana emergente, consulte “Configuraciones de teclado para emulaciones del editor” (Ayuda | Configuraciones de teclado) para ver una lista de teclas abreviadas de CodeInsight.

- Desplácese por la ventana utilizando las teclas de desplazamiento. Los elementos resaltados en negrita están en esta clase. Los elementos tachados son desaconsejados. Los elementos atenuados son heredados, pero pueden utilizarse.
- Busque `setText` escribiendo `setText` o desplazándose por la lista. Una vez seleccionado, haga doble clic sobre él o pulse *Intro*. El método `setText()` queda insertado en el editor tras el punto, reemplazando al nombre incorrecto de método `setTest`. Una ayuda inmediata muestra el tipo de parámetros más lógico para el método.



```

Marco1
double valueOneDoubleResult = valueOneDouble.doubleValue();
double valueTwoDoubleResult = valueTwoDouble.doubleValue();
multiplyResult = (valueOneDoubleResult * valueTwoDoubleResult);
multiplyStringResult = Double.toString(multiplyResult);
multiplyResultDisplay.setText(multiplyStringResult);
}

/*
 * Method that divides Value One and Value Two and displays result
 */
public void divideValues(Double valueOneDouble, Double valueTwoDouble)
{
    double valueOneDoubleResult = valueOneDouble.doubleValue();
    double valueTwoDoubleResult = valueTwoDouble.doubleValue();
}

```

**3** Haga clic en el botón Guardar todo en la barra de herramientas.

En el paso siguiente se examina la configuración de ejecución y se ejecuta el programa.

## Paso 4: Ejecución del programa

---

En este paso del tutorial se examina la configuración de ejecución del programa, y a continuación, se ejecuta el programa.

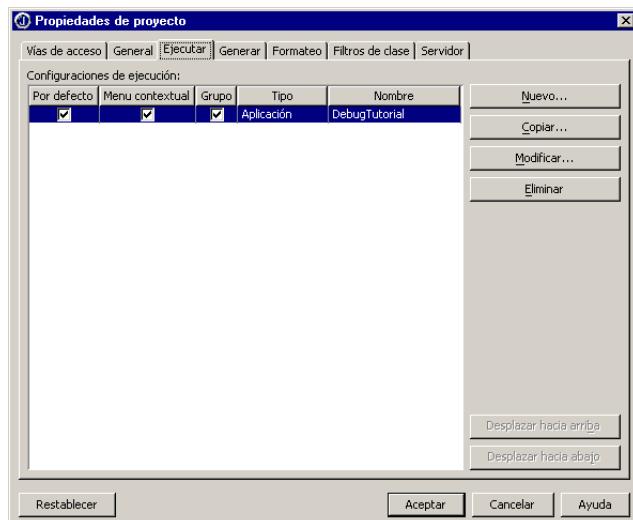
Las configuraciones de ejecución son parámetros de ejecución preestablecidos. La utilización de parámetros prestablecidos ahorra mucho tiempo durante la ejecución y la depuración, porque así sólo se definen una vez. Estas configuraciones prestablecidas permiten, cada vez que ejecute o depure la aplicación, simplemente seleccionar la configuración deseada.

Para obtener más información sobre las configuraciones de ejecución, consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7.

## Paso 4: Ejecución del programa

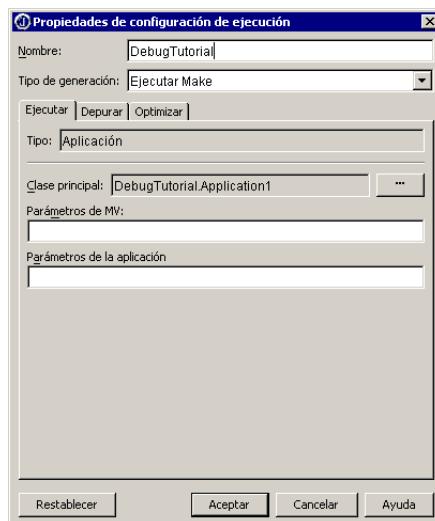
Para verificar la configuración para la ejecución de esta aplicación:

- 1 Seleccione Ejecutar | Configuraciones. A continuación se muestra la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto.



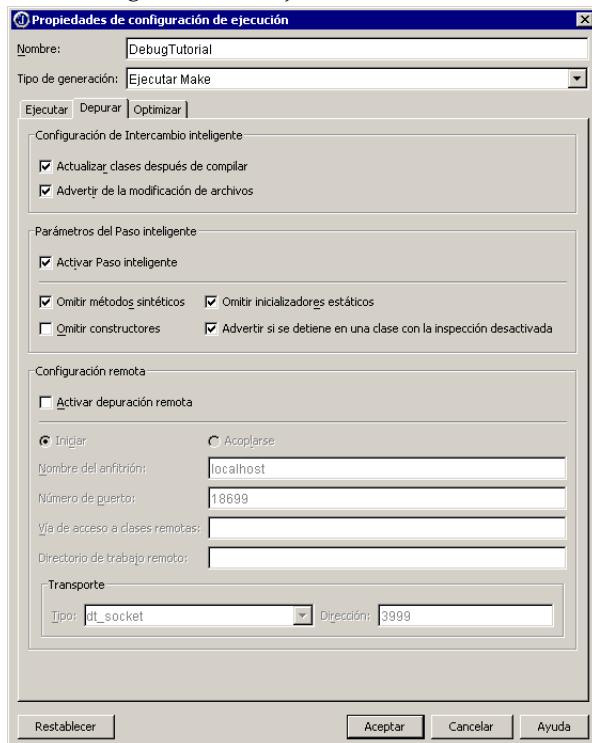
Hay una configuración predeterminada - DebugTutorial. Es una configuración de aplicación, lo que significa que para su ejecución se utiliza un ejecutor de aplicaciones. Es la configuración por defecto, y se mostrará también en el menú contextual cuando haga clic con el botón derecho sobre el archivo ejecutable `Application1.java`.

- 2 Pulse el botón Modificar. Se muestra el cuadro de diálogo Propiedades de configuración de ejecución.



Observe que el valor de Tipo es aplicación y que se utiliza el ejecutor de aplicaciones. El valor de Clase principal es DebugTutorial.Application1.

- Abra la pestaña Depurar para mostrar las propiedades de depuración de la configuración de ejecución. Se muestra la ficha Depurar.



Observe que Paso inteligente se encuentra activado.

- Pulse Aceptar dos veces para cerrar los cuadros de diálogo Propiedades de configuración de ejecución y Propiedades del proyecto.

## Almacenamiento de archivos y ejecución del programa

---

Guarde los cambios y ejecute el programa:



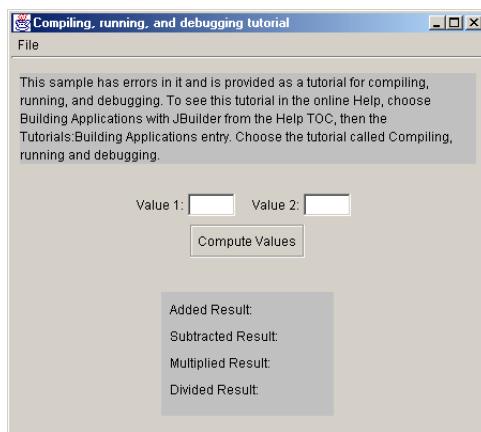
- Haga clic en el botón Guardar todo en la barra de herramientas.



- Haga clic en el botón Ejecutar en la barra de herramientas. El programa se ejecuta utilizando la configuración por defecto, DebugTutorial. La

## Paso 5: Corregir el método subtractValues()

salida del compilador aparece en la ficha Application1, en el panel de mensajes. Aparece la interfaz de usuario del programa.



- 3 Introduzca los valores en los campos de introducción de datos del programa Value 1 y Value 2. Pulse el botón Compute values. Los valores se calculan y aparecen en pantalla. Sin embargo, si observa con atención los resultados obtenidos, verá que hay algunos errores de ejecución; el programa se compila y ejecuta, pero da resultados incorrectos. En los siguientes pasos encontraremos y solucionaremos estos errores.
- 4 Elija File | Exit para cerrar la aplicación.
- 5 Haga clic sobre la X en la pestaña Aplicación1 del panel de mensajes para cerrarlo.

En el siguiente paso, podrá buscar y corregir un error de ejecución.

## Paso 5: Corregir el método subtractValues()

En este paso del tutorial, encontraremos y solucionaremos uno de tres errores de ejecución. Para encontrar este error, nos serviremos del depurador. Aprenderá cómo:

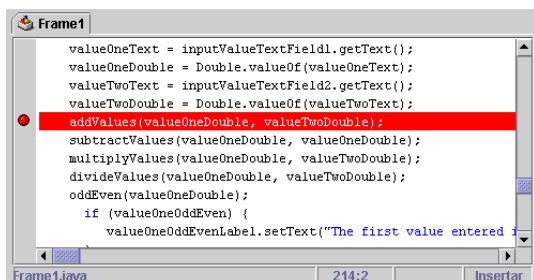
- Poner en marcha y detener el depurador.
- Crear una ventana flotante para una de las vistas del depurador.
- Definir un punto de interrupción.
- Inspeccionar un método y omitir su inspección.
- Seguir el rastro de un hilo.
- Definir un punto de observación `this`, un punto de observación de objeto y un punto de observación de una variable local.

- Utilizar el cuadro de diálogo Evaluar/Modificar.

En el paso anterior, se ejecutó el programa. Al introducir valores en los campos de introducción de datos Value 1 y Value 2 y pulsar Compute Values para obtener los valores de la suma, resta, multiplicación y división, quizás se diera cuenta de que el valor de la resta no era correcto. Por ejemplo, si introduce 4 en el campo Value 1 y 3 en el campo Value 2, el resultado de la resta es 0.0 en lugar de 1.0.

Para encontrar este error, utilizaremos el depurador. En primer lugar, fijaremos un punto de interrupción e iniciaremos el depurador.

- 1 Utilice el cuadro de diálogo Buscar/Reemplazar texto (Buscar | Buscar) para encontrar la línea de código que llama al método addValues(). Este es el primer método al que se llama cuando se pulsa el botón Compute Values. Escriba addValues en el campo Buscar del cuadro de diálogo para encontrar la llamada al método. Pulse el botón Buscar.
- 2 Haga clic en el margen gris del editor que está a la izquierda de la línea de código. En esta línea se fija un punto de interrupción. El círculo rojo indica que no se ha verificado el punto de interrupción.



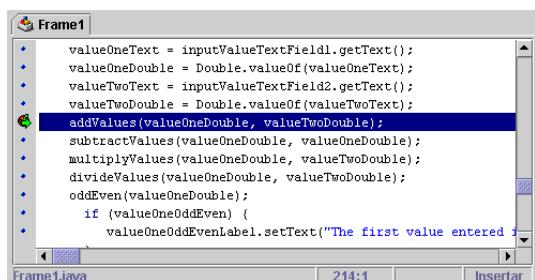
- 3 Haga clic en el botón Depurar programa en la barra de herramientas. JBuilder inicia el depurador MV, utilizando la configuración de ejecución de DebugTutorial.

El programa se está ejecutando y espera la entrada de datos (esto puede tardar unos segundos).

- 4 Escriba 4 en el campo Value 1 y 3 en el campo Value 2. Pulse Compute Values. Antes de que pueda examinar los resultados, el depurador toma el control. El programa queda minimizado y en el panel de mensajes aparece el depurador. En el editor aparecen ahora unos puntos azules junto a las líneas de código ejecutables, que indican dónde pueden fijarse puntos de interrupción válidos. El icono para el punto de interrupción que acaba de fijar ha cambiado a un punto rojo con una marca de comprobación verde que indica que el punto de interrupción es válido. La flecha indica el punto de ejecución (en este

## Paso 5: Corregir el método subtractValues()

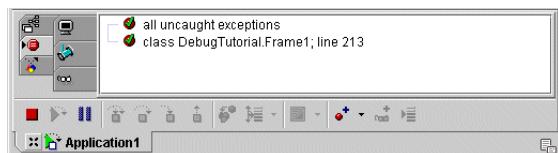
caso, la línea con punto de interrupción es también el punto de ejecución).



```
Frame1.java
Frame1.java 214:1 Insertar
public class Frame1 {
    ...
    valueOneText = inputValueTextField1.getText();
    valueOneDouble = Double.valueOf(valueOneText);
    valueTwoText = inputValueTextField2.getText();
    valueTwoDouble = Double.valueOf(valueTwoText);
    addValues(valueOneDouble, valueTwoDouble);
    subtractValues(valueOneDouble, valueOneDouble);
    multiplyValues(valueOneDouble, valueTwoDouble);
    divideValues(valueOneDouble, valueTwoDouble);
    oddEven(valueOneDouble);
    if (valueOneOddEven) {
        valueOneOddEvenLabel.setText("The first value entered is odd");
    }
}
```

Si desea obtener información sobre la interfaz de usuario del depurador, consulte ["Interfaz del depurador"](#) en la página 8-9.

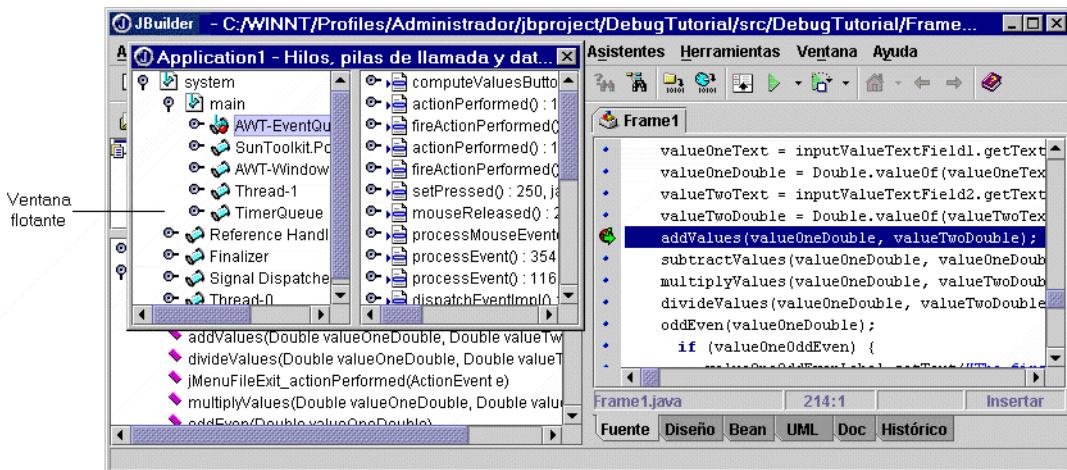
- 5 Haga clic en la pestaña Puntos de interrupción situada en la parte izquierda del depurador para ir a la vista Puntos de interrupción de datos y código. En pantalla aparecen el punto de interrupción por defecto y el punto de interrupción que acaba de fijar. La barra de estado del depurador muestra un mensaje que indica que el programa se ha detenido en el punto de interrupción establecida en el editor.



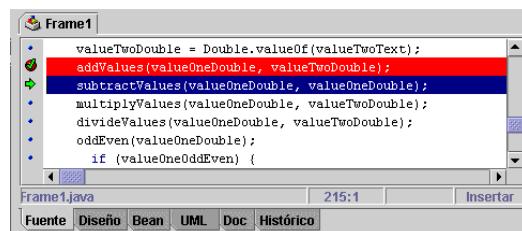
El siguiente paso será inspeccionar el hilo que se está ejecutando paso a paso. Esto nos permitirá ver desde dónde se llama a los métodos y fijar puntos de observación sobre esos métodos.

- 1 Abra la vista, Hilos, pilas de llamadas y datos. Observe que la vista está dividida, lo cual permite ver el contenido del elemento seleccionado en el panel izquierdo del panel derecho.
- 2 Haga clic con el botón derecho en un área vacía del panel izquierdo de la vista y seleccione Ventana flotante. La vista se convierte en una ventana flotante e inicialmente aparece en la parte superior izquierda de la pantalla. Puede cambiar las dimensiones de la ventana y moverla. Transformar una vista en ventana nos permite tener más de una vista del depurador simultáneamente. (Observe que todas las vistas pueden

transformarse en ventanas flotantes, excepto la vista Salida, entrada y errores de consola.)



- 3** Desplácese por el editor para ver simultáneamente el punto de interrupción y la ventana flotante.
- 4** También podría haber fijado el punto de interrupción en la llamada al método `subtractValues()` en lugar de en el método `addValues()`, lo cual le hubiera aproximado más a la parte del programa que realmente desea examinar con mayor detenimiento. Para esto, pulse el botón Omitir inspección de la barra de herramientas del depurador. De este modo se omite la inspección de la llamada al método `addValues()`, quedando el punto de ejecución en la llamada al método `subtractValues()`.

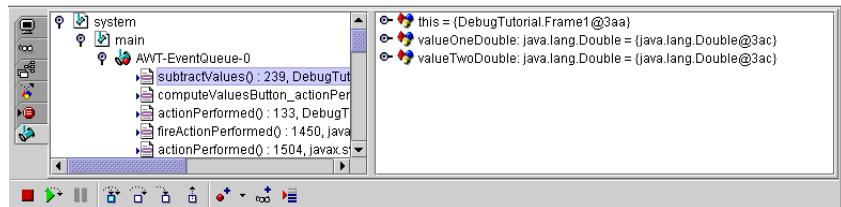


- 5** Pulse el botón Inspeccionar para inspeccionar el método `subtractValues()`. En estos momentos, el método `subtractValues()` aparece resaltado en el panel izquierdo de la vista flotante Hilos, pilas de llamadas y datos.
- 6** Haga clic con el botón derecho en un área vacía del panel izquierdo de la vista flotante Hilos, pilas de llamadas y datos y desactive la selección Ventana flotante para cerrarla. La ventana flotante vuelve a mostrarse como vista del depurador.

## Paso 5: Corregir el método subtractValues()

**Sugerencia** Si desea devolver a las pestañas del depurador su orden por defecto, haga clic con el botón derecho en un área vacía de la vista y seleccione Restablecer orden predeterminado.

- 7 Abra la vista Hilos, pilas de llamadas y datos. Observe que el método subtractValues() aparece ampliado en el panel derecho de la vista.



**Sugerencia** Puede pulsar el botón Mostrar marco de la pila actual para mostrar el hilo que se está inspeccionando.

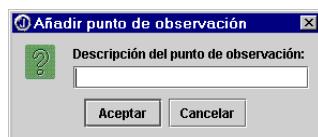
El siguiente paso será fijar puntos de observación sobre objetos y variables. Esto nos permitirá examinar los valores de los datos.

- 1 Cree un punto de observación sobre el objeto `this` haciendo clic con el botón derecho sobre el objeto `this` en la lista desplegada:

```
this = {DebuggerTutorial.Frame1@3c6}
```

Seleccione el comando Crear punto de observación 'this'. Un punto de observación sobre el objeto `this` permite inspeccionar la instancia actual de la clase.

- 2 Se muestra el cuadro de diálogo Añadir punto de observación con el campo Descripción del punto de observación. Pulse Aceptar.



No es necesario introducir una descripción del punto de observación. Si se introduce una descripción, aparecerá en la misma línea que la expresión con punto de observación de la vista Puntos de observación de datos. Una descripción puede facilitar la localización de los puntos de observación en la vista.

- 3 Vuelva a hacer clic con el botón derecho sobre el objeto `this`:

```
this = {DebuggerTutorial.Frame1@3c6}
```

Seleccione el comando Crear punto de observación de objetos. Se muestra el cuadro de diálogo Añadir punto de observación. Pulse Aceptar.

- 4 Haga clic con el botón derecho sobre el objeto `valueOneDouble` de la lista desplegada (en el panel derecho), con el fin de crear un punto de

observación sobre el primer valor que se pasa al método subtractValues():

```
valueOneDouble: java.lang.Double. = {java.lang.Double@3c7}
```

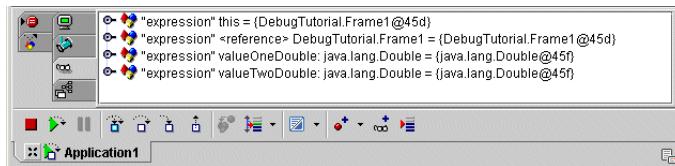
Seleccione el comando Crear punto de observación de variable local. Se muestra el cuadro de diálogo Añadir punto de observación. Pulse Aceptar.

- 5 Haga clic con el botón derecho sobre el objeto valueTwoDouble de la lista desplegada, con el fin de crear un punto de observación sobre el segundo valor que se pasa al método:

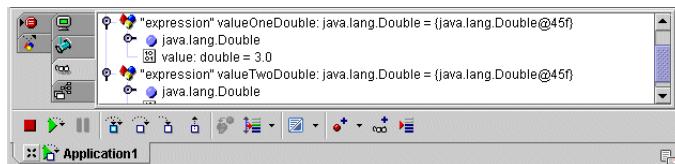
```
valueTwoDouble: java.lang.Double. = {java.lang.Double@3c7}
```

Seleccione el comando Crear punto de observación de variable local. Se muestra el cuadro de diálogo Añadir punto de observación. Pulse Aceptar.

- 6 Abra la vista Puntos de observación de datos.



- 7 Amplíe los dos primeros puntos de observación: el punto de observación `this` y el punto de observación `<reference>`. En este caso, ambos puntos de observación proporcionan los mismos datos, ya que son puntos de observación idénticos. Observe que en esta vista se pueden ver todos los datos de los objetos (excepto los datos estáticos). Los elementos atenuados son heredados. Contraiga estos dos puntos de observación. Los otros dos puntos de observación (puntos de observación de variable local) observan los valores de `valueOneDouble` y `valueTwoDouble`.
- 8 Pulse el botón Inspeccionar para inspeccionar el método `subtractValues()`.
- 9 Despliegue los puntos de observación sobre `valueOneDouble` y `valueTwoDouble`.



Los dos valores son iguales. Sin embargo, no introduje dos valores iguales en los dos campos de introducción de datos del programa.

- 10 Fije un punto de observación sobre `subtractStringResult`, el resultado de la resta. Este valor de tipo `String` aparece en la etiqueta de salida. Para

-  fijar el punto de observación, haga clic en el botón Añadir punto de observación de la barra de herramientas del depurador y escriba `subtractStringResult` en el campo Expresión. Pulse Aceptar. Quizá tenga que desplazarse por la vista Puntos de observación de datos para ver el punto de observación.
  -  11 Pulse el botón Inspeccionar tres veces para pasar a la siguiente línea en el editor:

```
subtractResultDisplay.setText(subtractStringResult)
```

En la vista Puntos de observación de datos, el valor de `subtractStringResult` es `0.0` en lugar de `1.0`, como sería de esperar.
- Nota** También puede utilizar el cuadro de diálogo Evaluar/Modificar para examinar el valor de `subtractStringResult`. Para ello, seleccione Ejecutar | Evaluar/Modificar. Escriba `subtractStringResult` en el campo de introducción de datos Expresión y haga clic en Evaluar. En el campo Resultado aparece el resultado de la evaluación. Observe que la imagen en pantalla es igual que si se ampliara el punto de observación. Haga clic en Cerrar para cerrar el cuadro de diálogo.
- 12 Haga clic sobre el método dos veces más. El punto de ejecución vuelve a la línea desde la que se llama al siguiente método, `multiplyValues()`.
  - 13 Fíjese en la llamada al método `subtractValues()`, la línea situada antes del punto de ejecución. Observe que `valueOneDouble` se pasa dos veces, en lugar de pasar `valueOneDouble` y `valueTwoDouble`. Cambie el segundo parámetro a `valueTwoDouble`.

## Almacenamiento de archivos y ejecución del programa

Guarde los cambios y ejecute el programa:

-  1 Haga clic en el botón Guardar todo en la barra de herramientas.
-  2 Haga clic en el botón Reiniciar el programa de la barra de herramientas del depurador.
-  3 Haga clic en el botón Ejecutar en la barra de herramientas. Introduzca los valores. Se ejecuta el programa. Al introducir los valores y pulsar el botón Compute Values, el valor de la resta que ahora aparece es correcto. Sin embargo, si observa detenidamente los demás resultados, verá que el resultado de la división también es incorrecto. Prosiga con el Paso 6 para encontrar y solucionar el error.
- 4 Salga del programa. Elimine las pestañas del panel de mensajes. Para ello, haga clic con el botón derecho en la pestaña Application1 y seleccione Eliminar “Application1”.

En el siguiente paso, se busca y corrige otro error de ejecución.

## Paso 6: Corregir el método divideValues()

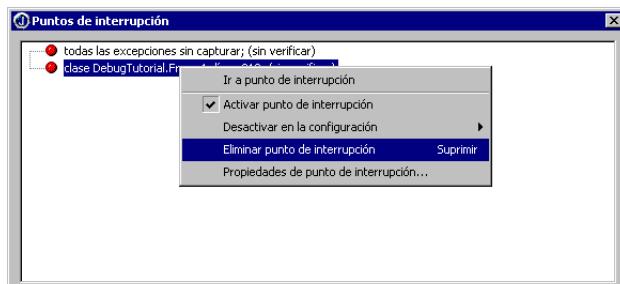
En este paso del tutorial, encontraremos y solucionaremos otro de los tres errores de ejecución. Fijaremos un punto de interrupción, inspeccionaremos un método y aprenderemos a utilizar la ayuda inmediata y ExpressionInsight para localizar errores.

En el paso anterior, se encontró y solucionó un error en la llamada al método `subtractValues()`. En este momento, al volver a ejecutar el programa, quizás se dé cuenta de que el resultado de la división también es incorrecto. Por ejemplo, si escribe `4` en el campo Value 1 y `2` en el campo Value 2, el resultado de la división es `8.0` en lugar de `2.0`.

En primer lugar fijaremos un punto de interrupción, inspeccionaremos el método dudoso y utilizaremos ExpressionInsight y la ayuda inmediata.

- 1 Seleccione Ejecutar | Ver puntos de interrupción para eliminar el punto de interrupción fijado en el Paso 5. En el cuadro de diálogo Puntos de interrupción, haga clic con el botón derecho en este punto de interrupción:

```
clase DebugTutorial.Frame1; línea 213; (sin verificar)
```



Seleccione Eliminar punto de interrupción y cierre el cuadro de diálogo pulsando Cerrar.

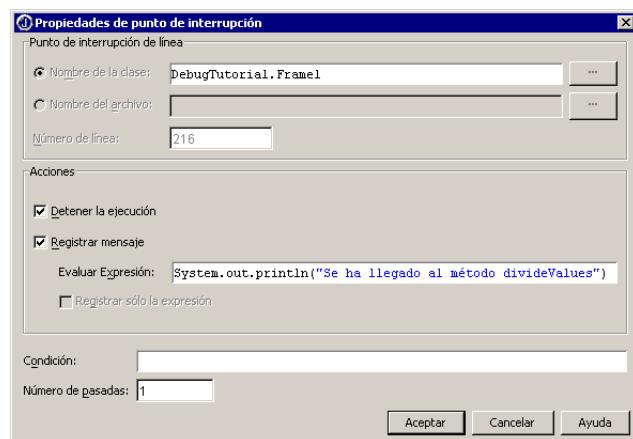
- 2 Utilice el cuadro de diálogo Buscar/Reemplazar texto para localizar la llamada al método `divideValues()`.
- 3 Fije un punto de interrupción en esta línea. Haga clic con el botón derecho en una línea con punto de interrupción, y elija Propiedades de punto de interrupción para abrir el cuadro de diálogo Propiedades de punto de interrupción.
- 4 Seleccione la opción Registrar mensaje. En el campo de introducción de datos Evaluar expresión escriba:

```
System.out.println("Se ha llegado al método divideValues")
```

El mensaje quedará escrito en la vista Salida, entrada y errores de consola cuando se llegue al punto de interrupción especificado. Si

## Paso 6: Corregir el método divideValues()

además se selecciona la opción Detener la ejecución, el programa se detendrá. El cuadro de diálogo tendrá un aspecto parecido a éste:



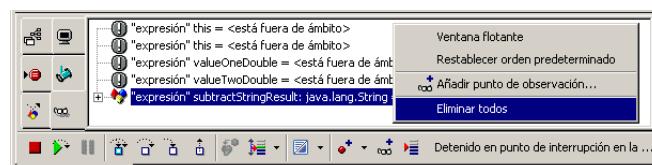
Pulse Aceptar para cerrar el cuadro de diálogo.

- 5 Haga clic en el botón Depurar de la barra de herramientas principal.
- 6 Escriba 4 en el cuadro Value 1 y 2 en el cuadro Value 2 cuando aparezca en pantalla la interfaz de usuario del programa. Pulse Compute Values. Recuerde que antes de que el usuario pueda examinar los resultados, el depurador toma el control y se muestra en el panel de mensajes.
- 7 Abra la vista Salida, entrada y errores de consola. En pantalla aparece este mensaje:

```
Alcanzado punto de interrupción en la clase DebugTutorial.Frame1 en la línea  
216  
Expresión del histórico: System.out.println("Se ha llegado al método  
divideValues") = void
```

Durante el ciclo de desarrollo, puede utilizar esta característica en lugar de añadir sentencias `println` en el código.

- 8 Abra la vista Puntos de observación de datos y código. Observe que la mayoría de los puntos de observación ya están fuera de ámbito. Haga clic con el botón derecho del ratón en una zona vacía de la vista y seleccione Eliminar todos.

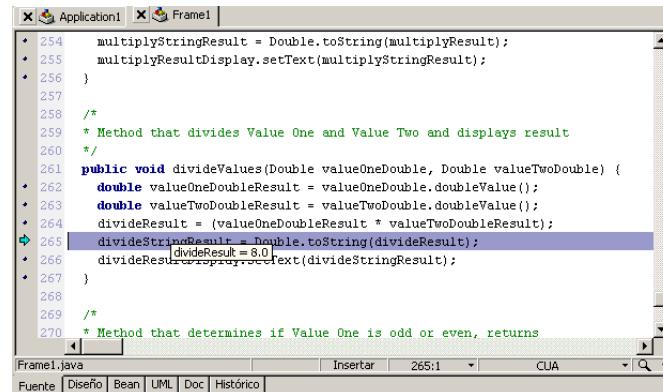


- 9 Pulse el botón Inspeccionar para inspeccionar el método `divideValues()`.

