



# **Guía del desarrollador de Enterprise JavaBeans™**

---



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.

JBE0080WW21002entjb 5E6R1002  
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 . . . . .	1-2	
Asistencia a los desarrolladores . . . . .	1-4	
Cómo ponerse en contacto con el servicio técnico de Borland . . . . .	1-4	
Recursos en línea . . . . .	1-5	
World Wide Web . . . . .	1-5	
Grupos de noticias de Borland . . . . .	1-5	
Usenet, grupos de noticias . . . . .	1-6	
Información sobre errores . . . . .	1-6	
<b>Capítulo 2</b>		
<b>Programación con Java 2 Platform Enterprise Edition</b>	<b>2-1</b>	
Ventajas de las aplicaciones J2EE . . . . .	2-1	
Ventajas del modelo multinivel. . . . .	2-4	
Ventajas de JBuilder . . . . .	2-5	
Tecnologías para el nivel del cliente . . . . .	2-6	
Tecnologías del nivel intermedio. . . . .	2-6	
Otras tecnologías J2EE . . . . .	2-8	
Preparación para distribuir aplicaciones J2EE . . . . .	2-9	
Más información acerca de J2EE . . . . .	2-9	
<b>Parte I</b>		
<b>Desarrollo de Enterprise JavaBeans</b>		
<b>Capítulo 3</b>		
<b>Introducción al desarrollo de EJB</b>	<b>3-1</b>	
Finalidad de los Enterprise JavaBeans. . . . .	3-1	
Competencias en el desarrollo de una aplicación EJB. . . . .	3-2	
Competencias de la aplicación . . . . .	3-3	
Competencias de infraestructura . . . . .	3-3	
Competencias de distribución y operación .	3-4	
Arquitectura de EJB . . . . .	3-4	
El servidor EJB . . . . .	3-5	
El contenedor EJB . . . . .	3-5	
Funcionamiento de los enterprise beans. .	3-6	
Tipos de enterprise beans . . . . .	3-7	
Beans sesión. . . . .	3-7	
Beans entidad. . . . .	3-7	
Beans gestionado por mensajes . . . . .	3-8	
Acceso local y remoto . . . . .	3-8	
<b>Capítulo 4</b>		
<b>Desarrollo de enterprise beans</b>	<b>4-1</b>	
<b>Capítulo 5</b>		
<b>Configuración del servidor de aplicaciones de destino</b>	<b>5-1</b>	
Servidores admitidos . . . . .	5-1	
Configuración de servidores con JBuilder . .	5-2	
Cómo añadir un service pack . . . . .	5-5	
Las bibliotecas generadas . . . . .	5-6	
Cómo poner el ORB a disposición de JBuilder	5-7	
Selección de un servidor . . . . .	5-9	
Configuración de los controladores JDBC . .	5-11	
Creación de los archivos .library y .config .	5-11	
Adición del controlador JDBC a proyectos .	5-12	
<b>Capítulo 6</b>		
<b>Creación de beans sesión y gestionados por mensajes 2.0 con el diseñador de EJB</b>	<b>6-1</b>	
Los módulos EJB . . . . .	6-2	
Creación de módulos EJB 2.0. . . . .	6-2	
Creación de módulos con el Asistente para grupos EJB vacíos . . . . .	6-2	
Creación de módulos EJB a partir de descriptores de distribución . . . . .	6-4	
El diseñador de EJB. . . . .	6-7	
Creación de beans sesión. . . . .	6-9	
Visualización del código fuente de los beans	6-11	
Modificación de beans . . . . .	6-11	
Modificación de los atributos del bean .	6-11	
Añadir un nuevo campo . . . . .	6-12	
Borrado de campos . . . . .	6-13	
Añadir un nuevo método. . . . .	6-13	
Eliminación de métodos . . . . .	6-14	
Los métodos ejbCreate() . . . . .	6-14	
Cómo regenerar interfaces de un bean .	6-15	
Importación de beans. . . . .	6-15	
Organización de beans con vistas. . . . .	6-17	
Búsqueda de beans . . . . .	6-18	
Cómo ordenar los beans . . . . .	6-19	

Creación de beans gestionados por mensajes . . . . .	6-19
Eliminación de beans . . . . .	6-20
Asignación de nombres a los archivos en el diseñoador de EJB . . . . .	6-21
Solución de errores en los beans . . . . .	6-22
Visualización de los descriptores de distribución . . . . .	6-23
Presentación del editor de descriptor de distribución . . . . .	6-23
Configuración de las opciones del IDE para el diseñador de EJB . . . . .	6-24
Siguiente paso . . . . .	6-25
<b>Capítulo 7 Creación de beans entidad 2.0 con el diseñador de EJB</b>	<b>7-1</b>
Creación de beans entidad CMP 2.0 a partir de una fuente de datos importada . . . . .	7-2
Importación de fuentes de datos . . . . .	7-2
jndi-definitions.xml . . . . .	7-4
Modificación del esquema de la fuente de datos importada . . . . .	7-4
Generación de las clases e interfaces de los beans entidad . . . . .	7-6
Modificación de las propiedades de los beans entidad . . . . .	7-7
Referencias a otra tabla . . . . .	7-10
Cómo añadir referencias de tabla de WebLogic . . . . .	7-13
Inspectores de campos y métodos de los beans entidad . . . . .	7-16
Establecimiento de relaciones entre beans entidad . . . . .	7-16
Establecimiento de relaciones por medio del inspector . . . . .	7-17
Eliminación de relaciones . . . . .	7-22
Adición de métodos de búsqueda . . . . .	7-22
Cómo añadir métodos ejbSelect . . . . .	7-23
Adición de métodos base empresariales . . . . .	7-24
Creación de esquemas a partir de beans entidad . . . . .	7-24
Exportación de fuentes de datos . . . . .	7-24
Creación de beans entidad con persistencia gestionada por bean . . . . .	7-25
<b>Capítulo 8 Creación de componentes EJB 1.x con JBuilder</b>	<b>8-1</b>
Módulos EJB . . . . .	8-1
Creación de módulos EJB 1.x . . . . .	8-2
Creación de módulos EJB 1.x con el Asistente para grupos EJB vacíos . . . . .	8-2
Creación de módulos EJB a partir de descriptores de distribución . . . . .	8-4
Creación de enterprise beans . . . . .	8-5
Creación de beans sesión . . . . .	8-6
Creación de beans entidad . . . . .	8-7
Cómo añadir lógica empresarial al bean . . . . .	8-9
Exposición de métodos empresariales mediante la interfaz remota . . . . .	8-11
Generación de la clase del bean desde una interfaz remota . . . . .	8-13
Creación de las interfaces base y remota para beans existentes . . . . .	8-15
<b>Capítulo 9 Creación de beans entidad EJB 1.x a partir de una tabla de base de datos</b>	<b>9-1</b>
Creación de beans entidad con el modelador de beans entidad EJB 1.x . . . . .	9-1
<b>Capítulo 10 Compilación de enterprise beans y creación de archivos JAR</b>	<b>10-1</b>
Compilación de beans . . . . .	10-1
Cambio de propiedades de generación de los módulos EJB o gruposEAR . . . . .	10-1
Cambio de las propiedades de generación de beans . . . . .	10-4
Cambio de las propiedades de generación de módulos EJB . . . . .	10-4
Compilación . . . . .	10-5
El archivo JAR generado . . . . .	10-5
Modificación del descriptor de distribución . . . . .	10-6
Comprobar descriptores . . . . .	10-8

<b>Capítulo 11</b>	
<b>Ejecución y prueba de enterprise beans</b>	<b>11-1</b>
Comprobación del bean . . . . .	11-3
Selección del tipo de cliente de prueba . . . . .	11-3
Las aplicaciones cliente de prueba . . . . .	11-4
Creación de aplicaciones cliente de prueba . . . . .	11-4
Uso de la aplicación cliente de prueba . . . . .	11-7
Uso de la aplicación cliente de prueba con el enterprise bean . . . . .	11-9
Creación de una configuración de ejecución del servidor . . . . .	11-9
Ejecución de la aplicación cliente de prueba EJB . . . . .	11-12
Los tests JUnit . . . . .	11-12
Creación de un test JUnit . . . . .	11-12
Creación de un test JUnit mediante el Asistente para cliente de prueba EJB . . . . .	11-13
Ejecución del test JUnit . . . . .	11-15
Los tests Cactus JUnit . . . . .	11-16
Creación de un test Cactus JUnit . . . . .	11-16
Configuración del proyecto para probar un EJB mediante Cactus . . . . .	11-17
Creación de un test Cactus JUnit mediante el Asistente para cliente de prueba EJB . . . . .	11-22
Ejecución del test Cactus JUnit . . . . .	11-24
Activación de la depuración de WebSphere 4.0 Advanced Edition . . . . .	11-25
Preparación para la depuración remota de aplicaciones WebSphere . . . . .	11-26
WebSphere Server 3.5 . . . . .	11-26
WebSphere Single Server 4.0 . . . . .	11-26
WebSphere Server Advanced Edition 4.0 . . . . .	11-27
Preparación para depurar las Páginas JavaServer de WebLobic de forma remota . . . . .	11-28
Preparación para la depuración remota de aplicaciones iPlanet . . . . .	11-30
<b>Capítulo 12</b>	
<b>Distribución de enterprise beans</b>	<b>12-1</b>
Creación de un archivo de descriptor de distribución . . . . .	12-2
Competencias del descriptor de distribución . . . . .	12-3
Tipos de información del descriptor de distribución . . . . .	12-3
Información de estructura . . . . .	12-4
Información de ensamblaje de la aplicación . . . . .	12-5
Seguridad . . . . .	12-5
Propiedades específicas del servidor de aplicaciones . . . . .	12-6
Creación de archivos EAR . . . . .	12-6
Distribución en un servidor de aplicaciones . . . . .	12-7
Distribución de archivos JAR . . . . .	12-8
Distribución a servidores ajenos a Borland . . . . .	12-9
Definición de las opciones de distribución mediante el cuadro de diálogo Propiedades . . . . .	12-10
Distribución en tiempo real en un servidor de aplicaciones . . . . .	12-11
<b>Capítulo 13</b>	
<b>El editor de descriptor de distribución</b>	<b>13-1</b>
Presentación del editor de descriptor de distribución . . . . .	13-2
Visualización del descriptor de distribución de enterprise beans . . . . .	13-2
Visualización de la ficha Propiedades de módulo EJB de WebLogic 6.x o 7.x . . . . .	13-4
Cambio de la información del bean . . . . .	13-5
Información del enterprise bean . . . . .	13-6
Panel General . . . . .	13-6
Panel de beans gestionados por mensajes . . . . .	13-9
Panel Environment . . . . .	13-11
Panel EJB References . . . . .	13-12
Panel Resource References (Referencias de recursos) . . . . .	13-14
Panel Security Role References . . . . .	13-15
Panel Properties . . . . .	13-16
Panel Security Identity . . . . .	13-21
Panel EJB Local References . . . . .	13-23
Panel Resource Env Refs . . . . .	13-25
Panel General de WebLogic 6.x o 7.x . . . . .	13-26
Panel Propiedades específicas del servidor . . . . .	13-27
Panel Editor DD de EAServer . . . . .	13-28
Configuración de propiedades de paquetes de Sybase . . . . .	13-28
Definición de las propiedades de los componentes de Sybase . . . . .	13-30
Importación de un descriptor de distribución de Sybase . . . . .	13-31
Panel Caché de WebLogic 6.x o 7.x . . . . .	13-31
Transacciones de contenedor . . . . .	13-32

Definición de las normas de transacción del contenedor . . . . .	13-32	Implementación de la interfaz SessionBean . . . . .	15-2
Panel Aislamiento de transacciones de WebLogic 6.x o 7.x . . . . .	13-35	Escrutura de métodos empresariales . . . . .	15-3
Panel Métodos idempotentes de WebLogic 6.x o 7.x . . . . .	13-36	Adición de métodos ejbCreate() . . . . .	15-3
Las fuentes de datos JDBC 1 . . . . .	13-37	JBuilder en la creación de beans sesión . . . . .	15-4
Definición de los niveles de aislamiento . .	13-39	Ciclo de vida de los beans sesión . . . . .	15-6
Definición de las propiedades de fuente de datos . . . . .	13-39	Beans sin estado . . . . .	15-6
Adición de competencias de seguridad y permisos para métodos . . . . .	13-42	Beans con estado . . . . .	15-7
Creación de competencias de seguridad . .	13-42	Estado preparado para métodos en transacciones . . . . .	15-8
Asignación de permisos para métodos . .	13-44		
Cómo añadir persistencia gestionada por contenedor para componentes EJB 1.1 . .	13-46		
Panel Finders . . . . .	13-48		
Definición de buscadores . . . . .			
WebSphere 4.0 . . . . .	13-50		
Comprobación de la información sobre descriptores . . . . .	13-51		
<b>Capítulo 14</b>			
<b>Uso de DataExpress para componentes EJB</b>	<b>14-1</b>		
Los componentes EJB DataExpress . . . . .	14-2		
Componentes para el servidor . . . . .	14-2		
Componentes para el cliente . . . . .	14-2		
Creación de beans entidad . . . . .	14-3		
Creación del bean sesión en el servidor . . . . .	14-3		
Adición de componentes proveedores y almacenadores al bean sesión . . . . .	14-4		
Escritura del método setSessionContext() .	14-5		
Cómo añadir referencias EJB o referencias EJB locales al descriptor de distribución . . . . .	14-5		
Adición de los métodos de suministro y almacenamiento . . . . .	14-6		
Llamadas a métodos de búsqueda . . . . .	14-7		
Generación del lado cliente . . . . .	14-8		
Tratamiento de relaciones . . . . .	14-10		
El proyecto de ejemplo . . . . .	14-10		
<b>Capítulo 15</b>			
<b>Desarrollo de beans sesión</b>	<b>15-1</b>		
Tipos de beans sesión . . . . .	15-1		
Beans sesión con estado . . . . .	15-1		
Beans sesión sin estado . . . . .	15-2		
Escritura de la clase del bean sesión . . . . .	15-2		
<b>Capítulo 16</b>			
<b>Desarrollo de beans entidad</b>	<b>16-1</b>		
La persistencia y los beans entidad . . . . .	16-1		
Persistencia gestionada por bean . . . . .	16-2		
Persistencia gestionada por contenedor . .	16-2		
Claves principales de los beans entidad . . .	16-3		
Escritura de la clase del bean entidad . . . . .	16-3		
Implementación de la interfaz EntityBean .	16-4		
Declaración e implementación de los métodos del bean entidad . . . . .	16-5		
Creación de métodos de creación . . . .	16-5		
Creación de métodos de búsqueda . . . .	16-7		
Escritura de métodos empresariales . . . .	16-8		
Ciclo de vida de los beans entidad . . . . .	16-8		
Estado de inexistencia . . . . .	16-9		
Estado de espera . . . . .	16-9		
Estado de activación . . . . .	16-9		
Vuelta al estado de espera . . . . .	16-10		
Ejemplo de bean entidad de banca . . . . .	16-10		
Interfaz base del bean entidad . . . . .	16-11		
La interfaz remota del bean entidad . . .	16-12		
Bean entidad con persistencia gestionada por contenedor . . . . .	16-12		
Bean entidad con persistencia gestionada por bean . . . . .	16-14		
La clase de la clave principal . . . . .	16-19		
El descriptor de distribución . . . . .	16-19		
Descriptor de distribución de un bean entidad con persistencia gestionada por bean . . . . .	16-20		
Descriptor de distribución de un bean entidad con persistencia gestionada por contenedor . . . . .	16-21		
<b>Capítulo 17</b>			
<b>Desarrollo de beans gestionados por mensajes</b>	<b>17-1</b>		
Funcionamiento de los beans gestionados por mensajes . . . . .	17-2		

Ciclo de vida de la instancia de un bean gestionado por mensajes . . . . .	17-2
Cómo escribir clases del bean gestionado por mensajes. . . . .	17-3
Implementación de la interfaz Message DrivenBean . . . . .	17-4
Implementación de la interfaz MessageListener . . . . .	17-4
Escritura del método onMessage() . . . . .	17-4
JBuilder en la creación de beans gestionados por mensajes . . . . .	17-5
Atributos del descriptor de distribución de los beans generados por mensajes . . . . .	17-6
Utilización de SonicMQ Message Broker con beans gestionados por mensajes . . . . .	17-8
<b>Capítulo 18 Creación de interfaces base y remota/local</b>	<b>18-1</b>
Creación de la interfaz base. . . . .	18-1
La interfaz EJBHome. . . . .	18-2
La interfaz LocalHome. . . . .	18-2
Creación de interfaces base o base locales para beans sesión. . . . .	18-3
Métodos create() en beans sesión . . . . .	18-3
Creación de interfaces base locales o base remotas para beans entidad . . . . .	18-4
Métodos create() en beans entidad . . . . .	18-5
Métodos de búsqueda en beans entidad . . . . .	18-6
Creación de la interfaz remota o local . . . . .	18-8
Las interfaces EJBObject y EJBLocal Object . . . . .	18-9
<b>Capítulo 19 Desarrollo de clientes de enterprise beans</b>	<b>19-1</b>
Localización de la interfaz base . . . . .	19-2
Obtención de la interfaz remota/local. . . . .	19-2
Beans sesión. . . . .	19-2
Beans entidad. . . . .	19-3
Los métodos de búsqueda y la clase de la clave principal . . . . .	19-4
Métodos de creación y eliminación . . . . .	19-4
Llamadas a métodos. . . . .	19-5
Eliminación de instancias de beans . . . . .	19-6
Referencia a beans mediante identificadores . . . . .	19-6
Gestión de las transacciones . . . . .	19-8
Detección de información sobre el bean. . . . .	19-9
Creación de clientes con JBuilder . . . . .	19-10
<b>Capítulo 20 Gestión de las transacciones</b>	<b>20-1</b>
Características de las transacciones . . . . .	20-1
Aceptación de transacciones en el contenedor. . . . .	20-2
Enterprise beans y transacciones . . . . .	20-3
Transacciones gestionadas por contenedor y gestionadas por bean . . . . .	20-3
Transacciones locales y globales. . . . .	20-4
API de transacciones . . . . .	20-5
Gestión de excepciones de transacción . . . . .	20-7
Excepciones de sistema . . . . .	20-7
Excepciones de aplicación . . . . .	20-7
Gestión de excepciones de aplicación . . . . .	20-8
Cancelación de transacciones. . . . .	20-8
Opciones para la continuación de transacciones . . . . .	20-9
<b>Capítulo 21 Tutorial: Desarrollo de beans sesión con el diseñador de EJB</b>	<b>21-1</b>
Creación de proyectos . . . . .	21-2
Definición del servidor de aplicaciones de destino . . . . .	21-2
Creación de módulos EJB . . . . .	21-3
Creación del bean . . . . .	21-5
Configuración de las propiedades del bean . . . . .	21-6
Adición de campos al bean Cart . . . . .	21-6
Adición de métodos empresariales al bean Cart . . . . .	21-10
Adición y eliminación de artículos del carro . . . . .	21-10
Recuperación de los artículos del bean y su coste . . . . .	21-11
Adición de un método purchase() . . . . .	21-11
El código fuente . . . . .	21-12
Inicialización de la lista de artículos . . . . .	21-13
Adición de sentencias de importación . . . . .	21-14
Implementación de ejbCreate() . . . . .	21-14
Implementación de addItem() y removeItem() . . . . .	21-15
Creación de una clase Item . . . . .	21-15
Implementación de los métodos restantes . . . . .	21-17
Utilización de los descriptores de distribución del bean . . . . .	21-18
Compilación del proyecto . . . . .	21-19
Ejecución del bean Cart . . . . .	21-19