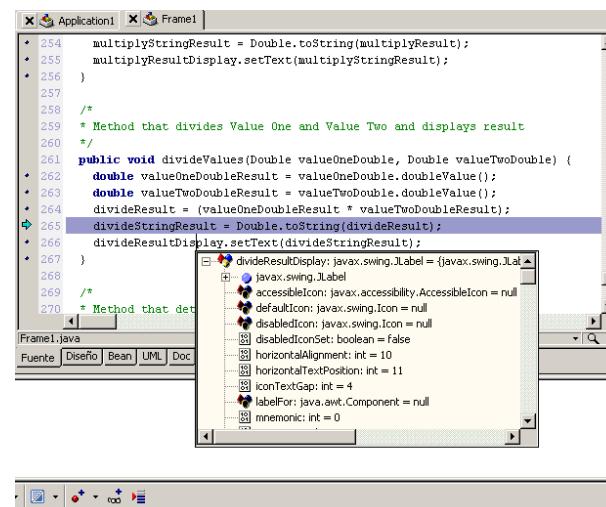
**10** Pulse el botón tres veces más para superar la línea:

```
divideResult = (valueOneDoubleResult * valueTwoDoubleResult)
```

**11** Coloque el cursor sobre la variable divideResult en el editor. En pantalla aparece una ayuda inmediata que muestra el valor de divideResult. Observe que el valor es incorrecto. Conforme a los datos introducidos, el resultado debería ser 2.0. Sin embargo, es 8.0.



También puede pulsar la tecla *Ctrl* y al mismo tiempo hacer clic con el botón derecho del ratón para abrir ExpressionInsight. Esta ventana emergente muestra el nombre, tipo y valor de la expresión. Si la expresión es un objeto, puede examinar los miembros del objeto, y también puede utilizar el menú emergente para fijar puntos de observación, modificar valores y cambiar los valores iniciales de presentación. Por ejemplo, coloque el cursor sobre divideResultDisplay, en la siguiente línea de código. Pulse *Ctrl* junto con el botón derecho del ratón. Aparecerán los miembros del objeto *JLabel*. Desplácese por la lista: los elementos atenuados son heredados.



Haga clic en el editor para cerrar la ventana ExpressionInsight. La ventana también se cerrará automáticamente al mover el cursor.

- 12** Lea detenidamente la línea del código fuente inmediatamente anterior al punto de ejecución:

```
divideResult = (valueOneDoubleResult * valueTwoDoubleResult)
```

¿Detecta el error? El método `divideResult()` multiplica los valores en lugar de dividirlos.

- 13** Para solucionar el error, cambie el operador `*` por `/`.

## Almacenamiento de archivos y ejecución del programa

---

Guarde los cambios y ejecute el programa:

- 1** Elimine el punto de interrupción en el editor.
- 2** Haga clic en el botón Guardar todo en la barra de herramientas.
- 3** Haga clic en el botón Reanudar el programa de la barra de herramientas del depurador.
- 4** Haga clic en el botón Ejecutar en la barra de herramientas. Introduzca valores en los campos Value 1 y Value 2. El programa se ejecuta y ahora el valor de la división es correcto. Sin embargo, si observa detenidamente los demás resultados, verá el último error. Si en el campo Value 1 escribe un número impar, el programa informa erróneamente de que el valor es par. Si escribe un valor par, el programa dice que es impar.
- 5** Salga del programa. Elimine la pestaña Application1 del panel de mensajes.

En el paso siguiente, se busca y corrige el último error de ejecución de este tutorial.

## Paso 7: Corregir el método oddEven()

---

En este paso del tutorial, encontraremos el último de los tres errores de ejecución. Utilizaremos el cuadro de diálogo Evaluar/Modificar para evaluar la llamada a un método, inspeccionar y omitir la inspección de un método, fijar un punto de observación y modificar un valor booleano sobre la marcha para probar una teoría.

En el Paso 6, se solucionó un error en el método `divideValues()`. Ahora, al volver a ejecutar el programa, observará que la sentencia que indica si el primer valor es par o impar es incorrecta.

Por ejemplo, si en el campo Value 1 escribe 4, el programa informa que es un número impar. Sin embargo, si escribe 3, el programa dice que el valor es par. En este paso, encontraremos y solucionaremos este error.

Encontraremos este error utilizando el cuadro de diálogo Evaluar/Modificar para evaluar el método que determina si el número es par o impar. A continuación fijaremos un punto de observación en el resultado dado por el método para ver si se muestra en pantalla correctamente.

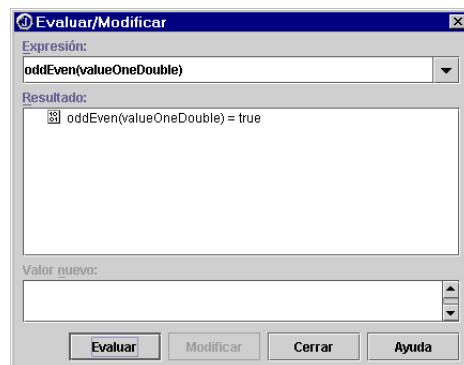
- 1 Utilice el cuadro de diálogo Buscar/Reemplazar texto para localizar la llamada al método `oddEven()` en `Frame1.java`. Observe que el nombre de la variable también contiene el texto `OddEven`. Para encontrar el método, puede activar la opción May./Min. en el cuadro de diálogo o buscar: `oddEven()`
- 2 Fije un punto de interrupción en esta línea:  

```
oddEven(valueOneDouble);
```
- 3 Pulse el botón Depurar.
- 4 Escriba 3 en el cuadro Value 1 y 4 en el cuadro Value 2 cuando aparezca en pantalla la interfaz de usuario del programa. Pulse el botón Compute Values. El foco vuelve al depurador.
- 5 Seleccione Ejecutar | Evaluar/Modificar para abrir el cuadro de diálogo Evaluar/Modificar.

#### Sugerencia

También puede hacer clic con el botón derecho en el editor y seleccionar Evaluar/Modificar.

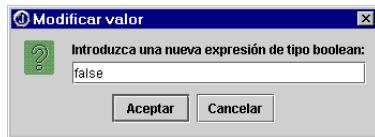
Escriba `oddEven(valueOneDouble)` en el cuadro Expresión. Pulse sobre Evaluar. El método devuelve `true`.



Cierre el cuadro de diálogo Evaluar/Modificar. A continuación inspeccionaremos el método para evaluar qué quiere decir el valor `true`.

- 6 Abra la vista Puntos de observación de datos. Fije un punto de observación sobre `valueOneIsOdd`.

- 7 Pulse el botón Inspeccionar de la barra de herramientas del depurador. Al inspeccionar el método `oddEven()`, el valor de `valueOneIsOdd` es `true`, porque el valor se inicializó como `true` (verdadero). (Para ver la inicialización, busque `boolean valueOneIsOdd` por medio del cuadro de diálogo Buscar texto. Seleccione Ejecutar | Mostrar punto de ejecución para volver a la posición del cursor.)
- 8 Haga clic en Inspeccionar tres veces más para avanzar en la inspección del método. Este método determina si el valor es par o impar. Al ir inspeccionando, el valor de `valueOneIsOdd` sigue siendo verdadero, `true`. ¿Es correcto? ¿El resultado de  $(3 \text{ módulo } 2)$  es igual a cero? En realidad no es igual a cero, y el valor de `valueOneIsOdd` debería ser `false`.
- 9 Para probar esta teoría, haga clic con el botón derecho sobre `valueOneIsOdd` en la vista Puntos de observación de datos y seleccione Modificar valor. Se muestra el cuadro de diálogo Modificar valor. Escriba `false` y haga clic en Aceptar. Ahora el valor de `valueOneIsOdd` es `false`. Acabamos de cambiar el valor devuelto por el método, de `true` a `false`.

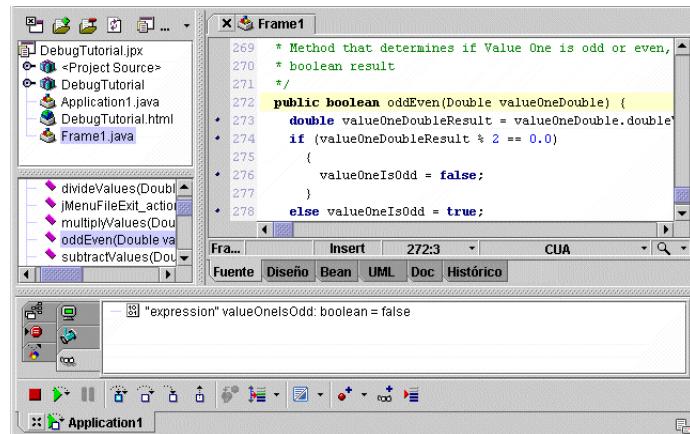


Pulse Aceptar para cerrar el cuadro de diálogo.

- 10 Haga clic en Abandonar inspección para salir del método, volver al punto en el cual se realizó la llamada y, a continuación, seleccionar Inspeccionar con el fin de inspeccionar la sentencia `if` de la siguiente línea de código.
- 11 Examine el contenido de la sentencia `if`. Es muy sencillo:

Si `valueOneIsOdd` es `true`, imprima el mensaje que dice que el número es par. Sin embargo, si el valor es `false`, imprima el mensaje que dice que el número es impar.
- 12 Haga clic de nuevo en el botón Inspeccionar. El punto de ejecución se desplaza a la sentencia `else`, la línea que dice: "Si el valor de `valueOneIsOdd` es falso, imprima el mensaje que indica que el número es impar."

- 13** Haga clic sobre el método `oddEven()` en el panel de estructura para llegar a la ubicación del método en el editor. (Quizá tenga que desplazarse por el panel de estructura para localizar el método.)



- 14** Examine la operación módulo y sus resultados. ¿Están asignados correctamente los resultados `true/false`? Si observa detenidamente, se dará cuenta de que los valores asignados `true` y `false` en realidad están trastocados. El código afirma que si el módulo es igual a cero, el valor devuelto es `false` y el número es impar. Si el módulo no es igual a cero, el valor devuelto es `true` y el número es par. Estas sentencias deberán invertirse de manera que queden como sigue:

```

if (valueOneDoubleResult % 2 == 0.0)
{
    valueOneIsOdd = true;
}
else valueOneIsOdd = false;

```

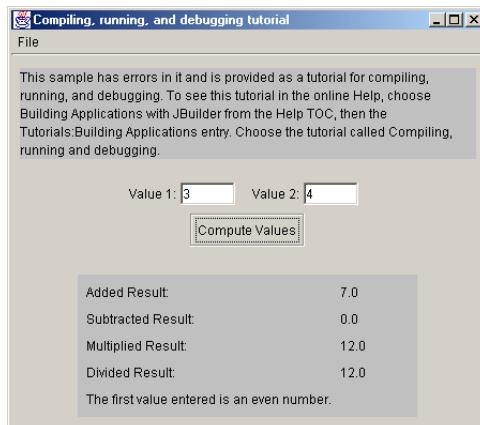
- 15** Intercambie los valores `true` y `false`.

Guarde los cambios y ejecute el programa:

- 1** Guarde los archivos.
- 2** Haga clic en el botón Reanudar el programa de la barra de herramientas del depurador.
- 3** Ejecute de nuevo la aplicación.

## Paso 8: Buscar excepciones producidas durante la ejecución

- 4 Escriba 3 en el cuadro Value 1 y 4 en el cuadro Value 2. Pulse el botón Compute Values. El resultado es correcto. Ahora el programa afirma que el Value 1 es un número impar.



- 5 Salga del programa seleccionando File | Exit. Elimine la pestaña Application1.

En el siguiente paso, veremos qué sucede cuando se genera una excepción durante la ejecución.

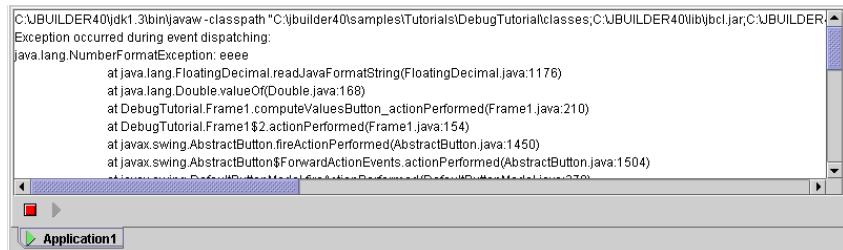
## Paso 8: Buscar excepciones producidas durante la ejecución

En este paso del tutorial, veremos lo que sucede cuando se genera una excepción durante la ejecución. El programa de ejemplo no cuenta con tratamiento de errores. Por ejemplo, si en los campos Value 1 y Value 2 se escribe un valor no numérico, el programa genera un seguimiento de la pila de excepciones producidas durante la ejecución. No informará que el valor no tiene el formato esperado ni facilitará información acerca de cuáles son los valores válidos.

Para ver un seguimiento de la pila de excepciones producidas durante la ejecución,

- 1 Ejecute la aplicación.
- 2 Escriba `eeee` en el campo de introducción de datos Value 1. Escriba 3 en el campo de introducción de datos Value 2. Pulse Compute Values.
- 3 Minimice el programa para ver el panel de mensajes.

La pestaña Application1 contiene un seguimiento de la pila NumberFormatException. Esto es un seguimiento de cómo el programa ha llegado a esta excepción.



```
C:\JBUILDER40\jdk1.3bin\javaw -classpath "C:\jbuilder40\samples\Tutorials\DebugTutorial\classes;C:\JBUILDER40\lib\jbc1.jar;C:\JBUILDER40\lib\jbc2.jar" Application1
Exception occurred during event dispatching:
java.lang.NumberFormatException: eeee
    at java.lang.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:1176)
    at java.lang.Double.valueOf(Double.java:168)
    at DebugTutorial.Frame1.computeValuesButtonActionPerformed(Frame1.java:210)
    at DebugTutorial.Frame1$2.actionPerformed(Frame1.java:154)
    at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:1450)
    at javax.swing.AbstractButton$ForwardActionEvents.actionPerformed(AbstractButton.java:1504)
```

- 4 Seleccione el primer nombre de clase subrayado del seguimiento de la pila para ver dónde se generó la excepción. En este caso, seleccione FloatingDecimal.

JBuilder abre el código fuente de `java.lang.FloatingDecimal` y resalta la línea del código donde se encuentra la excepción. Puede hacer clic en otras clases del seguimiento de la pila para examinar los pasos que llevaron al programa a esta excepción.

El tratamiento de esta excepción excede del ámbito de este tutorial. Para volver a ejecutar el programa sin la excepción, cierre el programa y ejecútelo de nuevo escribiendo valores numéricos.

¡Enhorabuena! Ha llegado al final de este tutorial. Hemos encontrado y solucionado errores de sintaxis, errores de compilación y errores de ejecución con el depurador integrado de JBuilder. También hemos visto un ejemplo de un seguimiento de la pila de excepciones producidas durante la ejecución.

Si desea información más detallada sobre la compilación, ejecución y depuración, consulte los siguientes apartados:

- [Capítulo 6, “Generación de programas en Java”](#)
- [Capítulo 5, “Compilación de programas en Java”](#)
- [Capítulo 7, “Ejecución de programas en Java”](#)
- [Capítulo 8, “Depuración de programas en Java”](#)



# 18

## Tutorial: Generación con archivos Ant

En este tutorial se utilizan funciones de JBuilder Enterprise

Este tutorial explica cómo trabajar con archivos de generación Ant para generar proyectos. Ant es una herramienta de generación basada en Java que construye proyectos según se especifica en los archivo de generación XML. Los archivos de generación definen los tipos y tareas de generación. Por ejemplo, un archivo de generación podría contener tipos de generación separados para generar un proyecto y crear Javadoc. Puede ejecutar tipos de generación individuales o el tipo por defecto del proyecto utilizando el archivo de generación Ant.

JBuilder reconoce automáticamente los archivos de generación Ant llamados build.xml y muestra estos nodos con un icono Ant en lugar del icono XML habitual. También puede utilizar el Asistente para Ant para importar archivos Ant, sea cual sea su nombre. Los tipos del archivo build.xml se muestran como nodos descendientes.

En este tutorial, se completan las siguientes tareas:

- Creación de un proyecto y una aplicación.
- Creación de un archivo de generación Ant.
- Ejecución de tipos de generación Ant.
- Ejecución del tipo Ant por defecto.
- Cómo introducir un error en un archivo fuente Java y examinar los mensajes de error Ant.
- Cómo añadir un tipo al menú Proyecto.
- Modificación de propiedades Ant.
- Cómo añadir bibliotecas Ant personalizadas.

### Consulte

- “Generación con archivos Ant externos” en la página 6-9
- El proyecto Jakarta en Apache: <http://jakarta.apache.org/ant>
- Documentación sobre Ant en el directorio de JBuilder extras/ant/docs/

El apartado Opciones de accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder para personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte “[Convenciones de la documentación](#)” en la página 1-5.

## Paso 1: Crear un proyecto y una aplicación

---

En este paso, utilizará los asistentes de JBuilder para crear un proyecto y una aplicación.

- 1 Elija Archivo | Nuevo proyecto para iniciar el Asistente para proyectos.
- 2 Escriba `AntProject` en el campo Nombre, y pulse Finalizar para cerrar el asistente y crear el proyecto.
- 3 Seleccione Archivo | Nuevo para abrir la galería de objetos y haga doble clic en el ícono Aplicación de la ficha General para abrir el Asistente para aplicaciones.
- 4 Acepte los valores por defecto y haga clic en Finalizar para cerrar el asistente.

## Paso 2: Crear el archivo de generación Ant

---

Ahora que ya tiene un proyecto, se crea un archivo de generación Ant que se utiliza para generar el proyecto. JBuilder identifica automáticamente los archivos llamados `build.xml` como archivos de generación Ant y muestra iconos Ant para estos nodos en el panel del proyecto.

Primero, cree el archivo de generación Ant.

- 1 Seleccione Archivo | Archivo nuevo.
- 2 Escriba `build` en el campo Nombre, seleccione XML como extensión de archivo en la lista desplegable Tipo y pulse Aceptar. El nuevo archivo se abre en el editor.
- 3 Pulse Aceptar para guardar el archivo en el proyecto.
- 4 Escriba el siguiente texto o cópielo y péguelo en el editor:

```

<?xml version="1.0"?>
<!DOCTYPE project>
<project name="AntProject" default="dist" basedir=".">
<property name="src" value="src"/>
<property name="build" value="build"/>
<property name="dist" value="dist"/>

<target name="init">
<tstamp/>
<mkdir dir="${build}"/>
</target>

<target name="compile" depends="init">
<javac srcdir="${src}" destdir="${build}"/>
</target>

<target name="dist" depends="compile">
<mkdir dir="${dist}/lib"/>
<jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
</target>

<target name="clean">
<delete dir="${build}"/>
<delete dir="${dist}"/>
</target>

</project>

```

**5** Guarde el proyecto.

**6** Seleccione Proyecto | Añadir archivos/paquetes, diríjase al directorio AntProject en la pestaña Explorador, seleccione build.xml y pulse Aceptar para añadirlo al proyecto.

**7** Examine el archivo build.xml para comprender su función:

- project (proyecto): incluye el nombre de proyecto, el objetivo por defecto a ejecutar si no se ejecuta ninguno de los otros tipos individuales, y la ubicación del directorio base.
- properties (propiedades): Los tipos y tareas Ant están habitualmente “enlazados a la propiedad”. Las propiedades se utilizan también para pasar parámetros a las tareas sin, por ello, redefinir las propiedades ya existentes en el archivo de generación.
- init target (iniciar): crea un directorio build para las clases compiladas.
- compile target (compilar): inicia primero el objetivo init, compila los archivos fuente Java y coloca los archivos .class generados en el directorio build.
- dist target: inicia primero el objetivo compile, a continuación crea un directorio dist/lib/ y, por último, crea un archivo JAR en dicho directorio.

- clean target (limpiar): elimina los directorios build y dist.

## Paso 3: Ejecutar tipos de generación individuales

---

A continuación, ejecutará dos tipos del archivo build.xml. Primero, ejecutará el init target para crear el directorio build para los archivos .class compilados. Finalmente, utilizará el objetivo clean para eliminar la salida y el directorio build.



- 1 Pulse el botón Actualizar en el panel del proyecto y amplíe el nodo Ant, para mostrar los nodos descendientes, que son los tipos Ant.
- 2 Haga clic con el botón derecho sobre el tipo Ant init del panel del proyecto y seleccione Ejecutar Make. Este objetivo crea el directorio build al que irán los archivos .class compilados.
- 3 Examine los mensajes de salida de un nodo Ant en el panel de mensajes. El objetivo init creó con éxito el directorio build.

```
StdOut
Buildfile: build.xml
init:
[mkdir] Created dir: C:\Documents and Settings\ktaylor\jbproject\
          AntProject\build
BUILD SUCCESSFUL
Total time: 2 segundos
```

- 4 Haga clic con el botón derecho sobre el tipo Ant compile del panel del proyecto y seleccione Ejecutar Make. Este objetivo compila los archivos fuente .java, genera los archivos .class, y los coloca en el directorio build creado por el objetivo init.
- 5 Haga clic con el botón derecho sobre el objetivo Ant clean y seleccione Ejecutar Make para eliminar toda la salida generada, incluido el directorio build.

## Paso 4: Ejecutar el objetivo por defecto

---

Si selecciona el nodo Ant y realiza un Ejecutar Make sobre él, JBuilder ejecuta el objetivo Ant por defecto, en este caso, el objetivo dist. El objetivo por defecto se especifica en el elemento <project>. El objetivo dist crea dist/lib/ y genera un archivo JAR en ese directorio. Observe que el objetivo dist depende de compile, que, a su vez, depende de init. Así que cuando el objetivo dist se ejecuta, ejecuta el objetivo compile, que, a su vez, ejecuta el objetivo init. Debido a estas dependencias, el orden de ejecución de los tipos es: init, compile, dist.

A continuación, se ejecutará el tipo por defecto del archivo de generación.

- 1** Haga clic con el botón derecho sobre el nodo Ant `build.xml` y seleccione Ejecutar Make para ejecutar el tipo de generación por defecto, `dist`.
- 2** Mire en el panel de mensajes para ver los resultados de la generación:

```
StdOut
    Buildfile: build.xml
    init:
        [mkdir] Created dir: C:\Documents and Settings\ktaylor\
                jbproject\AntProject\build
    compile:
        [javac] Compilando 2 archivos fuente en C:\Documents and Settings\
    ktaylor\
                jbproject\AntProject\build
    dist:
        [mkdir] Created dir: C:\Documents and Settings\ktaylor\
                jbproject\AntProject\dist\lib
        [jar] Building jar: C:\Documents and Settings\ktaylor\jbproject\
                AntProject\dist\lib\MyProject-20020826.jar
    BUILD SUCCESSFUL
    Total time: 4 segundos
```

Debido a que borró anteriormente las clases y su directorio con el tipo `clean`, el tipo `init` vuelve a crear el directorio `build`. El tipo `compile` compila una vez más los archivos `.class`. Finalmente, el tipo `dist` crea el directorio `dist/lib/` y genera un archivo JAR en él.

## Paso 5: Tratamiento de errores con Ant

---

En este paso, introducirá un error en un archivo fuente Java, ejecutará Make del archivo de generación Ant, y utilizará los mensajes de error para buscarlo en el archivo fuente.

- 1** Abra `Application1.java` en el editor.
- 2** Busque el método `main()` y añada etiquetas de comentarios como se muestra a continuación:

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

- 3** Haga clic con el botón derecho en `build.xml` y elija Ejecutar Make.
- 4** Examine el panel de mensajes y observe que aparece un nodo `StdErr` que muestra mensajes de error. En este ejemplo, añadir etiquetas de comentarios delante del método `main()` produce dos errores:

```
StdErr
    "Application1.java":      [javac] C:\Documents and Settings\ktaylor\jbproject\
        AntProject\src\antproject\Application1.java:43: error #203: Inicio
            de tipo no válido en la línea 43
            [javac]     try {
            [javac]         ^
    "Application1.java":      [javac] C:\Documents and Settings\ktaylor\jbproject\
        AntProject\src\antproject\Application1.java:51: error #202: 'clase' o
            'interfaz' esperada en la línea 51
            [javac] }
            [javac] ^
```

- 5 Seleccione un mensaje de error del panel para resaltarlo en el editor. Haga doble clic en el error para desplazar el cursor a la línea de código, en el editor.

**Sugerencia**

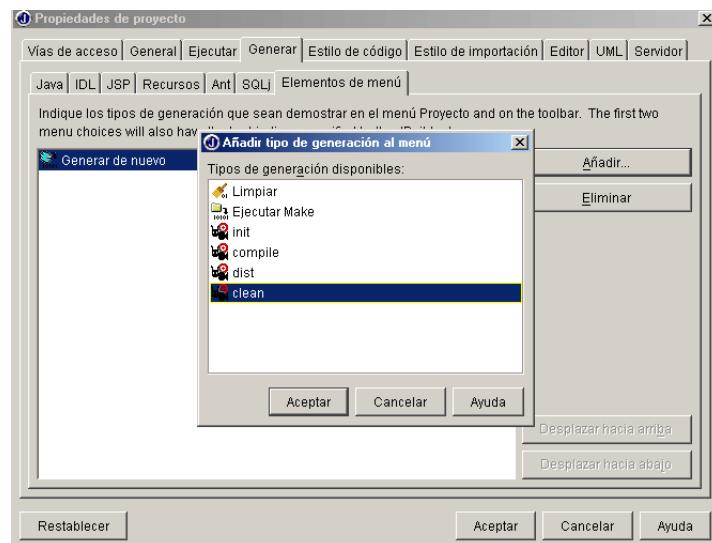
Si está utilizando el compilador Java de Borland (**bmj**) para crear su archivo Ant, los errores del compilador se listan por número. Para una lista completa de errores del compilador por número, consulte “Mensajes de error del compilador”. La opción Utilizar el compilador Java de Borland está activada por defecto. Consulte “[Configuración de las propiedades Ant](#)” en la página 6-15.

- 6 Elimine las etiquetas de comentario del método `main()` antes de pasar al siguiente paso.

## Paso 6: Añadir un tipo de generación al menú Proyecto

En este paso, se añade el tipo Ant clean al menú Proyecto y se reorganizan los tipos de generación. Para obtener más información sobre la configuración de este menú, consulte “[Configuración del menú Proyecto](#)” en la página 6-21.

- 1 Elija Proyecto | Propiedades de proyecto y se abrirá el cuadro de diálogo Propiedades de proyecto.
- 2 Seleccione la pestaña Generar y, a continuación, la pestaña Elementos de menú.
- 3 Pulse el botón Añadir para abrir el cuadro de diálogo Añadir tipo de generación al menú.



- 4 Seleccione el tipo Ant, clean (build.xml), **no** el tipo de JBuilder, Limpiar.

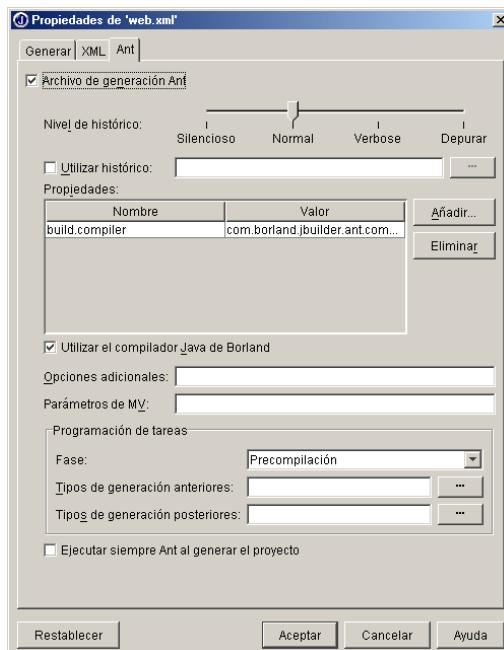
- 5 Pulse Aceptar para añadir al menú Proyecto el tipo de generación clean.
  - 6 Seleccione el botón Desplazar hacia arriba para subir el tipo Ant clean en la lista bajo Ejecutar Make. Ahora, el tipo clean será el segundo elemento en el menú Proyecto.
- Sugerencia** Los dos primeros elementos del menú Proyecto tienen asignadas teclas que pueden personalizarse en el Editor de configuración de teclado (Herramientas | Opciones del editor | Editor | Personalizar).
- 7 Haga clic en Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.
  - 8 Seleccione Proyecto | Limpiar para limpiar el proyecto. Observe en el panel de mensajes que el tipo clean se ha ejecutado.

## Paso 7: Configuración de las propiedades Ant

---

Puede que haya casos en los que quiera cambiar las propiedades Ant del fichero de generación sin sobreescribirlo. Puede realizar esto pasando parámetros en el cuadro de diálogo Propiedades Ant.

- 1 Haga doble clic sobre el nodo build.xml y seleccione Propiedades.
- 2 Abra la pestaña Ant y cambie la opción Nivel de histórico a Verbose, lo que proporciona más información al panel de mensajes.



## Paso 7: Configuración de las propiedades Ant

- 3 Pulse el botón Añadir, a la derecha de la lista Propiedades.
- 4 Seleccione build en la lista desplegable Nombre y escriba test el campo Valor.



- 5 Pulse dos veces sobre Aceptar para cerrar ambos cuadros de diálogo.  
Ahora, cuando ejecute el objetivo Ant compile, se creará un directorio test y los archivos de clase se crearán en este directorio en vez de en build.
- 6 Haga clic con el botón derecho sobre el tipo Ant compile y seleccione Ejecutar Make.
- 7 Examine el resultado obtenido en el panel de mensajes. Hay más información, porque se utiliza la opción verbose. Los resultados le indican que se creó el directorio test cuando se ejecutó el tipo init, en lugar del directorio build. Debería ver algo similar a esto en el panel de mensajes:

```
StdOut
    Apache Ant version 1.5 compilado el 9 de julio de 2002
    Buildfile: build.xml
    Detected Java version: 1.4 en: C:\jbuilder\jdk1.4\jre
    Detected OS: Windows 2000
        analizando archivo de generación build.xml with URI = file:C:/Documents and
        Settings/ktaylor
            /jbproject/AntProject/build.xml
        Project base dir set to: C:\Documents and
            Settings\ktaylor\jbproject\AntProject
        Redefinir ignorada por propiedad build
        Build sequence for target 'compile' is [init, compile]
        Complete build sequence is [init, compile, clean, dist]
        init:
            [mkdir] Created dir: C:\Documents and Settings\ktaylor\
                jbproject\AntProject\test
        compile:
            [javac] antproject\Application1.java añadida como C:\Documents and
            Settings\ktaylor\
                Settings\ktaylor\
                    jbproject\AntProject\test\antproject\Application1.class no existe.
            [javac] antproject\Frame1.java añadida como C:\Documents and Settings\
            ktaylor\
                jbproject\AntProject\test\antproject\Frame1.class no existe.
            [javac] Compilando 2 archivos fuente en C:\Documents and Settings\ktaylor\
                jbproject\AntProject\test
        BUILD SUCCESSFUL
        Total time: 4 segundos
```

A continuación, se configura una opción para generar siempre el proyecto con Ant cuando utilice el comando Crear proyecto o Generar el proyecto. En primer lugar, ha de limpiar el proyecto.

- 1 Seleccione Proyecto | Limpiar. Como puede ver en el panel de mensajes, se ha borrado toda la salida de Ant, incluyendo los archivos de clases y el directorio test.
- 2 Haga clic con el botón derecho del ratón en `AntProject.jpx` en el panel del proyecto y pulse Limpiar. Esto elimina las clases del directorio `classes` que generó el sistema de generación de JBuilder. Si mira en el administrador de archivos de su sistema operativo, verá que las clases y el directorio `classes` generado por JBuilder han sido eliminados.
- 3 Haga doble clic sobre el nodo `build.xml` y seleccione Propiedades.
- 4 Abra la pestaña Ant y active la opción Ejecutar siempre Ant al generar el proyecto en la ficha Ant y pulse Aceptar para cerrar el cuadro de diálogo. A partir de ahora, cuando elija Proyecto | Ejecutar Make del proyecto, Ant se ejecutará como parte del proceso de generación de JBuilder.
- 5 Seleccione Proyecto | Ejecutar Make del proyecto para generararlo.

Ant ejecuta el tipo Ant por defecto y JBuilder genera con Ejecutar Make. Los mensajes de Ant mostrados en el panel de mensajes le informan de que se han creado nuevos directorios, se han compilado las clases y se ha creado un JAR. Mire en el administrador de archivos de su sistema operativo para comprobar si JBuilder generó también los archivos de clases en el directorio `classes` cuando se ejecutó Make.

## Paso 8: Añadir tareas Ant personalizadas a su proyecto

---

Puede haber casos en los que tenga bibliotecas personalizadas que contengan tareas de generación Ant hechas a medida. Por ejemplo, puede tener tareas de generación en su archivo de generación Ant que necesiten ejecutar herramientas tales como ANTLR Translator generator, Java mail o JUnit testing. Puede crear una biblioteca personalizada que incluya las vías de acceso a estas herramientas y añadirla a su proyecto. Puede también utilizar una versión diferente de Ant si añade una biblioteca con los JAR de Ant. Si no especifica ningún JAR de Ant, JBuilder utiliza el Ant que se suministra en el directorio `lib` de JBuilder.

Puede añadir estas bibliotecas a su proyecto en la ficha de Propiedades de proyecto de la siguiente manera:

- 1 Seleccione Proyecto | Propiedades de proyecto.
- 2 Seleccione la pestaña Generar y, a continuación, la pestaña Ant.

- 3** Pulse el botón Añadir para abrir el cuadro de diálogo Seleccionar bibliotecas.
  - 4** Seleccione una biblioteca de la lista o pulse el botón Nueva para crearla con el Asistente para bibliotecas. Pulse Aceptar para cerrar el cuadro de diálogo Seleccione una biblioteca y añadir la biblioteca al proyecto.  
Para obtener más información sobre la creación de bibliotecas, consulte “[Adición de proyectos como bibliotecas necesarias](#)” en la página 3-5.
  - 5** Seleccione una biblioteca de la lista y pulse Desplazar hacia arriba o Desplazar hacia abajo si desea cambiar su orden en la lista. Las bibliotecas se buscan por el orden en que figuran en la lista.
  - 6** Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.
- Ahora que ha completado el tutorial, consulte “[Generación con archivos Ant externos](#)” en la página 6-9 para aprender más sobre la ejecución de archivos de generación Ant en JBuilder.

# 19

## Tutorial: Depuración remota

Este tutorial es una  
característica de JBuilder  
Enterprise

En este tutorial aprenderá paso a paso a:

- Utilizar las funciones de depuración remota para conectarse a un programa que ya se está ejecutando en un ordenador remoto.
- Depurar utilizando la inspección interprocesal.
- Utilizar configuraciones predefinidas para depurar tanto procesos de cliente como de servidor.

El tutorial emplea el proyecto de ejemplo incluido en la carpeta `<jbuilder>\samples\RMI`. Se trata de una aplicación RMI creada en JBuilder. Antes de ejecutar este tutorial, asegúrese de haber instalado la carpeta de ejemplos.

En este tutorial se presupone lo siguiente:

- El lector está utilizando Windows.
- El lector está familiarizado con la compilación, la ejecución y la depuración. En caso contrario, trabaje con el tutorial del [Capítulo 17, “Tutorial: Tutorial de compilación, ejecución y depuración”](#). También puede leer los siguientes capítulos:
  - [Capítulo 6, “Generación de programas en Java”](#)
  - [Capítulo 5, “Compilación de programas en Java”](#)
  - [Capítulo 7, “Ejecución de programas en Java”](#)
  - [Capítulo 8, “Depuración de programas en Java”](#)
- El lector está familiarizado con los procesos cliente/servidor en JBuilder.
- El lector ha leído el [Capítulo 9, “Depuración remota”](#).

## Paso 1: Abrir el proyecto de ejemplo

- Se está familiarizado con las ventanas de DOS y con la ejecución de comandos desde la línea de comandos.

Para ejecutar este tutorial, necesita:

- Dos equipos conectados en red. JBuilder debe estar instalado en uno de ellos y el JDK 1.3 o posterior en el otro. En este tutorial, llamaremos equipo “cliente” al que tiene JBuilder. Aquel donde se encuentra el JDK se conocerá como “remoto”. Este ordenador ejecutará el servidor.
- El nombre del host o la dirección IP del ordenador remoto. Este ID suele asignarlo el administrador de la red.
- Una forma de transferir archivos desde el ordenador cliente hasta el remoto.

**Nota** Para ejecutar el ejemplo sin depurarlo, siga las instrucciones del archivo HTML del proyecto, `SimpleRMI.html`.

El apartado Opciones de accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder para personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte “[Convenciones de la documentación](#)” en la página 1-5.

## Paso 1: Abrir el proyecto de ejemplo

---

Este tutorial emplea el proyecto de ejemplo incluido en la carpeta `samples\RMI` de la instalación de JBuilder. Antes de ejecutar este tutorial, asegúrese de haber instalado la carpeta de `ejemplos`.

En este paso se abrirá el archivo de proyecto. Para abrir el proyecto de ejemplo,

- 1 Seleccione Archivo | Abrir proyecto. Se abre el cuadro de diálogo Abrir proyecto.
- 2 Deplácese a la carpeta `<jbuilder>\samples\RMI`.
- 3 Haga doble clic en `SimpleRMI.jpx`. El proyecto se abre en el panel del proyecto. Los archivos del proyecto se enumeran en el panel del proyecto. Este proyecto consta de seis archivos:
  - `SimpleRMI.html` - El archivo HTML que proporciona una descripción general del proyecto. Este archivo incluye instrucciones para la creación y ejecución de una aplicación RMI en JBuilder.

- SimpleRMI.policy - El archivo de normas de seguridad. El archivo especifica los derechos que tiene el servidor RMI para monitorizar y aceptar peticiones del cliente en una red.
- SimpleRMIClient.java - La clase cliente que se conecta al objeto servidor.
- SimpleRMIClientImpl.java - La clase que implementa la interfaz del servidor RMI.
- SimpleRMIClientInterface.java - La interfaz RMI.
- SimpleRMIServer.java - La clase servidor que crea una instancia de la clase `Impl`.

En el Paso 2, se definirán las configuraciones de ejecución y depuración del cliente y del servidor

## Paso 2: Definir las configuraciones de ejecución y depuración

---

En este paso se definirán las configuraciones de ejecución y depuración para el cliente y el servidor. Para obtener más información sobre las configuraciones de ejecución y depuración consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7 y “[Configuración de las opciones de depuración](#)” en la página 8-76.

Las configuraciones de este tutorial se definen mediante las siguientes fichas del cuadro de diálogo:

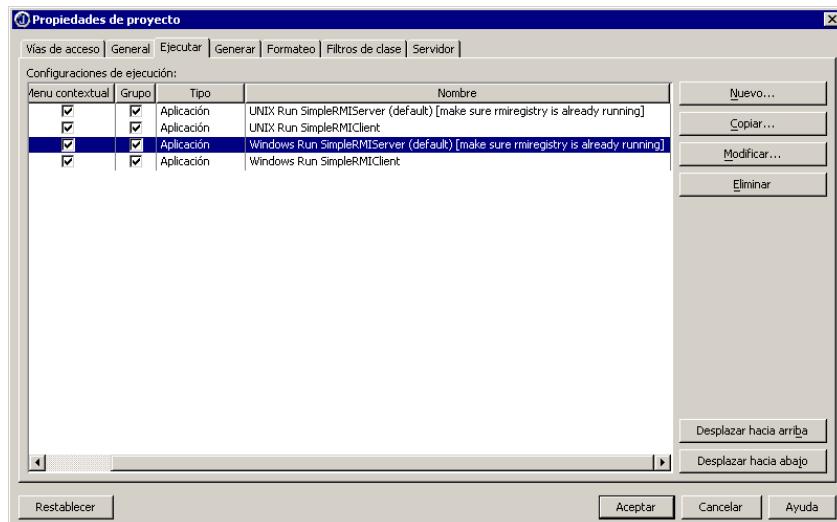
**Tabla 19.1** Fichas del cuadro de diálogo para definir las configuraciones de depuración y ejecución servidor y cliente

Fichas del cuadro de diálogo Propiedades de configuración de ejecución	Se aplica a	Descripción
Ficha Ejecutar	Servidor (ejecuta en el ordenador remoto)	Configura los parámetros de ejecución para el servidor RMI.
Ficha Depurar	Servidor (ejecuta en el ordenador remoto)	Configura cómo se asocia el servidor del ordenador del cliente al proceso del servidor remoto.
Ficha Ejecutar	Cliente (ejecuta en el ordenador con JBuilder)	Configura los parámetros de ejecución del servidor RMI.

## Paso 2: Definir las configuraciones de ejecución y depuración

Para definir las configuraciones de ejecución del servidor:

- 1 Seleccione Ejecutar | Configuraciones. A continuación se muestra la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto.



- 2 Elija la configuración denominada Windows Run SimpleRMIServer.
- 3 Pulse Modificar para mostrar la ficha Ejecutar del cuadro de diálogo Propiedades de configuración de ejecución.
- 4 Cerciórese de que el argumento codebase de los Parámetros de MV indica la ubicación de los archivos de clase de servidor. En una típica instalación de Windows, éste será la carpeta classes de la carpeta <jbuilder>\samples\RMI:

```
-Djava.rmi.server.codebase=file:C:\<jbuilder>\samples\RMI\classes\
```

**Nota** Hace falta incluir la última barra invertida del argumento, después de classes.

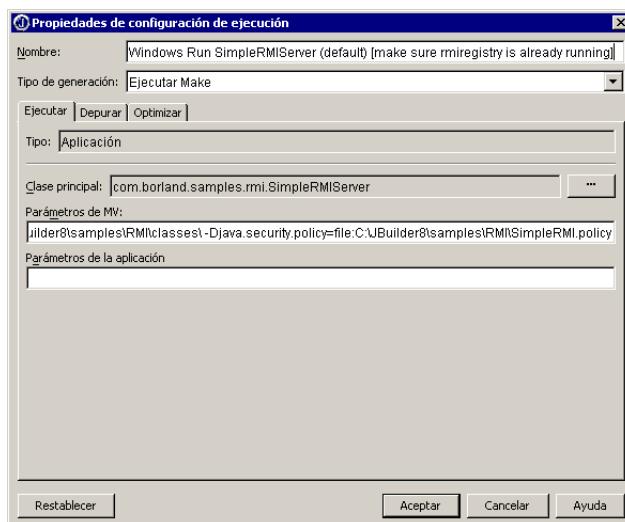
- 5 Cerciórese de que el argumento de normas de seguridad del campo Parámetros de MV indica la ubicación del archivo de normas de seguridad. El archivo de normas delimita los derechos que tiene el servidor RMI para monitorizar y aceptar peticiones del cliente RMI en una red. En una instalación típica de Windows, se tratará de la carpeta <jbuilder>\samples\RMI.

```
-Djava.security.policy=file:C:\<jbuilder>\samples\RMI\SimpleRMI.policy
```

- 6 Cerciórese de que la clase principal sea:

```
com.borland.samples.rmi.SimpleRMIServer
```

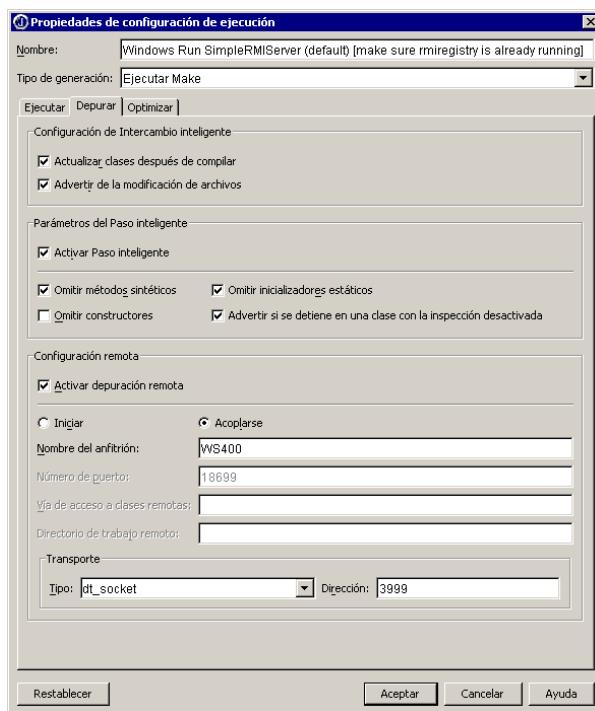
- 7 Cuando haya terminado, la ficha Ejecutar del servidor debería tener un aspecto parecido a éste:



Para definir la configuración de la depuración remota del servidor:

- 1 Haga clic en la pestaña Depurar.
- 2 Haga clic en la opción Activar depuración remota y luego en la opción Acoplarse.
- 3 En el campo Nombre del anfitrión, escriba el nombre del ordenador donde se ejecutará el servidor.
- 4 Deje el tipo de transporte en `dt_socket`.
- 5 Escriba la dirección del ordenador remoto en el campo Dirección. Volverá a utilizar este número al ejecutar el servidor en el ordenador remoto (en "["Paso 5: Iniciar el registro de RMI y el servidor en el equipo remoto"](#) en la página 19-10). Para los objetivos de este tutorial, déjelo como 3999.

- 6 Cuando haya terminado, la ficha Depurar del servidor debería tener un aspecto parecido a éste:

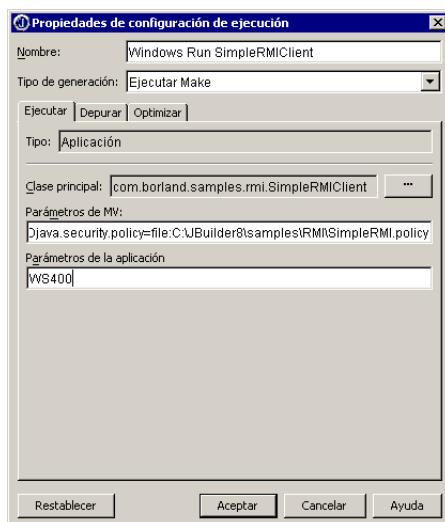


- 7 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de configuración de ejecución.

A continuación, se configurará la ejecución del cliente.

- 1 En la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto, seleccione la configuración llamada Windows Run SimpleRMIClient.
- 2 Pulse Modificar para mostrar la ficha Ejecutar del cuadro de diálogo Propiedades de configuración de ejecución.
- 3 Cerciórese de que el argumento del campo Parámetros de la MV indica la ubicación del archivo de normas de seguridad. En una instalación típica de Windows, se tratará de la carpeta <jbuilder>\samples\RMI.  
-Djava.security.policy=file:C:\<jbuilder>\samples\RMI\SimpleRMI.policy
- 4 Cerciórese de que la clase principal sea:  
`com.borland.samples.rmi.SimpleRMIClient`
- 5 En el campo Parámetros de la aplicación, escriba el nombre del ordenador remoto. Éste es el nombre que escribió en el campo Nombre de anfitrión de la ficha Depurar del cuadro de diálogo Propiedades de configuración de ejecución para el servidor (en el apartado anterior).

- 6** Cuando haya terminado, la ficha Ejecutar correspondiente al cliente debería tener un aspecto similar a éste:



- 7** Haga clic en Aceptar para cerrar el cuadro de diálogo.  
**8** Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

En el próximo paso, se definirán los puntos de interrupción en el cliente y el servidor.

## Paso 3: Definición de puntos de interrupción

---

En este paso se definirá un punto de interrupción en el proceso del cliente y un punto de interrupción interprocesal en el proceso del servidor. El primer punto de interrupción hará que el cliente se detenga cuando esté a punto de llamarse al punto de interrupción interprocesal. El punto de interrupción interprocesal detendrá al servidor. Esta técnica le permite inspeccionar procesos del servidor desde un proceso del cliente.

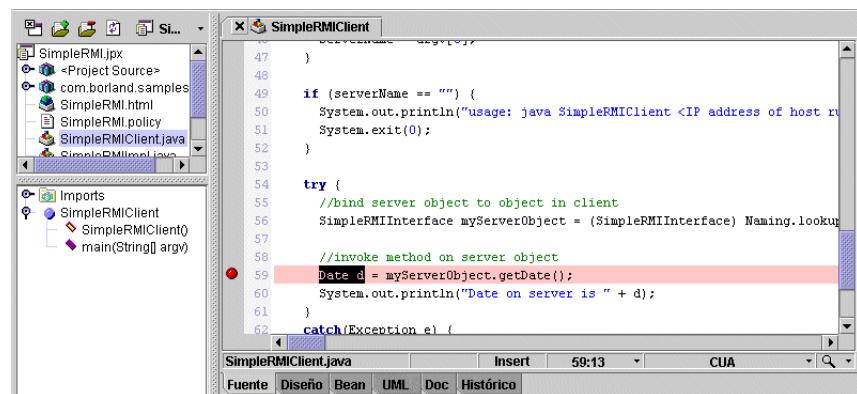
Para definir un punto de interrupción en una línea en el proceso del cliente:

- 1** Haga doble clic sobre `SimpleRMIClient.java` en el panel del proyecto. El archivo se abre en el editor.
- 2** Utilice Buscar | Buscar para encontrar la cadena `Date d`. El cursor se posicionará en la siguiente línea:

```
Date d = myServerObject.getDate();
```

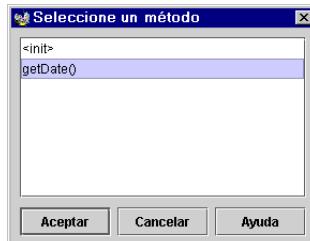
### Paso 3: Definición de puntos de interrupción

- 3 Haga clic sobre el margen, en el área gris a la izquierda de la línea de código, para definir un punto de interrupción en la línea.



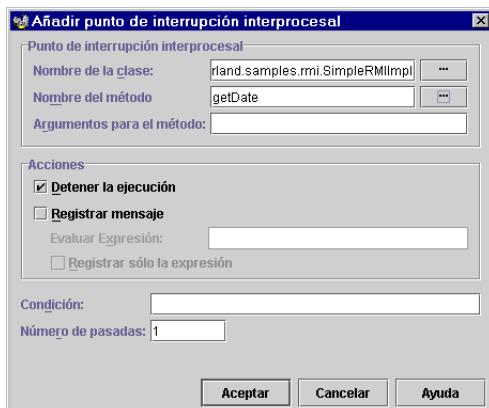
Para definir un punto de interrupción interprocesal en el proceso del servidor:

- 1 Seleccione Ejecutar | Añadir punto de interrupción | Añadir punto de interrupción interprocesal. Aparece el cuadro de diálogo Añadir punto de interrupción interprocesal.
- 2 Pulse el botón de puntos suspensivos a la derecha del campo Nombre de la clase.
- 3 En el campo Buscar del cuadro de diálogo Seleccionar clases, escriba SimpleRMIClient.
- 4 Pulse Aceptar para cerrar el cuadro de diálogo cuando haya seleccionado la clase.
- 5 Pulse el botón de puntos suspensivos a la derecha del campo Nombre del método.
- 6 Seleccione getDate() en el cuadro de diálogo Seleccione un método.



- 7 Pulse Aceptar para cerrar el cuadro de diálogo.
- 8 La opción Acción del cuadro de diálogo Añadir punto de interrupción interprocesal debe mostrar Detener la ejecución.

- 9 El cuadro de diálogo Añadir punto de interrupción interprocesal debe tener este aspecto:



- 10 Pulse Aceptar para cerrar el cuadro de diálogo.

En el próximo paso, compilaremos el servidor y copiaremos los archivos de clase del servidor al ordenador remoto.

## Paso 4: Compilar el servidor y almacenar archivos de clase al equipo remoto

---

En este paso se compilará el servidor y se copiarán sus archivos de clase en el equipo remoto.

Seleccione Proyecto | Ejecutar Make de “SimpleRMI.jpx” para compilar los archivos del servidor en JBuilder. La barra de estado indica cuándo el proyecto ha sido generado.

Observe que en el panel del proyecto aparece un ícono que indica que SimpleRMIImpl.java puede ampliarse. El compilador de RMI ha creado la clase stub SimpleRMIImpl\_Stub.java. No modifique este archivo, ya que se genera automáticamente.

Abra una ventana de DOS y desplácese a la carpeta <jbuilder>/samples/RMI. Debe incluir un subdirectorio classes. El subdirectorio classes contiene una estructura de carpetas idéntica a la del paquete. Los archivos .class de servidor se almacenan en la carpeta classes/com/borland/samples/rmi. La carpeta classes también incluye las carpetas dependency, cache y Generated Source.

Los archivos de clase del servidor han de copiarse al equipo remoto. En el caso de este ejemplo, puede copiarse toda la carpeta RMI al equipo remoto, con el mismo nombre, RMI. Para ello, puede proceder de una de las siguientes maneras:

## Paso 5: Iniciar el registro de RMI y el servidor en el equipo remoto

- Copie los archivos en una red y posteriormente al equipo remoto.
- Copie los archivos a un disquete y de allí llévelos al equipo remoto.
- Envíe los archivos al equipo remoto por FTP.

**Importante** A partir de este punto, si actualiza los archivos fuente en el equipo cliente (donde se ejecuta JBuilder), deberá copiar los archivos .class de nuevo en el equipo remoto. De no hacerlo así, los archivos fuente y los compilados diferirán, produciendo errores.

En el siguiente paso, se iniciarán el registro de RMI y el servidor en el equipo remoto.

## Paso 5: Iniciar el registro de RMI y el servidor en el equipo remoto

---

En este paso se iniciará el registro de RMI en el equipo remoto, así como el servidor, éste en modo depuración. Es imprescindible conocer los valores de RMI, así como la configuración de depuración Java de la línea de comandos que pone en marcha el servidor.

Para iniciar el registro de RMI en el equipo remoto:

- 1 Abra la ventana 4DOS o 4NT.
- 2 Cambie a la carpeta <jdk>\bin.
- 3 Inicie el registro de RMI mediante el siguiente comando:

```
start rmiregistry
```

El registro de RMI se pone en marcha en un proceso independiente. Si el registro no se inicia, puede que no haya memoria suficiente. Cierre las demás aplicaciones y la ventana de DOS e inténtelo de nuevo.

Para iniciar el servidor en el equipo remoto:

- 1 Abra una ventana de comandos. La línea de comandos de Java cuenta con más de 256 caracteres, por lo que no podrá ejecutarla en una ventana estándar de 4DOS o 4NT. Abra la ventana de comandos desde el menú Inicio, pulse sobre Ejecutar y escriba command. En el caso de NT, escriba cmd.
- 2 Cerciórese de que la carpeta <jdk>\bin está en el PATH.
- 3 Desplácese al directorio raíz del ejemplo RMI.
- 4 Escriba el siguiente comando en la ventana. Este comando iniciará el servidor en modo depuración y detendrá su ejecución. Este comando puede colocarse en un archivo .bat o un script del shell. Si lo hace así, cerciórese de no incluir retornos de carro.

**Paso 6: Iniciar el proceso del servidor y del cliente, en modo depuración.**

```
java -Xdebug -Xnoagent -Djava.compiler=NONE -  
Djava.rmi.server.codebase=file:\rmi\classes\ -Djava.security.policy=file:\  
rmi\SimpleRMI.policy -  
Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=y -classpath d:\  
rmi\classes\ com.borland.samples.rmi.SimpleRMIServer
```

La línea de comandos que el usuario introduce para ejecutar el servidor utiliza los argumentos para RMI y para el depurador. Sigue una descripción de cada uno de los parámetros.

**Tabla 19.2** Argumentos de línea de comandos para RMI y el depurador

Parámetro	Descripción
java	El comando que inicia la máquina virtual de Java.
-Xdebug	Ejecuta la MV en modo depuración.
-Xnoagent	No utiliza el agente de depuración.
-Djava.compiler=NONE	No utiliza ningún JIT.
-Djava.rmi.server.codebase= file:\rmi\classes\	Indica la ubicación de los archivos de clase del servidor.
-Djava.security.policy= file:\rmi\SimpleRMI.policy	Indica la ubicación del archivo de normas de seguridad de java.
-Xrunjdwp:transport=dt_socket,server=y, address=3999,suspend=y	Opciones del depurador, donde: <ul style="list-style-type: none"><li>• transport: El método de transporte. Debe coincidir con los valores asignados en la página de Depuración de propiedades de ejecución del servidor, consulte "<a href="#">"Paso 2: Definir las configuraciones de ejecución y depuración"</a> en la página 19-3.</li><li>• server: Ejecutar la MV en modo servidor.</li><li>• address: El número del puerto a través del cual el depurador se comunica con el equipo remoto. Debe coincidir con los valores asignados en la página de Depuración de propiedades de ejecución del servidor, consulte "<a href="#">"Paso 2: Definir las configuraciones de ejecución y depuración"</a> en la página 19-3.</li><li>• suspend: Indica si se ha de suspender el programa inmediatamente tras su inicio.</li></ul>
-classpath d:\rmi\classes\	La vía de acceso a las clases.
com.borland.samples.rmi.SimpleRMIServer	El archivo ejecutable del servidor (incluye el nombre del paquete).

En el siguiente paso se empleará el depurador para conectarse con el servidor e inspeccionar el método `getDate()` del servidor, donde se fijó el punto de interrupción interprocesal.

## Paso 6: Iniciar el proceso servidor y el cliente en modo depuración

En este paso se iniciarán el proceso servidor y el cliente en modo depuración, dentro de JBuilder, y se inspeccionará el punto de interrupción interprocesal. Una vez comenzada la inspección, JBuilder permite desplazarse entre el cliente y el servidor. Se seguirán los siguientes pasos:

- Iniciar el proceso del servidor en el equipo cliente, en modo depuración.
- Iniciar el cliente, en el ordenador cliente, en modo depuración.
- Inspeccionar el punto de interrupción interprocesal en el servidor que se está ejecutando en el equipo remoto.

Para iniciar el proceso del servidor en modo depuración, en el equipo cliente (donde se ejecuta JBuilder):



- 1 Pulse sobre la flecha que se encuentra a la derecha del botón Depurar programa en la barra de herramientas principal.
- 2 Seleccione la configuración Windows Run SimpleRMIServer.

**Nota**

No es preciso iniciar el registro de RMI en el equipo cliente. Ya se está ejecutando en el equipo remoto.

- 3 El debugador se pone en marcha y detiene la ejecución.
- 4 Haga clic en el botón Reanudar el programa de la barra de herramientas del depurador.

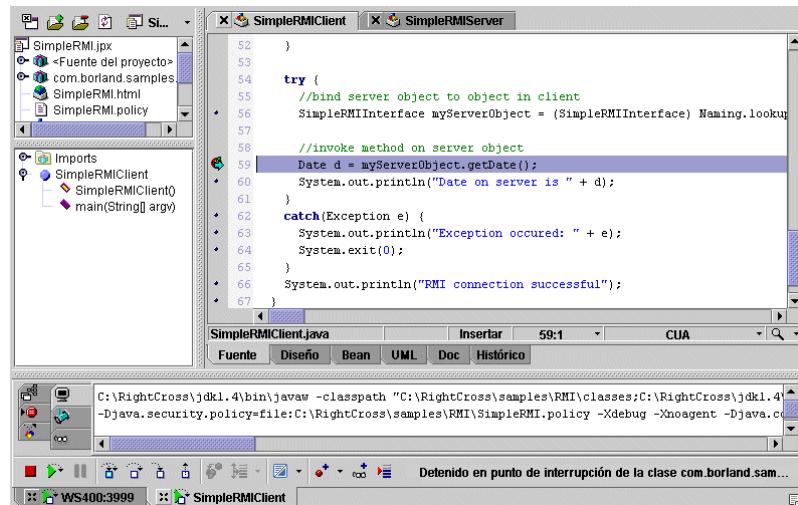
En el ordenador remoto se muestra el mensaje SimpleRMImpl ready. El nombre del ordenador y la dirección se muestran en la pestaña del depurador, que se encuentra en la parte inferior de la ventana del visualizador de aplicaciones JBuilder AppBrowser.

Para iniciar el cliente en modo depuración, en el equipo cliente (donde se ejecuta JBuilder):



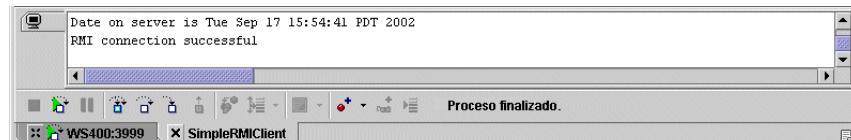
- 1 Pulse con el botón derecho sobre la flecha que se encuentra a la derecha del botón Depurar programa en la barra de herramientas principal.
- 2 Seleccione la configuración Windows Run SimpleRMIClient.

- 3 El depurador se pone en marcha y detiene la ejecución en la llamada al método `getDate()` del servidor. (Se estableció un punto de interrupción en esta línea en el Paso 2.)



Para inspeccionar el punto de interrupción interprocesal:

- 1 Pulse sobre la pestaña del proceso `SimpleRMIClient` en el depurador.
- 2 Pulse sobre el ícono Inspeccionar en la barra de herramientas del depurador del cliente, lo que comenzará la inspección del método que tiene la interrupción en el lado servidor. Si utiliza Omitir inspección, el depurador no se detendrá.
- 3 Pulse el botón Inspeccionar dos veces más. En el ordenador remoto se muestra el mensaje `SimpleRMImpl getDate()`.
- 4 Siga pulsando Inspeccionar en la pestaña `SimpleRMIClient` hasta que el cliente se ejecute por completo. El proceso `SimpleRMIClient` tendrá el siguiente aspecto en el depurador:



Este será el resultado en el servidor que se ejecuta en el equipo remoto:

```
SimpleRMImpl ready
SimpleRMImpl.getDate()
```

- 5 Para salir del servidor del ordenador remoto, pulse `Ctrl + C` desde la ventana de comandos. Para cerrar RMIServer, pulse el botón de cierre de la ventana de RMIServer.