Código para el bean sesión de carro de la compra . . . . .	21-20
--	-------

## Capítulo 22

### Tutorial: Creación de aplicaciones cliente de prueba **22-1**

Apertura del proyecto cart_session . . . . .	22-1
Utilización del Asistente para clientes de prueba EJB . . . . .	22-2
Examinar el código generado . . . . .	22-3
Adición de código al cliente de prueba . . . . .	22-4
Creación de una instancia del bean Cart . .	22-4
Adición y eliminación de artículos del carro . . . . .	22-5
Final de la compra . . . . .	22-7
Eliminar la instancia del bean . . . . .	22-7
Resumen de los artículos del carro . . . . .	22-9
Compilación de la aplicación cliente de prueba . . . . .	22-9
Ejecución del cliente de prueba . . . . .	22-10
Código para la aplicación cliente de prueba .	22-12

## Parte II

### Tecnologías compatibles

#### Capítulo 23 Creación de productores y consumidores JMS **23-1**

Utilización del asistente JMS . . . . .	23-2
Sistemas de mensajes de publicar y suscribir . . . . .	23-3
Sistemas de mensajes punto a punto . . . . .	23-4
Código adicional . . . . .	23-5

#### Capítulo 24 Las aplicaciones distribuidas basadas en CORBA **24-1**

Definición de CORBA . . . . .	24-1
Definición del ORB de VisiBroker . . . . .	24-2
Trabajo conjunto de JBuilder y el ORB de VisiBroker . . . . .	24-2
Configuración de JBuilder para aplicaciones CORBA . . . . .	24-4
Definición de interfaces en Java . . . . .	24-7
Acerca de los compiladores java2iiop y java2idl . . . . .	24-7

Acceso a los compiladores java2iiop y java2idl en JBuilder . . . . .	24-8
RMI . . . . .	24-9
Utilización del compilador java2iiop . . . . .	24-10
Creación de interfaces IIOP ejecutando java2iiop . . . . .	24-10
Correspondencia de tipos de datos primitivos a IDL . . . . .	24-12
Correspondencia de tipos de datos complejos . . . . .	24-13
Utilización del compilador java2idl . . . . .	24-14

## Capítulo 25

### Tutorial: Creación de aplicaciones CORBA

**25-1**

Paso 1: Configuración del proyecto . . . . .	25-2
Paso 2: Definición de las interfaces de los objetos CORBA en IDL . . . . .	25-3
Paso 3: Generación de stubs de cliente y servidores . . . . .	25-4
Archivos generados . . . . .	25-4
Paso 4: Implementación del cliente . . . . .	25-5
Realización de una asociación con el objeto AccountManager . . . . .	25-7
Asociación de la clase englobadora durante la ejecución . . . . .	25-7
Paso 5: Implementación del servidor . . . . .	25-9
Definición de POA . . . . .	25-10
Paso 6: Cómo obtener una implementación para la interfaz CORBA . . . . .	25-11
Paso 7: Compilación de la aplicación . . . . .	25-12
Paso 8: Ejecución de la aplicación Java . . . . .	25-12
Inicio del agente inteligente del ORB de VisiBroker . . . . .	25-12
Inicio del servidor . . . . .	25-13
Ejecución del cliente . . . . .	25-13
Distribución de la aplicación . . . . .	25-14
Otras aplicaciones de ejemplos . . . . .	25-15

## Apéndice A

### Sugerencias acerca de los servidores Borland

**A-1**

Configuración y uso de Borland Enterprise Server 5.0.2 -5.1.x . . . . .	A-1
Particiones, servicios de partición y API J2EE . . . . .	A-1
Archivos de propiedades del servidor . . . . .	A-2
Cambio del puerto de administración . . . . .	A-3

Configuración de la seguridad del agente de administración . . . . .	A-3
Configuración de la seguridad de la partición . . . . .	A-4
Configuración de la política del cargador de clases para la partición . . . . .	A-4
Configuración de la distribución de WAR expandidos para la partición . . . . .	A-4
Asuntos internacionales . . . . .	A-4
Borland Enterprise Server como servidor web de aplicaciones con JBuilder . . . . .	A-5
Configuración del servidor . . . . .	A-5
Agente de gestión . . . . .	A-6
Inicio del servidor . . . . .	A-6
Distribución . . . . .	A-7
Ejecución del cliente de prueba . . . . .	A-8
Soluciones a problemas de aplicaciones web . . . . .	A-8
La consola de Borland Enterprise Server con JBuilder . . . . .	A-9
Configuración y uso de Borland AppServer 4.5.1 . . . . .	A-9
<b>Apéndice B Sugerencias acerca de WebLogic Server</b>	<b>B-1</b>
Configuración de JBuilder . . . . .	B-1
WebLogic 7.x . . . . .	B-1
Configuración de WebLogic 7.x en JBuilder 8 . . . . .	B-2
WebLogic 6.x . . . . .	B-2
WebLogic 5.1 . . . . .	B-3
Utilización de código ya creado . . . . .	B-3
Creación de beans entidad WebLogic en JBuilder . . . . .	B-3
El editor de descriptor de distribución . . . . .	B-4
Compilación . . . . .	B-4
Distribuir . . . . .	B-4
Depuración remota de Páginas JavaServer . . . . .	B-5
Asuntos internacionales . . . . .	B-6
<b>Apéndice C Sugerencias acerca de WebSphere Application Server</b>	<b>C-1</b>
Instalación de WebSphere y configuración de JBuilder . . . . .	C-1
WebSphere Server 3.5 . . . . .	C-1
Generación de descriptores de distribución de WebSphere . . . . .	C-2
Importación de descriptores de WebSphere 4.0 Advanced Edition a JBuilder . . . . .	C-4
Distribución de enterprise beans en servidores WebSphere . . . . .	C-4
El editor de descriptor de distribución . . . . .	C-5
Ejecución y distribución . . . . .	C-5
Activar la depuración de WebSphere 4.0 Advanced Edition . . . . .	C-6
Activación de la depuración remota . . . . .	C-7
WebSphere Server 3.5 . . . . .	C-7
WebSphere Single Server 4.0 . . . . .	C-7
WebSphere Server Advanced Edition 4.0 . . . . .	C-8
<b>Apéndice D Sugerencias acerca de iPlanet Application Server</b>	<b>D-1</b>
iPlanet 6.0 Service Pack 3 . . . . .	D-1
Ejecución o depuración en iPlanet . . . . .	D-1
Iniciar y detener iPlanet desde dentro de JBuilder . . . . .	D-1
Ejecución o depuración de un servlet o JavaServer Page . . . . .	D-2
Preparación para la depuración remota de aplicaciones iPlanet . . . . .	D-2
Configuraciones de distribución . . . . .	D-3
Creación de clientes para iPlanet . . . . .	D-3
Inicio de la vista web . . . . .	D-3



# Tutoriales

Desarrollo de beans sesión con el diseñador de EJB . . . . .	21-1	Creación de aplicaciones CORBA. . . . .	25-1
Creación de aplicaciones cliente de prueba . .	22-1		



# 1

## Introducción

La *Guía del desarrollador de Enterprise JavaBeans* está dividida en tres partes:

- **Capítulo 2, “Programación con Java 2 Platform Enterprise Edition”**

Este capítulo explica las ventajas de las aplicaciones de Java<sup>TM</sup> 2 Platform, Enterprise Edition (J2EE<sup>TM</sup>) y describe cómo JBuilder le puede ayudar a desarrollarlas en equipo. También le sugiere los recursos que puede utilizar para aprender más sobre el desarrollo de aplicaciones J2EE.

- **Parte I, “Desarrollo de Enterprise JavaBeans”**

En Esta parte se explica la forma de crear Enterprise JavaBeans (EJB) con JBuilder y utilizarlos para generar sistemas distribuidos. JBuilder tiene un conjunto de diseñadores, asistentes y herramientas que simplifican enormemente la creación, la depuración, la comprobación y la distribución de enterprise beans. Se pueden crear beans enterprise para su distribución en el Borland Enterprise Server 5.0.2 – 5.1.x, el Borland AppServer 4.5, los BEA WebLogic Servers 6.x, 7.x y 5.1, los IBM WebSphere Application Servers 3.5 y 4.0, el Sybase Enterprise Application Server 4.5 y los Sun-Netscape iPlanet Application Servers 6.x+.

- **Parte II, “Tecnologías compatibles”**

Esta parte las funcionalidades de JBuilder para trabajar con dos tecnologías compatibles con la programación en J2EE, el Java Message Service<sup>TM</sup> (JMS<sup>TM</sup>) Common Object Request Broker Architecture (CORBA- Arquitectura común de agente de solicitud de objetos).

El Java Message Service, que es parte de J2EE, suministra las API necesarias para crear aplicaciones que utilizan un sistema enterprise de mensajes. JBuilder dispone de un Asistente para JMS que le permite generar aplicaciones que puede tanto producir como consumir mensajes.

Este apartado también explica lo que es CORBA y cómo puede utilizar JBuilder junto con el ORB de VisiBroker de Borland para desarrollar aplicaciones distribuidas basadas en CORBA.

- Apéndices

Los apéndices contienen sugerencias para trabajar con los servidores de aplicaciones compatibles. Aunque mucha de la información que contienen se encuentra distribuida a través de la *Guía del desarrollador de Enterprise JavaBeans*, puede encontrar información específica sobre su servidor en un lugar de los apéndices.

## 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 Hola 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 SET PATH.</li><li>• Código Java.</li><li>• Tipos de datos de Java, como boolean, int y long</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 void y static.</li></ul>
Negrita	<p>La negrita se utiliza para las herramientas java, bmj (Borland Make for Java), bcj (Borland Compiler for Java) y opciones del compilador. Por ejemplo: <b>javac, bmj, -vía de acceso a clases</b>.</p>

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

<b>Tipo de letra</b>	<b>Significado</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 <i>Esc</i> 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.
< >	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, <nombredarchivo> 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 <nombredarchivo> 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	<p>La ubicación del directorio inicial varía según la plataforma y se indica con una variable &lt;home&gt;.</p> <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.

# Programación con Java 2 Platform Enterprise Edition

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

Java<sup>TM</sup> 2 Platform Enterprise Edition (J2EE<sup>TM</sup>) es una arquitectura para una plataforma de desarrollo en Java para aplicaciones enterprise distribuidas, desarrollada por Sun Microsystems, con aportaciones de la comunidad de desarrolladores, incluido Borland. Los productos de la plataforma J2EE, como el servidor Borland Enterprise Server, ofrecen al desarrollador la posibilidad de crear aplicaciones con las siguientes ventajas:

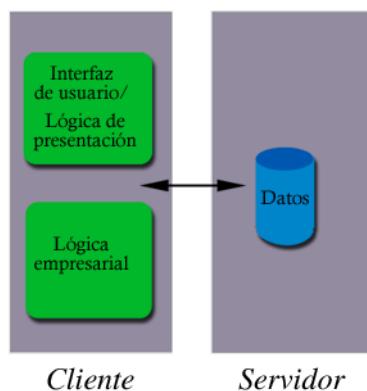
- Fiabilidad y flexibilidad de dimensión, con lo que las operaciones empresariales se procesan de forma más rápida y precisa.
- Excelente nivel de seguridad, que protege la privacidad del usuario y la integridad de los datos empresariales.
- Disponibilidad inmediata, para cumplir las demandas cada vez mayores del entorno empresarial mundial.

JBuilder es un entorno de desarrollo Java integrado que, cuando se acompaña de un servidor de aplicaciones compatible de empresas como Borland, BEA, IBM, Sun y Sybase, agiliza y simplifica en gran medida el desarrollo de aplicaciones J2EE.

## Ventajas de las aplicaciones J2EE

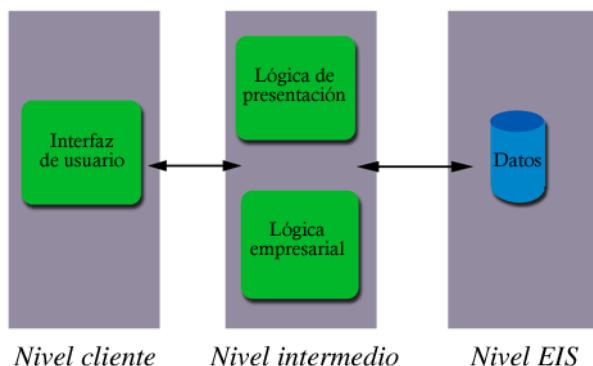
A principios de los años 90, los sistemas de información utilizaban, con frecuencia, una arquitectura cliente-servidor. La interfaz de usuario se ejecutaba en un ordenador de escritorio. Este era el nivel del cliente. Los datos empresariales a los que tenía acceso la aplicación cliente se encontraban en una base de datos y los “servía” un servidor. Este

acercamiento prometía una funcionalidad y flexibilidad de dimensión mejorada.



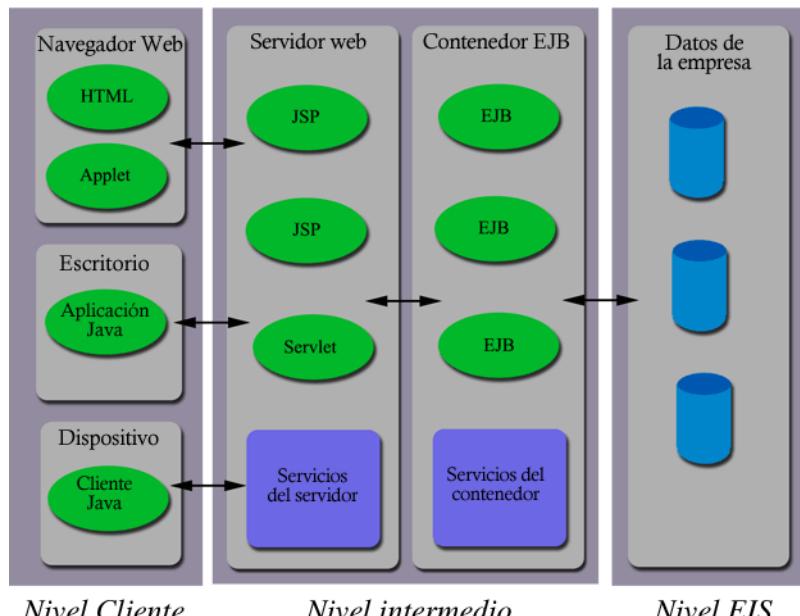
Con la experiencia, la comunidad de desarrolladores se dio cuenta de que es muy complicado crear y mantener un sistema distribuido flexible utilizando el modelo cliente-servidor. Por ejemplo, la lógica empresarial de la aplicación estaba en la aplicación cliente. Cada vez que había que modificar la lógica, había que instalar la aplicación cliente revisada en cada equipo cliente de la empresa. El mantenimiento se convirtió en una pesadilla. Estas aplicaciones también tenían que gestionar las transacciones, preocuparse de la seguridad y procesar los datos de forma eficaz, todo ello con una interfaz atractiva y fácil de comprender por parte del usuario. Muy pocos desarrolladores tenían aptitudes en todos estos campos. Aunque una arquitectura cliente-servidor puede ser adecuada para algunos entornos, la mayoría de las grandes empresas de hoy en día requieren mucho más de lo que puede ofrecer el modelo cliente-servidor.

Una vez que se hicieron evidentes las limitaciones del modelo cliente-servidor, la comunidad de desarrolladores empezó a buscar un modelo mejor. El resultado es el modelo multinivel.



En el modelo multinivel, la lógica necesaria para presentar al usuario la interfaz de usuario reside en el nivel intermedio. Ahora la lógica empresarial también se encuentra en el nivel intermedio. Si es necesario realizar cambios, se pueden llevar a cabo solamente en un lugar, sin necesidad de realizarlos en todos los equipos.

Este diagrama ampliado muestra los componentes que se ejecutan en cada nivel:



La aplicación cliente en una aplicación J2EE puede ser una página HTML o un applet que se ejecute en un visualizador, una aplicación Java de un ordenador de escritorio o, incluso, una aplicación cliente Java de algún dispositivo portátil, como un PDA (asistente personal digital) o un teléfono móvil.

El nivel intermedio puede contar con páginas de JavaServer™ o servlets que se ejecuten en un servidor web. Estos elementos componen la lógica de presentación del servidor. Los contenedores EJB proporcionan un entorno de ejecución para Enterprise JavaBeans™, que contiene la lógica empresarial de la aplicación. Tanto el servidor web como el contenedor Enterprise JavaBeans (EJB) ofrecen servicios a los componentes que se ejecutan en ellos. Gracias a que estos servicios siempre están disponibles, los programadores no necesitan incluirlos en los componentes que crean.

El nivel del Enterprise Information System (EIS - Sistema de información empresarial) es un repositorio de los datos de la empresa. Normalmente EIS se compone de la información contenida en un sistema de base de datos relacional.

Muy pocas aplicaciones J2EE cuentan con todos estos componentes. Por ello, se pueden mezclar y hacer que coincidan de forma muy flexible para que cubran las necesidades de la empresa.

## Ventajas del modelo multinivel

---

El concepto multinivel de la plataforma J2EE cuenta con numerosas ventajas:

- Reduce la complejidad del desarrollo distribuido gracias a una arquitectura simplificada y al hecho de que se comparte la carga de trabajo entre las competencias .

La lógica empresarial de la aplicación se ejecuta en el nivel intermedio dentro de un contenedor EJB y/o en un servidor web. Estos contenedores y servidores evitan al desarrollador multitud de tareas complejas. Por ejemplo, un contenedor EJB puede gestionar transacciones, agrupación de instancias y persistencia de datos sin necesidad de que el programador EJB deba escribir la lógica necesaria para llevar a cabo estas tareas. Un servidor web puede crear y agrupar instancias de clases de servlet y gestionar numerosos hilos y conexiones a través de sockets. En lugar de tener que escribir el código con el fin de que se realicen todas estas acciones, un miembro del equipo de desarrollo sólo tiene que definir el funcionamiento deseado en el momento de la distribución.

Los miembros del equipo de desarrollo desempeñan funciones distintas. Cada uno es especialista en una o más áreas. Por ejemplo, un diseñador gráfico o webmaster es el que debe crear el contenido de una página HTML u hoja de estilo. Un desarrollador con experiencia se encarga de la lógica empresarial de la aplicación encapsulada en los componentes EJB. Un desarrollador web se dedica a desarrollar la interfaz de usuario y la lógica de presentación mediante el uso de páginas JSP y servlets. Un ensamblador de aplicaciones fusiona todos los componentes de la aplicación a través de un archivo recopilatorio de Enterprise (EAR) y un descriptor de distribución que explica cómo se va a distribuir la aplicación. El distribuidor de aplicaciones y el administrador distribuyen la aplicación. Gracias a esta distribución del trabajo, cada parte del proceso de desarrollo y distribución lo lleva a cabo una persona cualificada en su área y, de esta forma, no es necesario que todos sean expertos en todos los campos.

- Esto hace que se ajuste a cualquier dimensión, por lo que el desarrollo de los sistemas se puede ir adaptando a las necesidades de cada momento.

Si aumenta la demanda al sistema, se puede actualizar la lógica fácilmente en un lugar del nivel intermedio, sin necesidad de descargar la nueva lógica en todos los equipos del cliente.

- Las nuevas aplicaciones se pueden integrar bien con los sistemas de información existentes.

JDBC, una tecnología J2EE, es una API de Java para bases de datos SQL, lo que permite que se acceda a cualquier tipo de información recogida en tablas que pueda existir en la empresa. La interfaz JNDI permite a las aplicaciones que utilizan tecnología Java tener acceso a los servicios de nomenclatura y directorio de la empresa. La arquitectura en desarrollo del conector J2EE aporta, a las aplicaciones Java, conexiones con sistemas heredados heterogéneos. El JMS (Java Message Service) es la API de Java utilizada para enviar y recibir mensajes mediante los sistemas de mensajes empresariales. Se llama a los servicios CORBA a través del JavaIDL.