## Paso 6: Iniciar el proceso servidor y el cliente en modo depuración

Durante la puesta en marcha del servidor o el cliente en modo depuración, en JBuilder, puede presentarse alguno de los siguientes mensajes de error:

**Tabla 19.1** Mensajes de error de RMI cliente/servidor

Mensaje de error	Descripción
connection refused	Puede que el registro de RMI no se esté ejecutando en el ordenador remoto. Detenga todos los procesos y ejecute el registro de RMI en el ordenador remoto, escribiendo <code>start rmiregistry</code> en la línea de comandos. (La carpeta <code>&lt;jdk&gt;\bin</code> debe estar incluida en PATH.) Reinicie el servidor remoto y comience el proceso de depuración de nuevo.
Java exception: <code>java.rmi.NotBoundException</code> <code>SimpleRMIClient</code>	No se ha iniciado el proceso de depuración del servidor. Haga clic en el botón Reanudar el programa  de la barra de herramientas del depurador. Vuelva a iniciar el cliente en modo depuración.

¡Enhorabuena! Ha finalizado este tutorial. Ha ejecutado un servidor RMI en un equipo remoto valiéndose de configuraciones de ejecución preestablecidas. Y, posteriormente, ha depurado el programa con las funciones de depuración remota de JBuilder.

# 20

## Tutorial: Visualización de código con el visualizador UML

En este tutorial se utilizan funciones de JBuilder Enterprise.

En este tutorial se le muestra paso a paso cómo utilizar las características UML de JBuilder para poder navegar por su código y analizarlo. El UML resulta de gran ayuda para examinar código, analizar el desarrollo de una aplicación y para transmitir el diseño del software. JBuilder utiliza diagramas UML para presentar código y buscar clases y paquetes. Los diagramas UML pueden ayudarle a comprender rápidamente la estructura de un código desconocido, reconocer áreas especialmente complejas, e incrementar su productividad resolviendo problemas con mayor rapidez.

Si desea más información sobre las funciones UML de JBuilder, consulte [Capítulo 11, “Presentación de código con UML”](#). Para conocer las definiciones de los términos de UML, consulte [“Términos de Java y UML en la página 11-2”](#).

En este tutorial, realizará tareas como:

- Visualización de un diagrama de paquete UML.
- Visualización de un diagrama de clase UML.
- Cómo añadir referencias de biblioteca.
- Filtrado de diagramas UML.

El tutorial utiliza el proyecto de ejemplo situado en la carpeta `samples/DataExpress/ProviderResolver` de la instalación de JBuilder. Antes de ejecutar este tutorial, asegúrese de haber instalado la carpeta de `ejemplos`. Los usuarios con acceso de sólo lectura a los ejemplos de JBuilder deben copiar el directorio `samples` en un directorio para el que tengan permiso de lectura y escritura.

## Paso 1: Compilación del ejemplo

Este ejemplo muestra cómo generar un proveedor y un almacenador de DataExpress personalizados. Utiliza una sencilla aplicación que muestra los datos proporcionados por el bean `ProviderBean` al conjunto de bases de datos `TableDataSet` en una tabla `JdbTable`. Incluye también una barra de herramientas `JdbNavToolBar` cuyo botón `Guardar` puede pulsarse para guardar los cambios en el archivo de texto, que contiene datos de ejemplo, vía `ResolverBean`. Si desea más información acerca del ejemplo, consulte el archivo de notas del proyecto en el ejemplo: `ProviderResolver.html`. El ejemplo incluye los siguientes archivos:

- `ProviderBean.java`: proporciona datos, a partir de un sencillo archivo de texto no delimitado, a un `TableDataSet`.
- `ResolverBean.java`: reemplaza los datos en el archivo de texto original.
- `TestApp.java`: una sencilla aplicación que muestra los datos proporcionados por el bean `ProviderBean` al conjunto de bases de datos `TableDataSet` en una tabla `JdbTable`. Incluye también una barra de herramientas `JdbNavToolBar` cuyo botón `Guardar` puede pulsarse para guardar los cambios en el archivo de texto vía `ResolverBean`.
- `TestFrame.java`: la aplicación interfaz de usuario.
- `data.txt`: el archivo de texto que contiene los datos del ejemplo.
- `DataLayout.java`: una interfaz que describe la estructura de `data.txt`.

### Consulte

- “[Términos de Java y UML](#)” en la página 11-2
- “[Glosario de los diagramas UML de JBuilder](#)” en la página 11-7

El apartado Opciones de accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder para personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte “[Convenciones de la documentación](#)” en la página 1-5.

## Paso 1: Compilación del ejemplo

---

En este paso, compilará el proyecto. Siempre es mejor compilar antes de seleccionar la pestaña UML, de esta forma el diagrama será exacto y estará actualizado. Cuando selecciona la pestaña UML, JBuilder carga los archivos de clase para determinar sus relaciones que el visualizador UML utilizará para obtener la información de paquete y clase para los diagramas.

Lo primero que hay que hacer es abrir el ejemplo:

- 1 Seleccione Archivo | Abrir proyecto y busque el ejemplo  
ProviderResolver: <jbuilder>/samples/DataExpress/ProviderResolver/ProviderResolver.jpx
- 2 Seleccione Proyecto | Ejecutar Make del proyecto para compilar el proyecto.

**Importante**

Antes de ver el diagrama UML de un proyecto o archivo, siempre debe compilar el proyecto para ver el diagrama completo. JBuilder carga entonces las clases compiladas para generar los diagramas. Seleccione Proyecto | Ejecutar Make del proyecto antes de ir a la pestaña UML.

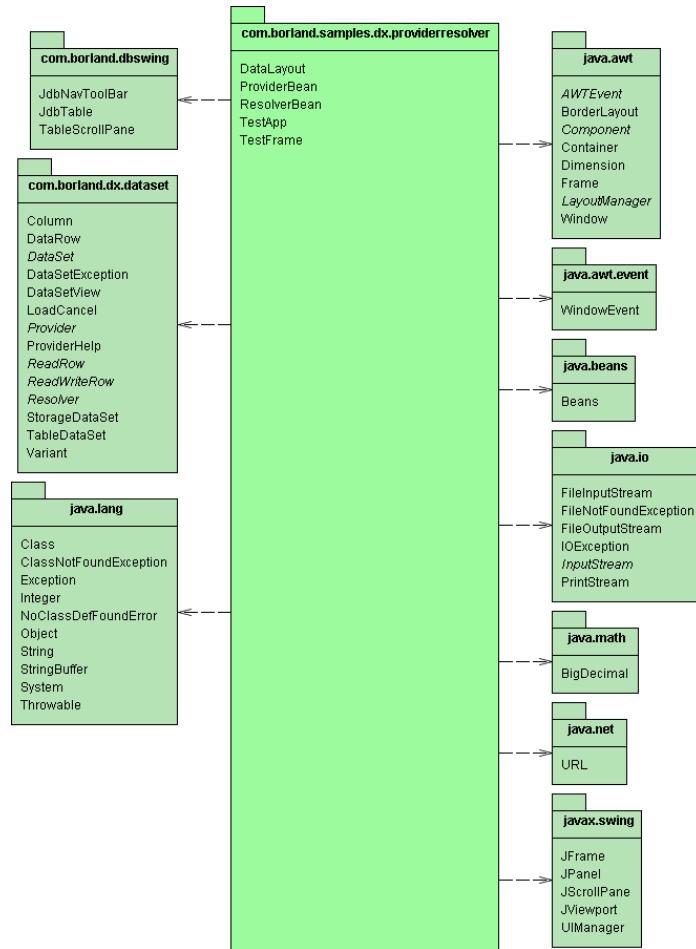
## Paso 2: Visualización de un diagrama de paquete UML

---

Ahora que el proyecto está compilado, JBuilder puede crear los diagramas UML a partir de los archivos de clase generados. Comience por mirar el diagrama de paquete UML.

- 1 Haga doble clic en el nodo del paquete com.borland.samples.dx.providerresolver en el panel del proyecto para abrirlo en el panel de contenido. Por defecto, se abre con la pestaña Paquete, que muestra el resumen del paquete, activada.
- Nota** Si el nodo del paquete no está disponible, configure la opción Activar la localización y compilación de paquetes fuente en la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto).

- 2 Seleccione la pestaña UML para ver el diagrama del paquete UML. Cuando selecciona la pestaña UML, JBuilder carga las clases para determinar sus relaciones y genera el diagrama UML.



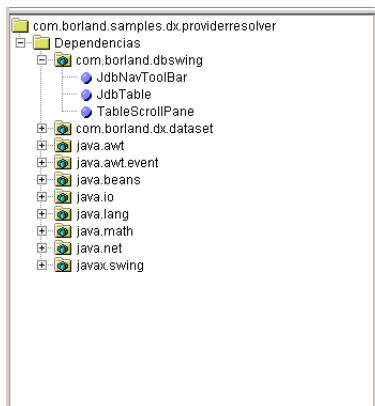
El diagrama representa cada paquete como un rectángulo con una pestaña y su nombre completo encima. El paquete actual, `com.borland.samples.dx.providerresolver`, está en el centro, con sus dependencias con otros paquetes a cada lado. Las dependencias en el diagrama UML se representan mediante una línea discontinua que va desde el paquete central hasta el paquete del que depende. Este paquete central, que tiene un fondo verde brillante, lista todas las clases que contiene. Los paquetes exteriores, que tienen un fondo verde más oscuro, sólo listan las clases de las que `com.borland.samples.dx.providerresolver` es dependiente.

- 3 Examine el panel de estructura del lado inferior izquierdo del diagrama UML. Hay dos carpetas que pueden aparecer con los diagramas de

paquete: Dependencias y Dependencias inversas. Este paquete sólo tiene dependencias, por lo que sólo hay una carpeta.

**Nota** Para conocer más acerca de las definiciones de las carpetas del panel de estructura, consulte “[Glosario de los diagramas UML de JBuilder](#)” en la [página 11-7](#).

- 4 Abra la carpeta Dependencias para ver los paquetes, clases e interfaces de los que el paquete central es dependiente. Como verá más adelante, puede utilizar el panel de estructura para navegar a otros diagramas UML.



**Nota** Para conocer las definiciones de los iconos del panel de estructura, consulte “Iconos UML y panel de estructura de JBuilder” en la ayuda en línea.

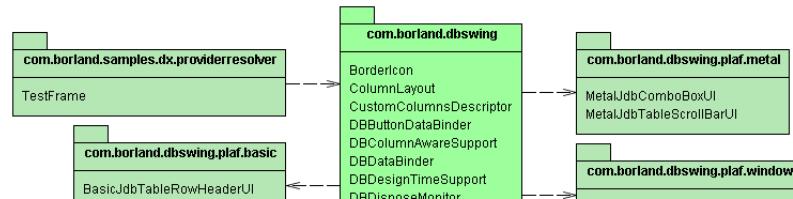
En los siguientes pasos, navegue hasta un paquete que también tiene dependencias inversas: `com.borland.dbswing`.

- 1 Desplácese hasta el paquete `com.borland.dbswing` que está en la parte superior izquierda del diagrama UML. Existen dos formas de hacer esto:
  - Haga doble clic sobre el nombre del paquete en el diagrama UML.
  - Abra una carpeta en el panel de estructura, haga clic con el botón derecho sobre el nombre del paquete y seleccione Abrir.

Puede tardar un poco en cargarse. Ahora, el paquete `com.borland.dbswing` es el paquete central del diagrama UML. Hay líneas discontinuas que apuntan en ambas direcciones en este diagrama. El paquete `com.borland.dbswing` tiene *dependencias y dependencias inversas*. Consulte “[Glosario de los diagramas UML de JBuilder](#)” en la [página 11-7](#) para obtener las definiciones de estos términos.

- 2 Mire al paquete del lado superior izquierdo, `com.borland.samples.dx.providerresolver`. Tiene una línea discontinua que apunta hacia el paquete central, y no desde el paquete central. Esto significa que es una dependencia inversa. La clase `TestFrame` del paquete

com.borland.samples.dx.providerresolver utiliza componentes dbSwing para el interfaz de usuario de la aplicación, tales como JdbNavBar y JdbTable. Puesto que TestFrame tiene dependencia de com.borland.dbswing, entonces com.borland.dbswing tiene una dependencia inversa de com.borland.samples.dx.providerresolver.



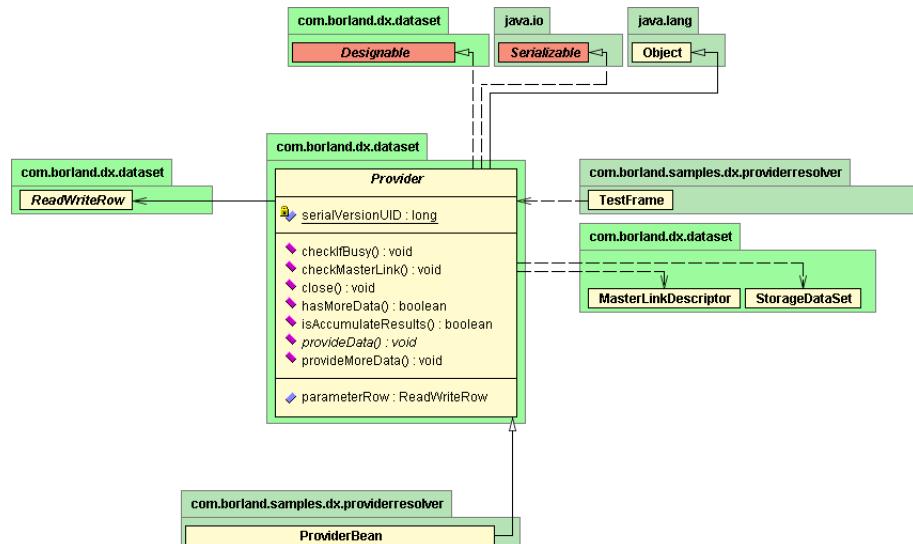
- 3 Mire las carpetas del panel de estructura. Hay dos para este paquete: Dependencias y Dependencias inversas.
- 4 Haga doble clic en la carpeta Dependencias inversas del panel de estructura y amplíe el nodo del paquete com.borland.samples.dx.providerresolver para ver que TestFrame también aparece listado como una dependencia inversa de com.borland.dbswing.
- 5 Haga doble clic sobre TestFrame en el diagrama UML o en el panel de estructura para ver los componentes de dbSwing que utiliza, así como otros componentes de otros paquetes. Ahora, está viendo un diagrama de clase UML con TestFrame como la clase central y los componentes listados directamente bajo el nombre de la clase.
- 6 Seleccione Ver | Ocultar todo para ampliar el visualizador UML y ocultar los paneles de proyecto y estructura.

## Paso 3: Visualización de un diagrama de clase UML

En este paso, navegará hasta otro diagrama de clase, en este caso una clase abstracta llamada Provider. En los diagramas UML, las clases abstractas aparecen en cursiva.

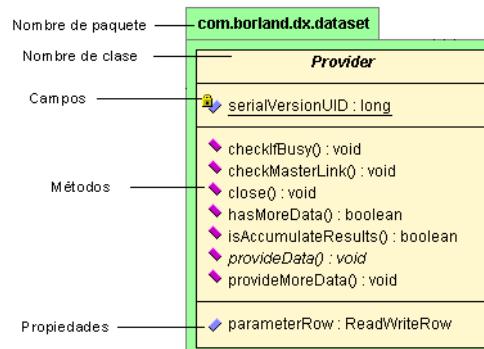
- 1 Desplácese a la derecha y haga doble clic sobre la clase Provider ubicada en el paquete com.borland.dx.dataset, el segundo a la derecha en el diagrama UML TestFrame.java. Ahora, Provider aparece como la clase

central de un diagrama de clase UML. Provider aparece resaltado para indicar que está seleccionado.



Y todos los diagramas del paquete actual, `com.borland.dx.dataset`, se muestran por defecto con fondo verde brillante, mientras que los otros paquetes lo hacen con verde más oscuro.

El diagrama de clase UML muestra la clase en el centro del diagrama. Rodeándola, aparece el paquete, con su nombre en una pestaña en la parte superior. La propia clase está dividida en varias secciones separadas por líneas horizontales en el orden siguiente:



- Nombre de la clase (arriba): las clases abstractas aparecen en cursiva.
- Campos y propiedades \*: los campos estáticos aparecen subrayados.

- Métodos y obtención\* y asignación\*: los métodos abstractos aparecen en cursiva, los estáticos subrayados.
- Propiedades\* (opcional) (abajo).

\*Por defecto, las propiedades se muestran en la sección inferior. La opción Mostrar propiedades por separado se configura en la ficha UML del cuadro de diálogo Opciones (Herramientas | Opciones del IDE). Si esta opción está desactivada, las propiedades se muestran en las secciones adecuadas, con campos y métodos. Consulte “[Configuración de las opciones del IDE](#)” en la página 11-21.

**Nota** Los iconos indican si un campo, método o propiedad es privado, público, o protegido. Para conocer las definiciones de los iconos, consulte “Iconos UML y panel de estructura de JBuilder en la ayuda en línea. Puede también cambiar estos iconos por iconos genéricos de UML. Consulte “[Configuración de las opciones del IDE](#)” en la página 11-21.

He aquí otras cosas a observar en el diagrama:

- Las dependencias y las dependencias inversas están a la derecha indicadas con líneas discontinuas.
- Las asociaciones están a la izquierda indicadas con líneas continuas.
- Las clases ampliadas (superclases) y las interfaces implementadas, están arriba. Las relaciones ampliadas se representan con una línea continua con un gran triángulo. Las relaciones implementadas utilizan una línea discontinua con un gran triángulo.
- Las clases ampliables están abajo. Las relaciones ampliadas se representan con una línea continua con un gran triángulo.
- Las interfaces se representan por defecto con rectángulos naranjas con su nombre en cursiva.
- Por defecto, las clases se representan con rectángulos amarillos.

Consulte “[Glosario de los diagramas UML de JBuilder](#)” en la página 11-7 para obtener las definiciones de estos términos.

- 2 Mire los métodos listados en la tercera sección de la clase central Provider. Uno de los métodos, `provideData()`, es un método abstracto. Por lo tanto, su nombre aparece en cursiva.
- 3 Haga doble clic sobre el nombre del método `provideData()` en el diagrama UML para leer los comentarios en el código fuente. Este método abstracto, heredado por la clase ProviderBean en la parte baja del diagrama, proporciona los datos para un DataSet. Además, este método está también en la clase abstracta Provider.

- 4 Haga clic en la pestaña UML para volver al diagrama de clase UML Provider. Bajo la clase central hay clases ampliables. ProviderBean, que proporciona los datos que se han de leer en una TableDataSet, deriva de (hereda de) la clase abstracta Provider.
- 5 Haga clic sobre la clase ProviderBean en la parte inferior del diagrama y seleccione Ir a código fuente para examinar algunos de sus métodos. Observe que hereda el método `provideData()` del método abstract `provideData()` de Provider del que se deriva.
- 6 Seleccione la pestaña UML para ver el diagrama de clase ProviderBean. La herencia se representa en los diagramas UML como una línea continua con un gran triángulo que apunta a la superclase. Debido a que ProviderBean hereda de y se deriva de Provider, hay una línea continua con un gran triángulo que apunta desde ProviderBean hacia Provider en la parte superior del diagrama.
- 7 Posicione el puntero sobre el método `provideData()` de la clase ProviderBean. Una herramienta de ayuda inmediata muestra el nombre del método con sus parámetros y tipo devuelto. Esta es una forma práctica de aprender más sobre un método sin abandonar el diagrama.

```
provideData(StorageDataSet, boolean): void
```

- 8 Haga doble clic sobre la clase Provider en la parte superior del diagrama o haga clic con el botón derecho y seleccione Ir a el diagrama para volver a su diagrama de clase.

Ahora, mire la última sección del diagrama de la clase central Provider. Por defecto, las propiedades se listan de forma separada. Puede cambiar la visualización de propiedades en la ficha UML del cuadro de diálogo Opciones del IDE(Herramientas | Opciones del IDE). Una propiedad existe cuando un nombre de método precedido de "is" "get", o "set" coincide con un nombre de campo. Por ejemplo, en este diagrama, `parameterRow` es un campo con métodos get y set. Por lo tanto, `parameterRow` es una propiedad.

- 1 Haga clic con el botón derecho en el campo `parameterRow`, en la parte inferior de la clase Provider, en el diagrama UML, y seleccione Ir a código fuente.
- 2 Seleccione Ver | Mostrar todo, y así podrá ver de nuevo el panel de estructura.
- 3 Examine los métodos del panel de estructura y busque cualquier método "is", "get", o "set". Hay dos métodos con el mismo nombre que el campo `parameterRow`: `getParameterRow()` y `setParameterRow(ReadWriteRow)`.

## Paso 4: Añadir referencias de bibliotecas

---

En alguna ocasión, puede querer incluir referencias desde bibliotecas del proyecto para ver un diagrama completo de todas las relaciones. Por defecto, los diagramas UML de JBuilder no muestran las referencias de bibliotecas del proyecto. Habitualmente, las bibliotecas proporcionan servicios a las aplicaciones que se construyen a partir de ellas, pero no saben nada acerca de sus usuarios. Para mostrar estas relaciones, necesita incluir referencias de las bibliotecas.

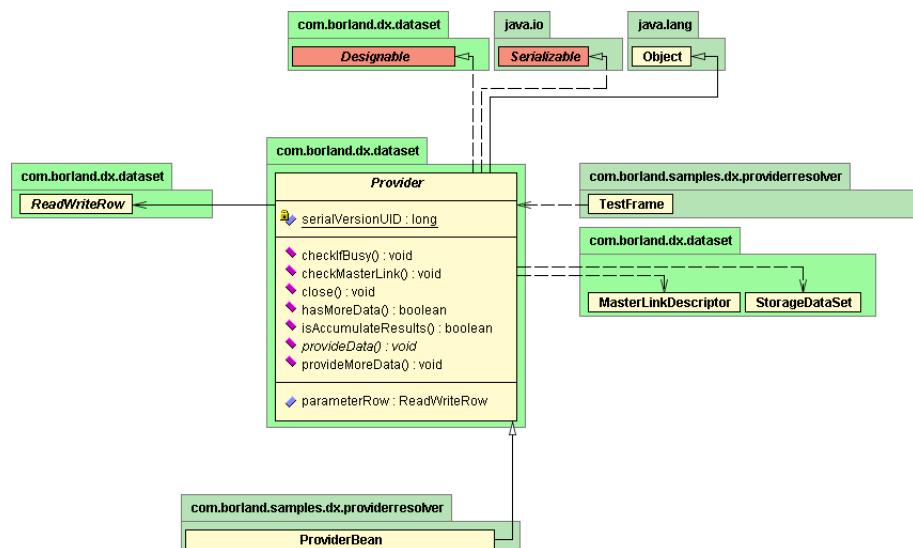
Cuando seleccione la opción Incluir referencias de los archivos de clase de las bibliotecas del proyecto en la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto), se incluirán también referencias de las clases de la biblioteca del proyecto en el diagrama UML. Después de seleccionar esta opción, JBuilder actualiza automáticamente el diagrama. Para obtener más información sobre esta opción, consulte “[Inclusión de referencias de bibliotecas de proyecto](#)” en la página 11-20.

**Advertencia** Si las bibliotecas son grandes, el diagrama UML puede tardar cierto tiempo en cargarse. JBuilder debe cargar primero todas las clases y la información de las dependencias para determinar las relaciones.

Antes de seleccionar la opción Incluir referencias de los archivos de clase de las bibliotecas del proyecto, revise el diagrama de la clase Provider. Después, cambie las propiedades del proyecto y observe cómo cambia el diagrama Provider.

1 Seleccione la pestaña UML para volver al diagrama de la clase Provider. Hay un paquete en el lado izquierdo que contiene la clase `ReadWriteRow` y un paquete en la parte inferior del diagrama que contiene la clase

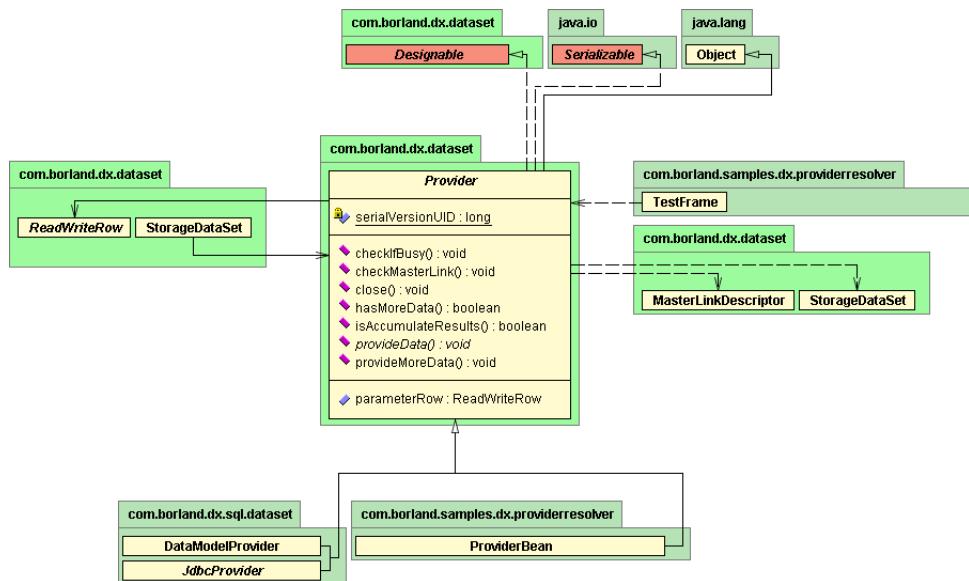
ProviderBean. Cuando añada las referencias de las bibliotecas, se añadirán más paquetes y clases al diagrama en estas áreas.



- 2** Haga doble clic sobre `ProviderResolver.jpx` en el panel del proyecto y seleccione Propiedades para abrir el cuadro de diálogo Propiedades de proyecto.
- 3** Abra la pestaña General.
- 4** Seleccione la opción Incluir referencias de los archivos de clase de las bibliotecas del proyecto en la parte inferior de la ficha.



- 5 Pulse Aceptar para cerrar el cuadro de diálogo. El diagrama UML se actualiza y ahora muestra referencias adicionales de las bibliotecas del proyecto.
- 6 Revise el diagrama UML y observe los paquetes y clases adicionales añadidos a la izquierda y en la parte inferior del mismo. En el lado izquierdo, se ha añadido StorageDataSet al paquete com.borland.dx.dataset. En la parte inferior se ha añadido otro paquete com.borland.dx.sql.dataset con dos clases. Estas clases en la parte inferior del diagrama se derivan de Provider. Las clases del paquete com.borland.dx.sql.dataset no se utilizan directamente por las clases del proyecto, pero, puesto que derivan de Provider, guardan relación con el mismo.



- 7 Pulse y arrastre el diagrama sin soltar el botón del ratón para trasladarlo al centro. Mire los dos paquetes com.borland.dx.dataset a izquierda y derecha. Observe que StorageDataSet aparece en dos sitios. Provider tiene dos relaciones con StorageDataSet: una asociación inversa y una dependencia. Aparece también en las carpetas apropiadas del panel de estructura.

Ahora, dedique un momento a examinar las asociaciones generadas en un diagrama de clase UML. Las asociaciones aparecen en el lado izquierdo de un diagrama de clase. Provider tiene dos asociaciones: una *asociación* y una *asociación inversa*. Estas asociaciones aparecen en las carpetas adecuadas del panel de estructura. Para ver las definiciones de estos términos, consulte “[Glosario de los diagramas UML de JBuilder](#)” en la página 11-7.

- 1 Abra la carpeta Asociaciones del panel de estructura del diagrama de la clase Provider y amplie el nodo del paquete. Aquí puede ver que ReadWriteRow es una asociación.
- 2 Seleccione ReadWriteRow en el panel de estructura para resaltarla en el diagrama. Las asociaciones en el diagrama se representan con una línea continua que apunta desde la clase central a la asociación. Provider tiene una asociación con ReadWriteRow, puesto que tiene la siguiente referencia a ReadWriteRow:

```
private ReadWriteRow parameterRow;
```

- 3 Abra la carpeta Asociaciones inversas en el panel de estructura y amplie el nodo del paquete. Aquí puede ver que StorageDataSet es una asociación inversa.
- 4 Seleccione StorageDataSet en el panel de estructura para resaltarla en el diagrama. Una asociación inversa se representa con una línea continua que apunta desde la asociación a la clase central. StorageDataSet tiene una referencia a Provider:

```
private Provider provider;
```

- 5 Haga clic con el botón derecho en StorageDataSet en el diagrama y seleccione Ir a código fuente para navegar directamente al código fuente.
- 6 Realice una búsqueda en el código fuente de provider (Buscar | Buscar) y seleccione Buscar todos. Examine los resultados de la búsqueda en el panel de mensajes y observe que hay muchas referencias a la clase Provider, así como a los métodos getProvider() y setProvider().
- 7 Haga clic con el botón derecho sobre la pestaña Buscar en el panel de mensajes y seleccione Eliminar la pestaña "Buscar" para cerrarlo.

## Paso 5: Filtrado de diagramas UML

---

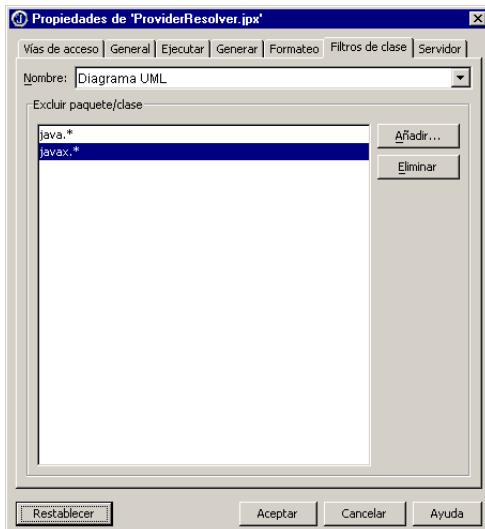
En algunos casos, puede que desee eliminar paquetes y clases de sus diagramas UML con el fin de simplificarlos. Para ello, ha de abrir el cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto). Para obtener más información, consulte “[Definición de las propiedades del proyecto](#)” en la página 11-18.

El filtrado de paquetes y clases se configura en la ficha Filtros de clase del cuadro de diálogo Propiedades de proyecto. El filtrado determina qué clases y paquetes se excluyen de los diagramas UML del proyecto. Una vez que haya configurado los filtros, puede activar y desactivar el filtrado desde el menú contextual del visualizador UML.

- 1 Seleccione la pestaña del archivo del paquete com.borland.samples.dx.providerresolver en la parte superior del panel de

contenido. Verá que ahora hay muchos paquetes java y javax en el diagrama. Estos paquetes aparecen también en el panel de estructura de la carpeta Dependencias.

- 2 Haga doble clic sobre ProviderResolver.jpx en el panel del proyecto y seleccione Propiedades para abrir el cuadro de diálogo Propiedades de proyecto.
- 3 Seleccione la pestaña Filtros de clase en el cuadro de diálogo.
- 4 Excluya los paquetes java y javax del diagrama del paquete com.borland.samples.dx.providerresolver de la siguiente manera:
  - a Seleccione Diagrama UML en la lista desplegable Nombre.
  - b Seleccione el botón Añadir para abrir el cuadro de diálogo Seleccionar paquete/clase.
  - c Seleccione java y haga clic en Aceptar.
  - d Pulse de nuevo el botón Añadir.
  - e Seleccione javax y haga clic en Aceptar. Ambos paquetes aparecen ahora en la lista Excluir paquete/clase.



- f Pulse Aceptar para cerrar el cuadro de diálogo.
- 5 Vuelva a revisar el diagrama y notará que los paquetes java y javax se han eliminado del diagrama. Sólo permanecen en el diagrama los paquetes borland.
- 6 Si examina el panel de estructura comprobará que el filtro no elimina los paquetes java y javax packages del panel de estructura, aunque los paquetes se muestran en un color más suave para indicar que no se encuentran en el diagrama.

- 7 Haga clic con el botón derecho en el visualizador UML y observe que Activar filtrado de clases está seleccionado en el menú contextual.
- 8 Desactive el filtrado seleccionando quitando la marca de Activar filtrado de clases. Los paquetes reaparecen en el diagrama.

**Importante** Si configura el filtrado en el cuadro de diálogo Propiedades de proyecto, todos los diagramas del proyecto se filtran. Deshabilitar el filtrado en un diagrama no lo desactiva para todos. Si se desplaza a otro diagrama del proyecto, el filtrado sigue activado. Cuando se cierra el archivo o el paquete, la configuración recupera los valores definidos para el proyecto.

Enhorabuena. Ha finalizado este tutorial de UML. Hay otras muchas características disponibles en el visualizador UML de JBuilder, tales como:

- Perfeccionando código
- Guardar e imprimir diagramas UML
- Visualización de Javadoc
- Personalización de diagramas UML

### Consulte

- [Capítulo 12, “Perfeccionamiento de símbolos de código”](#)
- [“Creación de imágenes de diagramas UML” en la página 11-21](#)
- [“Impresión de diagramas UML” en la página 11-22](#)
- [“Visualización de Javadoc” en la página 14-22](#)
- [“Configuración de las opciones del IDE” en la página 11-21](#)



# Tutorial: Creación y ejecución de tests y conjuntos de tests

El test de módulos una función de JBuilder Enterprise.

Este tutorial le enseñará cómo crear un test y un conjunto de test para analizar el código creado. Este tutorial utiliza el ejemplo ProviderResolver de <jbuilder>/samples/DataExpress como aplicación en test. Antes de ejecutar este tutorial, asegúrese de haber instalado la carpeta de ejemplos.

**Nota**

El tutorial del Capítulo 20, “Tutorial: Visualización de código con el visualizador UML” también utiliza el ejemplo ProviderResolver. Si ha previsto ejecutar ambos tutoriales, es recomendable que ejecute primero este último, o que haga una copia del ejemplo ProviderResolver antes de ejecutar éste, porque las modificaciones hechas en este tutorial pueden cambiar algunos de los diagramas descritos en el tutorial de UML.

Este tutorial asume que usted está familiarizado con Java, JUnit y con el IDE (entorno integrado de desarrollo) de JBuilder. Para obtener más información sobre las JSP, consulte *Procedimientos iniciales con Java*. Si desea más información sobre JUnit, visite del sitio web de JUnit, <http://www.junit.org>. Si desea obtener más información sobre el IDE de JBuilder, consulte “El entorno de JBuilder” en *Introducción a JBuilder*.

**Nota**

La opción elegida en Tipos de generación debe ser Ejecutar Make o Generar de nuevo en su configuración de ejecución actual para que todos los pasos de este tutorial funcionen correctamente. Ejecutar Make es el valor por defecto. Para obtener más información sobre las configuraciones de configuración, consulte “Definición de las configuraciones para la ejecución” en la página 7-7.

El apartado Opciones de accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder para personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte “[Convenciones de la documentación](#)” en la página 1-5.

## Paso 1: Apertura de proyectos

---

En este paso, se abre el ejemplo ProviderResolver. Para cumplir con los propósitos de este tutorial, la aplicación ProviderResolver será la aplicación bajo prueba.

- 1 Seleccione Archivo | Abrir proyecto para presentar el cuadro de diálogo homónimo.
- 2 Haga clic en el ícono de la carpeta Ejemplos.
- 3 Abra los nodos DataExpress y ProviderResolver en el árbol.
- 4 Seleccione ProviderResolver.jpx y pulse Aceptar.

El ejemplo ProviderResolver está abierto.

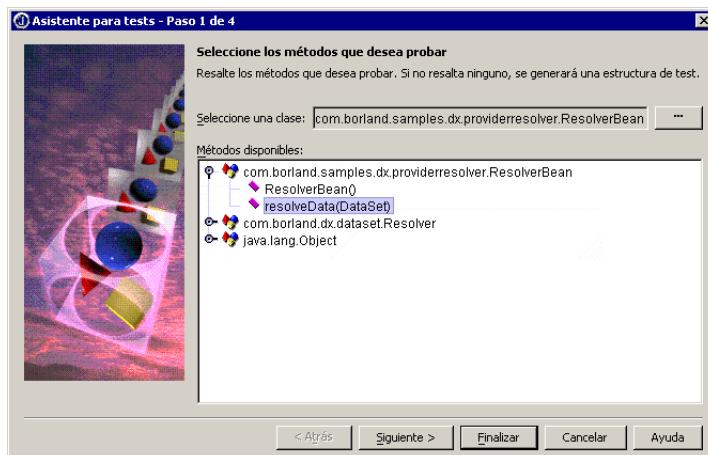
## Paso 2: Creación de montajes para tests

---

Este paso crea el montaje de un test utilizando el Asistente para tests. La clase del montaje para tests contendrá un método de test para uno de los métodos de la clase ResolverBean. Posteriormente añadirá un segundo método de test. Cuando escriba test en el mundo real, querrá realizar test más exhaustivos, pero para los propósitos de este tutorial, sólo implementará dos métodos de test.

- 1 Seleccione Proyecto | Generar de nuevo el proyecto “ProviderResolver.jpx”. Esto hace que los métodos de las clases del proyecto estén disponibles para el Asistente para tests.
- 2 Haga doble clic en `ResolverBean.java` en el panel del proyecto para abrirlo en el editor.
- 3 Para mostrar la galería de objetos, seleccione Archivo | Nuevo.
- 4 Seleccione Test en la ficha Test de la galería de objetos y pulse Aceptar. Se muestra el Asistente para tests con `ResolverBean` seleccionada como la clase a comprobar.

- 5 Seleccione el método `resolveData(DataSet)` de `ResolverBean`. Éste es el aspecto del Asistente para tests:



- 6 Pulse el botón Finalizar.
- 7 Abra el nodo del paquete `com.borland.samples.dx.providerresolver` en el panel del proyecto para ver el test que se ha creado. Se llama `TestResolverBean.java`. Haga doble clic en este archivo para abrirlo en el editor.

**Nota** Si el nodo del paquete no está disponible, configure la opción Activar la localización y compilación de paquetes fuente en la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto).

## Paso 3: Implementación de un método de test que lanza una excepción esperada

A veces es útil escribir un test para verificar que se lanza una excepción esperada. Su código del test debería ser lo suficientemente exacto como para determinar si la excepción lanzada es la misma que la esperada. El test debería fallar si se lanza otra excepción.

En este paso y en el siguiente, rellenará el montaje para el test escribiendo código de test. Este paso implementa el método `testResolveData()` en `ProbarResolverBean`. Para implementar el método:

- 1 Sustituya el cuerpo del método `testResolveData()` por el siguiente código:

```
resolverBean = new ResolverBean();
com.borland.dx.dataset.DataSet dataSetView=
    new com.borland.dx.dataset.StorageDataSet();
```

```
try {
    resolverBean.resolveData(dataSetView1);
    fail("fallo: resolveData() did not throw an exception");
}
catch(com.borland.dx.dataset.DataSetException e){
    System.out.println("TestResolverBean.testResolveData(): correcto ");
}
catch(Exception e) {
    System.err.println("Exception thrown: " +e);
    fallo("excepción incorrecta: " + e.getClass().toString());
}
```

-  **2** Haga clic en el botón Guardar todo de la barra de herramientas, o seleccione Archivo | Guardar todo.
- 3** Para ejecutar la comprobación, haga doble clic en `TestResolverBean.java` en el panel del proyecto y seleccione en el menú contextual Ejecutar test utilizando valores por defecto. Cuando termina la ejecución de la comprobación, la barra de progreso del ejecutor de tests aparece en verde. Junto a los nombres de las clases de test y los tests, en el ejecutor de tests, se muestra una marca de verificación que indica que la comprobación se ha realizado correctamente.

El código del test lanza una excepción `DataSetException`. Esto se debe a que el conjunto de datos no está abierto. Cuando la excepción `DataSetException` esperada es lanzada, ésta se captura y el test pasa porque no hay comunicación de fallo o error. Si fuera lanzada una excepción de otra clase, sería capturada por la segunda cláusula `catch` que llama al método `fail()` para asegurarse de que el test falla cuando se lanza una excepción equivocada.

Añadió también otra línea de código del test a continuación de la línea del bloque `try` que se espera que lance una excepción. Si no se lanza ninguna excepción, esta línea se ejecuta. La llamada a `fail()` en esta línea de código hace que el test falle y la cadena que se pasa explica el fallo. Por supuesto, esta nueva línea de código sólo puede ejecutarse si no se lanza una excepción.

## Visualización de la salida de fallo del test

Para mostrar el aspecto que tiene un test fallido en el ejecutor de tests:

-  **1** Anteponga signos de comentario (//) a la siguiente línea de código en el bloque `try`:

```
resolverBean.resolveData(dataSetView1);
```

- 2** Haga clic en el botón Guardar todo de la barra de herramientas, o seleccione Archivo | Guardar todo.
- 3** Para ejecutar la comprobación, haga doble clic en `TestResolverBean.java` en el panel del proyecto y seleccione en el menú contextual Ejecutar test

utilizando valores por defecto. La barra de avance se vuelve roja para indicar que ha habido al menos un fallo en el test. Se muestra la pestaña Fallos del test. Se lista un fallo para el método `testResolveData()`, que se indica mediante un icono con una X roja.

-  4 Haga clic en el nodo `testResolveData()` de la ficha Fallos del test. Esto nos muestra la siguiente salida:

```
junit.framework.AssertionFailedError: fallida: resolveData() no
lanzó una excepción

...(Click for full stack trace)...
at com.borland.samples.dx.providerresolver.TestResolverBean.testResolveData
(TestResolverBean.java:26)

...
```

Observe cómo la cadena que se pasa al método `fail()` ha sido escogida cuidadosamente para proporcionar información útil en caso de fallo. Este es un buen objetivo a tener en cuenta cuando escriba sus tests.

## Corrección del test para que pase

---

Ha precedido de signos de comentario una línea de código necesaria para que el test pase. El propósito de esto era ver el aspecto que tiene un fallo en el ejecutor de tests. Para hacer que el test pase de nuevo:

- 1 Quite los signos de comentario de la línea de código que llama a `resolverBean.resolveData(dataSetView1)`.
- 2  Haga clic en el botón Guardar todo de la barra de herramientas, o seleccione Archivo | Guardar todo.

Si el test se vuelve a ejecutar, el resultado será correcto. Hágalo ahora si lo desea.

## Paso 4: Creación de un segundo método de test

---

En este paso escribirá un método para probar el valor de una de las constantes definidas en la interfaz `DataLayout`, que es implementada por `ResolverBean`. Esta test verifica que el valor de la constante se asigna correctamente. Para ello:

- 1 Añada el siguiente método a la clase `TestResolverBean`:

```
public void testConstant() {
    resolverBean = new ResolverBean();
    assertEquals(6, resolverBean.COLUMN_COUNT);
}
```



- 2** Haga clic en el botón Guardar todo de la barra de herramientas, o seleccione Archivo | Guardar todo.

El código que acaba de añadir comprueba el valor de la constante COLUMN\_COUNT de la interfaz DataLayout para asegurarse de que concuerda con el valor esperado. Para ahorrar tiempo en este tutorial, sólo comprobamos esta constante, pero esto debe darle una idea de algunos posibles tests que podría escribir para verificar valores esperados. Si ejecuta el test ahora, debería pasar, puesto que el valor se ha configurado correctamente.

## Paso 5: Creación de un conjunto de tests

Un conjunto de tests es una colección de ellas que deberían ejecutarse en grupo. En este paso, creará un conjunto de tests utilizando el Asistente para conjuntos de tests. Este conjunto de tests llamará a TestResolverBean. Normalmente, un conjunto de tests llama a más de uno, pero, para ahorrar tiempo en este tutorial, sólo se ha creado un test. Si dispusiera de más de un test, el proceso para crear el conjunto que llama a varios test es el mismo.

- 1** Seleccione Archivo | Nuevo.
- 2** Seleccione Conjunto de tests en la ficha Test de la galería de objetos y haga clic en Aceptar.
- 3** Seleccione TestResolverBean.java como test a incluir en el conjunto. Use el botón Añadir si es necesario. Si tiene otras test, puede también añadirlas en este momento. Éste es el aspecto del Asistente para tests:



- 4** Pulse Siguiente.

- 5** Escriba ProviderResolverSuite en el campo Clase. Ahora el Asistente para conjuntos de tests tiene este aspecto:



- 6** Pulse el botón Finalizar. Se añade un archivo ProviderResolverSuite.java a su proyecto.
- 7** Haga doble clic sobre ProviderResolverSuite.java en el paquete com.borland.samples.dx.providerresolver en el panel del proyecto para abrirlo. Examine la siguiente línea de código del método suite():

```
suite.addTestSuite(
    com.borland.samples.dx.providerresolver.TestResolverBean.class);
```

Si posteriormente desea añadir otros tests a este conjunto, deberá escribir una línea de código similar a ésta por cada uno de ellos, sustituyendo el nombre de clase del nuevo test por TestResolverBean.

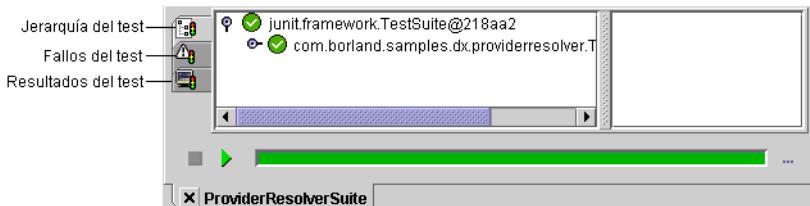
## Paso 6: Ejecución de tests

---

En este paso ejecutará el conjunto de tests que acaba de crear. El proceso para ejecutar un conjunto de tests es el mismo que para ejecutar un test excepto por el hecho de que cuando se ejecuta un conjunto, automáticamente se ejecutan todos los tests que forman parte de él. Para ejecutar su conjunto de tests:

- 1** Haga doble clic sobre ProviderResolverSuite.java en el panel del proyecto y seleccione Ejecutar test en el menú contextual. El test se ejecuta.

**2** Examine la salida. JBTestRunner tiene el siguiente aspecto:



La pestaña superior situada a la izquierda de la ficha JBTestRunner en la Vista de mensajes es la vista Jerarquía del test. Observe que el árbol que se muestra en esta vista indica que todas los tests han pasado.



Todos los iconos de los tests tienen una marca de comprobación verde. Bajo el nodo para ProviderResolverSuite, verá un subnodo para su test. Si tenía otros test, habrá un subnodo por cada uno de ellos. Expanda el nodo del test para ver los test individualmente. Haga clic en un test o nodo individual para ver sus resultados.



**3** Pulse sobre la pestaña Fallos del test. Actualmente no hay ninguna salida en esta vista. Si hubiera habido fallos, los listaría y mostraría la salida generada por sus aserciones fallidas.



**4** Pulse sobre la pestaña Resultados del test. Esta pestaña lista los resultados de los tests. El resultado del test que escribió en este tutorial es:

TestResolverBean.testResolveData(): correcto

Encima de todo esto, la pestaña Resultados del test muestra también el comando que se utilizó para ejecutar los tests

Puede también depurar los tests haciendo clic con el botón derecho sobre ellos en el panel del proyecto y seleccionando Depurar test en el menú contextual. El depurador de tests funciona igual que el depurador normal, por lo que no se explica en este tutorial. La única diferencia radica en que al depurar un test, aparecen las pestañas Jerarquía del test y Fallos del test de JBTestRunner además de la interfaz del usuario normal del depurador. Si desea obtener más información sobre el depurador, consulte el [Capítulo 8, "Depuración de programas en Java"](#).

¡Enhorabuena! Ha completado el tutorial sobre creación y ejecución de tests y conjuntos de tests. Si desea información más detallada sobre la comprobación de módulos, consulte [Capítulo 13, "Test de módulos"](#).

# Tutorial: Utilización de montajes para tests

El test de módulos una función de JBuilder Enterprise.

Este tutorial le muestra cómo utilizar el Asistente para montajes para JDBC y el Asistente para montajes para tests que le permitirán crear montajes para sus tests. Los montajes son código compartido que puede ser utilizado por múltiples clases de comparaciones para realizar tareas rutinarias. Este tutorial le muestra también cómo utilizar el Montaje de comparación y el montaje JDBC juntos en un test.

Este tutorial asume que usted está familiarizado con Java, JUnit, JDataStore y con el IDE (entorno integrado de desarrollo) de JBuilder. Para obtener más información sobre las JSP, consulte *Procedimientos iniciales con Java*. Si desea más información sobre JUnit, visite del sitio web de JUnit, <http://www.junit.org>. Si desea obtener más información sobre JDataStore, consulte la *Guía del desarrollador de JDataStore*. Si desea obtener más información sobre el IDE de JBuilder, consulte “El entorno de JBuilder en *Introducción a JBuilder*.

**Nota** La opción elegida en Tipos de generación debe ser Ejecutar Make o Generar de nuevo en su configuración de ejecución actual para que todos los pasos de este tutorial funcionen correctamente. Ejecutar Make es el valor por defecto. Para obtener más información sobre las configuraciones de configuración, consulte “[Definición de las configuraciones para la ejecución](#)” en la página 7-7.

El apartado Opciones de accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder para personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte “[Convenciones de la documentación](#)” en la página 1-5.

## Paso 1: Creación de proyectos

---

- 1 Seleccione Archivo | Nuevo proyecto para abrir el Asistente para proyectos.
- 2 En el campo Nombre, introduzca un nombre de proyecto, fixturestutorial.
- 3 Pulse Finalizar para cerrar el asistente para Proyectos y crear el proyecto. No se necesita hacer ningún cambio en los valores por defecto en los Pasos 2 y 3 del asistente.

Se crea un nuevo proyecto.

## Paso 2: Creación de un módulo de datos

---

En este paso, creará una clase `DataModule` que se utilizará para el test. En la mayoría de los casos, cuando se escriben tests reales ya se tiene un código que se desea comprobar. En este tutorial se va a crear un código de ejemplo para comprobarlo a continuación.

Para crear el módulo de datos:

- 1 Seleccione Archivo | Nuevo.
- 2 Seleccione Módulo de datos en la ficha General de la galería de objetos. Pulse Aceptar. Se abre el Asistente para módulos de datos.
- 3 Acepte los valores por defecto para Paquete y Clase, desactive la opción Llamar al Modelador de datos y pulse Aceptar. Se crea un módulo de datos.
- 4 Añada la siguiente sentencia de importación a la parte superior del archivo `DataModule1.java`, justo después de la línea `package fixturestutorial;`:

```
import com.borland.dx.sql.dataset.*;
```
- 5 Añada la siguiente línea de código justo después de la línea `private static DataModule1 myDM;`:

```
Database database1 = new Database();
```
- 6 Añada la siguiente línea de código al método `jbInit()`. <unidad> y <jbuilder> son la unidad real y el directorio donde se encuentra instalado JBuilder:

```
database1.setConnection(new ConnectionDescriptor  
    ("jdbc:borland:dslocal:<drive>:\\<jbuilder>\\samples\\JDataStore\\  
     datastores\\employee.jds",  
     "user", "", false, "com.borland.datastore.jdbc.DataStoreDriver"));
```

- 7** Añada el siguiente método al módulo de datos:

```
public Database getDatabase1() {
    return database1;
}
```

- 8** Elija Proyecto | Generar de nuevo el proyecto “fixturestutorial.jpx”.

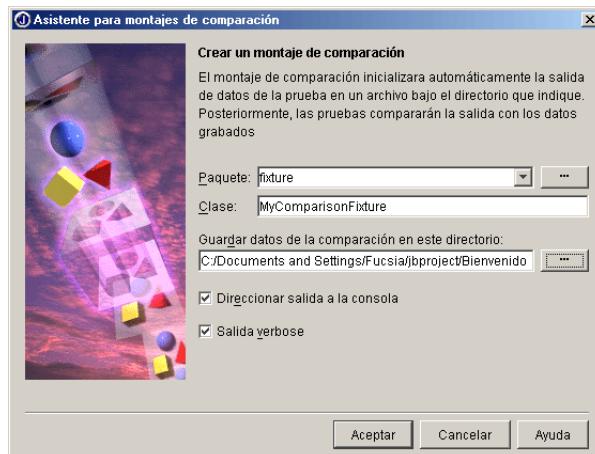
Ha finalizado el módulo de datos. En los pasos siguientes se crearán montajes para tests y un test, que se utilizarán para comprobar este módulo de datos.

## Paso 3: Creación de un montaje de comparación

---

El Asistente para montajes para tests genera un montaje que es de utilidad para grabar los resultados de los tests y compararlos con resultados de tests previos. Un Montaje de comparación amplía `com.borland.jbuilder.unittest.TestRecorder`. Para crear un Montaje de comparación:

- 1** Seleccione Archivo | Nuevo.
- 2** Haga clic en la pestaña Test de la galería de objetos. Seleccione Montaje de comparación y pulse Aceptar. Se abre el Asistente para montajes de comparación.
- 3** Acepte el valor por defecto en el campo Paquete.
- 4** Escriba `MyComparisonFixture` como nombre de clase.
- 5** Acepte el valor por defecto para la ubicación del directorio de datos de comparación .
- 6** Marque Direccional salida a la consola y Salida verbose. Éste es el aspecto del Asistente para montajes de comparación:



- 7** Pulse Aceptar. Se crea una clase de montaje de comparación llamada MyComparisonFixture. Expanda el nodo fixturestutorial del panel del proyecto para poder verlo.

**Nota** Si el nodo del paquete no está disponible, configure la opción Activar la localización y compilación de paquetes fuente en la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto).

- 8** Haga doble clic sobre MyComparisonFixture.java en el panel del proyecto para abrirlo en el editor (está en el paquete fixturestutorial). Anote la siguiente línea de código:

```
super.setMode(UPDATE);
```

Esta línea de código configura el modo de salida del montaje de comparación. He aquí los posibles valores de las constantes pasadas a setMode():

- UPDATE - El montaje de comparación compara la nueva salida con un archivo de salida existente, o lo crea si no existe y graba la salida en él.
- COMPARE - El montaje de comparación siempre compara la nueva salida con la ya existente.
- RECORD - El montaje de comparación graba todas las salidas, sobrescribiendo cualquiera ya existente en el archivo de salida.
- OFF - El montaje de comparación está desactivado.

**Sugerencia**

Si un archivo de salida contiene datos incorrectos, defina el modo de salida en RECORD después de resolver el problema. Cuando haya registrado la salida deseada, vuelva a definir el modo en UPDATE.

## Paso 4: Creación de un montaje JDBC

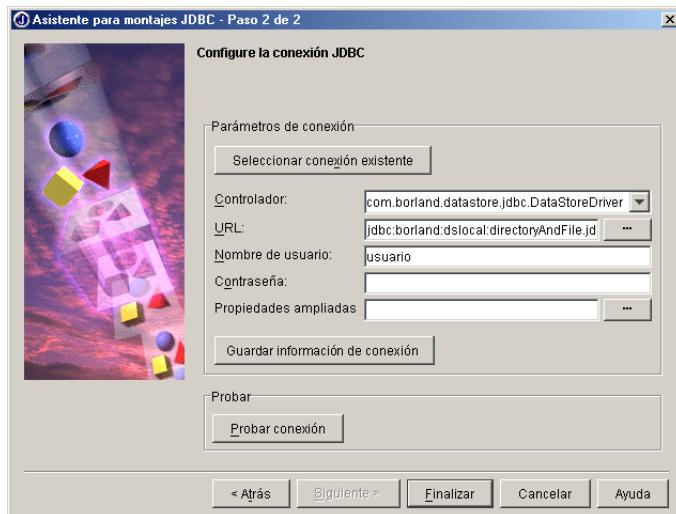
---

El Asistente para montajes JDBC genera un montaje que se utiliza para administrar conexiones con fuentes de datos JDBC.

Para crear un montaje JDBC:

- 1** Seleccione Archivo | Nuevo.
- 2** Haga clic en la pestaña Test de la galería de objetos. Seleccione Montaje JDBC y pulse Aceptar. Se abre el Asistente para montajes JDBC.
- 3** Acepte los valores por defecto para Paquete, Clase, Clase base y pulse en Siguiente.
- 4** Seleccione el siguiente controlador:  
com.borland.datastore.jdbc.DataStoreDriver

- 5 Escriba o navegue hasta la siguiente URL:  
`jdbc:borland:dslocal:<unidad>\<jbuilder>\samples\JDataStore\datastores\employee.jds` (<unidad> y <jbuilder> se sustituyen por la ubicación real de JBUILDER).
- 6 Escriba user en Nombre de usuario.
- 7 Haga clic en el botón Probar conexión. Debería haber un mensaje indicando que la acción se ha realizado con éxito a la derecha del botón Probar conexión. Éste es el aspecto del Asistente para montajes JDBC:



Si la conexión falla, puede deberse a que aún no haya introducido la información de licencia correcta de JDataStore en el Administrador de licencias de JDataStore. El Administrador de licencias de JDataStore esta disponible en el menú Archivo del Explorador de JDataStore.

- 8 Pulse el botón Finalizar. Se crea una clase de montaje para JDBC llamada `JdbcFixture1`.
- 9 Haga doble clic en `JdbcFixture1.java`, en el panel del proyecto, para abrirlo en el editor. Observe los métodos `setUp()` y `tearDown()` en este montaje. En el paso siguiente, utilizará estos métodos para ejecutar scripts SQL para gestionar datos que podrían utilizarse en sus tests.

## Paso 5: Modificación del montaje JDBC para ejecutar scripts SQL

---

En este paso modificará los métodos `setUp()` y `tearDown()` del método del montaje JDBC para hacer que ejecuten scripts SQL que generen automáticamente datos de test antes se ejecuten los tests y borren los datos cuando hayan finalizado. Para ello:

- 1** Asegúrese de que `JdbcFixture1.java` está abierto en el editor.
- 2** Añada las variables `String` que aparecen en **negrita** al montaje:

```
public class JdbcFixture1 extends JdbcFixture {

    String createSQL =
        "create table TESTTABLE (i int, j int);"+
        "insert into TESTTABLE values(1, 2);"+
        "insert into TESTTABLE values(2, 3);"+
        "insert into TESTTABLE values(3, 4);"+
        "insert into TESTTABLE values(4, 5);"

    String deleteSQL =
        "drop table TESTTABLE;";
```

Estas variables `String` contienen sentencias SQL que se utilizarán para administrar los datos de test.

- 3** Añada el código que aparece en **negrita** al método `setUp()`:

```
public void setUp() {
    super.setUp();

    Connection con = getConnection();
    if(con != null)
        runSqlBuffer(new StringBuffer(createSQL), true);
```

Este código consigue una conexión con `JDataStore` y ejecuta el script SQL para crear la tabla de tests.

- 4** Añada el código que aparece en **negrita** al método `tearDown()`:

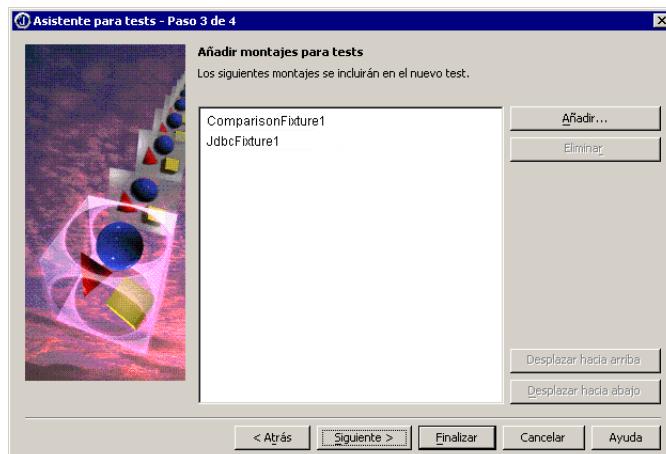
```
Connection con = getConnection();
if(con != null)
    runSqlBuffer(new StringBuffer(deleteSQL), true);
super.tearDown();
```

Este código consigue una conexión con `JDataStore` y ejecuta el script SQL para eliminar la tabla de tests.

## Paso 6: Creación de un test utilizando montajes para tests

En este caso utilizará el Asistente para tests para incluir el montaje de comparación y el montaje JDBC en un tests.

- 1 Elija Proyecto | Generar de nuevo el proyecto “fixturestutorial.jpx”. Esto hace que los métodos de las clases del proyecto estén disponibles para el Asistente para tests.
- 2 Haga doble clic en `DataModule1.java` para abrirlo en el editor.
- 3 Seleccione Archivo | Nuevo.
- 4 Haga clic en la pestaña Test de la galería de objetos. Seleccione Test y pulse Aceptar. Se abre el Asistente para tests.
- 5 Acepte `fixturestutorial.DataModule1` como la clase a probar. No seleccione ningún método.
- 6 Pulse Siguiente.
- 7 Acepte los detalles de clase por defecto en el Paso 2 del asistente y pulse en Siguiente.
- 8 Utilice el botón Añadir para añadir `MyComparisonFixture` y `JdbcFixture1` a la lista de montajes para tests seleccionados, si no lo están ya. Éste es el aspecto del Asistente para tests:



- 9 Pulse el botón Finalizar. Se añade un nuevo test al proyecto llamado `TestDataModule1`. Abra el nodo `fixturestutorial` en el panel del proyecto para verlo.

## Paso 7: Implementación del test

---

En este paso se escribe el código que llama a los dos montajes para tests, con el fin de utilizarlos en un test.