- Se incrementa la seguridad.

Las tecnologías J2EE se han diseñado teniendo muy presente la seguridad. Por ejemplo, solamente los usuarios con competencias asignadas pueden tener acceso a determinados métodos de los beans enterprise. No es en estos beans donde se codifica quién tiene acceso a los métodos. Esta información está definida en los descriptores de distribución de los beans enterprise; el distribuidor utiliza estos descriptores para establecer su comportamiento una vez distribuidos. Este tipo de seguridad se denomina declarativa.

En el caso de una seguridad compleja, J2EE permite también seguridad programable. En estos casos, la lógica de seguridad se sitúa dentro del código.

- Los desarrolladores pueden elegir entre una gran variedad de herramientas, servidores y componentes para desarrollar las aplicaciones que necesiten.

El equipo de desarrollo puede seleccionar las soluciones que mejor se adapten a sus necesidades, sin tener que ajustarse a las ofertas de un solo fabricante.

## Ventajas de JBuilder

---

JBuilder Enterprise Edition cuenta con numerosas características que pueden serle de ayuda en el desarrollo de aplicaciones J2EE. Estas son las tecnologías con las que cuenta JBuilder para ayudarle a desarrollar el nivel del cliente.

## Tecnologías para el nivel del cliente

---

- Applets

Las applets son un tipo especial de aplicación Java que se descarga y se ejecuta desde un navegador en un equipo cliente. Para comenzar a desarrollar un applet en JBuilder, consulte en primer lugar el Asistente para applets. Si desea obtener más información sobre las applets, consulte el tutorial “Las applets” de la *Guía del desarrollador de aplicaciones Web*.

- Aplicaciones de interfaz de usuario de Java

JBuilder cuenta con numerosas características que pueden ayudarle a desarrollar aplicaciones que se ejecuten en un equipo cliente. Comience a crear una aplicación Java utilizando el Asistente para aplicaciones. Continúe el diseño de la interfaz de usuario utilizando el diseñador de interfaces de usuario de JBuilder. La interfaz de usuario se crea añadiendo componentes de interfaz de usuario de la paleta de componentes de JBuilder. Si desea más información acerca de la utilización del diseñador de interfaces de JBuilder, consulte “Diseño visual en JBuilder” en *Diseño de aplicaciones con JBuilder*.

Si desea crear sus propios componentes JavaBean para utilizarlos en la interfaz de usuario, BeansExpress puede facilitarle la tarea. Si desea obtener más información sobre la utilización de BeansExpress, consulte “Creación de JavaBeans con BeansExpress” en *Creación de aplicaciones con JBuilder*. Estos componentes para beans enterprise también le facilitan la generación de aplicaciones clientes con la ayuda de componentes visuales enlazados a bases de datos, tales como dbSwing o InternetBeans Express. Puede encontrar más detalles sobre DataExpress para EJB en el [Capítulo 14, “Uso de DataExpress para componentes EJB”](#).

Tanto un servidor web como un contenedor EJB pueden ejecutarse en el nivel intermedio. JBuilder se suministra con Tomcat, un contenedor de servlets que se puede utilizar como un servidor web, y con el Borland Enterprise Server 5.0.2. -5.1.x, que lleva el contenedor EJB. Puede generar aplicaciones para estos servidores, o bien, puede configurar JBuilder para que le permita desarrollar aplicaciones para los servidores BEA WebLogic 5.1 y 6.x, IBM WebSphere 3.5 y 4.0, Sun-Netscape iPlanet 6.0 y 6.5 y Borland AppServer 4.5.

## Tecnologías del nivel intermedio

---

Estas son las tecnologías J2EE de nivel intermedio que utiliza un servidor web:

- Servlets

Un servlet es una aplicación Java en el servidor que puede procesar peticiones de los clientes. El servlet responde a la petición generando una salida dinámica que se envía de vuelta al cliente. Puede comenzar a utilizar servlets con la ayuda del Asistente para servlets de JBuilder. Si desea obtener más información sobre los servlets y cómo desarrollarlos, consulte “Los servlets en la *Guía del desarrollador de aplicaciones Web*”.

- Páginas de JavaServer

Las páginas JSP son una extensión de la tecnología para servlets, y ofrecen un modo simplificado de desarrollarlos. Al igual que los servlets, generan una salida dinámica que se envía de vuelta al navegador del cliente. Puede comenzar a desarrollar páginas JSP con el Asistente para JSP de JBuilder. Si desea obtener más información sobre las JSP y cómo desarrollarlas, consulte “Las páginas JavaServer (JSP)” en la *Guía del desarrollador de aplicaciones Web*.

InternetBeans Express es una biblioteca de componentes que amplía la tecnología de servlets y JSP con la que cuenta JBuilder. Esta biblioteca facilita la presentación y manipulación de datos desde una base de datos para que pueda crear servlets y páginas JSP enlazados a datos.

Esta es la tecnología J2EE de nivel intermedio que utiliza un contenedor EJB:

- Enterprise JavaBeans (EJB)

Enterprise JavaBeans son componentes del servidor que contienen la lógica empresarial de la aplicación. JBuilder le ayuda a crear componentes EJB 1.x y EJB 2.0. Puede comenzar a generar beans enterprise utilizando los asistentes para EJB que se encuentran en la ficha EJB de la galería de objetos (Archivo | Nuevo | EJB). Si lo que desea es generar componentes EJB 2.0, JBuilder le ofrece el diseñador EJB, una herramienta bidireccional Two-Way ToolTM que permite diseñar visualmente los beans mientras mantiene sincronizados el código, los descriptores de distribución y el diseño. Si desea más detalles acerca de la creación, prueba y distribución de beans enterprise, consulte el [Capítulo 3, “Introducción al desarrollo de EJB”](#).

Mientras crea los beans enterprise, JBuilder genera los descriptores de distribución EJB. Puede utilizar el editor del descriptor de distribución de JBuilder para modificarlos como deseé. Si desea más información sobre la verificación del descriptor de distribución, consulte el [Capítulo 13, “El editor de descriptor de distribución”](#).

Todos los componentes de aplicaciones web y de enterprise beans se pueden combinar y enviar a un archivo EAR (Enterprise Archive). JBuilder cuenta con un Asistente para EAR que puede ayudarle a crear sus archivos EAR.

## Otras tecnologías J2EE

---

Gracias al hecho de que no están confinadas en un nivel particular de la arquitectura, estas tecnologías son activadores que hacen posible que todo funcione:

- JDBC (Conectividad con bases de datos Java)

JDBC es el estándar utilizado para acceder a una base de datos en el nivel de los sistemas EIS. Define la API de Java que se utiliza para crear las sentencias SQL que se envían a la base de datos.

JBuilder incluye DataExpress, una biblioteca de componentes para acceder a la información de la base de datos. Esta biblioteca conecta la aplicación con la base de datos mediante la utilización de controladores JDBC.

JBuilder también cuenta con JDataStore, una biblioteca Java de bases de datos y componentes incrustados. Se accede a JDataStore utilizando JDBC.

Los beans entidad, que son beans enterprise que acceden a las filas de la base de datos, también conectan con los datos mediante JDBC.

- Java Message Service (JMS)

JMS es un servicio de mensajería empresarial que dirige los mensajes entre los componentes y los procesos dentro de una aplicación distribuida.

Borland Enterprise Server incluye SonicMQ 4.0.1, una implementación JMS. JBuilder también admite componentes EJB 2.0, que incluyen beans gestionados por mensajes. Estos beans integran JMS en los beans enterprise. JBuilder incluye además un Asistente JMS. Si desea más información acerca de la creación de clases y aplicaciones que crean y consumen mensajes JMS, consulte el [Capítulo 23, “Creación de productores y consumidores JMS”](#).

- JNDI (Interfaz de Java para servicios de nomenclatura y directorio)

Todos los servidores J2EE utilizan JNDI, un servicio de nomenclatura Java que se utiliza para ubicar objetos distribuidos.

- Lenguaje de marcas ampliable (XML).

Aunque no es exactamente una tecnología J2EE, casi todas utilizan XML. Por ejemplo, los descriptores de distribución de los componentes web y los beans enterprise están escritos en XML. Estos descriptores de distribución describen el comportamiento de los componentes una vez distribuidos.

JBuilder cuenta con varias características XML que le ayudan a llevar a cabo tareas comunes de programación de sus proyectos J2EE. Si desea más información acerca de las características XML de JBuilder, consulte "Introducción" en *Guía del desarrollador de aplicaciones XML*.

## Preparación para distribuir aplicaciones J2EE

---

A la vez que crea y compila aplicaciones web y beans enterprise, JBuilder puede crear los archivos WAR y EJB-JAR (Recopilatorio EJB) de forma automática. Puede elegir si agrupa los componentes de una aplicación J2EE dentro del archivo EAR. JBuilder proporciona un Asistente para EAR que le ayuda en esta tarea.

## Más información acerca de J2EE

---

Si ha leído hasta este punto, le habrá podido parecer que todos los conceptos y siglas de las tecnologías J2EE que aparecen se asemejan a una sopa de letras. Para poder profundizar en los conceptos de J2EE y en sus ventajas, comience a investigar en la página web de Sun, [www.java.sun.com](http://www.java.sun.com). El enlace <http://java.sun.com/j2ee/docs.html> le llevará a la página principal de la documentación de Sun acerca de J2EE, donde encontrará abundante información de gran utilidad.

Si la programación en J2EE le resulta una novedad, consulte el tutorial J2EE en <http://java.sun.com/j2ee/tutorial/index.html>. Si le interesa un acercamiento más profundo sobre la programación en J2EE y las prácticas de programación recomendadas para sus aplicaciones J2EE, consulte el detallado J2EE Blueprints que se encuentran en <http://java.sun.com/j2ee/blueprints/index.html>. J2EE Blueprints constituye una parte fundamental de J2EE. Aunque no lo lea todo de una sola vez, le será muy útil para comprender conceptos más complicados y para encontrar la mejor manera de acercarse al desarrollo de J2EE. Este material también lo puede encontrar en el libro *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition*, escrito por Nicholas Kassem y el equipo Enterprise de Sun. En el enlace <http://java.sun.com/j2ee/blueprints/aboutthebook.html> encontrará más detalles sobre este libro.

En la página web de Sun, puede encontrar documentación adicional y detalles acerca de las diversas tecnologías J2EE. También existen excelentes manuales de otros autores, pero, ya que J2EE es un producto en desarrollo, compruebe a qué versiones de las diferentes tecnologías se refieren.



P a r t e

I

# Desarrollo de Enterprise JavaBeans



# Introducción al desarrollo de EJB

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

La “Especificación Enterprise JavaBeans (EJB)” define formalmente un modelo de componentes de servidor Java y una interfaz de programación para servidores de aplicaciones. Los desarrolladores construyen los componentes, llamados enterprise beans, que contienen la lógica empresarial. Los enterprise beans se ejecutan en un servidor EJB que proporciona a los beans servicios como la gestión de transacciones y la seguridad. Los desarrolladores no necesitan preocuparse por programar estos complejos servicios de bajo nivel, por lo que pueden concentrarse en encapsular las reglas empresariales de una organización o sistema dentro de los beans, conscientes de que estos servicios están a disposición de los beans cuando se necesitan.

Aunque la especificación Enterprise JavaBeans dicta las normas del subsistema EJB, resulta especialmente útil a los proveedores como Borland que construyen los servidores y contenedores EJB en los que se ejecutan los beans. Este libro ayuda a los desarrolladores de JBuilder a aprender lo necesario sobre el desarrollo de enterprise beans.

## Finalidad de los Enterprise JavaBeans

El modelo cliente-servidor de desarrollo de aplicaciones goza de una popularidad considerable. La aplicación cliente se encuentra en el ordenador local y accede a los datos de un almacén, como un sistema de gestión de bases de datos (SGBDR). Este modelo funciona bien mientras el sistema tenga sólo unos pocos usuarios. A medida que haya más usuarios que necesiten acceso a los datos, estas aplicaciones tendrán dificultades para cumplir sus necesidades. El cliente contiene el código lógico, por lo

que es necesario instalarlo en todos los ordenadores, y la gestión se hace cada vez más difícil.

Las ventajas de dividir las aplicaciones en más niveles que los dos que componen el modelo cliente-servidor resultan evidentes. En las aplicaciones multinivel, sólo la interfaz de usuario permanece en los ordenadores locales, mientras la lógica de la aplicación se ejecuta en el nivel intermedio, en un servidor. El nivel final sigue siendo el de los datos almacenados. Cuando es necesario actualizar la lógica de una aplicación se efectúan cambios en el software del nivel intermedio del servidor, con lo que se simplifica enormemente la gestión de actualizaciones.

Sin embargo, la creación de aplicaciones distribuidas fiables, seguras y fáciles de gestionar resulta muy compleja. Por ejemplo, la tarea de gestionar transacciones en un sistema distribuido es considerable.

Afortunadamente, el uso de componentes acordes con la especificación EJB para crear sistemas distribuidos aligera gran parte de la carga, ya que:

- Divide el desarrollo de un sistema distribuido en tareas específicas que se asignan a especialistas.

Por ejemplo, si la aplicación es un sistema de contabilidad, el desarrollador de enterprise beans debe tener conocimientos de contabilidad. El administrador del sistema debe saber cómo controlar una aplicación distribuida en ejecución. Cada especialista desempeña un papel distinto.

- Pone el servidor EJB y los servicios de contenedor a disposición de los desarrolladores de enterprise beans y aplicaciones.

El proveedor del servidor EJB y el proveedor del contenedor EJB (que a menudo son el mismo distribuidor) gestionan muchas de las tareas más difíciles para que no necesiten efectuarlas los desarrolladores. Por ejemplo, el contenedor que ejecuta un enterprise bean puede proporcionar servicios transaccionales y de seguridad al bean de forma automática.

- Hace portables los enterprise beans.

Cuando se escribe un bean, se puede distribuir en cualquier servidor EJB que cumpla la norma Enterprise JavaBeans. Sin embargo, cada bean puede incluir elementos específicos del fabricante.

## Competencias en el desarrollo de una aplicación EJB

---

El trabajo de desarrollo de aplicaciones EJB distribuidas se divide en seis competencias independientes. Cada una de ellas está a cargo de un experto en el área. Al dividir así el trabajo, la tarea de crear y gestionar sistemas distribuidos se hace mucho más fácil.

## Competencias de la aplicación

---

Quienes desempeñan las competencias de aplicación escriben el código de los enterprise beans y las aplicaciones que los utilizan. Las dos competencias requieren la comprensión del funcionamiento del negocio, aunque a distintos niveles. Éstas son las dos competencias de la aplicación:

- Proveedor de beans

Los proveedores o desarrolladores de beans crean los enterprise beans y escriben el código de los métodos empresariales que contienen.

También definen las interfaces base local o base remota y remota o local de los beans y crean sus descriptores de distribución. Los proveedores de beans no necesitan saber cómo se ensamblarán y distribuirán sus beans.

- Ensamblador de aplicaciones

Los ensambladores escriben las aplicaciones que utilizan los enterprise beans. Normalmente, estas aplicaciones incluyen otros componentes, como interfaces de usuario clientes, applets, páginas del servidor Java (JSP) y servlets. Estos componentes se ensamblan en una aplicación distribuida. Los ensambladores añaden instrucciones de ensamblaje a los descriptores de distribución de beans. Aunque los ensambladores de aplicaciones deben conocer los métodos que contienen los enterprise beans para poder llamarlos, no es necesario que sepan cómo se implementan.

Normalmente, los usuarios de JBuilder interesados en Enterprise JavaBeans son proveedores de beans y ensambladores de aplicaciones. Por tanto, este libro está dirigido a ellos principalmente. JBuilder tiene asistentes, diseñadores y otras herramientas que simplifican el desarrollo de enterprise beans y las aplicaciones que los utilizan.

## Competencias de infraestructura

---

Los enterprise beans y la aplicación que los utiliza no se pueden ejecutar sin el apoyo de una infraestructura. Aunque las competencias de las dos infraestructuras son distintas, lo habitual es que las efectúe el mismo proveedor. Juntas proporcionan a los enterprise beans servicios de sistema y un entorno de ejecución. Éstas son las dos competencias de infraestructura:

- Proveedor de servidores EJB

Están especializados en la gestión de transacciones distribuidas, los objetos distribuidos y otros servicios de bajo nivel. Proporcionan un marco de aplicación en el que se ejecutan los contenedores EJB. Los proveedores de servicios EJB deben facilitar a los beans, como mínimo, un servicio de denominación y otro de transacciones.

- Proveedor de contenedores EJB

Proporcionan las herramientas de distribución necesarias para desarrollar enterprise beans y su aceptación durante la ejecución. El contenedor proporciona servicios de gestión a uno o varios beans. Llevan a cabo la comunicación con el servidor EJB para acceder al servicio que necesita el bean.

En casi todas las clases, el proveedor de servidores EJB y el proveedor de contenedores EJB son el mismo. Borland Enterprise Server proporciona tanto el servidor como el contenedor.

## Competencias de distribución y operación

---

Los últimos pasos en el desarrollo de una aplicación distribuida EJB consisten en distribuir la aplicación y controlar la ejecución en los ordenadores de la empresa y la infraestructura de la red. Éstas son las competencias de distribución y operación:

- Distribuidor

Los distribuidores entienden el entorno de operación de las aplicaciones distribuidas. Adaptan la aplicación EJB al entorno de operación de destino modificando las propiedades del enterprise bean con las herramientas que suministra el proveedor de contenedores. Por ejemplo, los distribuidores establecen las normas de transacciones y seguridad definiendo las propiedades correspondientes en el descriptor de distribución. También integran la aplicación con los programas de gestión de la empresa.

- Administrador del sistema

Cuando una aplicación está distribuida, el administrador del sistema controla su ejecución y lleva a cabo las acciones adecuadas si se comporta de forma anómala. Los administradores del sistema son responsables de la configuración y la administración de la infraestructura informática y de red que incluye el servidor EJB y el contenedor EJB.

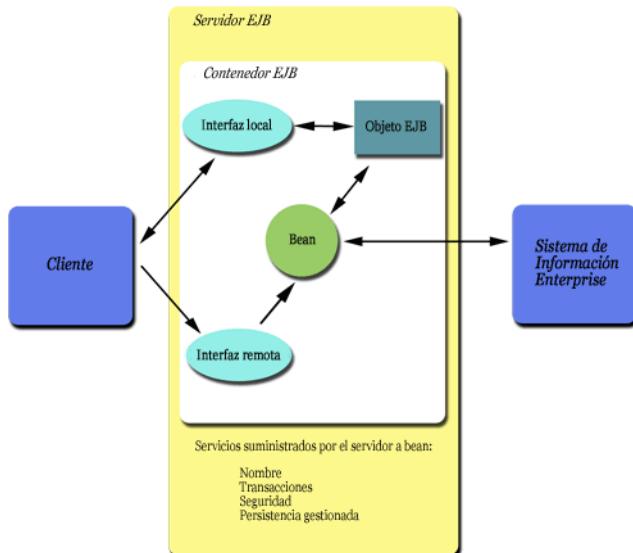
## Arquitectura de EJB

---

Las aplicaciones distribuidas multinivel constan a menudo de un cliente que se ejecuta en un ordenador local, un nivel intermedio que se ejecuta en un servidor que contiene la lógica empresarial y un nivel de fondo que consiste en un sistema de información empresarial (EIS). El EIS puede ser un sistema de base de datos relacional, un sistema de ERP, una aplicación antigua o cualquier almacén de datos que guarda los datos a los que es necesario acceder. En esta figura se muestra un sistema multinivel

distribuido típico de EJB, con tres niveles: el cliente, el servidor, el contenedor, y los beans distribuidos en estos, más el sistema de información empresarial.

**Figura 3.1** Diagrama de arquitectura EJB



Dado que este libro trata principalmente sobre la forma de desarrollar enterprise beans, nos concentraremos en el nivel intermedio.

## El servidor EJB

El servidor EJB proporciona a los enterprise beans servicios de sistema y gestiona los contenedores en los que se ejecutan los beans. Debe facilitar un servicio de denominación accesible para la JNDI y un servicio de transacciones. Con frecuencia, los servidores EJB proporcionan funciones adicionales que los distinguen de los competidores. Borland Enterprise Server, AppServer Edition 5.0.2 -5,1.x es un ejemplo de servidor EJB.

## El contenedor EJB

Un contenedor es un sistema de ejecución para uno o varios enterprise beans. Proporciona la comunicación entre los beans y el servidor EJB. También gestiona las transacciones, la seguridad y la distribución de la red. Los contenedores son a la vez código y herramientas que generan código específico para un enterprise bean. También proporcionan herramientas para la distribución de enterprise beans y permiten al contenedor controlar y gestionar la aplicación.

El servidor EJB y el contenedor EJB proporcionan en conjunto el entorno de ejecución del bean. El contenedor proporciona servicios de gestión a uno o varios beans. El servidor proporciona servicios al bean, pero el contenedor interacciona en representación de los beans que obtienen estos servicios. En la mayoría de los casos, el servidor EJB y el contenedor EJB son del mismo fabricante y forman parte de un mismo servidor de aplicaciones, como Borland Enterprise Server, AppServer Edition 5.0.2 - 5.1.x.

Aunque forman una parte fundamental de la arquitectura de Enterprise JavaBeans, los desarrolladores de enterprise beans y los ensambladores de aplicaciones no necesitan preocuparse por el contenedor. En los entornos EJB distribuidos funciona en segundo plano. Por tanto, no es necesario extenderse en este libro sobre el funcionamiento de los contenedores. Si desea más información sobre ellos, consulte la "Enterprise JavaBeans 1.1 Specification" en <http://java.sun.com/products/ejb/docs.html>. Si desea información más específica acerca del contenedor EJB de Borland, consulte la documentación del Borland Enterprise Server.

## Fucionamiento de los enterprise beans

---

El desarrollador de beans debe crear estas interfaces y clases:

- La interfaz base remota y/o base local del bean.

La interfaz base define los métodos que emplea el cliente para crear, localizar y destruir instancias de un enterprise bean.

- La interfaz remota y/o local del bean.

La interfaz remota o local define los métodos empresariales que se implementan en el bean. Los clientes acceden a estos métodos a través de la interfaz remota.

- La clase enterprise Bean.

La clase del enterprise bean implementa la lógica empresarial del bean. Los clientes acceden a estos métodos a través de la interfaz remota del bean.

Cuando el bean se distribuye en el contenedor EJB, el cliente llama al método `create()` definido en la interfaz base para que cree una instancia del bean. La interfaz local no se implementa en el bean, sino en el contenedor. Otros métodos declarados en la interfaz base permiten al cliente localizar una instancia de un bean y eliminarla cuando deja de ser necesaria. Los beans EJB 2.0 también permiten que la interfaz base posea métodos empresariales llamados métodos `ejbHome`.

Cuando se crea una instancia del enterprise bean, el cliente llama a los métodos empresariales que contiene. Sin embargo, el cliente no llama nunca directamente a un método de la instancia del bean. Los métodos

que se encuentran a disposición del cliente están definidos en la interfaz remota o local del bean, implementada por el contenedor. Cuando el cliente llama a un método, el contenedor recibe la solicitud y la delega en la instancia del bean.

## Tipos de enterprise beans

---

Un enterprise bean puede ser un bean sesión, entidad o gestionado por mensajes.

### Beans sesión

---

Los beans sesión pueden ser de dos tipos: con estado y sin estado. Los beans sin estado no mantienen el estado para un cliente. Debido a que no mantienen estado conversacional, estos beans se pueden utilizar para admitir varios clientes.

Los beans sesión con estado se ejecutan para un solo cliente. En cierto modo, los beans sesión representan al cliente en el servidor EJB. Los beans sesión pueden mantener el estado del cliente, lo que significa que éste puede utilizarlos para retener información. El ejemplo clásico de uso de beans sesión es el carro de una persona que compra en una tienda por Internet. A medida que el comprador selecciona los artículos para añadirlos al carro, el bean sesión retiene una lista de los elementos seleccionados.

La duración de los beans sesión puede ser muy breve. Normalmente, cuando termina la sesión, el cliente elimina el bean.

### Beans entidad

---

Los beans entidad permiten a los objetos consultar bases de datos. Normalmente, el bean representa a una fila en el seno de un conjunto de tablas de base de datos relacional. Los beans entidad se suelen utilizar para más de un cliente.

A diferencia de los beans sesión, normalmente tienen una duración larga. Mantienen un estado persistente y duran mientras los datos permanecen en la base de datos, y no sólo mientras los necesita un cliente.

El contenedor puede gestionar la persistencia del bean, o la puede gestionar el bean mismo. Si el bean gestiona la persistencia, su desarrollador debe escribir código que incluya llamadas a la base de datos.

## Beans gestionado por mensajes

---

La especificación EJB 2.0 introduce los beans gestionado por mensajes. Se comportan como monitores de Java Message Service (JMS - Servicio de mensajes Java) que procesan mensajes asíncronos. El contenedor EJB gestiona todo el entorno del bean.

Los beans gestionados por mensajes son parecidos a los beans sesión sin estado, ya que no conservan el estado conversacional, pero a diferencia de los beans sesión y entidad, los clientes no acceden a ellos por medio de interfaces. Los beans gestionados por mensajes no tienen interfaces; sólo tienen una clase de bean. Un solo bean de este tipo puede procesar mensajes de varios clientes. Básicamente, los beans gestionados por mensajes son bloques de código de aplicación que se ejecutan cuando llega un mensaje a un destino JMS determinado.

## Acceso local y remoto

---

Se puede acceder a los componentes EJB 2.0 de forma remota o local. Los clientes que acceden a beans remotos utilizan sus interfaces remota y base remota. Con frecuencia, la interfaz base remota se considera como la interfaz base. Los clientes con acceso remoto a un bean pueden ejecutarlo en otra máquina y utilizar una máquina virtual de Java (MVJ) distinta de la del bean. En las llamadas a métodos de beans remotos, los parámetros se pasan por valor, lo que ayuda a mantener una conexión fluida entre el cliente y el bean.

Los clientes con acceso local a un bean deben ejecutarlo en la misma MVJ del bean al que se accede. Los clientes locales no son aplicaciones cliente externas, sino otros enterprise beans y componentes web. En las llamadas a métodos de beans locales, los parámetros se pasan por referencia, con lo que la conexión entre el bean o el componente web que realiza la llamada y el bean llamado es más estable.

Al igual que la interfaz remota, la interfaz local proporciona acceso a los métodos empresariales del bean, mientras que la interfaz base local proporciona acceso a los métodos que controlan el ciclo de vida y los métodos de búsqueda del bean. Es frecuente que los beans entidad que tienen una relación gestionada por contenedor con otros beans entidad cuenten con acceso local a ellos.

Dado que los beans con interfaces locales se deben ejecutar en la misma MVJ, no es necesario realizar llamadas remotas. De este modo, se reduce la sobrecarga de serialización y transporte de objetos. Por lo general, esto mejora el rendimiento.

# 4

## Desarrollo de enterprise beans

En los siguientes capítulos se explica la forma de utilizar los asistentes, diseñadores y herramientas de JBuilder con el objeto de agilizar y facilitar la creación de enterprise beans. Para estudiarlo es necesario saber qué son los enterprise beans, cómo funcionan y cuáles son sus requisitos.

Si no cuenta con los conocimientos suficientes sobre los EJB o desea más información sobre su desarrollo antes de empezar a utilizar los asistentes y herramientas para EJB de JBuilder, empiece por leer el [Capítulo 15, “Desarrollo de beans sesión”](#) y los siguientes.

El desarrollo de Enterprise JavaBeans con JBuilder tiene varios pasos:

Paso	Para obtener más información, consulte el
1 Configuración del servidor de aplicaciones de destino	<a href="#">Capítulo 5, “Configuración del servidor de aplicaciones de destino”</a>
2 Creación de módulos EJB	<a href="#">“Los módulos EJB” en la página 6-2</a>
3 Creación de beans sesión y gestionados por mensajes 2.0 con el diseñador de EJB y Creación de beans entidad 2.0 con el diseñador de EJB, o bien  Creación de componentes EJB 1.x con JBuilder y Creación de beans entidad EJB 1.x a partir de una tabla de base de datos	<a href="#">Capítulo 6, “Creación de beans sesión y gestionados por mensajes 2.0 con el diseñador de EJB”</a> y <a href="#">Capítulo 7, “Creación de beans entidad 2.0 con el diseñador de EJB”</a> , o bien  <a href="#">“Creación de enterprise beans” en la página 8-5</a> y <a href="#">Capítulo 9, “Creación de beans entidad EJB 1.x a partir de una tabla de base de datos”</a>
4 Compilación de enterprise beans y creación de archivos JAR	<a href="#">Capítulo 10, “Compilación de enterprise beans y creación de archivos JAR”</a>
5 Modificación de los descriptores de distribución	<a href="#">Capítulo 13, “El editor de descriptor de distribución”</a>
6 Ejecución y prueba de enterprise beans	<a href="#">Capítulo 11, “Ejecución y prueba de enterprise beans”</a>
7 Distribución en un servidor de aplicaciones	<a href="#">“Distribución en un servidor de aplicaciones” en la página 12-7</a>



# Configuración del servidor de aplicaciones de destino

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

Antes de empezar a crear Enterprise JavaBeans es necesario configurar el servidor de aplicaciones en el que se van a distribuir los enterprise beans.

**Sugerencia**

Asegúrese de consultar los apéndices de sugerencias de los distintos servidores para encontrar información de configuración adicional para cada servidor. Éstos son los apéndices:

- [Apéndice A, “Sugerencias acerca de los servidores Borland”](#)
- [Apéndice B, “Sugerencias acerca de WebLogic Server”](#)
- [Apéndice C, “Sugerencias acerca de WebSphere Application Server”](#)
- [Apéndice D, “Sugerencias acerca de iPlanet Application Server”](#)
- [Apéndice E, “Sugerencias acerca de Sybase Enterprise Application Server”](#)

## Servidores admitidos

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

JBuilder es compatible con los siguientes servidores:

- Borland App Server 4.5.1
- Borland Enterprise Server 5.0.2, 5.1, 5.1.1
- WebLogic Application Server 5.1 SP 11 (y todas las versiones anteriores de 5.1 service packs).

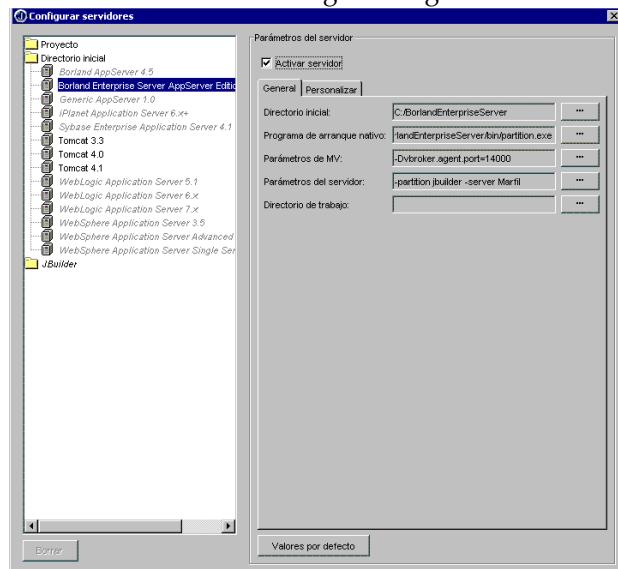
- WebLogic Application Server 6.0 SP 2 (y todas las versiones anteriores de 6.0 service packs).
- WebLogic Application Server 6.1 SP 3 (y todas las versiones anteriores de 6.1 service packs).
- WebLogic Application Server 7.0 SP 1 (y todas las versiones anteriores de 7.0 service packs).
- WebSphere Application Server 3.5 Fix Pack 6 (y todas las versiones anteriores de WebSphere 3.5 service packs).
- WebSphere Application Server 4.0.1 Single Server.
- WebSphere Application Server 4.0.1 Advanced Edition.
- iPlanet Application Server 6.0 SP 4
- iPlanet Application Server 6.5
- Sybase EAServer 4.1.1
- Tomcat 3.3.1
- Tomcat 4.0.6
- Tomcat 4.1.12

## Configuración de servidores con JBuilder

Para configurar uno o varios servidores de aplicaciones de destino:

1 Seleccione Herramientas | Configurar servidores.

Se abre el cuadro de diálogo Configurar servidores:



En la parte izquierda del cuadro de diálogo aparecen los servidores para los que JBuilder encuentra una OpenTool registrada.

- 2** Seleccione el servidor que deseé configurar pulsando sobre él en el panel de la izquierda.

En la parte derecha del cuadro de diálogo aparece la lista de las opciones por defecto para el servidor seleccionado. Todos los servidores de aplicaciones excepto el Generic App Server 1.0 tienen una ficha General y una Personalizar, para modificar la configuración. La ficha General cuenta con campos que tienen todos los servidores de aplicaciones, y la ficha Personalizar tiene campos específicos del servidor seleccionado. En algunos casos, la modificación de la configuración de Personalizar actualizará la configuración de la ficha General.

La opción Generic AppServer 1.0 es una opción genérica. Representa un servidor de aplicaciones básico que es compatible con el desarrollo de EJB 1.1 y/o EJB 2.0. Selecciónela si JBuilder no acepta actualmente el servidor de aplicaciones que va a utilizar. Probablemente será necesario modificar el descriptor de distribución resultante con las herramientas del servidor de aplicaciones, con el fin de obtener la configuración deseada. También se puede elegir esta opción si no se pretende utilizar un servidor de aplicaciones determinado.

- 3** Marque la casilla Activar servidor de la parte superior del cuadro de diálogo.

Al marcar esta opción se activan los campos del servidor seleccionado. Hasta que no la seleccione no podrá modificar ningún campo. La casilla de selección Activar servidor también determina si el servidor aparece en la lista de servidores cuando se selecciona uno para el proyecto a través de la ficha Servidores del cuadro de diálogo Proyecto | Propiedades de proyecto.

JBuilder proporciona una configuración por defecto para el servidor seleccionado. Si la configuración por defecto de JBuilder no es la apropiada, puede modificarla con el fin de adaptarla a sus necesidades. Algunos servidores necesitan que se introduzcan los parámetros además de la configuración por defecto. Pulse el botón Aceptar en este cuadro de diálogo cuando no estén definidos todos los valores necesarios o cuando al seleccionar otro servidor mientras que el actual está activado aparezca un cuadro de diálogo que informe sobre los valores que faltan.

- 4** Si desea modificar alguna opción, pulse el botón de puntos suspensivos junto al campo y realice los cambios.

Puede añadir las dependencias necesarias del cualquier aplicación a la vía de acceso a clases del servidor pulsando la pestaña Bibliotecas

necesarias y utilizando el botón Añadir para añadir las bibliotecas que necesita el servidor.

- 5 Abra la pestaña Personalizar para ver los campos que son exclusivos del servidor. La mayoría de los servidores de aplicaciones le permiten especificar la localización del JDK que utiliza el servidor; algunos requieren que especifique la localización JDK para completar satisfactoriamente la configuración. Las actualizaciones que realice en la ficha Personalizar pueden actualizar a su vez la configuración de la ficha General.

A continuación se relacionan los JDK y sus ubicaciones para los servidores de aplicaciones admitidos por JBuilder 7

- Borland Enterprise Server 5.0.2/5.1+: <raíz de bes>/jdk
- Borland AppServer 4.5: <sun jdk1.2.2>
- WebLogic 5.1: <raíz de weblogic>/jre1\_2/jre
- WebLogic 6.0: <raíz de bea>/jdk130
- WebLogic 6.x: <raíz de bea>/jdk131
- WebLogic 7.0: <raíz de bea>/jdk131\_02
- WebSphere 3.5: <raíz de websphere>/jdk
- WebSphere 4.0 (tanto Advanced Edition como Single Server): <raíz de websphere>/java
- Sybase Enterprise Application Server 4.1: <raíz de Sybase>/\_jvm
- IPlanet: ninguna

Una vez realizadas las modificaciones, pulse el botón Aceptar con el fin de que se activen los cambios y cierre el cuadro de diálogo. A continuación, JBuilder realiza una pasada de validación en la configuración del servidor. Si prefiere que JBuilder no intente validar las modificaciones, quite la marca del cuadro de diálogo Activar servidor. Las modificaciones se guardan al pulsar Aceptar, aunque puede volver más tarde con el fin de realizar más cambios antes de validar la configuración. Sin embargo, debe marcar la casilla Activar servidor antes de poder seleccionar el servidor que desea configurar para utilizar en el proyecto. Al seleccionar Aceptar, se graban los cambios en todos los servidores que tenga activados, desactivados o que haya modificado.

- 6 Pulse Aceptar cuando haya finalizado la configuración del servidor de aplicaciones que desea utilizar para desarrollar los beans enterprise.

Si pulsa Aceptar y está marcada la casilla Activar servidor, JBuilder intenta verificar la configuración. Si se detecta algún error, aparece un mensaje con los datos acerca de esos errores. Si no se ha marcado la opción Activar servidor, se cierra el cuadro de diálogo sin validar la configuración.

Una vez que se ha configurado y validado de forma adecuada, el servidor seleccionado aparece en negrita en el panel de la izquierda. Los caracteres en gris indican qué servidores no se han configurado todavía. Los caracteres en rojo indican que hay algún error, normalmente debido a que se ha leído la biblioteca, pero no se ha podido encontrar el servidor. Puede eliminar el servidor que aparezca en rojo si lo selecciona y pulsa el botón Borrar.

El botón Valores por defecto asigna a las opciones los valores que tenían originalmente cuando se instaló JBuilder, y desactiva el servidor.

Si aparece un cuadro de diálogo con un mensaje informándole de que debe reiniciar JBuilder, debe cerrar y reiniciar JBuilder en ese momento .

- Nota** En el caso de Borland Enterprise Server 5.0.2 -5.1.x, se define el puerto del Agente inteligente de Visibroker en la ficha Corba del cuadro de diálogo Herramientas | Configurar Enterprise. Consulte "["Cómo poner el ORB a disposición de JBuilder"](#) en la página 5-7.
- Nota** En el caso de WebSphere 3.5, debe instalar el WebSphere FixPack2 con el objeto de conseguir la versión más actualizada del JDK de IBM.

## Cómo añadir un service pack

---

Si desea añadir un service pack a la configuración de su servidor de aplicaciones, siga estos pasos:

- 1 Seleccione Herramientas | Configurar servidores.
- 2 Seleccione su servidor en la lista de servidores del panel del lado izquierdo del cuadro de diálogo.
- 3 Haga clic en la pestaña General y después en la pestaña Clase, si no están ya seleccionadas.
- 4 Pulse Añadir.
- 5 Navegue hasta la ubicación del archivo JAR del service pack (como `weblogic51sp10boot.jar`).
- 6 Haga clic en Aceptar para cerrar todos los cuadros de diálogo.

## Las bibliotecas generadas

---

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

Al configurar el servidor, se crean automáticamente una o más bibliotecas; estas bibliotecas contienen todos los archivos del servidor de aplicaciones necesarios para el desarrollo de beans enterprise en el servidor que elija. Estas son las bibliotecas que se han creado, ordenadas por servidor de aplicaciones:

- **Borland Enterprise Server 5.0.2 – 5.1.x**

Borland Enterprise Server 5.0.2 -5,1.x Client: Todos los archivos JAR necesarios para ejecutar un cliente.