- 1 Haga doble clic en `TestInputModule1.java` en el panel del proyecto para abrirlo en el editor. El test crea una instancia para los montajes y llama a sus métodos `setUp()` y `tearDown()`.
- 2 Añada la siguiente línea de código a las sentencias `import` en la parte superior de `TestInputModule1.java`:

```
import java.awt.*;
```

- 3 Añada el siguiente método al cuerpo del archivo `TestInputModule1.java`:

```
public void testQuery() throws Exception{  
    DataModule1 dm = new DataModule1();  
    Connection con = dm.getDatabase1().getJdbcConnection();  
    Statement stmt = con.createStatement();  
    ResultSet rs = stmt.executeQuery("SELECT * FROM TESTTABLE");  
    jdbcFixture1.dumpResultSet(rs, myComparisonFixture);  
    dm.getDatabase1().closeConnection();  
}
```

Este método hace lo siguiente:

- Crea una instancia `DataModule1`.
- Establece una conexión utilizando el método `getJdbcConnection()` del objeto `Database` de `DataExpress` utilizado en `DataModule1`.
- Llama al método `Connection.createStatement()` con el fin de prepararse para la ejecución de una consulta en SQL.
- Ejecuta la consulta, almacenando el resultado en un objeto `ResultSet`.
- Utiliza el método `dumpResultSet()` del montaje JDBC para volcar el conjunto de resultados al montaje de comparación. El método `dumpResultSet()` pasa un `ResultSet` y un `Writer` como parámetros. Un Montaje para comparación puede utilizarse como el `Writer` ya que lo amplía.
- Llama al método `closeConnection()` del objeto `Database` con el fin de garantizar que la conexión con la fuente de datos está cerrada.

## Paso 8: Adición de una biblioteca necesaria

---

Para que sea posible ejecutar el test es necesario añadir al proyecto la biblioteca JDataStore. Para ello:

- 1 Seleccione Proyecto | Propiedades de proyecto.
- 2 Seleccione la pestaña Bibliotecas necesarias en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto.
- 3 Pulse el botón Añadir.
- 4 Seleccione en la lista la biblioteca JDataStore y pulse Aceptar.
- 5 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

## Paso 9: Ejecución del test

---

En este paso ejecutará el test.

- 1 Haga clic con el botón derecho sobre `TestInputModule1.java` en el panel del proyecto y seleccione Ejecutar test utilizando valores por defecto en el menú. El test se ejecuta. Cuando se ejecuta el test sucede lo siguiente:
  - El ejecutor del test crea una instancia `TestInputModule1`.
  - Se llama al `TestInputModule1.setUp()` que, como respuesta, llama a los métodos `setUp()` de los dos montajes en el orden adecuado.
  - Se llama al método `testQuery()`. La salida del montaje de comparación se graba en un archivo de datos en el mismo directorio fuente en que se encuentra `TestInputModule1.java`, el subdirectorio `test/fixtures/tutorial` del directorio del proyecto.
  - Se llama al método `tearDown()` que, como respuesta, llama a los métodos `tearDown()` de los dos montajes en el orden adecuado.

¡Enhorabuena! Ha finalizado este tutorial. Si desea información más detallada sobre la comprobación de módulos, consulte [Capítulo 13, “Test de módulos”](#).



# A

## Creación de archivos de configuración para ejecutables nativos

Los archivos de configuración proporcionan flexibilidad y personalización en el inicio de aplicaciones y utilidades. Por ejemplo, se pueden utilizar para pasar parámetros a la Máquina Virtual de Java (MV), pasar parámetros a OpenTools, depurar OpenTools y personalizar el inicio de las aplicaciones.

El *programa de inicio* de la aplicación contiene los ejecutables adecuados, scripts shell y/o iconos de escritorio, y se utiliza para iniciar la aplicación. Antes de iniciarla, busca un *archivo de configuración* para obtener instrucciones adicionales. Un archivo de configuración es un archivo de texto con una lista de directivas que distinguen entre mayúsculas y minúsculas y que tienen que ejecutarse antes de iniciar el ejecutable.

Una vez encontrado el archivo de configuración, el programa de inicio procesa cada línea de texto de forma secuencial antes de que se cargue la MV de Java. Si el programa de inicio no encuentra un archivo de configuración, se abre como un archivo y busca el archivo de configuración que tiene almacenado como un comentario zip. El programa de inicio declara un error y se cierra si no puede leer el archivo de configuración, si hay líneas con directivas no reconocidas, si falta la directiva mainclass o si la MV de Java no se inicia. También se declara un error si se omite la directiva javapath y no se puede determinar una ubicación por defecto de la MV de Java.

Las directivas del archivo de configuración pueden especificar la clase principal, añadir archivos JAR, pasar parámetros a la MV, añadir una entrada a la vía de acceso a clases Java, etc. Los archivos de configuración

también pueden incluir otros archivos de configuración. Por ejemplo, el programa de inicio de JBuilder se refiere al archivo `jbuilder.config`, que a su vez utiliza la directiva `include` para referirse a `jdk.config`, tal y como se muestra en el siguiente ejemplo.

Si está creando ejecutables con el Creador de recopilatorios o con el Creador de ejecutables nativos, puede redefinir el archivo de configuración por defecto que hayan creado estos asistentes. Para obtener más información, consulte “[Distribución con el Creador de recopilatorios](#)” en la página 15-18, y “[Creación de ejecutables con el Creador de ejecutables nativos](#)” en la página 15-32.

## Ejemplo de archivo de configuración

```
# Leer la definición del JDK compartido
include jdk.config

# Ajustar esta MV para que ofrezca espacio suficiente para trabajar con grandes
# aplicaciones
vmparam -Xms32m
vmparam -Xmx128m

# Colocar el Englobador AWT ligero en la vía de acceso de inicio
addbootpath ..\lib\lawt.jar
addbootpath ..\lib\TabbedPaneFix.jar

# Añadir todos los archivos JAR ubicados en el directorio parche, lib y lib/ext
addjars ..\patch
addjars ..\lib
addjars ..\lib\ext

# Incluir la API del Servlet 2.3 de Tomcat 4 en la vía de acceso a clases del
IDE
addpath ..\jakarta-tomcat-4.0.3\common\lib\servlet.jar

# Activar la integración shell
socket 8888

# Añadir todos los archivos de configuración ubicados en el directorio lib/ext
includedir ..\lib\ext

# JBuilder necesita tener acceso al entorno
exportenv

# Iniciar JBuilder mediante la clase principal
mainclass com.borland.jbuilder.JBuilder
```

Al crear ejecutables nativos para sus aplicaciones en el Creador de recopilatorios o en el Creador de ejecutables nativos de JBuilder, puede personalizar el funcionamiento de inicio de la aplicación con un archivo de configuración en la ficha Ejecutables. Por ejemplo, puede pasar determinados parámetros de ejecución y de la MV a su aplicación antes de que se inicie. Para obtener más información sobre la creación de

ejecutables nativos, consulte “[Creación de ejecutables con el Creador de ejecutables nativos](#)” en la página 15-32.

## Inicio de la MV

---

El comando de la línea de comandos para iniciar la MV tiene la siguiente forma:

```
<javapath> [-Xbootclasspath/p:<bootpath>]  
[-classpath  
classpath>] <vmparams> <mainclass> {params}
```

Los elementos entre <corchetes angulares> representan valores derivados del archivo de configuración, y los que están entre [corchetes] representan las partes opcionales de la línea de comandos que se omiten si los elementos relevantes de la vía de acceso no están definidos en el archivo de configuración. El componente {params} es por defecto un duplicado de los parámetros de la línea de comandos del programa de inicio, pero algunas directivas del archivo de configuración pueden alterarlo.

## Requisitos del archivo de configuración

---

El archivo de configuración debe cumplir determinadas especificaciones para que se pueda analizar correctamente.

### Tipo de archivo y ubicación

---

El archivo de configuración debe ser un archivo de texto sin formato. Debe encontrarse en el mismo directorio que el programa de inicio, y tener el mismo nombre pero con la extensión .config. Por ejemplo, bcj.exe, ubicado en el directorio bin de JBuilder, tiene un archivo de configuración denominado bcj.config. Si el programa de inicio no encuentra el archivo de configuración, se abre como un archivo y busca el archivo .config que tiene almacenado.

### Líneas en blanco y comentarios

---

Las líneas en blanco y las que empiezan con el carácter # se omiten para permitir que el archivo de configuración se estructure y se documente.

### Convenciones de las vías de acceso

---

Todas las vías de acceso del archivo de configuración pueden ser relativas o absolutas. Las vías de acceso relativas son relativas para el directorio

que contiene el archivo de configuración y el programa de inicio. Puede utilizar “..” en la vía de acceso para desplazarse hasta el directorio superior. Todos los separadores de las vías de acceso deben ser barras diagonales, sin tener en cuenta el separador de vías de acceso estándar de la plataforma local.

## Directivas

---

En el archivo de configuración se pueden especificar las siguientes directivas. Algunas son necesarias, y otras optativas.

### **javapath**

---

La directiva `javapath` ofrece la ubicación exacta y el nombre del intérprete Java. El que el programa de inicio sea un ejecutable o una librería compartida depende del programa de inicio. Por ejemplo, un programa de inicio Win32 necesita una de las siguientes:

```
javapath ../jdk14/bin/java.exe  
javapath ../jdk14/jre/bin/client/jvm.dll
```

Si no se encuentra una directiva `javapath`, el programa de inicio intenta determinar una ubicación por defecto de la MV de Java de acuerdo con el modo adecuado de esa plataforma.

Los ejecutables específicos de la plataforma buscan el JDK instalado en la siguiente ubicación:

- Windows: Registro.
- Linux/Solaris variable de entorno de JAVA\_HOME y en la vía de acceso del usuario.
- Mac OS X ubicación predeterminada para el JDK.

**Nota** Puede redefinir este comportamiento por defecto si especifica la ubicación del JDK en un archivo de configuración personalizado. El archivo ejecutable aparece entonces en la ubicación especificada. Si desea obtener más información sobre los archivos de configuración, consulte el siguiente paso.

La directiva `javapath` es necesaria si el JDK no está instalado en la ubicación habitual de la plataforma.

### **mainclass**

---

La directiva `mainclass`, necesaria, ofrece el nombre completo de la clase que se utiliza para iniciar la aplicación. Por ejemplo:

```
mainclass com.borland.jbuilder.JBuilder
```

## **addpath**

---

La directiva `addpath` añade una vía de acceso única a la vía de acceso a clases utilizada para iniciar la aplicación. Por ejemplo:

```
addpath ..\lib\jbuilder.jar
```

Compruebe cómo se aplican las siguientes reglas adicionales:

- Las vías de acceso que se refieren a un directorio o a un archivo que no existe no se añaden.
- Las vías de acceso que ya están en la vía de acceso a clases, la vía de acceso de inicio y la vía de acceso a excluir no se añaden.
- Las vías de acceso que contengan espacios se colocan automáticamente entre comillas al generar la línea de comandos.

## **addjars**

---

La directiva `addjars` añade todos los archivos JAR del directorio especificado a la vía de acceso a clases utilizada para iniciar la aplicación. Por ejemplo:

```
addjars ..\lib
```

Las mismas reglas que se aplican a la directiva `addpath` se aplican a cada una de las vías de acceso añadidas como resultado de la directiva `addjars`.

## **addbootpath**

---

La directiva `addbootpath` añade una única vía de acceso a la vía de acceso de inicio utilizada para iniciar la MV de Java. Por ejemplo:

```
addbootpath ..\lib\lawt.jar
```

Compruebe cómo se aplican las siguientes reglas adicionales:

- Las vías de acceso que se refieren a un directorio o a un archivo que no existe no se añaden.
- Las vías de acceso que ya se encuentren en la vía de acceso a clases se eliminan de la vía de acceso a clases y se añaden a continuación a la vía de acceso de inicio.
- Las vías de acceso que ya están en la vía de acceso de inicio y la vía de acceso a excluir no se añaden.
- Las vías de acceso que contengan espacios se colocan automáticamente entre comillas al generar la línea de comandos.

## addbootjars

---

La directiva `addbootjars` añade todos los archivos JAR del directorio especificado a la vía de acceso de inicio utilizada para iniciar la aplicación. Por ejemplo:

```
addbootjars ..\lib
```

Las mismas reglas que se aplican a la directiva `addbootpath` se aplican a cada una de las vías de acceso añadidas como resultado de la directiva `addbootjars`.

## addskippath

---

La directiva `addskippath` define una única vía de acceso que no se debe añadir nunca a las vías de acceso a clases o de inicio utilizadas para iniciar la MV de Java. Esto resulta de gran utilidad para eliminar las vías de acceso individuales que, de otro modo, añadirían las directivas `adjars` o `addbootjars`. Por ejemplo:

```
addskippath ..\lib\dbswing.jar
```

Compruebe cómo se aplican las siguientes reglas adicionales:

- La vía de acceso se elimina de la vía de acceso a clases si ya se ha añadido.
- La vía de acceso se elimina de la vía de acceso de inicio si ya se ha añadido.

## vmparam

---

La directiva `vmparam` proporciona parámetros que se pasan directamente a la MV de Java cuando se inicia. Por ejemplo, la siguiente directiva establece los tamaños mínimos y máximos de memoria dinámica en 8MB y 128MB respectivamente:

```
vmparam -Xms8m -Xmx128m
```

Los efectos de esta directiva son acumulativos. Cada uno de los sucesos siguientes se añade al conjunto de parámetros que se pasan a la MV con espacios insertados automáticamente entre los parámetros.

## include

---

La directiva `include` hace que se analice el contenido del archivo antes de continuar con el archivo de configuración actual. Esta directiva no se debe utilizar para anidar archivos de configuración en un número arbitrario de

niveles. El programa de inicio declara un error y finaliza si no se puede leer el nombre del archivo.

```
include jdk.config
```

Al igual que todas las vías de acceso del archivo de configuración, la vía de acceso puede ser relativa o absoluta. Consulte “[Convenciones de las vías de acceso](#)” en la página A-3.

## includedir

---

La directiva `includedir` hace que todos los archivos con la extensión `.config` del directorio especificado se procesen como con la directiva `include`.

```
includedir ../lib/ext
```

Al igual que todas las vías de acceso del archivo de configuración, la vía de acceso puede ser relativa o absoluta. Consulte “[Convenciones de las vías de acceso](#)” en la página A-3.

## copyenv

---

La directiva `copyenv` hace que se exponga el contenido de una variable de entorno, definiendo la variable de entorno Java correspondiente.

```
copyenv PROMPT
```

La variable Java tiene un prefijo para evitar conflictos de espacios de nombres, por lo que esta directiva define una variable de entorno Java de nombre `borland.copyenv.PROMPT`, cuyo valor se deriva originariamente de la variable de entorno `PROMPT`.

## exportenv

---

La directiva `exportenv` hace que se exponga el contenido de todas las variables de entorno del sistema, escribiéndolas en un archivo temporal. Cada llamada del programa de inicio crea un archivo único utilizando el formato de archivo de Java `.properties` para representar una colección completa de pares nombre/valor para todas las variables de entorno.

```
exportenv
```

La variable de entorno Java `borland.exportenv` está configurada para contener el nombre del archivo en el que se ha escrito el entorno. Una vez que se cierra la MV de Java, el programa de inicio se encarga de eliminar el archivo temporal.

## addparam

---

La directiva `addparam` añade un nuevo parámetro o conjunto de parámetros al conjunto existente de parámetros de aplicaciones.

```
addparam <parámetro>
```

## clearparams

---

La directiva `clearparams` descarta el conjunto existente de parámetros de aplicaciones. Esta directiva se utiliza normalmente junto con las siguientes directivas `addparam`.

```
clearparams
```

## restartcode

---

La directiva `restartcode` especifica un código de salida para el proceso Java que se debe interpretar como una solicitud para reiniciar el proceso de inicio. Esta directiva es utilizada normalmente por una aplicación que necesita realizar cambios en su configuración, por lo que el programa de inicio debe volver a leer los archivos de configuración como si se iniciase por primera vez.

```
restartcode 22
```

**Advertencia** Debe tener cuidado al utilizar esta directiva, ya que puede derivar fácilmente en un ciclo interminable de inicios de la MV. Por motivos de seguridad, el código de salida de reinicio 0 nunca se interpreta como una solicitud de reinicio, incluso si se encuentra una directiva explícita `restartcode 0`.

# Funciones optativas de inicio en un solo paso

---

Para posibilitar una distribución sencilla de las aplicaciones, las implementaciones del programa de inicio pueden admitir dos funciones optativas:

- 1 Determinar una ubicación por defecto de la MV de Java en ausencia de una directiva `javavm`.
- 2 Si no existe un archivo `<exename>.config` y el programa de inicio parece tener el formato de un archivo JAR:
  - a Añadir automáticamente el ejecutable a la vía de acceso como si lo hiciera una directiva explícita `addpath`
  - b Leer el comentario del ejecutable o del archivo JAR e interpretarlo como un archivo de configuración

Estas dos opciones optativas hacen posible crear un ejecutable autónomo que contenga el programa de inicio, el código Java, y toda la información necesaria, incluido el nombre de la clase principal. Todo lo que hace es encadenar el ejecutable de inicio con un archivo JAR cuyo comentario tiene el formato de un archivo de configuración.



# B

## Las herramientas de línea de comandos

JBuilder cuenta con las siguientes herramientas de línea de comandos:

**bmj** y **bcj** son funciones  
de JBuilder SE y  
Enterprise

- Interfaz de la línea de comandos de JBuilder
- Make de Borland para Java (**bmj**)
- El compilador de Borland para Java (**bcj**)

El JDK incluye las siguientes herramientas de línea de comandos:

- **javac** - compila el lenguaje de programación Java.
- **java** - inicia las aplicaciones Java.
- **jar** - gestiona los archivos recopilatorios Java (JAR).
- **javadoc** - extrae los comentarios del código y genera documentación HTML a partir de ellos.
- **appletviewer** - permite ejecutar applets sin utilizar un visualizador de Web.
- **native2ascii** - convierte archivos de caracteres de codificación nativa en archivos con secuencias de escape de Unicode.

### Consulte

- La documentación de Sun Tools en <http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html>.

## Definición de la vía de acceso a clases para herramientas de línea de comandos

---

La vía de acceso a clases indica a las herramientas de Java dónde se encuentran las clases que no forman parte de la plataforma Java. Es posible definir la vía en las clases que contienen la opción **-classpath**, y también se puede definir la variable de entorno **CLASSPATH** descrita a continuación. La opción **-classpath** redefine provisionalmente la variable de entorno **CLASSPATH** para la sesión de línea de comandos actual. Es recomendable utilizar **-classpath**, ya que se puede definir para una aplicación sin influir sobre las otras.

Los directorios de classpath se muestran separados por dos puntos en UNIX y por punto y coma en Windows. Debe incluir siempre las clases de sistema al final de la vía de acceso. La vía de acceso a clases se utiliza también para buscar los archivos fuente si no se ha especificado ninguna vía de acceso a ellos.

Si desea más información sobre las vías de acceso a clases, consulte “Setting the classpath” en la documentación de Java.

Si desea más información sobre las vías de acceso a clases, consulte “[Cómo construye JBuilder las vías de acceso](#)” en la página 4-10 y “[Localización de los archivos](#)” en la página 4-13.

### Opción -classpath

---

La opción **-classpath** permite definir provisionalmente la vía de acceso a clases.

- **UNIX**

La opción **-classpath** tiene la siguiente forma:

```
% jdkTool -classpath path1:path2
```

- **Windows**

La opción **-classpath** tiene la siguiente forma:

```
C:>jdkTool -classpath path1;path2
```

## Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos

---

La opción **-classpath** de línea de comandos sólo modifica temporalmente la vía de acceso a clases y no interfiere con otras aplicaciones. No obstante, se puede configurar de forma permanente la variable de entorno **CLASSPATH**.

Si desea más información sobre la opción **-classpath** y la variable de entorno **CLASSPATH**, consulte “Setting the classpath” en la documentación de Java.

## UNIX: variable de entorno CLASSPATH

Para visualizar la variable **CLASSPATH**:

- 1 Abra una ventana shell de línea de comandos.
- 2 Presente el valor actual de la variable de entorno **CLASSPATH** mediante el siguiente formato de línea de comando:
  - en el shell csh:

```
env
```

- en el shell sh:

```
CLASSPATH
```

Para definir la variable de entorno **CLASSPATH**,

- 1 Abra una ventana shell de línea de comandos.
  - 2 Defina la variable de entorno **CLASSPATH** con el siguiente formato de línea de comandos:
    - en el shell csh:
- ```
setenv CLASSPATH path1:path2
```
- en el shell sh:
- ```
CLASSPATH = path1:path2
export CLASSPATH
```

Para eliminar la variable **CLASSPATH**,

- 1 Abra una ventana shell de línea de comandos.
  - 2 Borre la variable de entorno de **CLASSPATH** con el siguiente formato de línea de comando:
    - en el shell csh:
- ```
unsetenv CLASSPATH
```
- en el shell sh:
- ```
unset CLASSPATH
```

## Windows: variable de entorno CLASSPATH

Para visualizar la variable **CLASSPATH** actual, utilice el comando **set**.

```
C:> set
```

Para definir la variable de entorno **CLASSPATH**:

- 1 Abra una ventana DOS.

**2** Modifique la variable de entorno CLASSPATH con el comando set.

```
set CLASSPATH=path1;path2 ...
```

Las vías de acceso deben comenzar con la letra de la unidad, por ejemplo C:\.

Si desea borrar la vía de acceso, puede eliminar los valores de la variable CLASSPATH de la siguiente manera:

```
C:> set CLASSPATH=
```

Este comando elimina los valores de la variable CLASSPATH sólo para la sesión DOS actual. Para que los valores de CLASSPATH sean los correctos, deberá modificar o borrar los valores de la configuración de inicio.

Si la variable CLASSPATH queda establecida al iniciar el sistema, se deberá buscar en sitios diferentes según el sistema operativo.

- Windows NT: Para que se abra el cuadro Propiedades del sistema, seleccione Inicio | Configuración | Panel de control | Sistema. Seleccione la pestaña Entorno y modifique la variable CLASSPATH en la sección Variables de usuario.
- Windows 2000: Para que se abra el cuadro Propiedades del sistema, seleccione Inicio | Configuración | Panel de control | Sistema. Haga clic sobre la pestaña Avanzado y pulse el botón Variables de entorno para modificar la variable CLASSPATH en la sección Variables de usuario. Si no está conectado en el equipo local como administrador, sólo puede cambiar variables de usuario.
- Windows XP: Para que se abra el cuadro Propiedades del sistema, seleccione Inicio | Configuración | Panel de control | Sistema. Pulse la pestaña Avanzadas y, a continuación, el botón Variables de entorno. Si no está conectado en el equipo local como administrador, sólo puede cambiar variables de usuario.

## Interfaz de la línea de comandos de JBuilder

---

JBuilder cuenta con una interfaz de línea de comandos que incluye argumentos como:

- Creación de proyectos
- Visualización de información de la configuración
- Visualización del administrador de licencias
- Desactivación de la pantalla de presentación
- Activación del modo de depuración con comentarios para autores de OpenTools

**Nota** Estos argumentos varían según la edición de JBuilder.

JBuilder ejecuta su propio programa de inicio, que es un ejecutable. El ejecutable puede pasar argumentos a JBuilder.

## Acceso a una lista de opciones

---

Para acceder a la lista de argumentos disponibles en su versión de JBuilder, abra una ventana de línea de comandos, sitúese en el directorio bin de JBuilder y escriba jbuilder -help.

```
/<jbuilder>/bin> jbuilder -help
```

JBuilder presenta una lista de argumentos disponibles:

Argumentos de línea de comandos disponibles:

- build: Generar proyectos de JBuilder (no disponible en Personal)
- help: Mostrar la ayuda en la línea de comandos
- info: Mostrar información de configuración
- license: Mostrar el administrador de licencias
- nosplash: Desactivar pantalla de inicio
- verbose: Mostrar el diagnóstico de carga de OpenTools.

Si desea obtener más información sobre cada argumento, escriba jbuilder -help <nombre\_del\_argumento>, tal y como aparece en este ejemplo.

```
/<jbuilder>/bin> jbuilder -help info
```

JBuilder devuelve esta información:

-info

Muestra la información sobre la configuración del sistema durante el inicio.

También es posible presentar una lista de argumentos tal y como se muestra en este ejemplo:

```
/<jbuilder>/bin> jbuilder -help info nosplash verbose
```

## Sintaxis

---

```
jbuilder [argumentos]
```

## Opciones

---

- **-build** <args>

Genera uno o más proyectos JBuilder suministrados como argumentos. Las configuraciones se toman directamente de los archivos de proyecto correspondientes. Se pueden definir destinos, que se ejecutan en el orden por el que aparecen. Si no se indica ninguno, Ejecutar Make es el destino por defecto.

Entre los argumentos **-build**, los proyectos se distinguen de los destinos por la extensión .jpx o .jpr. Se supone que todos los

**Es una característica de  
JBuilder SE y Enterprise.**

argumentos que terminen por .jpx o .jpr son proyectos. Todos los argumentos que no tengan la extensión .jpx o .jpr son nombres de tipos.

- Nota** Si un proyecto falla al completarse, los proyectos restantes no se generan.

El comando tiene esta forma:

```
jbuilder -build <project1.jpx> [ [<target1> <target2> ...]
[<project2.jpx> [<target3> ...] ] ... ]
```

Por ejemplo:

```
jbuilder -build myproject.jpx rebuild
```

```
jbuilder -build myproject.jpx clean make myotherproject.jpx
```

- Nota** Si el proyecto no se encuentra en el directorio actual, incluya la vía de acceso completa en el archivo de proyecto. Por ejemplo, si los proyectos están en el directorio /usuario/nombreusuario/ la vía de acceso completa sería:

```
jbuilder -build /usuario/nombreusuario/myproject.jpx clean make
/usr/username/myotherproject.jpx
```

El proceso de generación desde la línea de comandos presenta los errores y las advertencias de texto que pueden redirigirse a otro proceso o archivo. Por ejemplo, en las plataformas Windows, puede redirigirse a un archivo de texto del siguiente modo:

```
jbuilder -build myproject.jpx > myproject.txt
```

El proceso de generación desde la línea de comandos devuelve un código de resultado que indica que si hay fallos o si es correcto. Si la generación es correcta, se devuelve un valor cero. Si no lo es, se devuelve un valor distinto de cero. El valor real declarado depende del error.

En este ejemplo, el archivo .BAT de Windows genera el proyecto y refleja un mensaje de error o de éxito:

```
rem This has to be in a .BAT file:
jbuilder -build myproject.jpx
if not errorlevel 0 echo ERROR
if errorlevel 0 echo SUCCESS
rem End of .BAT file
```

Además, el proceso de generación desde la línea de comandos permite automatizar los procesos con otras herramientas de la línea de comandos. Por ejemplo, puede ejecutar tareas más complicadas, como generar otro proyecto si la generación es correcta y, a continuación, copiar los archivos o cancelar el archivo por lotes si falla la generación.

Consulte la documentación de su sistema operativo si desea obtener más información sobre los scripts o programas por lotes que admite su sistema.

- **-help <args>**

Enumera los argumentos de línea de comando de JBuilder disponibles.

Cuando se utiliza sin argumentos, **-help** muestra una lista de argumentos de línea de comandos reconocidos con breves descripciones. Cuando se llama a help con uno o varios argumentos se obtiene una descripción más detallada de los comandos especificados y sus argumentos.

```
-help <argumento1> <argumento2> <argumento3>
```

Los argumentos han de escribirse sin guión inicial.

- **-info**

Muestra la información sobre la configuración del sistema durante el arranque.

- **-license**

Muestra el administrador de licencias en lugar de iniciar JBuilder.

- **-nosplash**

Desactiva la pantalla de presentación que se muestra cuando se lanza JBuilder.

- **-verbose <args>**

Muestra el diagnóstico de carga de OpenTools.

## El compilador de Borland para Java (bcj)

---

**Es una función de  
JBuilder SE y Enterprise.**

### Sintaxis

```
bcj [ opciones ] {file.java}
```

### Descripción

El compilador de Borland para Java (**bcj**) compila código fuente Java para crear bytecodes Java desde la línea de comandos. **bcj** genera el programa Java en forma de archivos .class que contienen bytecodes, que son el código máquina de la máquina virtual Java. La compilación de un archivo fuente produce un archivo .class para cada declaración de clase o de interfaz. Cuando se ejecuta el programa de Java resultante en una plataforma concreta, como por ejemplo Windows NT, el intérprete de Java de dicha plataforma ejecuta los bytecodes de los archivos .class.

**bcj** compila el archivo .java especificado y todos los archivos indicados en la línea de comandos. **bcj** compila el archivo .java especificado, independientemente de que el archivo .class correspondiente esté o no desfasado. Un archivo .class desfasado es aquél que no se ha generado al compilar la versión actual del archivo fuente .java correspondiente. Los archivos .java importados que ya tienen archivos .class no vuelven a compilarse, aunque sus archivos .class no estén actualizados; después de utilizar **bcj**, algunas clases importadas pueden seguir teniendo archivos .class desfasados.

**bcj** no comprueba las dependencias entre archivos. Para obtener más información sobre **bmj** y la comprobación inteligente de dependencias, consulte “[Make de Borland para Java \(bmj\)](#)” en la página B-12 y “[Comprobación inteligente de dependencias](#)” en la página 5-2.

Para ver la sintaxis y la lista de opciones de la línea de comandos, introduzca el comando **bcj** sin argumentos que se encuentra en <jbuilder>/bin.

En ocasiones se debe utilizar la opción **-classpath** o definir la variable de entorno **CLASSPATH** para la línea de comandos, con el fin de encontrar las clases necesarias.

### Consulte

- “[Compilación desde la línea de comandos](#)” en la página 5-11
- “[Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos](#)” en la página B-2
- “[Setting the classpath](#)” en la documentación de las herramientas de Java

## Opciones

---

**Nota** Los directorios de las vías se muestran separados por dos puntos en UNIX y por punto y coma en Windows. Los siguientes ejemplos representan la plataforma UNIX.

- **-classpath** vía

Vía de acceso en la que se buscarán las clases. Redefine el valor por defecto o la variable de entorno **CLASSPATH**. Debe incluir siempre la vía de salida al principio de la vía. La vía de acceso a clases se utiliza también para buscar los archivos fuente si no se ha especificado ninguna vía de acceso a ellos. Por ejemplo:

```
bcj -classpath jbproject/testing/classes/test3:  
      jbproject/project1/classes tester.java
```

- **-d** directorio

El directorio raíz de la jerarquía de archivos de clase (destino). También se denomina "vía de salida".

Por ejemplo, la siguiente sentencia:

```
bcj -d jbproject/project1/classes tester.java
```

hace que los archivos de las clases definidas en el archivo fuente `tester.java` se guarden en el directorio `jbproject/project1/classes/test/test3`, suponiendo que `tester.java` contenga la siguiente sentencia de paquete: `package test.test3;`

Los archivos se leen desde la vía de acceso a clases y se escriben en el directorio de destino. El directorio de destino puede ser parte de la vía de acceso a clases. El destino por defecto coincide con la estructura del paquete en los archivos fuente y se inicia desde el directorio raíz de archivos fuente.

- **-deprecation**

Visualiza todas las clases, métodos, propiedades, variables y sucesos desaconsejados empleados en la API.

Si se produce una advertencia, durante la compilación, de que se han utilizado API desaconsejadas, puede ver estas últimas activando esta opción.

- **-encoding nombre**

Puede especificar un nombre de codificación de archivo (o un nombre de ficha de códigos) para controlar cómo interpreta el compilador los caracteres que no pertenecen al conjunto de caracteres ASCII. Por omisión se utiliza el convertidor de codificación nativa por defecto de la plataforma. Para obtener más información, consulte “[Elección de una codificación nativa para el compilador](#)” en la página 16-13.

Por ejemplo, la siguiente sentencia:

```
bcj -encoding EUC_JP tester.java
```

compila `tester.java`. Se considera que todos los archivos de código fuente están codificados en el conjunto de caracteres EUC\_JP, que es el que se utiliza normalmente para los entornos UNIX japoneses. Se puede crear cualquier codificación aceptada por la plataforma Java 2. En <http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html#intl> puede ver una lista de todas las codificaciones disponibles.

- **-exclude nombre de la clase**

Excluye todas las llamadas a los métodos `static void` en el archivo `.class` seleccionado de una compilación. Esto también excluye la evaluación de los parámetros pasados a esos métodos.

Por ejemplo, la exclusión de la clase A elimina todas las llamadas a los métodos `static void` de A desde las OTRAS clases.

- **-g**

Genera toda la información de depuración en el archivo de clase, incluidas las variables locales. Por defecto, sólo se generan números de línea y la información del archivo fuente.

- **-g:none**

No genera ninguna información de depuración

- **-g:{lista de palabras clave}**

Genera algunos tipos de información de depuración. La lista de palabras clave es una lista de palabras separadas por comas, como, por ejemplo: bcj -g:source,lines

Entre las palabras clave se incluyen:

- **source**: información de depuración del archivo fuente.
- **lines**: información de depuración del número de línea.
- **vars**: información de depuración de la variable local.

- **-nowarn**

Compila sin mostrar advertencias.

- **-obfuscate**

El enmascaramiento hace que los programas sean menos vulnerables a las manipulaciones. Una vez descompilado el código confuso, el código fuente generado representa los símbolos privados mediante otros nombres de símbolo.

- **-quiet**

Compila sin mostrar ningún mensaje.

- **-source** versión

Habilita la compilación de código fuente que contiene órdenes.

- En la versión 1.4, esta opción de línea de comandos activa la palabra clave assert y la función de órdenes de JDK 1.4.
- En la versión 1.3 el compilador no acepta las órdenes.
- Si no se utiliza la opción **-source**, el comportamiento por defecto del compilador es el de la versión 1.3.

### Consulte

- “Assertion Facility” en <http://java.sun.com/j2se/1.4/docs/guide/lang/assert.html>

- **-sourcepath** vía

Vía de acceso en la que se buscarán los archivos fuente. Si no se ha especificado ninguna vía de acceso a archivos fuente, se buscan por medio de la vía de acceso a clases.

Al igual que la vía de acceso a clases, la vía de acceso a archivos fuente debe situarse en el directorio raíz del árbol de paquetes y no directamente en el directorio de los archivos fuente.

Por ejemplo, para compilar `tester.java`, que contiene la sentencia de paquete `paquete test.test3` y que se encuentra en `jbproject/project1/src/test/test3`, debe definir la vía de acceso de archivos fuente como `jbproject/project1/src` y no como `jbproject/project1/src/test/test3`.

A continuación, escriba lo siguiente:

```
bcj -sourcepath jbproject/project1/src
      jbproject/project1/src/test/test3/tester.java
      -d jbproject/project1/classes
```

- **-verbose**

Esta opción proporciona más información sobre la compilación, como por ejemplo, los archivos de clase que se cargan y su ubicación en la classpath, además de la siguiente información:

- Qué archivos fuente se compilan.
- Qué clases se están cargando.
- Qué clases se generan.

## Opciones de compilación cruzada

---

**bcj** acepta la compilación cruzada, que consiste en compilar las clases con un inicio y clases de extensión de una plataforma Java distinta. Utilice **bootclasspath** y **-extdirs** con el fin de realizar una compilación cruzada.

- **-target** versión

Restringe los archivos de clase, de modo que funcionen sólo con una versión de MV determinada.

**bcj** admite:

- 1.1 - Genera archivos de clase que se ejecuten con 1.1 y las MV del SDK de Java 2. Cuando seleccione esto como MV destino, sus archivos de clase pueden ser cargados por cualquier MV.
- 1.2 - Genera archivos de clase que pueden ejecutarse **únicamente** en las MV del SDK de Java 2, versiones 1.2 y posteriores, pero **no** en las MV 1.1. Ésta es la opción por defecto.
- 1.3 - Genera archivos de clase que pueden ejecutarse **únicamente** en las MV del SDK de Java 2, versiones 1.3 y posteriores, pero **no** en las MV 1.1 ni 1.2.

- 1.4 - Genera archivos de clase que se ejecutan **únicamente** en máquinas virtuales de la versión 1.4 y posteriores del SDK de Java 2, pero **no** se ejecutan en 1.1, 1.2 ni 1.3.
- **-bootclasspath** vía de acceso a clases de arranque  
Compilación cruzada con las clases de inicio especificadas. Se pueden utilizar directorios, recopilatorios JAR y ZIP.
- **-extdirs** directorios  
Compilación cruzada con los directorios de extensión determinada. Se buscan archivos de clase para todos los archivos JAR de los directorios elegidos.

## Opciones de MV

---

- **-J** opción

Pasa opciones al programa de inicio de `java` al que llama **bcj**. Por ejemplo, **-J-Xms48m** asigna a la memoria inicial 48 megabytes, que es lo habitual cuando **-J** pasa opciones a la MV subyacente que ejecuta las aplicaciones escritas en Java.

Observe que **CLASSPATH**, **-classpath**, **-bootclasspath** y **-extdirs** no especifican las clases utilizadas para ejecutar **bcj**. Si debe hacerlo, utilice la opción **-J** con el fin de pasar opciones al método de inicio de **bcj**.

## Make de Borland para Java (bmj)

---

Es una función de JBuilder SE y Enterprise.

### Sintaxis

---

`bmj [opciones] rootClasses`

### Descripción

---

La utilidad Make de Borland para Java (**bmj**), que es el compilador por defecto del IDE y un compilador de línea de comandos, compila el código fuente Java a bytecodes Java desde la línea de comandos y comprueba las dependencias con el objeto de determinar qué archivos necesitan una nueva compilación. **bmj** genera el programa Java en forma de archivos `.class` que contienen bytecodes, que son el código máquina de la máquina virtual Java. La compilación de un archivo fuente produce un archivo `.class` para cada declaración de clase o de interfaz. Cuando se ejecuta el programa de Java resultante en una plataforma concreta, como por ejemplo Windows NT, el intérprete de Java de dicha plataforma ejecuta los bytecodes de los archivos `.class`.

**bmj** busca archivos de dependencia en la vía de acceso a clases. Si se especifica un conjunto de archivos fuente, es posible que no se recompilen todos ellos. Por ejemplo, podría determinarse que los archivos de clase están actualizados si se han guardado pero no se han editado desde la última compilación. Es posible forzar la recompilación por medio de la opción **-rebuild**.

Para verificar un conjunto (o gráfico) de clases interdependientes, basta con llamar a **bmj** en la clase raíz (o en varias clases raíz, si no está una debajo de otra). Es posible especificar las clases raíz utilizando nombres de clase, de paquete, nombres de fuentes que declaren clases o una combinación de éstos.

Es posible que necesite definir la variable de entorno **CLASSPATH** de la línea de comandos, de tal forma que se encuentren las clases necesarias.

Para ver la sintaxis y la lista de opciones de la línea de comandos, introduzca el comando **bmj** sin argumentos que se encuentra en el directorio `<jbuilder>/bin`. Muchas de estas opciones también se pueden especificar en el IDE de JBuilder en la ficha General y en la ficha Generar de Propiedades de proyecto (Proyecto | Propiedades de proyecto).

### Consulte

- “Comprobación inteligente de dependencias” en la página 5-2
- “Compilación desde la línea de comandos” en la página 5-11
- “Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos” en la página B-2
- “Setting the classpath” en la documentación de las herramientas de Java
- “El compilador de Borland para Java (bcj)” en la página B-7

### Opciones

---

**Nota** Los directorios de las vías se muestran separados por dos puntos en UNIX y por punto y coma en Windows. Los siguientes ejemplos representan la plataforma UNIX.

- **-classpath** vía

La vía de acceso que se utiliza para encontrar archivos de clases y de dependencia. Redefine el valor por defecto o la variable de entorno **CLASSPATH**. Debe incluir siempre la vía de salida al principio de la vía. La vía de salida es el directorio raíz de la jerarquía de archivos de clase. La vía de acceso a clases se utiliza también para buscar los archivos fuente si no se ha especificado ninguna vía de acceso a ellos.

Por ejemplo:

```
bmj -classpath jbproject/testing/classes/test3:  
jbproject/project1/classes tester.java
```

- **-d directorio**

El directorio raíz de la jerarquía de archivos de clase (destino). También se le denomina vía de salida.

Por ejemplo, la siguiente sentencia:

```
bmj -d jbproject/project1/classes tester.java
```

hace que los archivos de las clases definidas en el archivo fuente `tester.java` se guarden en el directorio `jbproject/project1/classes/test/test3`, suponiendo que `tester.java` contenga la siguiente sentencia de paquete: `package test.test3;`

El archivo de dependencias actualizado, `test.test3.dep2`, se guarda en `jbproject/project1/classes/package cache`.

Los archivos se leen desde la vía de acceso a clases y se escriben en el directorio de destino. El directorio de destino puede ser parte de la vía de acceso a clases. El destino por defecto de los archivos de clase coincide con la estructura de paquetes de los archivos origen y se inicia desde el directorio raíz de archivos fuente. El destino por defecto de los archivos de dependencia coincide con la estructura de paquetes y se inicia en el directorio actual.

- **-deprecation**

Visualiza todas las clases, métodos, propiedades, variables y sucesos desaconsejados empleados en la API.

Si se produce una advertencia, durante la compilación, de que se han utilizado API desaconsejadas, puede ver estas últimas activando esta opción.

- **-encoding nombre**

Puede especificar un nombre de codificación de archivo (o un nombre de ficha de códigos) para controlar cómo interpreta el compilador los caracteres que no pertenecen al conjunto de caracteres ASCII. Por omisión se utiliza el convertidor de codificación nativa por defecto de la plataforma. Para obtener más información, consulte “[Elección de una codificación nativa para el compilador](#)” en la página 16-13.

Por ejemplo, la siguiente sentencia:

```
bmj -encoding EUC_JP tester.java
```

compila `tester.java` y todos los archivos `.java` importados directamente que no tengan un archivo `.class`. Se considera que todos los archivos de código fuente están codificados en el conjunto de caracteres EUC\_JP, que es el que se utiliza normalmente para los entornos UNIX japoneses. Se puede crear cualquier codificación aceptada por la plataforma Java

2. En <http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html#int1> puede ver una lista de todas las codificaciones disponibles.

- **-exclude** nombre de la clase

Excluye todas las llamadas a los métodos static void en el archivo .class seleccionado de una compilación. Esto también excluye la evaluación de los parámetros pasados a esos métodos.

Por ejemplo, la exclusión de la clase A elimina todas las llamadas a los métodos static void de A desde las OTRAS clases.

- **-g**

Genera toda la información de depuración en el archivo de clase, incluidas las variables locales. Por defecto, sólo se generan números de línea y la información del archivo fuente.

- **-g:none**

No genera ninguna información de depuración.

- **-g:{lista de palabras clave}**

Genera algunos tipos de información de depuración. La lista de palabras clave es una lista de palabras separadas por comas, como, por ejemplo: bmj -g:source,lines

Entre las palabras clave se incluyen:

- **source**: información de depuración del archivo fuente.
- **lines**: información de depuración del número de línea.
- **vars**: información de depuración de la variable local.

- **-nocompile**

Verifica si las clases están actualizadas pero no compila ninguna. Es útil para verificar rápidamente si una clase o un paquete están actualizados. Se detiene en el primer archivo que deba volver a compilarse y comunica que "la clase <clase> necesita volver a compilarse debido a <motivo>".

- **-nowarn**

Compila sin mostrar advertencias.

- **-obfuscate**

El enmascaramiento hace que los programas sean menos vulnerables a las manipulaciones. Una vez descompilado el código confuso, el código fuente generado representa los símbolos privados mediante otros nombres de símbolo.

- **-quiet**

Compila sin mostrar ningún mensaje.

- **-rebuild**

Compila las clases raíz especificadas y sus archivos importados, independientemente de si se han modificado.

### Consulte

- “El comando Generar de nuevo” en la página 6-5

- **-source** versión

Habilita la compilación de código fuente que contiene órdenes.

- En la versión 1.4, esta opción de línea de comandos activa la palabra clave assert y la función de órdenes de JDK 1.4.
- En la versión 1.3 el compilador no acepta las órdenes.
- Si no se utiliza la opción -source, el comportamiento por defecto del compilador es el de la versión 1.3.

### Consulte

- “Assertion Facility” en <http://java.sun.com/j2se/1.4/docs/guide/lang/assert.html>

- **-sourcepath** vía

Vía de acceso en la que se buscarán los archivos fuente. Si no se ha especificado ninguna vía de acceso a archivos fuente, se buscan por medio de la vía de acceso a clases.

Al igual que la vía de acceso a clases, la vía de acceso a archivos fuente debe situarse en el directorio raíz del árbol de paquetes y no directamente en el directorio de los archivos fuente.

Por ejemplo, para crear tester.java, que contiene la sentencia de paquete test.test3 y que se encuentra en jbproject/project1/src/test/test3, debe definir la vía de acceso de archivos fuente como jbproject/project1/src/ y no como jbproject/project1/src/test/test3.

A continuación, escriba lo siguiente:

```
bmj -sourcepath jbproject/project1/src  
jbproject/project1/src/test/test3/tester.java  
-d jbproject/project1/classes
```

- **-sync**

Borra del directorio de archivos generados, los archivos de clase para los que no se disponía de archivos fuente antes de la compilación. Se debe especificar el directorio de vía de salida por medio de la opción -d.

Esta opción puede resultar útil para evitar que el compilador encuentre en el directorio de archivos generados un archivo de clase que no se pueda compilar desde el código fuente. (Probablemente ha eliminado

el archivo fuente o ha renombrado una de las clases declaradas en un archivo fuente.)

- **-verbose**

Esta opción proporciona más información sobre la compilación, como por ejemplo, los archivos de clase que se cargan y su ubicación en la classpath. Se presenta la siguiente información:

- La vía de acceso a clases, de archivos fuente y el directorio de archivos generados que se utilizan.
- Qué archivos fuente se compilan.
- Qué archivos de clases se cargan.
- Qué clases se generan.
- Qué archivos de dependencia se generan.

## Opciones de compilación cruzada

---

**bmj** acepta la compilación cruzada, que consiste en compilar las clases con un inicio y clases de extensión de una plataforma Java distinta. Utilice **bootclasspath** y **-extdirs** con el fin de realizar una compilación cruzada.

- **-target** versión

Restringe los archivos de clase de modo que funcionen con una versión de MV determinada.

**bmj** admite:

- 1.1 - Genera archivos de clase que se ejecuten con 1.1 y las MV del SDK de Java 2. Cuando seleccione esto como MV destino, sus archivos de clase pueden ser cargados por cualquier MV.
- 1.2 - Genera archivos de clase que pueden ejecutarse **únicamente** en las MV del SDK de Java 2, versiones 1.2 y posteriores, pero **no** en las MV 1.1. Ésta es la opción por defecto.
- 1.3 - Genera archivos de clase que pueden ejecutarse **únicamente** en las MV del SDK de Java 2, versiones 1.3 y posteriores, pero **no** en las MV 1.1 ni 1.2.
- 1.4 - Genera archivos de clase que se ejecutan **únicamente** en máquinas virtuales de la versión 1.4 y posteriores del SDK de Java 2, pero **no** se ejecutan en 1.1, 1.2 ni 1.3.
- **-bootclasspath** vía de acceso a la clase de arranque

Compilación cruzada con las clases de arranque especificadas. Se pueden utilizar directorios, recopilatorios JAR y ZIP.

- **-extdirs** directorios

Compilación cruzada con los directorios de extensión determinada. Se buscan archivos de clase para todos los archivos JAR de los directorios elegidos.

## Especificadores para las clases raíz

---

Las clases raíz se especifican de la siguiente forma:

```
{[-s] {source.java} | -p {package} | -c {class}}
```

- **-s** nombredearchivofuente

Indica que las clases raíz especificadas son las definidas en los archivos fuente indicados. Ésta es la interpretación por defecto.

Por ejemplo, la siguiente sentencia:

```
bmj -sourcepath jbproject/project1/src  
-s jbproject/project1/src/tester.java
```

es igual que

```
bmj -sourcepath jbproject/project1/src  
jbproject/project1/src/tester.java
```

Si enumera algunos paquetes con la opción **-p** antes de enumerar los archivos fuente, debe especificar la opción **-s**. Si enumera los archivos fuente antes que los paquetes y las clases, se da por supuesto el uso de la opción **-s** por lo que no es necesario especificarla.

- **-p** nombre de paquete

Nombre de los paquetes que se deben compilar.

Por ejemplo, la siguiente sentencia:

```
bmj -sourcepath jbproject/project1/src -p test.test3
```

ejecuta Make sobre todas las clases del paquete `test.test3` y todas las clases importadas.

- **-c** nombredeclase

Los nombres de las clases que se van a procesar con Make.

Por ejemplo, la siguiente sentencia:

```
bmj -sourcepath jbproject/project1/src  
-c test.test3.tester
```

ejecuta Make sobre la clase `tester` del paquete `test.test3` y todas las clases importadas.

Otro ejemplo es la sentencia siguiente:

```
bmj -sourcepath jbproject/project1/src tester.java -p package1 package2  
-s jbproject/project1/src/*.java
```

crea el archivo fuente `tester.java`, los paquetes `package1` y `package2` y todos los archivos java del directorio `jbproject/project1/src`.

El primer nombre de archivo fuente (`tester.java`) va antes que la opción `-p` (paquete), por lo que no hace falta especificar de modo explícito la opción `-s`, ya que `-s` está implícita. Sin embargo, si después de especificar la opción `-p` desea especificar otro nombre de archivo fuente, tiene que utilizar explícitamente la opción `-s`.

## Opciones de MV

---

- **-J**opción

Pasa opciones al método de inicio de `java` al que llama **bmj**. Por ejemplo, `-J-Xms48m` asigna a la memoria inicial 48 megabytes, que es lo habitual cuando `-J` pasa opciones a la MV subyacente que ejecuta las aplicaciones escritas en Java.

Observe que `CLASSPATH`, `-classpath`, `-bootclasspath` y `-extdirs` no especifican las clases utilizadas para ejecutar **bmj**. Si la ejecución es necesaria, utilice la opción `-J` para pasar opciones al método de inicio de **bmj**.



# Índice

## Símbolos

---

@author 14-5  
@deprecated 14-5  
@docRoot 14-5  
@exception 14-5  
@link 14-5  
@param 14-5  
@return 14-5  
@see 14-5  
@serial 14-5  
@serialData 14-5  
@serialField 14-5  
@since 14-5  
@tags lista 14-5  
@throws 14-5  
@version 14-5

## A

---

Abandonar inspección 8-38  
adicción de bibliotecas 4-2  
añadir  
    al proyecto 2-14  
    directorio de navegación al proyecto 2-18  
    sucesos  
        a JavaBeans 10-12, 10-15, 10-16  
        un JDK 2-23  
Ant 6-9  
    añadir tipos al menú Proyecto 6-21  
    archivos de generación 6-9  
    asignar valores a propiedades 6-15  
    Asistente para Ant 6-9  
    bibliotecas 6-17  
    configurar JDK 6-14  
    generar 6-9  
    importar proyectos Ant 6-9  
    opciones 6-17  
    tipos 6-9  
    tutorial 18-1  
    utilización del make de línea de comandos  
        bmj 6-15  
aplicaciones  
    compilar 5-1  
    comprobar 13-1  
    de ejemplo multilingüe 16-3  
    depurar 8-1  
    distribuidas  
        depurar 9-1  
        distribuir 15-16

distribuir 15-1, 15-11  
ejecutar 7-1  
generar 6-1  
applets  
    distribuir 15-1  
    ejecutar 7-1  
    Versiones JDK 15-9  
archivos  
    abrir fuera del proyecto 2-17  
    cambio 2-10  
    código fuente de stub 8-44  
    de clase 4-11  
        cómo los encuentra JBuilder 4-14  
        ubicación de los directorios 4-8  
de configuración A-1  
de generación  
    añadir al proyecto 6-9  
    Ant  
        añadir al proyecto 6-9  
    de grupos de proyectos 3-1  
    de imagen  
        guardar diagramas UML 11-21  
    de paquetes  
        documentación de Javadoc 14-20  
    de proyecto 2-1  
        establecer vías de acceso 4-10  
        guardar 2-11  
        vías de acceso 4-13  
        visualizar 4-14  
    de salida estándar para Javadoc 14-9  
    de salida JDK 1.1 para Javadoc 14-9  
descriptor 15-3  
    editar 15-27  
    ver en JBuilder 15-35  
ejecutables  
    crear 15-29  
    ejecutar 15-29  
ejecutar en el proyecto 7-2  
fuente  
    vía de acceso 4-10  
generados para Javadoc 14-18  
JAR 15-5  
    acceso de línea de comandos 15-6  
    actualización desde la línea de  
        comandos 15-7  
    añadir al proyecto 4-1  
    archivo descriptor 15-3  
    crear con el Constructor de  
        recopilatorios 15-5  
    crear con herramienta Jar 15-6

ejecución desde la línea de comandos 7-14, 15-27  
extraer 15-7  
ver contenido 15-7

**java**  
ubicación de los directorios 4-7

**recopilatorios**  
archivo descriptor 15-3  
archivos JAR 15-5  
archivos ZIP 15-5  
borrar 15-36  
crear archivos ejecutables 15-29  
crear para distribuir 15-2  
crear para la documentación 14-28  
definir opciones de la configuración de ejecución 15-31  
eliminar 15-36  
Java 15-5  
renombrar 15-36  
tipos aceptados por el Creador de recopilatorios 15-19  
ver contenido 15-7, 15-35

renombrar 2-17  
ubicación 4-13  
ver  
archivos de proyecto 4-14  
en JBuilder 2-10

**ZIP** 15-2  
añadir al proyecto 4-1  
crear con el Constructor de recopilatorios 15-19  
visualizar 15-7

asignar valores a propiedades  
en JavaBeans 10-4, 10-8  
proyectos 2-20

asistentes  
Ant 6-9  
Cadenas de recursos 16-6  
Conjunto de tests 13-7

**Creador**  
de ejecutables nativos 15-32  
de recopilatorios 15-18

**Grupos de proyectos** 3-1

**Javadoc** 14-9  
Montaje para comparación 13-10  
Montaje para JNDI 13-10

para  
Ant 6-9  
cliente de prueba EJB 13-8, 13-13  
extracción de recursos 16-6  
**JavaBean** 10-2  
**Javadoc** 14-1  
ámbito de documentación 14-13  
archivos de salida estándar 14-9

archivos de salida JDK 1.1 14-9  
cambiar propiedades para el nodo 14-25  
dar formato a los archivos de salida 14-9  
directorio de salida 14-11  
nombre del nodo  
documentación 14-11  
opciones de generación 14-11  
opciones de línea de comandos 14-14  
la configuración de Cactus 13-12  
proyectos  
crear proyectos 2-2  
Paso 1 definir raíz, tipo y plantilla 2-3  
Paso 2 establecer vías de acceso, bibliotecas, JDK 2-4  
Paso 3 opciones generales del proyecto 2-5  
tareas externas de generación 6-19  
tests 13-6

para conjuntos de tests 13-7  
Proyecto para código existente 2-8  
Tareas externas de generación 6-19  
Test 13-6

aspectos relativos a la distribución 15-8  
applets  
escritas con JDK 1.1.x o Java 2 15-9  
y aplicaciones 15-10  
bibliotecas 15-9  
bibliotecas de CLASSPATH 15-8  
redistribución de clases 15-16  
tiempo de descarga de archivos 15-11

assertEquals() 13-6  
assertNotNull() 13-6  
assertTrue() 13-6  
ayuda inmediata  
depurador 8-30  
diagramas UML 11-14

## B

---

**barras**  
de estado  
depurador 8-27  
progreso de la generación 7-2

de herramientas  
depurador 8-27

**BeanInfo (clases)**  
crear 10-9  
generar automáticamente 10-10

**BeanInsight** 10-23

**BeansExpress** 10-1  
asignar valores a propiedades 10-4, 10-8  
borrar propiedades 10-8  
cambiar clases BeanInfo 10-11  
cambiar propiedades 10-7

crear JavaBeans 10-2  
bibliotecas  
añadir  
    al proyecto 2-4  
    proyectos como necesarias 4-5  
de libre distribución 15-9  
definición 4-1  
distribuir 15-9  
editar 4-5  
establecer vías de acceso 2-25  
incluir referencias en diagramas UML 11-20  
mostrar listas 4-6  
necesarias  
    añadir proyectos 3-5, 4-5  
bloque de código  
    insertar en sentencia try/catch 12-28  
Borland  
asistencia  
    a desarrolladores 1-7  
    técnica 1-7  
contacto 1-7  
e-mail 1-9  
grupos de noticias 1-8  
informar sobre errores 1-9  
recursos en línea 1-7  
    World Wide Web 1-8  
borrar 2-16  
buscar  
    definición  
        campo 12-8  
        class 12-8  
        method 12-8  
        variable 12-8  
    referencias 12-9  
        a campo 12-9  
        a class 12-9  
        a method 12-9  
        a variable 12-9  
    en el visualizador UML 11-22

**C**

---

Cactus 13-2, 13-12  
configurar el proyecto 13-12  
ejecutar pruebas 13-14  
probar un EJB 13-8  
cadena de inicialización de Java 10-19  
cambiar  
    código durante la depuración 8-72  
    el JDK 2-23  
    valores de las variables 8-71  
campos  
    buscar definición 12-8  
    buscar referencias a 12-9

diagramas UML 11-7  
Javadoc  
    definir en el Asistente para proyectos 2-5  
carpeta Filtros de paquetes 6-27  
clases  
    actualizar después de compilar 8-72  
        (opciones) 8-74  
    Assert 13-6  
    buscar  
        definición 12-8  
        referencias a 12-9  
    diagramas UML 11-5, 11-7  
    documentación API para 14-2  
    ejecutable  
        configurar el proyecto 7-4  
        especificar en archivo descriptor 15-27  
Java  
    colecciones 10-1  
    PackageTestSuite 13-3  
    principal  
        definir 7-4  
    redistribución en la distribución 15-16  
    referencia 4-9  
    TestCase 13-1, 13-3  
        ampliar 13-5  
        setUp() 13-5  
        tearDown() 13-5  
    TestRecorder 13-10  
    TestSuite 13-1, 13-3  
codificaciones  
    añadir y redefinir 16-15  
    definición nativa 16-2  
    definir opciones 16-14  
    nativas  
        admitidas 16-15  
            admitidas 16-14  
            definir 2-5  
        nativas admitidas 16-13  
            definición 16-2  
        native 16-13  
código  
    código confuso en diagramas UML 11-3  
LegacyJ  
    depurar 8-30  
modificar durante la depuración 8-72  
mostrar desde un diagrama UML 11-14  
SQLJ  
    depurar 8-30  
comandos  
    Buscar definición 12-8  
    Ejecutar 5-5, 7-4  
        generar Ant 6-14  
        hasta el cursor 8-41  
        hasta el final del método 8-40

Make 6-4  
Eliminar punto de observación 8-69  
Generar de nuevo 6-5  
Limpiar 6-5  
Nuevo directorio de navegación 2-18  
Optimizar importaciones 12-15  
comentarios Javadoc 14-2  
añadir  
    para campos 14-3  
    para clases 14-3  
    para interfaces 14-3  
    para métodos 14-3  
colocación 14-3  
conflictos 14-8  
ejemplos de 14-2  
etiquetas @todo 14-8  
generar automáticamente etiquetas para 14-7  
usar etiquetas 14-5  
compiladores  
    bcj 5-11, B-7  
    bmj 5-11, B-12  
    de línea de comando  
        bcj B-7  
    de línea de comandos bcj B-7  
    definir en el IDE 5-9  
    definir opciones 5-7  
    javac 5-9  
    javac del proyecto 5-9  
    Make de Borland 5-9  
compiladores de línea de comando  
    bmj 5-11, B-12  
compilar  
    antes de perfeccionar 12-6  
    archivos fuente 5-3  
    barra de estado 7-2  
    comandos  
        Ejecutar 5-5  
        Make 6-4  
        Generar de nuevo 6-5  
        Limpiar 6-5  
    cómo encuentra JBuilder los archivos 4-14  
    comprobación inteligente de dependencias 5-2  
    con referencias de bibliotecas de proyecto 12-6  
configurar vía de salida 5-10  
copiar recursos en la vía de salida 6-29  
definir  
    MV destino 5-7  
    opciones 5-7  
    un compilador diferente 5-9  
descripción general 5-1  
desde la línea de comandos 5-11  
en el IDE  
    con javac 5-9  
    con javac del proyecto 5-9  
            con Make de Borland para Java 5-9  
            errores 5-5  
            al abrir proyectos 5-6  
            excluir paquetes 6-27  
            generación con archivos Ant 18-1  
            grupos de proyectos 6-6  
            parámetros de todo el proyecto 5-7  
            programas  
                con información de depuración 8-4  
                en Java 5-1  
            proyectos en un grupo de proyectos 5-11  
            tutorial 17-1  
componentes  
    multiplataforma 10-2  
    reutilizables 10-1  
comprobación  
    código del servidor 13-12  
    de dependencias 5-2, 5-7  
    inteligente de dependencias 5-2  
    un EJB 13-12  
configuraciones  
    bibliotecas 4-2  
    de ejecución  
        crear 7-9  
        definir 7-7  
        depurar 8-4  
        para comprobación de módulos 13-17  
    de Inspeccionar código  
        para clases sin archivo fuente 8-44  
    JDK 2-23  
    JDK en JBuilder 2-24  
    mostrar con argumentos de la línea de comandos B-4  
    tipos de ejecución 7-13  
configuraciones de ejecución  
    Test 13-3  
    tipo de generación 7-11  
conflictos 8-36  
conjuntos  
    de sucesos  
        creación personalizada 10-16  
    de tests 13-5  
convenciones  
    de la documentación 1-5, 1-6  
convenciones de nomenclatura  
    paquetes 4-9  
correspondencia  
    directorio-paquete 5-7  
    paquete-directorio 5-7  
creador  
    de ejecutables nativos 15-32  
    de recopilatorios  
        crear archivos ejecutables 15-29  
        crear archivos recopilatorios 15-5