- **Borland AppServer 4.51**

BAS 4.5 Client: Todos los archivos JAR necesarios para ejecutar un cliente.

- **Sybase Enterprise Application Server 4.1**

EAServer 4.1 Client: Todos los archivos JAR necesarios para ejecutar un cliente.

- **WebLogic 7.x**

WebLogic 7.x Client: Todos los archivos JAR necesarios para ejecutar un cliente.

WebLogic 7.x Deploy: Archivos JAR necesarios para ejecutar la herramienta de distribución WebLogic 7.x.

- **WebLogic 6.x**

WebLogic 6.x Client: Todos los archivos JAR necesarios para ejecutar un cliente.

WebLogic 6.x Deploy: Archivos JAR necesarios para ejecutar la herramienta de distribución WebLogic 6.x.

- **WebLogic 5.1**

WebLogic 5.1 Client: Todos los archivos JAR necesarios para ejecutar un cliente.

WebLogic 5.1 Deploy: Archivos JAR necesarios para ejecutar la herramienta de distribución WebLogic 5.1.

- **WebSphere 4.0 Single Server**

WebSphere AES 4.0 Client: Archivos JAR necesarios para ejecutar un cliente.

WebSphere AES 4.0 ExtDirs: JAR utilizados durante el inicio del servidor.

WebSphere AES 4.0 EjbDeploy: Archivos JAR utilizados para compilar beans enterprise y crear stubs.

WebSphere AES 4.0 SeAppInstaller: Archivos JAR necesarios para ejecutar la herramienta de distribución de WebSphere 4.0 (SeAppInstaller).

- **WebSphere 4.0 Advanced Edition**

WebSphere AE 4.0 Client: Archivos JAR necesarios para ejecutar un cliente.

WebSphere AE 4.0 ExtDirs: JAR utilizados durante el inicio del servidor.

WebSphere AE 4.0 XmlConfig: Archivos JAR necesarios para ejecutar la herramienta de distribución de WebSphere 4.0 (SeAppInstaller).

WebSphere AE 4.0 EjbDeploy: Archivos JAR utilizados para compilar beans enterprise y crear stubs.

- **WebSphere 3.5**

WebSphere 3.5 Client: Archivos JAR necesarios para ejecutar un cliente. Los jar distribuidos para WebSphere se añaden automáticamente a esta definición de biblioteca.

WebSphere 3.5 JetAce : Archivos JAR utilizados para compilar beans enterprise y crear stubs.

WebSphere 3.5 XMLConfig : Archivos JAR necesarios para ejecutar la herramienta de distribución de WebSphere 3.5.

- **iPlanet 6.x+**

iPlanet 6.x+ Client: Archivos JAR necesarios para ejecutar un cliente.

**Importante** Cuando importa proyectos desarrollados con una versión anterior de JBuilder, debe actualizarlos para poder utilizar las últimas bibliotecas cliente.

## Cómo poner el ORB a disposición de JBuilder

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

Si utiliza Herramientas | Configurar servidores para configurar el servidor Borland Enterprise Server 5.02 – 5.1.x, la configuración de CORBA se realiza automáticamente al mismo tiempo. Puede consultar la configuración actual en la ficha CORBA del cuadro de diálogo Herramientas | Configurar Enterprise. Si lo desea, puede utilizar esta ficha para asignar al puerto del Agente inteligente de VisiBroker un número irrepetible. Además, si desea agregar un comando en el menú Herramientas para iniciar el Agente inteligente, marque la opción Añadir el agente inteligente de VisiBroker al menú Herramientas. Esta opción aparece seleccionada por defecto.

- Nota** Si está utilizando Borland AppServer 4.5, debe configurar CORBA manualmente.
- Nota** Si no ha instalado Borland Enterprise Server, deberá completar los siguientes pasos con el fin de poder utilizar idl2java o java2iop de VisiBroker:
- 1 Seleccione Proyecto | Propiedades de proyecto.
  - 2 Pulse sobre la pestaña Generar y, a continuación, en la ficha Generar, pulse sobre la pestaña IDL.
  - 3 Seleccione VisiBroker como compilador IDL (si aún no está seleccionado).
  - 4 En el campo Opciones adicionales añada lo siguiente:  

```
-VBJprop borland enterprise.licenseDir=<directorio de licencia de visibroker>  
-VBJjavavm<jdk1.3.1 java o vía de acceso a javaw>
```

(Para Borland Enterprise Server 5.1, el directorio de licencia correcto es \<Raíz de BES51>\var\servers\<directorío del servidor>\adm.) El <directorío del servidor> coincide con el nombre del servidor asignado al configurar mediante la opción de menú Herramientas | Configurar servidores. Puede encontrar el nombre del servidor en la ficha Personalizar de este cuadro de diálogo.
- En Borland Enterprise Server 5.0.2 -5.1.x**
- Asegúrese de que Borland Enterprise Server está utilizando JDK 1.3.1.
- 5 Pulse Aceptar.
- Todavía debe llevar a cabo un paso más: Iniciar el Agente inteligente de VisiBroker. Este agente gestiona los primeros pasos del arranque como, por ejemplo, la localización a cargo del cliente del servicio de nombres. Para iniciar el agente inteligente, elija Herramientas | VisiBroker Smart Agent.
- En Borland Enterprise Server 5.0.2 -5.1.x**
- Si selecciona Herramientas | Agente de gestión de Borland Enterprise, el Agente inteligente de VisiBroker también se iniciará como parte del proceso. Si utiliza Borland Enterprise Server AppServer Edition 5.0.2 - 5.1.x, debe iniciar el agente de gestión de Borland Enterprise Server en lugar del agente inteligente de VisiBroker. Si no lo inicia, se inicia automáticamente cuando inicia en servidor en JBuilder.

# Selección de un servidor

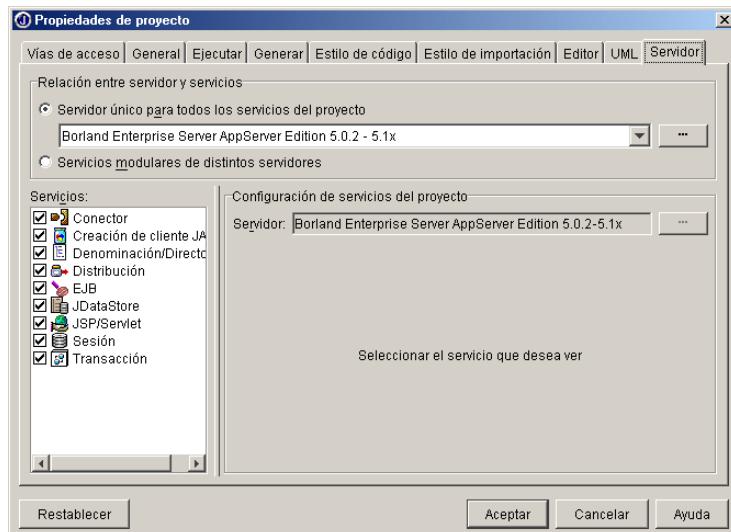
**La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.**

JBuilder puede utilizar como destino una gran variedad de servidores. Puede seleccionar un solo servidor de aplicaciones para todos los pasos de EJB y del desarrollo de aplicaciones web, o bien, puede seleccionar diferentes servidores para los diferentes aspectos del desarrollo. Por ejemplo, se puede escoger un servidor para su uso en el desarrollo de Enterprise Beans y otro para desarrollar aplicaciones web.

Para seleccionar uno o más servidores para su proyecto:

- 1 Seleccione Proyecto | Propiedades de proyecto.
- 2 Pulse la pestaña Servidor.

Se muestra la ficha Servidor:



- 3 Decida si desea utilizar un solo servidor para todos los aspectos del desarrollo o varios servidores que gestionen las distintas áreas.

- Para utilizar un único servidor:

- 1 Seleccione la opción Servidor único para todos los servicios del proyecto y elija el servidor en la lista desplegable.
- 2 Si desea realizar cambios a los parámetros de configuración del servidor seleccionado, haga clic sobre el botón con puntos suspensivos y modifique los parámetros deseados en las fichas General y Personalizar. Pulse Aceptar.

También puede utilizar este cuadro de diálogo para seleccionar un servidor diferente.

- 3 Si no desea que se inicie un servicio determinado cuando se inicia el servidor, desactive la casilla de selección próxima al Servicio en la lista Servicios. Esta función se aplica actualmente sólo a Borland Enterprise Server como el único que le permite iniciar y parar servicios seleccionados. Si desactiva los servicios Distribución o Creación de cliente JAR, los correspondientes elementos del menú se desactivarán en el menú Herramientas de JBuilder.
  - Para utilizar diferentes servidores en diferentes servicios,
    - 1 Seleccione la opción Servicios modulares de distintos servidores.
    - 2 Marque las casillas correspondientes a aquellos servicios para los que deseé especificar un servidor de aplicaciones en la lista Servicios.
    - 3 Pulse en uno de los servicios marcados para seleccionarlo en la lista Servicios.
    - 4 En el cuadro de grupo Configuración de servicios del proyecto, utilice la lista desplegable Servidor para seleccionar el servidor que deseé utilizar para este servicio.
- Si las propiedades del servidor que ha seleccionado están disponibles, aparecen bajo la lista desplegable Servidor. Si se selecciona el servicio Distribución, aparece una lista desplegable llamada Tipo de generación para distribuir al servidor, en el panel Configuración de servicios del proyecto. Seleccione el tipo de generación que deseé. Si selecciona el servicio EJB, aparece una información de sólo lectura que debe resultarle de ayuda para decidir qué servidor se ajusta más a sus necesidades.
- Para Borland Enterprise Server**

La opción jndi-definitions.xml de Distribución determina si el archivo jndi-definitions.xml se incluye en alguno de las acciones de distribución que seleccione. No la marque si no lo quiere incluir. jndi-definitions.xml es un descriptor XML de distribución de recursos EJB 2.0.
- Para WebLogic 6.x – 7.x**

Si selecciona el servicio JSP/Servlet, las propiedades de servicio incluyen una opción Mapear las aplicaciones del proyecto durante la ejecución. Cuando esta opción se selecciona (el valor por defecto) todas las aplicaciones web del proyecto se distribuyen desde el directorio de la aplicación web y no como un WAR cuando se inicia el servidor.

  - 5 Si desea realizar modificaciones en la configuración del servidor seleccionado para este servicio, pulse el botón de puntos suspensivos y cambie las opciones que deseé en las fichas General y Personalizar. Pulse Aceptar.
  - 6 Repita estos pasos para cada servicio al que deseé acceder.
- 4 Pulse Aceptar.

Si los asistentes para EJB de la ficha Enterprise de la galería de objetos están desactivados, no se selecciona ningún servidor de aplicaciones para el proyecto actual o el servidor seleccionado no está configurado. Siga los pasos detallados anteriormente para seleccionar un servidor de aplicaciones para el desarrollo de EJB.

## Configuración de los controladores JDBC

---

Para permitir que los asistentes para EJB de JBuilder y el diseñador de EJB tengan acceso a una base de datos, debe instalar el controlador JDBC que suministra el fabricante de bases de datos, y configurarlo en JBuilder. Instale el controlador JDBC siguiendo las indicaciones del fabricante.

Para comenzar la configuración del controlador en JBuilder, seleccione Herramientas | Configurar Enterprise de modo que aparezca el cuadro de diálogo Configurar Enterprise. Seleccione la pestaña Controladores de bases de datos para ver la ficha homónima. Utilice esta ficha para añadir controladores de bases de datos a JBuilder.

### Creación de los archivos .library y .config

---

Son tres los pasos para añadir un controlador de base de datos a JBuilder:

- Crear un archivo de biblioteca que contenga las clases de controladores, generalmente un archivo JAR, y algún archivo adicional como los de documentación o código fuente.
- Derivar un archivo .config del archivo de biblioteca que JBuilder añade a su vía de acceso a clases al iniciarse.
- Añadir una biblioteca al proyecto, o si desea que esté disponible para todos los nuevos proyectos, al proyecto por defecto.

Los primeros dos pasos pueden realizarse desde la ficha Controladores de bases de datos:

- 1 Abra JBuilder y seleccione Herramientas | Configurar Enterprise. Haga clic en la pestaña Controladores de base de datos que muestra los archivos .config de todos los controladores de bases de datos actualmente conocidos.
- 2 Haga clic en Añadir con el fin de incorporar un controlador y, a continuación, pulse Nuevo para crear un archivo de biblioteca para el controlador. El archivo de biblioteca se utiliza para añadir un controlador a la lista de bibliotecas necesarias de los proyectos.

**Nota** También puede crear una biblioteca desde Herramientas | Configurar bibliotecas, pero como después tendría que utilizar Configurar

Enterprise para obtener el archivo .config, es más sencillo hacerlo todo desde aquí.

- 3** Escriba un nombre y seleccione una ubicación para el nuevo archivo, en el cuadro de diálogo Crear biblioteca.
- 4** Haga clic en Añadir y desplácese hasta la ubicación del controlador. Puede seleccionar el directorio que contiene el controlador y todos sus archivos de apoyo, o puede simplemente seleccionar el archivo recopilatorio del controlador. Ambas opciones funcionarán. JBuilder extrae la información necesaria.
- 5** Pulse Aceptar para cerrar el visualizador de archivos. Esto muestra la nueva biblioteca en la última posición de la lista de bibliotecas y la selecciona.
- 6** Pulse Aceptar. JBuilder crea un archivo de biblioteca en el directorio /lib de JBuilder con el nombre asignado (por ejemplo, InterClient.library). Además lo lleva a la ficha Controladores de bases de datos que muestra el nombre del archivo .config correspondiente, que se derivará del archivo de biblioteca (por ejemplo, InterClient.config).
- 7** Seleccione el nuevo archivo .config en la lista de controladores de bases de datos y haga clic en Aceptar. Esto sitúa el archivo .config en el directorio /lib/ext de JBuilder.
- 8** Cierre y reinicie JBuilder con el fin de que los cambios realizados en los controladores de bases de datos tengan efecto, y el nuevo controlador se ubique en la vía de acceso a clases de JBuilder.

**Importante** Si se modifica el archivo .library tras la derivación del archivo.config habrá de volver a generar este último utilizando Configurar Enterprise, y reiniciar JBuilder.

Ahora que JBuilder puede ver al controlador de bases de datos, debe añadir la biblioteca del controlador a la lista Bibliotecas necesarias en Proyecto | Propiedades, o Proyecto | Propiedades por defecto.

## Adición del controlador JDBC a proyectos

---

Los proyectos que se ejecutan desde dentro de JBuilder sólo utilizan su propia vía de acceso a clases. Por lo tanto, para asegurarse de que el controlador JDBC esté disponible para los nuevos proyectos que lo necesiten, defina la biblioteca y añádala a la lista por defecto de bibliotecas necesarias. Esto se hace desde JBuilder, de la siguiente forma:

- 1** Inicie JBuilder y cierre los proyectos abiertos.
- 2** Elija Proyecto | Propiedades de proyecto por defecto.

- 3 Haga clic en la pestaña Bibliotecas necesarias, de la ficha Vías de acceso, y pulse Añadir.
- 4 Seleccione el nuevo controlador JDBC de la lista de bibliotecas y haga clic en Aceptar.
- 5 Haga clic en Aceptar para cerrar el cuadro de diálogo Propiedades del proyecto por defecto.

**Nota** También puede añadir el controlador JDBC a un proyecto existente. Simplemente abra el proyecto, seleccione Proyecto | Propiedades y utilice el procedimiento arriba descrito.



# Creación de beans sesión y gestionados por mensajes 2.0 con el diseñador de EJB

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic

En este capítulo se explica la forma de utilizar JBuilder para crear beans sesión y gestionados por mensajes compatibles con la especificación Enterprise JavaBeans™ 2.0 de Sun Microsystems. Si desea crear beans entidad EJB 2.0 y no está familiarizado con JBuilder y su diseñador de EJB, empiece por este capítulo, ya que abarca los fundamentos básicos para poder trabajar con el diseñador. Después puede pasar al [Capítulo 7, “Creación de beans entidad 2.0 con el diseñador de EJB”](#), en el que se explican las peculiaridades de la creación de beans entidad.

Si desea información sobre la forma de crear componentes compatibles con la especificación Enterprise JavaBeans™ 1.1, consulte el [Capítulo 8, “Creación de componentes EJB 1.x con JBuilder”](#).

Para ayudarle a crear sus propios beans compatibles con EJB 2.0, el diseñador de EJB le proporciona un entorno de desarrollo rápido de aplicaciones (RAD) para el desarrollo de EJB 2.0. El diseñador de EJB es una herramienta bidireccional (Two-Way Tool™) que permite diseñar visualmente enterprise beans mientras JBuilder genera el código a partir del diseño. El diseño se puede modificar por medio del diseñador de EJB o escribiendo directamente en el código fuente generado. El código y el diseño permanecen sincronizados. Cuando se trabaja con el diseñador de EJB se crean los descriptores de distribución, con lo que se prepara el bean para su distribución en el servidor de aplicaciones de destino.

# Los módulos EJB

---

Todos los enterprise beans deben pertenecer a un módulo EJB de JBuilder. Un módulo EJB es una agrupación lógica de uno o más beans que se distribuirán en un solo archivo JAR. Contiene la información necesaria para producir los descriptores de distribución del archivo JAR. Puede modificar el contenido de un módulo EJB mediante el Editor de descriptor de distribución.

Después de crear un módulo EJB y modificarlo a voluntad con el editor del descriptor de distribución, se puede Ejecutar Make o Generar un módulo para crear el archivo JAR. JBuilder utiliza el descriptor de distribución para ayudar en la identificación de los archivos de clase que se incluyen en el archivo.

Los módulos EJB pueden tener dos formatos: XML y binario. Dado que los módulos EJB de formato XML son básicamente archivos de texto, resulta más fácil trabajar con ellos si se utiliza un sistema de control de versiones. Los módulos EJB en formato binario son, en esencia, los descriptores de distribución en un recopilatorio .zip.

Es posible tener varios modulos EJB en un proyecto. Todos los módulos EJB de un proyecto utilizan la misma vía de acceso a clases y el mismo JDK, y están configurados para el mismo servidor de aplicaciones de destino.

Si todavía no lo ha hecho, siga las instrucciones que se detallan en el capítulo anterior, [Capítulo 5, “Configuración del servidor de aplicaciones de destino”](#).

## Creación de módulos EJB 2.0

---

Hay varias formas de crear un módulo EJB:

- Si aún no ha creado los enterprise beans, cree un módulo EJB con el asistente.
- Creación de módulos EJB a partir de descriptores de distribución.

### Creación de módulos con el Asistente para grupos EJB vacíos

Si aún no ha creado los enterprise beans, empiece por crear un módulo EJB con el asistente.

Para crear un módulo EJB:

- 1 Seleccione Archivo | Nuevo y haga clic en la pestaña Enterprise.

**Nota**

Si los asistentes para EJB de la ficha Enterprise están desactivados, significa que no tiene instalada la versión Enterprise de JBuilder o que no ha seleccionado un servidor de aplicaciones configurado para el

proyecto. Asegúrese también de que ha seleccionado un servidor de aplicaciones y no un servidor web. Si desea información sobre la forma de configurar y seleccionar el servidor de aplicaciones, consulte el [Capítulo 5, “Configuración del servidor de aplicaciones de destino”](#).

- 2** Haga doble clic en el ícono Módulo EJB y aparecerá el asistente:



Si no hay un proyecto abierto cuando se ejecuta el Asistente para grupos EJB vacíos, el asistente Proyecto aparecerá en primer lugar. Cree un proyecto y aparecerá el Asistente para grupos EJB vacíos.

- 3** Indique el nombre del módulo EJB.
- 4** Indique el formato del nuevo módulo.

Puede elegir entre binario, que almacena internamente los descriptores de distribución con formato .zip, y que se utilizaba antes de JBuilder 5, o bien XML, que almacena los descriptores de distribución con formato XML. El formato XML permite a los usuarios fusionar los cambios si se están comprobando en un sistema de control de fuentes. Se recomienda utilizar el formato XML.

- 5** En la lista desplegable Versión, indique que la versión utilizada es compatible con EJB 2.0.

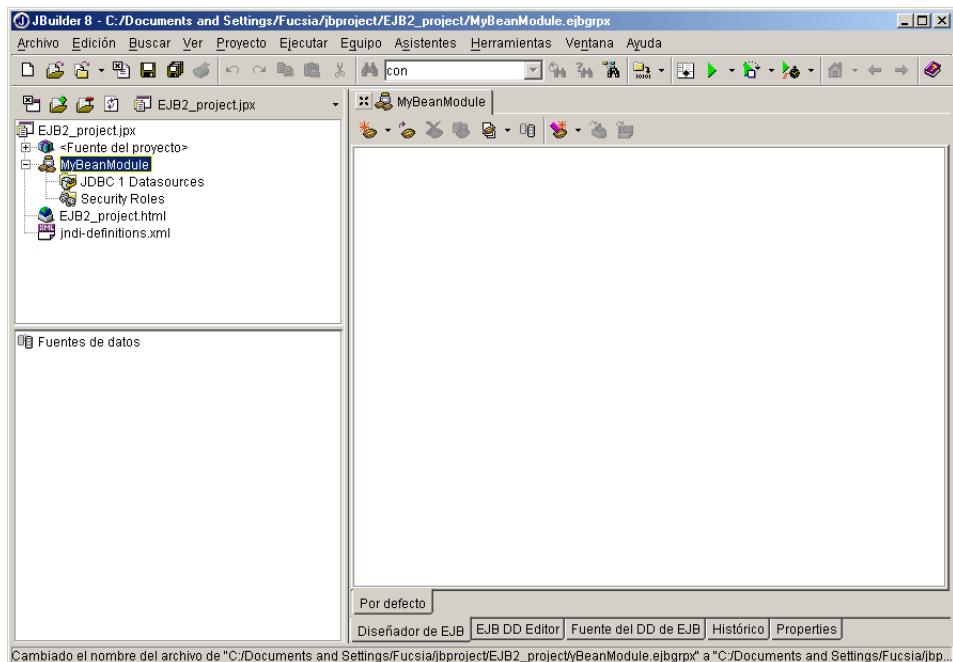
Si el servidor de aplicaciones de destino no es compatible con EJB 2.0, esta opción se encontrará desactivada.

- 6** Indique el nombre del archivo JAR en el que se encontrarán los enterprise beans.

JBuilder ha introducido un nombre por defecto, que es el mismo que el del módulo EJB. Basta con aceptar este nombre o escribir otro. JBuilder también ha introducido una ruta basada en la vía de acceso de su proyecto. Puede modificarla a su gusto o bien aceptar la ruta predeterminada.

## 7 Pulse Aceptar para crear el módulo EJB.

Aparece el diseñador de EJB. Por ejemplo, si se inicia un proyecto llamado EJB2\_Project y se indica que el módulo EJB se llama MyBeanModule, el diseñador de EJB, el panel de proyecto y el panel de estructura tienen el siguiente aspecto:



## Creación de módulos EJB a partir de descriptores de distribución

Si tiene descriptores de distribución para enterprise beans EJB 2.0 creados previamente, puede crear un módulo EJB que los contenga. Existen dos posibilidades:

- Utilizar el Asistente para módulo EJB a partir de descriptores, que crea un módulo EJB que contiene los descriptores de distribución.
- Utilizar el asistente para Proyecto para código existente, que crea un proyecto de JBuilder con nuevos módulos EJB que contienen los descriptores de distribución. El asistente sólo reconoce Borland AppServer 4.5, Borland Enterprise Server AppServer Edition 5.0.2 - 5.1.x, WebLogic 5.1 y posteriores descriptores de distribución específicos de un fabricante.

Tanto el Asistente para módulo EJB a partir de descriptores como el asistente para Proyecto para código existente integran la información específica del fabricante en un formato habitual de JBuilder, con el que se

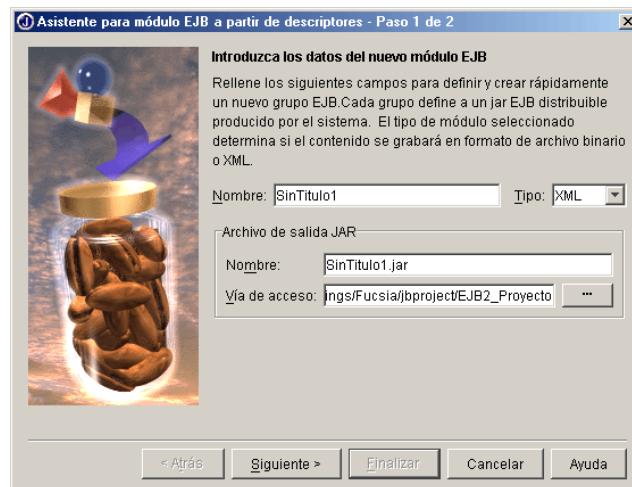
generan los descriptores de distribución del servidor de aplicaciones seleccionado para el proyecto.

### **Utilización del Asistente para módulos EJB a partir de descriptores**

El Asistente para módulos EJB a partir de descriptores sólo importa el descriptor de distribución ejb-jar.xml y los descriptores de distribución específicos de BEA WebLogic y Borland Enterprise Server a un nuevo módulo EJB.

Para utilizar el Asistente para módulos EJB a partir de descriptores:

- 1** Seleccione Archivo | Nuevo y haga clic en la pestaña Enterprise.
- 2** Haga doble clic en el ícono del Asistente para módulo EJB a partir de descriptores y éste aparecerá:



Si no hay un proyecto abierto cuando se ejecuta el Asistente para módulo EJB a partir de descriptores, el asistente para Proyectos aparecerá en primer lugar. Cree un proyecto y aparecerá el Asistente para módulos EJB a partir de descriptores.

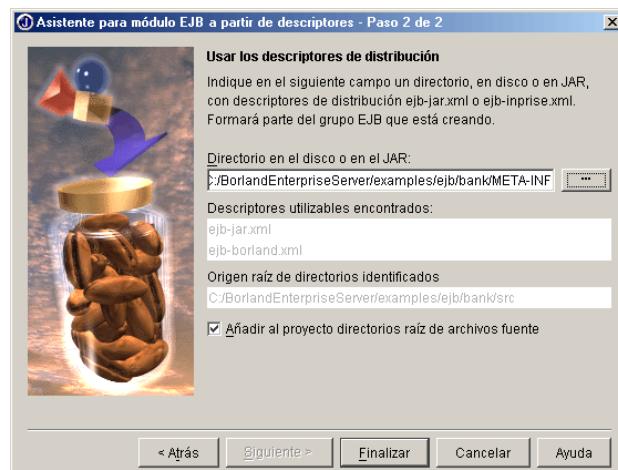
- 3** Indique el nombre del nuevo módulo EJB.
- 4** Indique el formato del nuevo módulo.

Puede elegir entre binario, que almacena internamente los descriptores de distribución con formato .zip, y que se utilizaba antes de JBuilder 5, o bien XML, que almacena los descriptores de distribución con formato XML. El formato XML permite a los usuarios fusionar los cambios si se están comprobando en un sistema de control de fuentes. Se recomienda utilizar el formato XML.

- 5** Indique el nombre y la vía de acceso del archivo JAR en el que se encontrará el enterprise bean.

JBuilder ha introducido un nombre por defecto, que es el mismo que el del módulo EJB. Basta con aceptar este nombre o escribir otro.

- 6** Pulse Siguiente e indique el directorio que contiene los descriptores de distribución que desea utilizar para el módulo. (Habitualmente se encuentra en el directorio META-INF del JAR.) Al hacerlo, el asistente elabora una lista de los descriptores de distribución en el directorio especificado en el campo Descriptores EJB identificados. También muestra el directorio raíz de archivos fuente de estos descriptores en el campo Directorio raíz de archivos fuente.



- 7** Si desea añadir al proyecto los directorios raíz de archivos fuente, active la casilla de selección Añadir al proyecto directorios raíz de archivos fuente.
- 8** Pulse Finalizar para crear el módulo EJB que contiene los descriptores de distribución de los beans.

### Utilización del Asistente para Proyecto para código existente

El asistente para Proyecto para código existente crea un proyecto de JBuilder a partir de un cuerpo de trabajo anterior. Este cuerpo de trabajo puede incluir descriptores de distribución EJB 1.1 o 2.0 específicos del fabricante, si se han desarrollado con Borland Enterprise Server 5.0.2 -5.1, WebLogic Server 5.1 o posteriores.

Para utilizar el Asistente para Proyecto para código existente con el fin de crear un proyecto de JBuilder y un módulo EJB por cada directorio que contenga descriptores de distribución EJB que el asistente busca, siga estos pasos:

- 1** Para mostrar la galería de objetos, seleccione Archivo | Nuevo.
- 2** Pulse sobre la pestaña Proyecto.
- 3** Haga doble clic sobre el ícono Proyecto para código existente. Se inicia el asistente.
- 4** Pulse el botón de puntos suspensivos para indicar el directorio raíz en el que desea que el asistente empiece a buscar.

Por ejemplo, suponga que el directorio `clients` contiene dos subdirectorios llamados `personal_info` y `accounts`, que contienen EJB con descriptores de distribución. Se desea crear dos módulos EJB: `personal_info` y `accounts`. Si se indica `clients` como el Directorio, el asistente empieza a buscar por el directorio `clients` y a continuación examina los directorios `personal_info` y `accounts`. Todos los módulos EJB generados reciben un nombre por defecto basado en el nombre del subdirectorio en el que se han encontrado los descriptores. De esta forma, el proyecto creado por el asistente contiene dos módulos EJB: `personal_info` y `accounts`.

Si desea más información sobre la utilización del resto de las fichas del asistente para Proyecto para código existente, pulse sobre el botón Ayuda del asistente o consulte “Creación de proyectos a partir de archivos existentes” en el capítulo “Creación y gestión de proyectos” de *Creación de aplicaciones con JBuilder*.

## El diseñador de EJB

---

El diseñador de EJB se utiliza para desarrollar enterprise beans EJB 2.0. Existen dos formas de abrir este diseñador para dar comienzo a la creación de un enterprise bean:

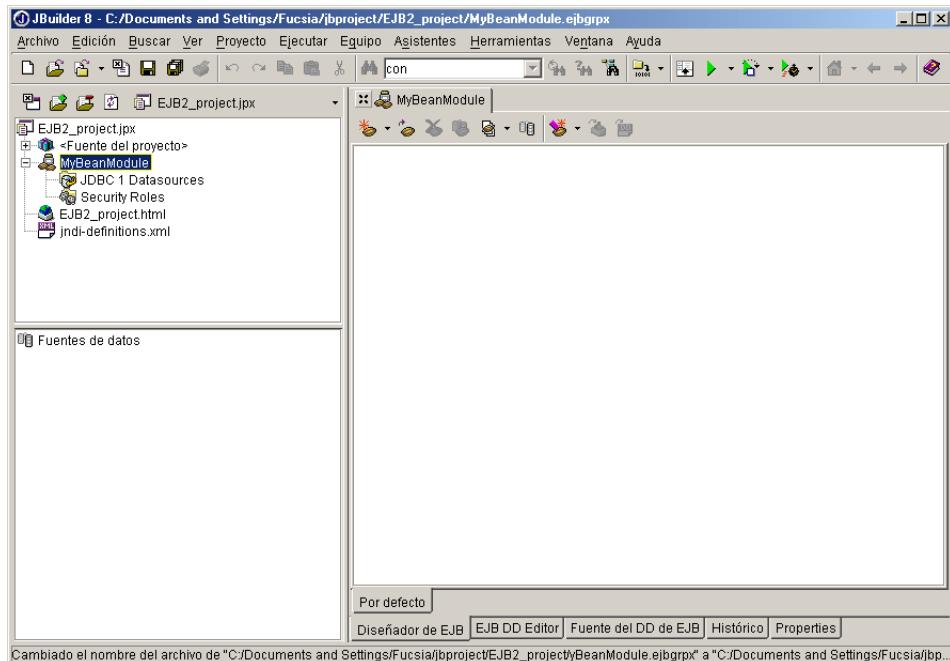
- Cree un módulo con el Asistente para grupos EJB vacíos. Si elige que el módulo que está creando sea compatible con EJB 2.0, el asistente muestra el módulo en el panel de proyecto y abre el diseñador de EJB, listo para la configuración del enterprise bean. Consulte “[Creación de módulos con el Asistente para grupos EJB vacíos](#)” en la página 6-2 para obtener más información.
- Utilice el asistente Diseñador de EJB 2.0. Éstos son los pasos necesarios:
  - a Elija Archivo | Nuevo, abra la pestaña Enterprise y haga doble clic en Diseñador de EJB 2.0.
  - b Seleccione en la lista Módulos EJB disponibles el módulo del que desea que forme parte el bean que va a crear.

En la lista aparecen únicamente los módulos EJB 2.0 disponibles. Si aún no tiene ningún módulo, pulse Nuevo. Se abre el Asistente para

módulos EJB. Cuando finaliza la ejecución de este asistente se vuelve al asistente Diseñador de EJB 2.0.

**c** Pulse Aceptar.

Aparece el diseñador de EJB. Por ejemplo, si se inicia un proyecto llamado **EJB2\_Project** y se indica que el módulo EJB se llama **MyBeanModule**, el diseñador de EJB, el panel de proyecto y el panel de estructura tienen el siguiente aspecto:



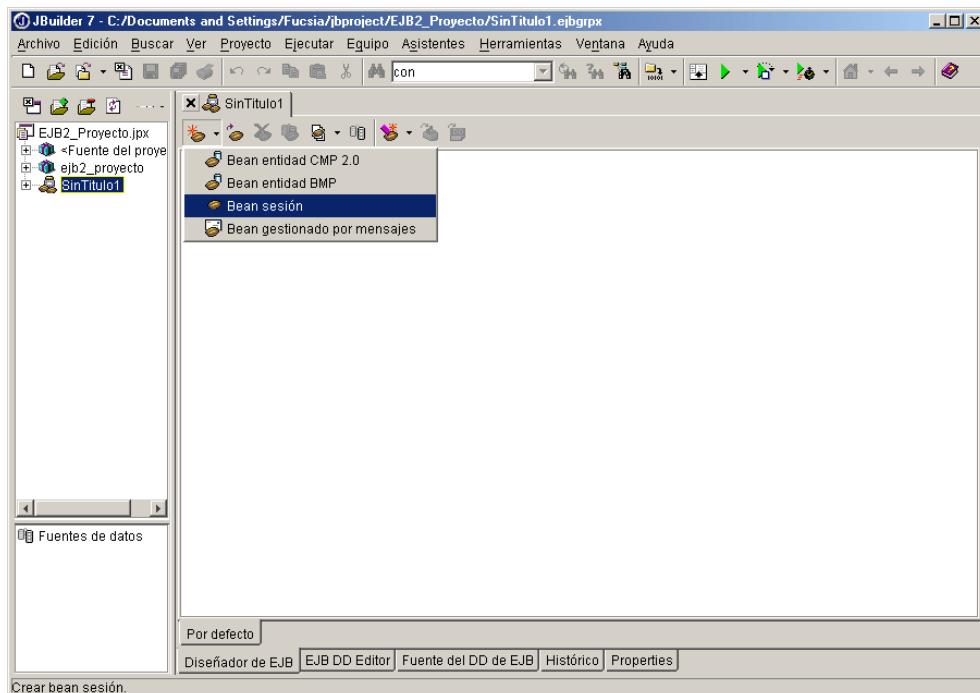
Si se encuentra en otra vista, como Fuente o en un panel del descriptor de distribución, puede pasar al diseñador de EJB mediante uno de los métodos siguientes:

- Haga doble clic en el nodo del módulo EJB, en el panel de proyecto, y elija la pestaña Diseñador de EJB en el panel de contenido.
- Haga clic en la pestaña Módulo EJB en la parte superior del panel de contenido. Esta opción sólo se encuentra disponible si el módulo está abierto en el proyecto actual.

# Creación de beans sesión

Para crear un bean sesión:

- 1 Haga clic con el botón derecho del ratón en el panel del diseñador de EJB y elija Crear EJB | Bean sesión, o pulse el icono Crear EJB de la barra de herramientas y elija Bean sesión:

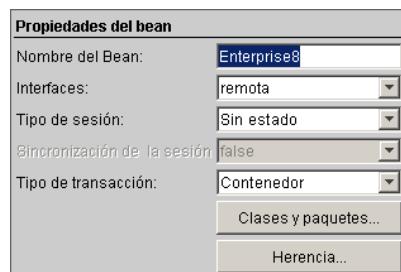


En el diseñador de EJB se muestra una representación del bean sesión llamada Enterprise<n> (<n> representa a un número) y en el panel de proyecto aparecen tres archivos.

Éstos son los archivos generados que aparecen en el panel de proyecto:

- Enterprise<n>: La interfaz remota del bean sesión.
- Enterprise<n>Bean: La clase de bean del bean sesión.
- Enterprise<n>Home: La interfaz base del bean sesión.

- 2** Pulse la representación del bean en la fila superior del diseñador de EJB. Aparece un inspector:



- 3** Dentro del inspector, cambie el nombre del bean por otro de su elección.

Los archivos del panel de proyecto cambian de nombre automáticamente.

- 4** En la lista desplegable Interfaces, seleccione Remota, Local o Remota/Local.

Si se elige Remota, el diseñador de EJB genera la clase del bean con una interfaz base remota y una interfaz remota. Éste es el valor por defecto.

Si se elige Local, el diseñador de EJB genera la clase del bean con una interfaz base local y una interfaz local. Dado que, por defecto, el diseñador de EJB genera interfaces remotas para los beans sesión, si se elige Local, el nombre y el contenido de los archivos del proyecto correspondientes al bean cambian para indicar que sólo se puede acceder a él localmente. Se añade LocalHome al nombre de la interfaz base remota, y Local al de la interfaz local.

Si se elige Remota/Local, en el panel de proyecto aparecen todos los archivos remotos y locales, con lo que se obtienen cinco archivos de bean en total.

- 5** Indique en Tipo de sesión el tipo del bean sesión: Sin estado o Con estado. Para obtener más información sobre los tipos de bean sesión, consulte “[Tipos de beans sesión](#)” en la página 15-1.

- 6** Indique el tipo de transacción: Contenedor (para la gestión por contenedor) o Bean (para la gestión por bean). Si desea información sobre los dos tipos de gestión de persistencia, consulte el apartado “[Transacciones gestionadas por contenedor y gestionadas por bean](#)” en la página 20-3.

- 7** Si desea que el bean implemente la interfaz SessionSynchronization, asigne el valor true al atributo SessionSynchronization. Esta opción está disponible sólo para beans sesión con estado. Para obtener información acerca de la interfaz SessionSynchronization, consulte “[La interfaz SessionSynchronization](#)” en la página 15-8.

- 8 Si desea realizar cambios en los nombres del paquete, el bean, la clase del bean y la interfaz, pulse Clases y paquetes. Pulse Aceptar.  
Los cambios realizados aparecen en el panel de proyecto.
- 9 Si desea cambiar el antecesor de la clase del bean por otro distinto de `java.lang.Object`, pulse Herencia.