- crear recopilatorio de documentación 14-28  
definir opciones de la configuración de ejecución 15-31  
distribuir archivos 15-18  
tipos de recopilatorios 15-19
- crear  
archivos de proyecto 2-13  
configuraciones de ejecución 7-7, 7-9  
doclet personalizado 14-30  
documentación API 14-1  
editores de propiedades 10-18  
JavaBeans 10-1, 10-2  
proyectos a partir de archivos 2-8  
tests 13-6
- cuadros de diálogo  
Archivos modificados 8-74  
Nuevo editor de propiedades 10-18  
Propiedad localizable 16-8  
Propiedades de la paleta 10-24
- D**
- 
- Datos de BeanInfo 10-10  
cambiar 10-10
- definición  
de las configuraciones para la ejecución 7-7  
duplicadas de clases 5-7
- dependencias  
diagramas UML 11-7
- depurador 8-1  
ayuda inmediata 8-30  
barra de estado 8-27  
barra de herramientas 8-27  
definición de los intervalos de actualización 8-78  
ejecución bajo su control 8-8  
ExpressionInsight 8-29  
finalización de la sesión 8-8  
interfaz de usuario 8-9  
pausar la ejecución del programa 8-8  
personalización  
de la presentación 8-75  
de parámetros 8-76  
teclas de método abreviado 8-28  
vistas 8-10  
y tests de módulos 13-19
- depurar 8-1  
apertura remota de programas 9-2  
aplicación web 8-6  
aplicaciones distribuidas 9-1  
archivo de clase 8-6  
borrar puntos de observación 8-69  
cambiar valores de datos 8-64  
código
- LegacyJ 8-30  
local que se ejecuta en un proceso independiente 9-10  
SQLJ 8-30
- compilar programas con información de depuración 8-4
- configuraciones de ejecución 8-4
- controlar  
inspección de clases 8-42  
la ejecución del programa 8-31
- crear puntos de interrupción 8-46
- definir puntos de ejecución 8-33
- depurar un programa en un ordenador remoto 9-6
- descripción general 8-3
- desde la línea de comandos 8-1
- desplazarse a través del código 8-37
- detectar conflictos 8-36
- edición de puntos de observación 8-69
- ejecución  
hasta el final de un método 8-40  
hasta la posición del cursor 8-41  
hasta un punto de interrupción 8-40
- elegir hilo para inspección 8-35
- en proceso independiente 9-10
- enlazar con un programa en ejecución 9-6
- errores lógicos 8-3
- examen de los valores de datos del programa 8-62
- excepciones de ejecución 8-2
- fuente distinta de Java 8-30
- gestionar hilos 8-34
- iniciar sesión con -classic 8-7
- inspeccionar  
clases con archivo fuente no disponible 8-44  
código de llamadas a métodos 8-37, 8-38
- interrupción  
prolongada de un hilo 8-35  
y ejecución del depurador 8-31
- JSP 8-30  
la opción -classic 2-22
- modificar  
código 8-72  
valores 8-71
- mostrar  
el hilo actual 8-35  
marco de pila superior 8-35  
variables estáticas y locales 8-63
- observación de expresiones 8-66
- paso inteligente 8-39
- proyecto 8-6
- puntos  
de ejecución 8-32  
de interrupción

interprocesal 9-10  
y configuración de Inspección  
desactivada 8-45  
reinicio del programa 8-32  
remoto 9-1  
salir de un método 8-38  
sesión de inicio 8-6  
sesiones 8-10  
tests de módulos 8-6, 13-3, 13-19  
tutorial 17-1  
visualizar llamadas a métodos 8-41  
desde la línea de comandos  
argumentos de JBuilder 5-12  
cambio a IDE durante la compilación 5-12  
compilar 5-11  
ejecutar  
    archivos JAR 7-14  
    programas distribuidos 7-14  
generar proyectos 5-12  
interfaz de la línea de comandos de  
    JBuilder B-4  
desplazamiento  
    diagramas UML 11-16  
diagramas UML 11-3  
    ayuda inmediata 11-14  
Buscar referencias 11-22  
carpetas del panel de estructura 11-7  
código confuso 11-3  
definición 11-7  
diagrama  
    de clases 11-12  
    de paquetes 11-12  
diagrama de clases  
    definición 11-5  
diagrama de paquetes  
    definición 11-4  
filtrar 11-19  
guardar como imágenes 11-21  
iconos de accesibilidad 11-9  
 impresión 11-22  
 incluir referencias  
    de biblioteca 11-20  
    del código generado 11-20  
perfeccionar código fuente 11-22  
personalizar 11-18  
presentar clases y paquetes 11-11  
ver clases internas 11-13  
directorio  
    de navegación  
        añadir 2-18  
    de trabajo 4-13  
directorio fuente  
    test 13-6  
diseñador de BeanInfo 10-10

diseño  
    Interfaces de usuario JavaBean 10-4  
    patrones (JavaBean) 10-1  
distribuir  
    aplicaciones 15-1, 15-11  
    aplicaciones como recopilatorios 15-2  
    applets 15-12  
    en Internet 15-15  
    sugerencias 15-15  
doctet  
    creación personalizada 14-30  
    estándar 14-9  
    JDK 1.1 14-9  
documentación  
    API  
        crear 14-1  
        de campo 14-2  
        de interfaz 14-2  
        de método 14-2  
        generar archivos de salida 14-18  
        ver  
            desde el panel del proyecto 14-22  
            en el Visualizador de ayuda 14-22  
            en pestaña Documentación 14-22, 14-24  
convenciones 1-5  
    convenciones de plataformas 1-6  
de Javadoc 14-1  
    ámbito de documentación 14-13  
    añadir comentarios 14-2  
    archivos  
        de detalles de paquetes 14-20  
        de salida  
            estándar 14-9  
            JDK 1.1 14-9  
        generados 14-18  
    archivos de comentarios de aspectos  
        generales 14-20  
    dar formato a los archivos de salida 14-9  
    directorio de salida 14-11  
    generar  
        archivos de paquetes 14-20  
        archivos de salida 14-18  
        mantener 14-25  
        nombre del nodo 14-11  
        opciones  
            de generación 14-11  
            de línea de comandos 14-14  
    ver  
        ver desde el panel del proyecto 14-22  
        ver desde un diagrama UML 14-22  
        ver en el Visualizador de ayuda 14-22  
        ver en pestaña Documentación 14-22, 14-24  
        ver todo el proyecto 14-22  
        ver un solo archivo 14-22, 14-24

ver Javadoc 14-24

## E

### editor

- de propiedades 10-10
  - cadena de inicialización de Java 10-19
  - componente personalizado 10-21
  - crear 10-18
  - Lista de cadenas 10-18
  - Lista de etiquetas de cadena 10-19
- optimizar importaciones 12-15
- y ejecutores de test 13-15

### EJB

- comprobar 13-12
- tests con Cactus 13-13

### ejecución del programa

- controlar 8-31

### ejecutables

- archivos de configuración A-1
- programa de inicio A-1

### ejecutar

- aplicaciones 7-1
- aplicaciones desde la línea de comandos 7-14
- applets 7-1
- archivos individuales 7-2
- archivos web 7-3
- bajo control del depurador 8-8
- proyectos 7-3
- proyectos agrupados 7-5
- tests Cactus 13-14
- tests de módulos 13-3, 13-15
- tutorial 17-1

### ejecutores de test

- definir 13-17
- disponibles en JBuilder 13-1
- JBTestRunner 13-15
- JUnit AwtUI 13-1
- JUnit SwingUI 13-17
- JUnit TextUI 13-17

### ejemplos

- aplicación multilingüe 16-3

### errores

- de sintaxis 5-5
- lógicos 8-3
- mensajes de error 5-5
- tiempo de ejecución 8-2
- tipos 8-2

### establecer

- punto de ejecución 8-33
- vías de acceso 4-10
  - archivos
  - fuente 4-10

bibliotecas necesarias 2-25

### CLASSPATH

- B-2
- copia de seguridad 4-12
- doc 4-12
- herramientas de línea de comandos B-2
- JDK 2-21
- opción -classpath B-2
- output 4-11
- vía de acceso a clases 4-11
- vía de búsqueda 4-11
- vías de acceso de proyecto 2-4

### etiquetas

- @todo 14-8
- en tests 13-6

### @todo para Javadoc

### Javadoc

- @author 14-5
- @deprecated 14-5
- @docRoot 14-5
- @exception 14-5
- @link 14-5
- @param 14-5
- @return 14-5
- @see 14-5
- @serial 14-5
- @serialData 14-5
- @serialField 14-5
- @since 14-5
- @throws 14-5
- @version 14-5

generar automáticamente 14-7

introducir en archivos fuente 14-5

lista 14-5

### para Javadoc

### excepciones

- y tests de módulos 13-6

### Exponer BeanInfo de superclase (opción)

10-10

### expresiones

evaluar y modificar 8-70

observar 8-66

### ExpressionInsight

8-29

### extraer método

12-4

## F

### fases

- Compilar 6-3
- de generación
  - Distribuir 6-3
  - Ejecutar Make 6-3
  - Generar de nuevo 6-3
  - Limpiar 6-3
  - Paquete 6-3
  - Postcompilar 6-3
  - Precompilar 6-3

Distribuir 6-3  
Ejecutar Make 6-3  
Empaquetar 6-3  
Generar de nuevo 6-3  
Limpiar 6-3  
Postcompilar 6-3  
Precompilar 6-3  
fichas Jerarquía del test 13-15  
filtrar  
  excluir paquetes de la generación 6-27  
  paquetes 6-27  
filtro de seguimiento de pila  
  test de módulos 13-18  
fuente  
  Convenciones empleadas en la documentación de JBuilder 1-5  
  inteligente 8-30  
  mostrar fuentes internacionales 16-11  
funciones acordes con la versión localizada 16-10

## G

---

generar  
  Ant 6-9  
  archivos  
    de salida Javadoc 14-18  
    SQLJ 6-17  
  comando  
    Ejecutar 5-5  
    Ejecutar Make 6-4  
    Ejecutar y Ant 6-14  
    Generar de nuevo 6-5  
    Limpiar 6-5  
  copiar recursos en la vía de salida 6-29  
  creación de tareas externas de generación 6-19  
  descripción general 6-1, 6-2  
  desde la línea de comandos B-4  
  etiquetas Javadoc 14-7  
  excluir paquetes 6-27  
  fases 6-2, 6-3  
  JavaBeans 10-2  
  Javadoc 14-1  
  mensajes  
    Ant 6-13  
      de generación 6-20  
  programas en Java 6-1  
  recopilación automática de fuentes 6-25  
  tareas  
    de generación 6-2  
      externas de generación 6-19  
  terminos definidos 6-2  
  tipos 6-2  
grupos  
  de noticias

Borland 1-8  
public 1-8  
de proyectos 3-1  
  añadir proyectos 3-1, 3-3  
  añadir tipos al menú Proyecto 6-23  
  compilar 6-6  
  configurar menú Grupo de proyectos 6-23  
  crear 3-1  
  dependencias entre proyectos 3-5  
  desplazamiento 3-4  
  ejecutar 7-5  
  eliminar proyectos 3-3  
  generar 6-6  
  orden de generación de proyectos 3-1  
  ventajas 3-1  
guardar  
  varios proyectos 2-27

## H

---

herramientas  
de línea de comandos  
  bcj B-7  
  bmj B-12  
  definir CLASSPATH B-2  
  establecer vías de acceso B-2  
  jar 15-6, B-1  
  java B-1  
  javac B-1  
  javadoc B-1  
  JBuilder B-4  
  native2ascii B-1  
  visualizador de applets B-1  
de recopilación de Java 15-6  
interfaz de la línea de comandos de  
  JBuilder B-4  
  jar 15-6, B-1  
  java B-1  
  javac B-1  
  javadoc B-1  
  línea de comandos B-1  
  native2ascii B-1  
  visualizador de applets B-1  
hilos  
  detectar conflictos 8-36  
  elegir para inspección 8-35  
  gestionar 8-34  
  interrupción prolongada 8-35  
  mostrar el actual 8-35  
  mostrar marco de pila superior 8-35  
  panel dividido 8-34

---

**iconos**

- de accesibilidad
  - diagramas UML 11-9
- especificar JavaBean 10-10
- Iconos de accesibilidad de UML 11-9
- JBTestRunner 13-15
- ímágenes UML 11-21
  - guardar diagramas 11-21
  - imprimir diagramas 11-22
- importaciones
  - optimizar 12-2
- inclusión en recursos
  - cadenas 16-5
  - definición 16-2
- información de depuración 8-4
- insertar en sentencia try/catch 12-5
- inspección 8-6, 8-37
  - código de clases
    - activar 8-42
    - desactivar 8-42
  - inteligente 8-37, 8-39
  - llamadas a métodos 8-37, 8-38
  - salir de métodos 8-38
- instalar
  - JavaBeans en la paleta de componentes 10-24
- Intercambio inteligente 8-72
  - y la MV de destino 8-72
- interfaces
  - de la línea de comandos JBuilder B-4
  - de usuario diseñar JavaBean 10-4
    - diagramas UML 11-7
- internacionalización 16-3, 16-17
  - características del depurador 16-13
  - controles homólogos no nativos 16-11
  - dbSwing 16-9
  - definición 16-1
  - funciones de compilador 16-13
  - funciones de JBuilder 16-3
  - funciones del diseñador visual 16-11
  - opciones de codificación 16-14
  - programas 16-1
  - términos y definiciones 16-1
- introducir variable 12-5

- 
- Java y UML 11-2
- JavaBeans 10-2
- añadir sucesos 10-12, 10-15, 10-16
  - cambiar propiedades 10-7
  - crear 10-1, 10-2
  - definición 10-1

- diseñar una interfaz de usuario 10-4
- distribuir 15-14
- eliminar propiedades 10-8
- establecer propiedades 10-4, 10-8
- instalar 10-24
- mostrar valores de las propiedades 10-10
- personalizar 10-9
- serialización 10-22
- validez 10-23
- ventajas 10-2

**JavaBeans válidos**

- comprobar 10-23

**JBTestRunner**

- Fallos del test 13-15
- iconos 13-15
- Jerarquía del test 13-15
- Resultados del test 13-15
- y editor 13-15

**JBuilder**

- versiones internacionales 16-17

**JDK**

- cambiar y configurar 2-23
- configuración en JBuilder 2-24
- configurar proyectos Ant 6-14
- definir en el asistente para proyectos 2-4
- depurar con -classic 2-22
- editar 2-22

**JSP**

- depurar 8-30

**JUnit**

- Assert (clase) 13-6
- assertEquals() 13-6
- assertNotNull() 13-6
- assertTrue() 13-6
- AwtUI 13-1
- clases
  - TestCase 13-1, 13-3
  - TestSuite 13-1, 13-3
- ejecutores de test 13-1
- integración en JBuilder 13-1
- probar un EJB 13-8
- setUp() 13-5
- SwingUI 13-1, 13-17
- tearDown() 13-5
- TextUI 13-1, 13-17

- 
- línea de comandos
  - herramientas B-1
- llamadas a métodos
  - buscar 8-41
  - evaluar 8-70
  - visualizar 8-41

localización 16-1  
de paquetes 6-25  
definición 16-2  
de fuentes 6-25

## M

---

make  
de Borland para Java (bmj) B-12  
de línea de comandos bmj 5-11, B-12  
utilización con Ant 6-15  
línea de comandos bmj B-12  
mantener Javadoc 14-25  
mensajes de error 5-5  
depurador 8-2  
menús  
configurar el menú Proyecto 6-21  
configurar menú Grupo de proyectos 6-23  
de generación 5-4  
añadir tipos 6-21, 6-23  
Grupo de proyectos  
añadir tipos 6-23  
Proyecto  
añadir tipos 6-21  
configurar 6-21  
métodos  
buscar definición 12-8  
buscar referencias a 12-9  
de test  
requisitos 13-6  
diagramas UML 11-7  
ejecución hasta el final 8-40  
modificación  
del JDK 2-22  
parámetros de métodos 12-24  
valores de datos en el depurador 8-64  
montaje  
de comparación 13-10  
asistente 13-10  
tutorial 22-1  
JDBC 13-9  
asistente 13-9  
tutorial 22-1  
JNDI 13-10  
asistente 13-10  
para tests  
ejemplo 13-8  
montaje de comparación 13-10  
Montaje JDBC 13-9  
Montaje JNDI 13-10  
personalizada 13-11  
predefinidos 13-8  
tutorial 22-1

personalizado 13-11  
asistente 13-11  
mostrar  
archivos  
de proyecto 4-14  
multiplataforma (entornos) 10-2  
MV  
definir MV destino 5-7  
depurar con -classic 2-22

## N

---

nodo  
de documentación  
ampliar 14-22  
Asistente para Javadoc 14-9  
cambiar propiedades 14-26, 14-27  
generar 14-9  
propiedades para 14-9  
de ejecutables nativos  
asignar valores a propiedades 15-34  
de recopilatorios  
borrar 15-36  
eliminar 15-36  
generar 15-34  
propiedades 15-36  
renombrar 15-36  
restablecer propiedades 15-35  
ver contenido 15-35  
fuente del proyecto 6-25, 6-27  
filtrar paquetes 6-27

## O

---

observación de expresiones 8-66  
omitir  
inspección 8-6, 8-38  
opciones  
-classic para depurar 8-7  
de la configuración de ejecución  
definir en el archivo recopilatorio 15-31  
de línea de comandos 5-12  
en el Asistente para Javadoc 14-14  
para Javadoc 14-14  
de recopilación automática de paquetes  
fuente 2-5  
de UML  
filtrado de paquetes y clases 11-19  
incluir referencias de biblioteca 11-20  
incluir referencias del código  
generado 11-20  
personalizar IDE 11-21  
propiedades  
proyecto 11-18

del IDE  
    UML 11-21  
    MV destino 5-7

OpenTools  
    activar el modo de depuración verbose con  
        argumentos de la línea de comando B-4  
    ejecutar 7-6  
optimizar importaciones 12-2  
organizar proyectos 2-14

**P**

---

páginas de código 16-15  
palabras clave  
    assert  
        activar 2-5, B-10, B-16  
paleta de componentes  
    instalar JavaBeans 10-24  
panel de estructura  
    carpeta Errores 5-5  
    carpeta Por hacer 14-8  
    conflictos Javadoc 14-8  
    UML 11-17  
pantalla de inicio  
    desactivar con argumentos de la línea de  
        comandos B-4  
paquetes 4-6  
    activar la localización de paquetes fuente 6-25  
    convenciones  
        de nomenclatura 4-9  
    diagramas UML 11-4, 11-7  
    excluir del proceso de generación 6-27  
    filtrar 6-27  
    localización de paquetes fuente 6-25  
    recopilación automática de fuentes 6-25  
    referencias a clases 4-9  
parámetros de métodos  
    modificación 12-4  
Paso inteligente 8-37, 8-38, 8-39  
    opciones 8-39  
pausar la ejecución del programa 8-8  
perfeccionamiento  
    cambiar  
        de nombre 12-1  
        parámetros de métodos 12-4, 12-24  
    desde el visualizador UML 11-22  
    extraer método 12-4, 12-26  
    insertar  
        bloque en sentencia try/catch 12-28  
        en sentencia try/catch 12-5  
introducir variable 12-5, 12-27  
mover 12-1  
optimizar importaciones 12-2  
por cambio de nombre

campos 12-2, 12-23  
clases 12-2, 12-19  
compilar antes 12-6  
descripción general 12-1  
deshacer 12-28  
detección de símbolos 12-8  
guardar 12-29  
métodos 12-2, 12-21  
paquetes 12-2, 12-18  
propiedades 12-2, 12-24  
variables locales 12-2, 12-22  
vista previa 12-11

por desplazamiento  
    clases 12-4, 12-20  
    compilar antes 12-6  
    descripción general 12-1  
    deshacer 12-28  
    detección de símbolos 12-8  
    guardar 12-29  
    vista previa 12-11

plantilla de proyecto  
    definir 2-3

plataformas  
    convenciones 1-6

posición del cursor  
    ejecutar hasta 8-41

presentar código 11-1

probar directorio fuente 13-6

programa de inicio  
    archivos de configuración A-1  
    ejecutables A-1

programas  
    compilar 5-1  
    comprobar 13-1  
    depurar 8-1  
    distribuir 15-1  
    ejecutar 8-31  
    interrupción 8-31

PropertyChangeSupport (clase) 10-8

propiedades  
    Ant 6-15  
    componentes JavaBean 10-4, 10-8  
    Creador de ejecutables nativos 15-34  
    diagramas UML 11-7  
    modificación 10-18  
    monitorizables  
        definir para JavaBeans 10-8  
    nodo de recopilatorios 15-35  
    ocultar 10-10  
    personalizar 10-10

proyecto  
    configurar para detectar referencias 12-6  
    configurar para perfeccionar 12-6  
    configurar vía de salida 5-10

- definición para UML 11-18
- directorio fuente para los tests 13-6
- restringidas
  - definir para JavaBeans 10-8
  - tareas externas de generación 6-21
- proyectos 2-1
  - abrir 2-12
  - agrupados
    - ejecutar 7-5
  - añadir
    - a grupos de proyectos 3-1
    - archivos o paquetes 2-14
    - archivos ZIP o JAR 4-1
    - archivos, paquetes 2-15
    - como bibliotecas necesarias 3-5, 4-5
    - directorio de navegación 2-18
  - Ant 6-9
  - asignar valores a propiedades 2-20
  - cambiar de uno a otro 2-26
  - cerrar 2-11
  - cómo añadir carpetas 2-15
  - compilar 5-1
  - configurar el menú Proyecto 6-21
  - crear 2-2
    - con el asistente Proyecto para código existente 2-8
    - y añadir archivos 2-13
  - definir
    - clase principal 7-4
    - parámetros generales 2-5
  - ejecutar 7-1, 7-3
  - eliminar de 2-16
  - generar 6-1
    - Ant con el comando Ejecutar 6-14
    - con el comando Ejecutar 5-5
    - desde la línea de comandos 5-12, B-4
  - guardar 2-11
    - varios 2-27
  - renombrar 2-17
  - uso de varios proyectos 2-26
  - ver archivos 2-10
- pruebas de regresiones 13-1
- puntos
  - de ejecución
    - definir 8-33
    - descripción general 8-32
  - de interrupción 8-46
    - acciones 8-57
    - activar 8-60
    - buscar 8-62
    - campo 8-53
    - class 8-50
    - condicional 8-59
    - condicionales 8-59
  - definir 8-59
  - de campo 8-53
  - de clase 8-50
  - de línea 8-47
  - de método 8-51
  - definir 8-46
  - desactivar 8-60
  - ejecutar hasta 8-40
  - eliminar 8-61
  - exception 8-49
  - interprocesal 8-53, 9-10
  - línea 8-47
  - method 8-51
  - número de pasadas 8-60
  - para depurar tests 13-19
  - por excepción 8-49
  - por número de pasadas 8-60
  - propiedades 8-56
  - sobreescribir Inspección desactivada 8-45
- de observación 8-66
  - borrar 8-69
  - de objetos 8-69
  - de variables 8-67
  - editar 8-69
  - puntos de observación de variables 8-67

## R

---

- readObject() 10-22
- recopilación
  - archivo descriptor 15-3
  - archivos
    - fuente 15-19
  - con el Creador de recopilatorios 15-5
  - de tests de JUnit 13-3
- recopilación automática de fuentes
  - niveles jerárquicos mostrados 6-25
  - Nodo Fuente del proyecto 6-25
- recopilar
  - con el Creador de ejecutables nativos 15-32
  - con herramienta Jar 15-6
  - documentación 15-19
  - proyectos para distribución 15-2
- recopilatorio de documentación
  - crear 14-28
- recursos
  - copiar en vía de salida 6-29
  - recopilación automática de fuentes 6-25
- Redefinir clases después de compilar 8-74
- redistribución de clases 15-16
- referencias
  - configurar para detectar 12-6
  - de bibliotecas 2-5

renombrar  
campos 12-2, 12-23  
clases 12-2, 12-19  
deshacer 12-28  
guardar 12-29  
métodos 12-2, 12-21  
paquetes 12-2, 12-18  
propiedades 12-2, 12-24  
variables locales 12-2, 12-22

## S

---

sentencias  
import 4-9  
optimizar 12-15  
serialización  
admitir 10-22  
setUp() 13-5  
símbolos de código  
detectar antes de perfeccionar 12-8  
perfeccionamiento 12-1  
SQLJ  
generar archivos 6-17  
stubs  
archivos fuente 8-44  
sucesos  
añadir a JavaBeans 10-12, 10-15, 10-16  
superclases 10-10  
SwingUI 13-17

## T

---

tareas de generación 6-19  
asignar valores a propiedades 6-21  
Asistente para tareas externas de  
generación 6-19  
configuración de las propiedades Ant 6-15  
generación de tareas externas 6-20  
tearDown() 13-5  
términos de UML  
definición 11-2  
términos correspondientes de Java 11-2  
test de módulos 13-1  
Asistente para conjuntos de tests 13-7  
Asistente para tests 13-6  
Cactus 13-2, 13-12  
clases  
TestCase 13-1, 13-3  
TestSuite 13-1, 13-3  
código del servidor 13-2, 13-12  
configuración de ejecución 13-3, 13-17  
crear tests 13-5  
definición 13-1  
depuración de tests 13-19  
deteccción de tests 13-3

directorio fuente 13-6  
ejecutar tests 13-15  
JBTestRunner 13-15  
JUnit 13-1  
JUnit SwingUI 13-17  
JUnit TextUI 13-17  
lista de funciones 13-2  
métodos de test 13-6  
montaje de comparación 13-10  
Montaje JDBC 13-9  
Montaje JNDI 13-10  
montaje personalizado 13-11  
montajes para tests 13-8  
objetivos 13-5  
opción de menú Depurar test 13-3  
opción de menú Ejecutar test 13-3  
recopilador de tests de JUnit 13-3  
tutorial 22-1  
un EJB 13-8, 13-12, 13-13  
test del servidor 13-2, 13-12  
tests  
crear 13-6  
ejecutar desde main() 13-17  
etiquetas @todo 13-6  
orden de ejecución 13-5  
setUp() 13-5  
tearDown() 13-5  
TextUI 13-17  
tiempo de ejecución  
errores 8-2  
excepciones 8-2  
tipos  
Ant 6-9  
configuración  
de ejecución 7-11  
de las propiedades Ant 6-15  
propiedades de tareas externas 6-21  
de configuraciones de ejecución 7-13  
de ejecutor 7-13  
de generación 5-5  
de referencias  
antecesores 12-9  
declaraciones 12-9  
descendientes 12-9  
escrituras 12-9  
lecturas 12-9  
referencias de miembros 12-9  
referencias de miembros descendientes 12-9  
referencias de tipo 12-9  
referencias de tipo descendientes 12-9  
usos directos 12-9  
usos indirectos 12-9  
generar 5-5

- trasladar  
  clases 12-4, 12-20  
  deshacer 12-28  
  guardar 12-29  
try/catch  
  insertar bloque de código en 12-28  
tutoriales  
  compilación, ejecución y depuración 17-1  
  de depuración remota 19-1  
  generación con archivos Ant 18-1  
  montajes para tests 22-1  
  UML 20-1
- U**
- ubicación de documentos 4-12  
UML 11-1  
  definición 11-1  
  descripción general 11-1  
  tutorial 20-1  
  y Java 11-2  
Unicode 16-13  
  ASCII de 7 bits 16-16  
  definición 16-2  
  formato de 16 bits 16-16  
  introducir 16-11  
  mostrar 16-11  
Usenet, grupos de noticias 1-8
- V**
- valores  
  de datos  
    cambiar en el depurador 8-64  
    del programa 8-62  
    examinar durante la depuración 8-62  
  examinar durante la depuración 8-62  
variables  
  buscar definición 12-8  
  buscar referencias a 12-9  
  de entorno CLASSPATH  
    configurar para línea de comandos B-2  
    introducir 12-27  
    modificar valores 8-71  
varios proyectos 2-26  
  cambiar de uno a otro 2-26  
ver  
  archivos 2-10  
  archivos de proyecto 4-14  
  documentación de Javadoc  
    desde un diagrama UML 11-14  
  documentación del API  
    en el visualizador UML 11-14  
versión localizada  
  definición 16-2
- versiones internacionales  
  JBuilder 16-17
- vías  
  de acceso 4-1  
    compilación, ejecución y depuración 4-14  
    configurar con la opción -classpath B-2  
    configurar vía de salida 5-10  
    definir B-2  
    ubicación de archivos de clase 4-8  
    ubicación de archivos Java 4-7  
  de acceso a  
    archivos fuente  
      archivos de clase 4-8  
      Los archivos java 4-7  
    clases 4-11  
      establecer vías de acceso B-2  
      copias de seguridad 4-12  
  de acceso de proyecto  
    definir 2-4  
  de archivos generados  
    definir 5-10  
  de búsqueda 4-11  
  de salida 4-11
- vistas  
  clases cargadas y datos estáticos 8-23  
  Clases con inspección desactivada 8-13  
  del depurador  
    Vista  
      Clases cargadas y datos estáticos 8-23  
      Clases con inspección desactivada 8-13  
      Hilos, pilas de llamada y datos 8-16  
      Monitores de sincronización 8-26  
      Puntos de interrupción de datos y  
        código 8-14  
      Puntos de observación de datos 8-20  
      Salida, entrada y errores de consola 8-12  
    Hilos, pilas de llamada y datos 8-16  
    Hilos, pilas de llamadas y datos  
      panel dividido 8-34  
  Monitores de sincronización 8-26  
  Puntos  
    de interrupción de datos y código 8-14  
    de observación de datos 8-20  
  Salida, entrada y errores de consola 8-12
- visualizador UML 11-11  
  ayuda inmediata 11-14  
  definición 11-11  
  desplazamiento por diagramas 11-16  
  menú contextual 11-15  
  perfeccionar código fuente 11-22  
  tutorial 20-1  
  ver código 11-14  
  ver Javadoc 11-14

visualizar

variable estática 8-63

variable local 8-63

variables estáticas y locales 8-63

## **W**

---

writeObject() 10-22