## Visualización del código fuente de los beans

---

Mientras se trabaja con el diseñador de EJB, en cualquier momento se puede hacer doble clic en un archivo generado (`.java`), en el panel de proyecto, con el fin de mostrar su código fuente. Si lo prefiere, también puede pulsar sobre la representación del bean en el diseñador de EJB y elegir Ver código fuente del bean en el menú contextual, o simplemente, pulse el icono Ver código fuente del bean de la barra de herramientas. JBuilder muestra el código fuente de la clase del bean. Si hace clic con el botón derecho del ratón en un campo o un método de la representación del bean y elige Ver código fuente del bean, cuando aparezca el editor, el cursor se sitúa en la definición del campo o del método, dentro del código fuente de la clase del bean. Para volver al diseñador de EJB, haga doble clic en el nodo del módulo EJB, en el panel de proyecto.

## Modificación de beans

---

Hasta ahora, JBuilder sólo ha generado clases esqueleto para el bean que se está desarrollando. Siga trabajando con el diseñador de EJB para dar al bean la forma deseada. Es posible añadir campos y métodos, y modificar sus atributos. En cualquier momento se puede cambiar directamente al código fuente del bean con el propósito de realizar adiciones y modificaciones. Cuando se vuelve al diseñador de EJB aparecen en él todos los cambios.

### Modificación de los atributos del bean

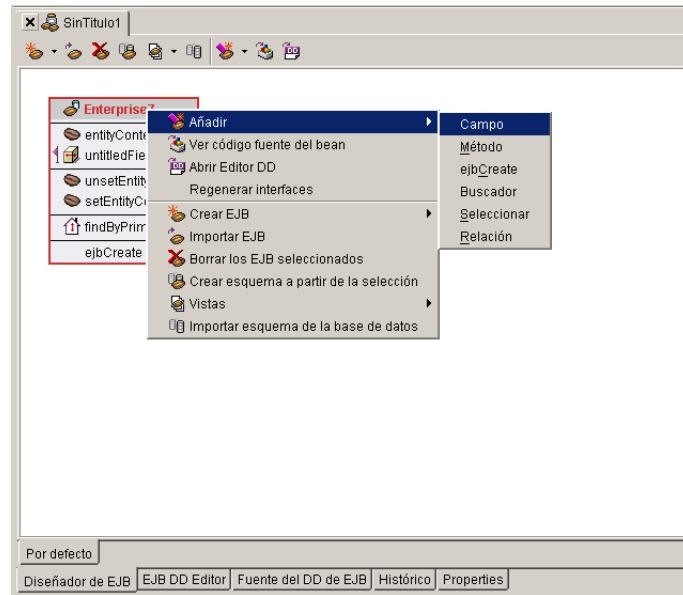
---

Para modificar los atributos de un bean, pulse el elemento deseado. Aparece un inspector en el que se pueden realizar los cambios.

## Añadir un nuevo campo

Para añadir un campo a un bean:

- Haga clic con el botón derecho del ratón en la representación del bean en el diseñador de EJB, y elija Añadir | Campo o pulse el icono Añadir de la barra de herramientas y elija Campo:



En el bean aparece un nuevo campo llamado `untitledField<n>`. Se abre el inspector del campo:

Nombre del campo:	<code>untitledField1</code>
Tipo:	<code>java.lang.String</code> ...
De obtención:	<code>ninguno</code>
De asignación:	<code>ninguno</code>

- Modifique en este inspector los atributos de `untitledField<n>`, incluido el nombre del campo:
  - Asigne al campo un nombre explicativo.
  - En el cuadro Tipo, indique el tipo Java del campo que está declarando: por ejemplo, `java.lang.String`. El botón de puntos suspensivos permite buscar un objeto Java.
  - Indique en las listas desplegables dónde se declaran los métodos de acceso de obtención y definición: en la interfaz local, en la remota, en ambas o en ninguna.

Para completar la creación del campo, haga doble clic en la clase del bean, en el panel de proyecto. Se muestra el código fuente. Busque los métodos de obtención y definición y añada la lógica deseada.

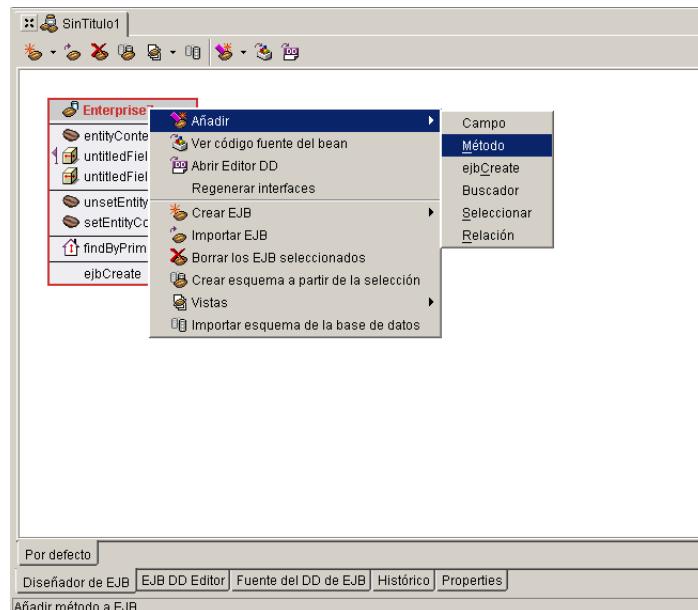
## Borrado de campos

Para borrar un campo de un bean, haga clic en él con el botón derecho del ratón, en la representación del bean, y elija Borrar campo.

## Añadir un nuevo método

Para añadir un método a un bean:

- 1 Haga clic con el botón derecho del ratón en la representación del bean, en el diseñador de EJB, y elija Añadir | Método o pulse el icono Añadir de la barra de herramientas y elija Método:



En el bean aparece un nuevo método llamado `untitledMethod<n>`. Se abre el inspector del método:

Nombre del método:	Método1
Tipo devuelto:	void
Parámetros de entrada:	
Interfaces:	ninguno

- 2 Modifique en este inspector los atributos de `untitledMethod<n>`:
  - Asigne al método un nombre explicativo.

- b** Indique el tipo devuelto del método. Para ello, escriba un tipo devuelto Java adecuado o pulse el botón de puntos suspensivos para buscar un objeto Java.
  - c** Introduzca en el campo Parámetros de entrada los argumentos que desee que se pasen. Los argumentos se declaran igual que en el código, por ejemplo, Integer age. Separe los distintos elementos por medio de comas.
  - d** Indique en la lista desplegable Interfaces dónde desea que se declare el método: en la interfaz base (base remota), en la interfaz base local, en las interfaces base y base local, en la interfaz local, en la interfaz remota, en las interfaces remota y local o en ninguna.
- 3** Para completar la creación del método, haga doble clic en la clase del bean, en el panel de proyecto. Se muestra el código fuente. Busque el método esqueleto que acaba de añadir al código fuente y escriba la lógica en el cuerpo.

## Eliminación de métodos

Para borrar un método de un bean, haga clic en él con el botón derecho del ratón, en la representación del bean, y elija Borrar método.

## Los métodos ejbCreate()

---

El diseñador de EJB permite especificar los parámetros necesarios para pasar a un método `ejbCreate()` y generar varios métodos `ejbCreate()` para un bean.

Para indicar los parámetros para pasar al método `ejbCreate()` por defecto:

- 1** Haga clic en `ejbCreate` en la representación del bean en el diseñador de EJB. Se abre un inspector.
- 2** Introduzca en el campo Parámetros de entrada los parámetros que desee que se pasen. Escriba el tipo de parámetro seguido de su nombre. Separe los parámetros con comas, como si estuviera definiéndolos en el código. Por ejemplo, puede utilizar `String group_id, String name` para pasar los parámetros `group_id` y `name` a `ejbCreate`.

**Nota** No es posible especificar parámetros por defecto para el método `ejbCreate()` de los beans sesión sin estado. Los beans sesión sin estado no necesitan más de un método `ejbCreate()` sin parámetros ya que no mantienen el estado.

Para añadir otro método `ejbCreate()` al bean:

- 1** Haga clic con el botón derecho del ratón en la representación del bean, en el diseñador de EJB.

- 2** Elija Añadir | ejbCreate para añadir un método ejbCreate a la representación del bean.
- 3** Acepte el nombre por defecto o indique uno distinto como valor de Nombre del método, en el inspector del método recién creado.
- 4** Escoja los parámetros que desea pasar al método como valor de Parámetros de entrada.

A continuación, entre en el código fuente de la clase bean y añada la lógica a los métodos ejbCreate() .

## Cómo regenerar interfaces de un bean

---

Cuando se trabaja directamente en el código fuente del bean, es posible que se realicen cambios en la clase del bean y se olvide de reproducir estos cambios en las interfaces base/base local y remota/local. Si lo desea, el diseñador de EJB puede volver a generar estas interfaces a partir del estado actual de la clase del bean.

Para volver a generar las interfaces del bean:

- 1** Haga clic con el botón derecho del ratón en la parte superior de la representación del bean, en el diseñador de EJB.
- 2** Elija Volver a generar interfaces en el menú contextual.

Si prefiere que el diseñador de EJB vuelva a generar automáticamente las interfaces del bean siempre que se realicen cambios en el código fuente de la clase, realice las siguientes acciones:

- 1** Seleccione Herramientas | Opciones del IDE.
- 2** Haga clic en la pestaña Diseñador de EJB.
- 3** Active la casilla de selección Volver a generar interfaces siempre.
- 4** Pulse Aceptar para cerrar el cuadro de diálogo.

## Importación de beans

---

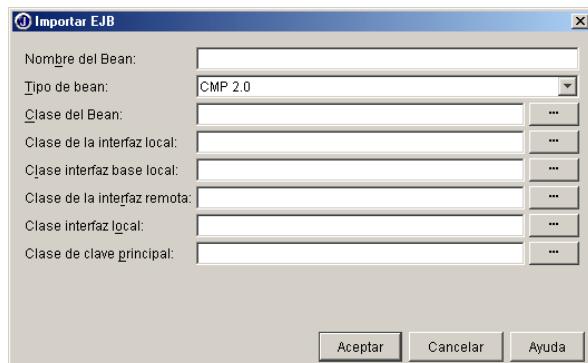
El diseñador de EJB permite importar enterprise beans a un módulo EJB. Encontrará esto útil cuando quiera importar un bean de un módulo EJB a otro o si ha obtenido un bean sin un descriptor de distribución adjunto. El código fuente del bean que está importando debe estar en la vía de acceso a código fuente del proyecto.

Para importar un enterprise bean:

- 1** Abra el módulo EJB al que desea importar el bean en el diseñador de EJB.

- 2** Haga clic con el botón derecho del ratón en el panel del diseñador de EJB y elija Importar EJB o pulse el ícono Importar EJB de la barra de herramientas:

Aparece el cuadro de diálogo Importar EJB.



- 3** Escriba el nombre deseado para el bean en el campo correspondiente.
- 4** Seleccione en la lista desplegable el tipo de bean que desea importar. Puede elegir entre CMP 2.0, BMP, sesión y gestionado por mensajes.
- 5** Indique el nombre de la clase del bean y su posición en el campo Clase de bean. Si lo prefiere, pulse el botón de puntos suspensivos para buscar el bean y la información del paquete se completará automáticamente.
- 6** Indique en los campos de clase de interfaz las interfaces que desea importar y su posición. Si lo prefiere, pulse el botón de puntos suspensivos para buscar las interfaces y la información del paquete se completará automáticamente.
- 7** Si desea importar una clase principal para un bean entidad, indique la clase y su posición en el campo Clase de clave principal.
- 8** Pulse Aceptar.

El bean se importa al módulo EJB. Su representación aparece en el diseñador de EJB junto con el inspector, donde se pueden modificar las propiedades.

## Organización de beans con vistas

---

Ahora, el diseñador de EJB permite agrupar en vistas conjuntos de enterprise beans, lo que ayuda a organizar y desarrollar proyectos de EJB complejos. Estas vistas funcionan de forma parecida a las hojas de una hoja de cálculo o a las fichas de un cuadro de diálogo con varias fichas. Aunque un módulo EJB puede contener muchos enterprise beans, en el diseñador de EJB se presentan únicamente los pertenecientes a la vista actual. Por ello, el uso de las vistas evita el desorden en la pantalla y le permite concentrarse en los beans que interesan en cada momento.

Cuando se crea un módulo EJB, la vista por defecto aparece en el diseñador de EJB. Si lo desea, puede cambiarle el nombre o crear otra.

Para cambiar el nombre a la vista por defecto:

- 1** Haga clic con el botón derecho del ratón en el panel del diseñador de EJB y elija Vistas | Cambiar el nombre a la vista, o pulse el icono Vistas de la barra de herramientas y elija Cambiar el nombre a la vista:
- 2** En el cuadro de diálogo que aparece, escriba el nombre que desea utilizar para la vista y pulse el botón Aceptar o la tecla *Intro*.

El nuevo nombre de la vista aparece en la pestaña correspondiente, en la parte inferior del panel del diseñador de EJB.

Para crear una vista :

- 1** Haga clic con el botón derecho del ratón en el panel del diseñador de EJB y elija Vistas | Nueva vista, o pulse el icono Vistas de la barra de herramientas y elija Nueva vista.
- 2** En el cuadro de diálogo que aparece, escriba el nombre de la nueva vista y pulse el botón Aceptar o la tecla *Intro*.

En la parte inferior del panel del diseñador de EJB aparece una pestaña con el nombre indicado. Pulse esta pestaña para convertir la vista recién creada en la vista actual.

Para desplazar EJB de una vista a otra:

- 1** Seleccione uno o varios EJB en una vista.
- 2** Haga clic con el botón derecho en uno de los EJB seleccionados.
- 3** Elija Vistas | Mover selección o pulse el icono Vistas de la barra de herramientas y elija Mover selección.
- 4** Seleccione en la lista desplegable la vista a la que desea desplazar los beans.

Para copiar un EJB de una vista a otra:

- 1 Seleccione uno o varios EJB en una vista.
- 2 Haga clic con el botón derecho en uno de los EJB seleccionados.
- 3 Elija Vistas | Copiar selección o pulse el icono Vistas de la barra de herramientas y elija Copiar selección.
- 4 Seleccione en la lista desplegable la vista a la que desea desplazar los beans.

Para eliminar EJB de una vista:

- 1 Seleccione uno o varios EJB en una vista.
- 2 Haga clic con el botón derecho en uno de los EJB seleccionados.
- 3 Elija Vistas | Eliminar selección

Cuando se utiliza el comando Vistas | Eliminar selección, las representaciones de los EJB se eliminan de la vista actual, pero eso no significa que se han eliminado permanentemente del módulo EJB ni del proyecto. Si los EJB eliminados no se encuentran en ninguna otra vista, vuelven a aparecer en la vista por defecto después de la eliminación.

Si desea eliminar a la vez los EJB y el código fuente correspondiente, consulte “[Eliminación de beans](#)” en la página 6-20.

Para eliminar una vista:

- 1 Convierta la vista que desea eliminar en la vista actual.
- 2 Haga clic con el botón derecho en el panel del diseñador de EJB.
- 3 Elija Vistas | Eliminar vista o pulse el icono Vistas de la barra de herramientas y elija Eliminar vista.

La vista desaparece. Si la vista eliminada contiene beans, éstos vuelven a aparecer en la vista por defecto.

## Búsqueda de beans

---

Si un módulo EJB contiene varias vistas es posible olvidar dónde se encuentra un bean. Para localizar un bean rápidamente:

- 1 Seleccione Buscar | Buscar EJB o pulse *Ctrl+F*.  
Aparece el cuadro de diálogo Buscar EJB. Se enumeran todos los EJB del módulo, junto con la vista en la que se encuentran entre paréntesis.
- 2 Seleccione el EJB que busca en la lista desplegable Nombres de EJB, o escriba su nombre en el cuadro Nombre de EJB, en la parte superior del cuadro de diálogo.

**3** Pulse Aceptar.

La vista que contiene el EJB buscado se convierte en la actual, aparece el EJB y se abre su inspector.

## Cómo ordenar los beans

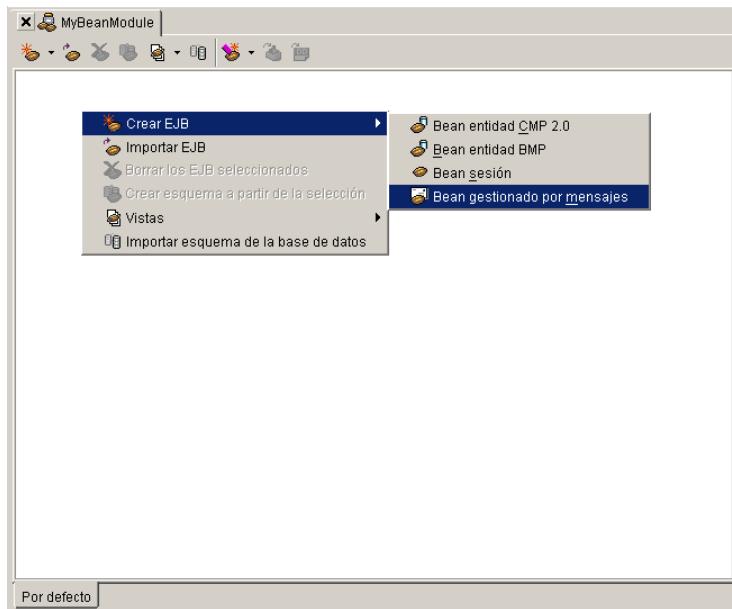
Si lo desea, el diseñador de EJB puede reorganizar los enterprise beans en una vista.

- 1** Haga clic con el botón derecho en el panel del diseñador de EJB.
- 2** Elija Vistas | Organizar los EJB o pulse el icono Vistas de la barra de herramientas y elija Organizar los EJB.

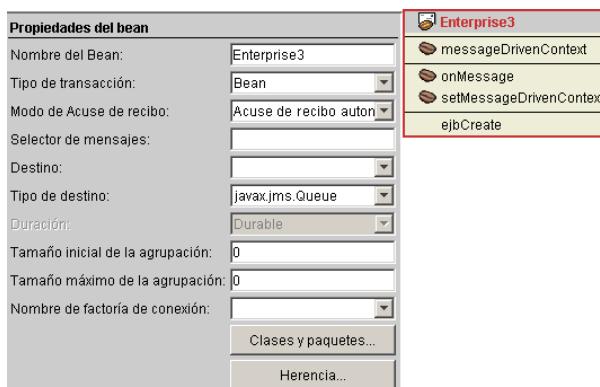
El diseñador de EJB intenta reorganizar los beans de forma lógica.

## Creación de beans gestionados por mensajes

Para crear un bean gestionado por mensajes, haga clic con el botón derecho del ratón en el diseñador de EJB y elija Crear EJB | Nuevo bean gestionado por mensajes, o pulse el icono Crear EJB de la barra de herramientas y elija Bean gestionado por mensajes:



En el diseñador de EJB aparece una representación del bean gestionado por mensajes, junto con su inspector.



La mayoría de las propiedades que se muestran en el inspector son atributos del descriptor de distribución. Si desea información sobre algunas de estas propiedades, consulte el [Capítulo 17, “Desarrollo de beans gestionados por mensajes”](#).

Igual que ocurre con los beans sesión, para añadir campos y métodos se hace clic con el botón derecho del ratón en la representación del bean y se elige un elemento del menú contextual. Cuando pulse un campo o un método para modificar sus atributos, utilice el inspector que se abre. Consulte [“Modificación de beans” en la página 6-11](#) para obtener más información.

Tenga en cuenta que cuando se añaden métodos no se pueden especificar las interfaces, ya que los beans gestionados por mensajes carecen de ellas.

Para completar el bean, diríjase al código fuente de su clase. Busque el método `onMessage()` en la clase y añada al cuerpo la lógica de reacción a un mensaje entrante.

## Eliminación de beans

Para eliminar beans del diseñador de EJB y borrar los archivos fuente correspondientes en el panel de proyecto:

- 1** Seleccione la representación del bean en el diseñador de EJB:
  - Para seleccionar un solo bean, púlselo.
  - Para seleccionar varios beans, vaya haciendo clic en ellos con la tecla *Ctrl* pulsada, o arrastre el puntero alrededor de la selección.
- 2** Haga clic con el botón derecho del ratón en uno de los beans seleccionados y elija Borrar los EJB seleccionados, pulse la tecla *Supr* o

pulse el icono Borrar los EJB seleccionados de la barra de herramientas del diseñador de EJB.

## Asignación de nombres a los archivos en el diseñador de EJB

---

El nombre de los archivos que genera JBuilder cuando se crean enterprise beans con el diseñador de EJB varía según el tipo de enterprise bean. En JBuilder, los beans sesión son remotos por defecto y los beans entidad son locales por defecto. Por ello, el diseñador de EJB genera interfaces base remotas para los beans sesión e interfaces base locales para los beans entidad. En todos los casos el nombre de la interfaz base, tanto si es remota como si es local, es <Nombre del bean>Home. De igual forma, la interfaz que declara los métodos empresariales es la remota para los beans sesión y la local para los beans entidad. Por ejemplo, si decide generar las interfaces local y remota de un *bean sesión* llamado Component, el diseñador de EJB genera estos archivos:

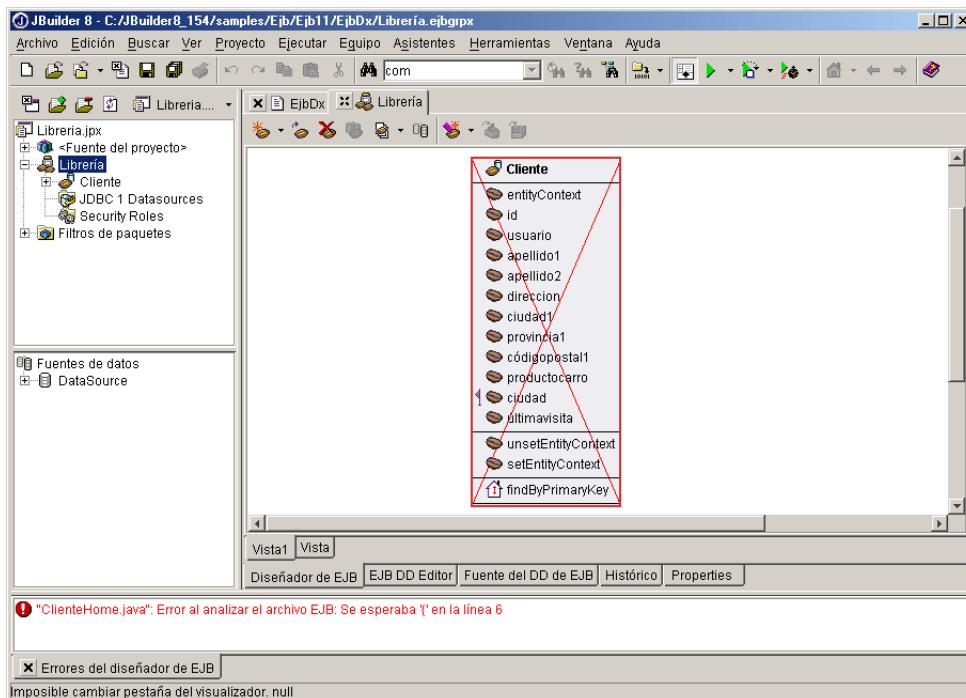
- ComponentHome: La interfaz base remota.
- ComponentBean: La clase del bean.
- Component: La interfaz remota.
- ComponentLocalHome: La interfaz base local.
- ComponentLocal: La interfaz local.

Si se hace lo mismo con un *bean entidad*, llamado también Component, el diseñador de EJB genera estos archivos:

- ComponentHome: La interfaz base local.
- ComponentBean: La clase del bean.
- Component: La interfaz local.
- ComponentRemoteHome: La interfaz base remota.
- ComponentRemote: La interfaz remota.

## Solución de errores en los beans

Cuando se trabaja entre el diseñador de EJB y el código fuente para crear beans es posible que se cometan errores de sintaxis. Si esto ocurre, cuando se abre el diseñador de EJB aparece un mensaje de error en el panel de mensajes, y la representación del bean se muestra tachada.

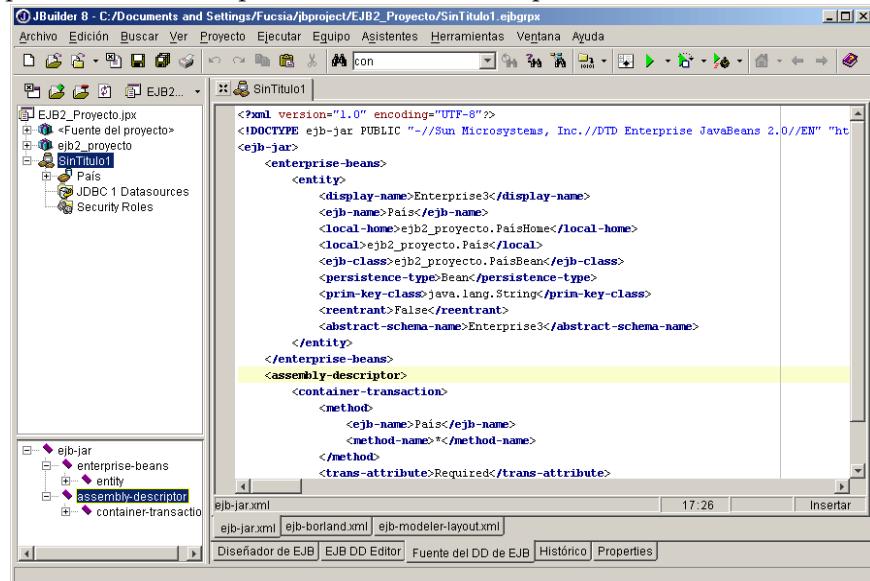


Para ir rápidamente a la línea de código fuente que tiene el error, haga doble clic en el mensaje de error. Repare el error en el código y vuelva al diseñador de EJB. El bean se representa normalmente.

Para eliminar la pestaña Errores del Diseñador de EJB del panel de mensajes, púlsela con el botón derecho del ratón y elija Eliminar pestaña de Errores del Diseñador de EJB.

## Visualización de los descriptores de distribución

A medida que se trabaja con el diseñador de EJB para generar enterprise beans y modificar sus atributos, se crean los descriptores de distribución de los beans. Si desea presentar el código fuente XML de los descriptores de distribución, pulse la pestaña Fuente del DD de EJB, en la parte inferior del panel de contenido. Aquí se presenta otra serie de pestañas. Las pestañas visibles dependen del servidor de aplicaciones de destino:



## Presentación del editor de descriptor de distribución

JBuilder proporciona un editor del descriptor de distribución que permite ver la configuración actual y modificarla. La información de los descriptores de distribución de los beans se muestra en diversos paneles. Existen varias formas de mostrar el Editor de descriptor de distribución:

- Haga doble clic en el módulo EJB, en el panel de proyecto, y pulse la pestaña Editor DD de EJB, en la parte inferior del panel de contenido. Aparece un panel que muestra varios de los atributos del archivo JAR.
- Para visualizar y modificar la información del descriptor de distribución de un solo bean, amplíe el nodo del módulo EJB en el panel de proyecto y haga doble clic en el bean, en la lista de beans del módulo EJB. Se abre el panel General del editor de descriptores de distribución. Para ver otros paneles, pulse cualquiera de las pestañas de la parte inferior del Editor de descriptor de distribución.

- Para visualizar y modificar el descriptor de distribución de un solo bean mientras se trabaja en él en el diseñador de EJB, pulse con el botón derecho del ratón la parte superior de la representación del bean, en el diseñador, y elija Abrir Editor DD en el menú contextual o bien pulse el icono de esta opción en la barra de herramientas.

Para volver al diseñador de EJB, haga doble clic en el nodo del módulo EJB, en el panel de proyecto, o pulse la pestaña del módulo EJB, en la parte superior del panel de contenido.

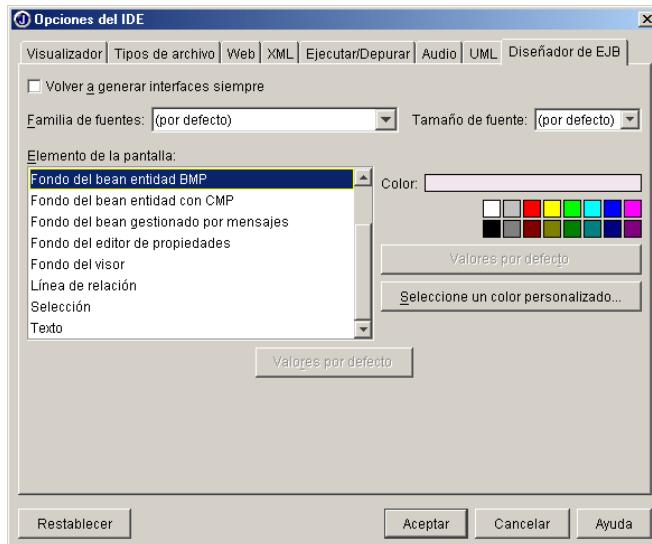
Si desea información más detallada sobre el editor del descriptor de distribución, consulte el [Capítulo 13, “El editor de descriptor de distribución”](#).

## Configuración de las opciones del IDE para el diseñador de EJB

Es posible modificar la fuente y cambiar el color de los elementos del diseñador de EJB:

- 1 Seleccione Herramientas | Opciones del IDE.
- 2 Haga clic en la pestaña Diseñador de EJB.

Esta ficha aparece en el cuadro de diálogo Opciones del IDE



- 3 Modifique a su gusto los elementos de la pantalla y pulse Aceptar cuando termine.

## Siguiente paso

---

Ahora que ha diseñado los beans sesión o gestionados por mensajes puede proceder a compilarlos. Consulte el [Capítulo 10, “Compilación de enterprise beans y creación de archivos JAR”](#).

Si busca información sobre la creación de beans entidad en el diseñador de EJB, consulte el [Capítulo 7, “Creación de beans entidad 2.0 con el diseñador de EJB”](#).



# Creación de beans entidad 2.0 con el diseñador de EJB

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

En este capítulo se explica la forma de utilizar JBuilder para crear beans entidad compatibles con Enterprise JavaBeans™ 2.0 de Sun Microsystems. Si desea información sobre la forma de crear beans entidad compatibles con Enterprise JavaBeans™ 1.1 specification, consulte el [Capítulo 8, “Creación de componentes EJB 1.x con JBuilder”](#) y el [Capítulo 9, “Creación de beans entidad EJB 1.x a partir de una tabla de base de datos”](#).

Antes de leer este capítulo es necesario haberse familiarizado con los asuntos tratados en el [Capítulo 6, “Creación de beans sesión y gestionados por mensajes 2.0 con el diseñador de EJB”](#). En ese capítulo se explica el manejo de las funciones del diseñador de EJB, que también se emplean en la generación de beans entidad. Sin embargo, el diseñador de EJB cuenta con funciones adicionales que ayudan a crear beans entidad. Su uso se describe en el presente capítulo.

Es posible dar comienzo a la creación de beans entidad de dos formas distintas:

- 1 Importar una fuente de datos al diseñador de EJB y utilizarla para crear beans entidad CMP 2.0.
- 2 Hacer clic en el diseñador de EJB con el botón derecho del ratón y elegir Bean entidad CMP 2.0 o Bean entidad BMP. Si lo prefiere, pulse el icono Crear EJB de la barra de herramientas del diseñador de EJB y elija Bean entidad CMP 2.0 o Bean entidad BMP. Añada y modifique los campos y métodos de la misma forma que con los beans sesión y gestionados por mensajes.

# Creación de beans entidad CMP 2.0 a partir de una fuente de datos importada

El diseñador de EJB permite importar una fuente de datos JDBC y utilizarla para crear beans entidad.

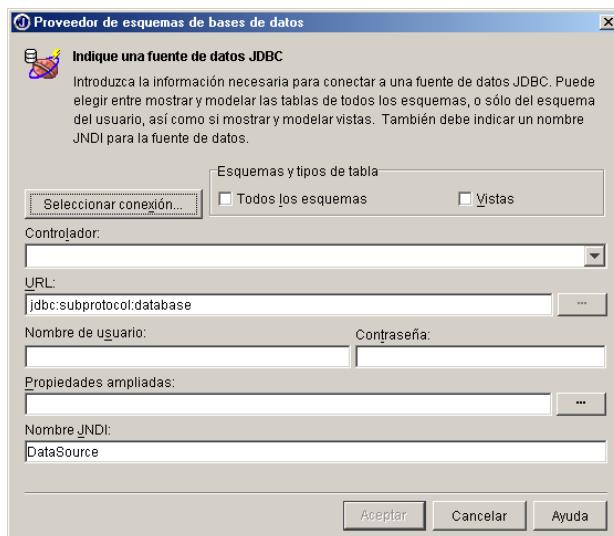
## Importación de fuentes de datos

Cuando se importa una fuente de datos, en realidad se importa su esquema o estructura. A continuación, dicho esquema se puede modificar según las necesidades.

Para importar una fuente de datos:

- 1 Haga clic con el botón derecho del ratón en el nodo DataSources, en el panel de estructura, y elija Importar esquema de la base de datos, o haga clic con el botón derecho del ratón en el panel del diseñador de EJB y elija esta opción. Si lo prefiere, pulse el icono Importar esquema de la base de datos de la barra de herramientas del diseñador de EJB.

Se abre el cuadro de diálogo Proveedor de esquemas de bases de datos:



- 2 Indique un origen de datos JDBC.

Introduzca la información necesaria para conectarse a una fuente de datos JDBC.

Si la conexión ya está creada, pulse Seleccionar conexión. El resto de la información necesaria para esta ficha se rellena automáticamente, con

excepción de la contraseña, que se debe introducir de forma manual si la conexión la requiere.

Si no tiene una conexión o desea crear una, seleccione un controlador de la lista desplegable correspondiente y elija una dirección URL.

Aparecen los controladores configurados por medio de Herramientas | Configurar Enterprise, en la ficha Controladores de bases de datos.

Consulte ["Configuración de los controladores JDBC"](#) en la página 5-11 para obtener más información.

Indique el nombre de usuario requerido para acceder a la fuente de datos y escriba la contraseña si es necesaria. Seleccione todas las propiedades ampliadas que necesite. Por último, elija un nombre JNDI para la fuente de datos o acepte el que proporciona por defecto JBuilder.

### 3 Especifique las opciones Esquemas y tipos de tabla que deseé.

Si se ha activado la opción Todos los esquemas, se utilizan todos aquellos en los que el usuario tiene derecho de conexión. Si deja la opción Todos los esquemas desactivada, sólo se usan los esquemas con el mismo nombre que el nombre de usuario, reduciendo el tiempo requerido para hacer la conexión y cargar los datos.

Active la opción Vistas si desea que las vistas se carguen en el diseñador de EJB. Si no desea descargar las vistas, no marque esta opción.

El diseñador de EJB intenta establecer una conexión con la fuente de datos especificada. Si la conexión se realiza correctamente, la fuente de datos especificada aparece en el panel de estructura:



Si el diseñador de EJB detecta otra fuente de datos en el proyecto con la misma URL, preguntará si desea usar el mismo nombre.

## jndi-definitions.xml

---

Todas las definiciones de fuente de datos de un proyecto se almacenan en jndi-definitions.xml. Para eliminar una fuente de datos, modifique el archivo directamente:

- 1 Haga doble clic sobre el nodo jndi-definitions.xml que aparece en el panel de proyecto para visualizar el código fuente.
- 2 Seleccione la definición JNDI que desea eliminar y bórrela.

Una vez que se haya eliminado la fuente de datos del archivo, ésta no se distribuirá si el servidor de aplicaciones de destino es Borland Enterprise Server AppServer Edition 5.0.2 -5.1.x.

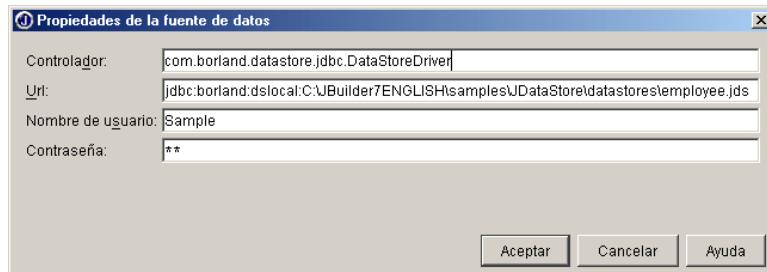
## Modificación del esquema de la fuente de datos importada

---

Cuando se importa una fuente de datos es posible realizar cambios en ella antes de utilizarla en la creación de los beans entidad. Haga clic con el botón derecho del ratón en diversos elementos, en el panel de estructura, y examine las opciones de modificación disponibles. Éstas son las opciones de los distintos elementos:

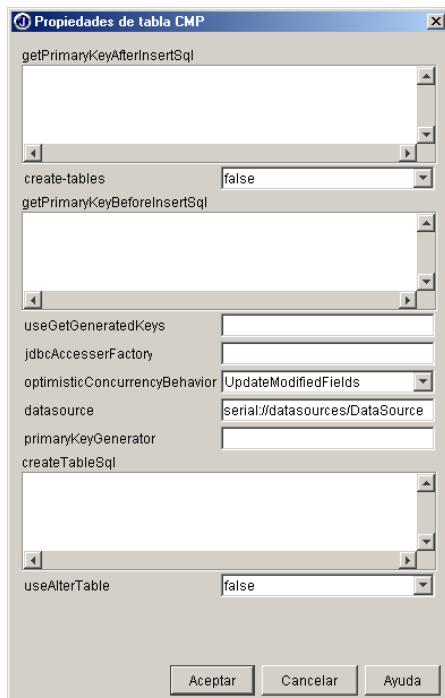
- En el nodo superior de la fuente de datos, el menú contextual permite añadir tablas y modificar las propiedades de la fuente de datos, así como cambiarle el nombre y borrarla.

Cuando se elige Modificar propiedades de fuente de datos, se abre el cuadro de diálogo Propiedades de la fuente de datos, en el que se puede cambiar la configuración de las propiedades de acceso a la fuente de datos:



- En los nodos de las tablas, el menú contextual permite añadir columnas y modificar las propiedades de las tablas, así como cambiar de nombre y borrar los nodos de tabla de la fuente de datos.

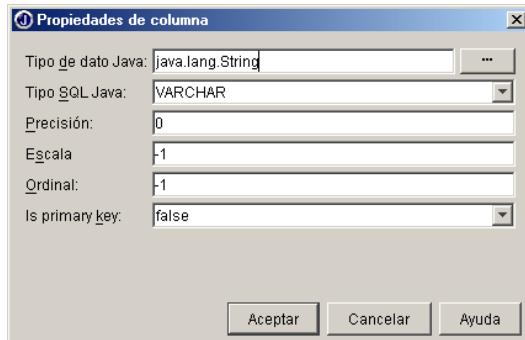
Cuando se elige Modificar propiedades CMP de tabla, se abre el cuadro de diálogo Propiedades de tabla CMP, en el que se puede especificar la configuración de las propiedades de persistencia gestionada por contenedor del bean entidad. Estas propiedades son exclusivas de los servidores de aplicaciones Borland:



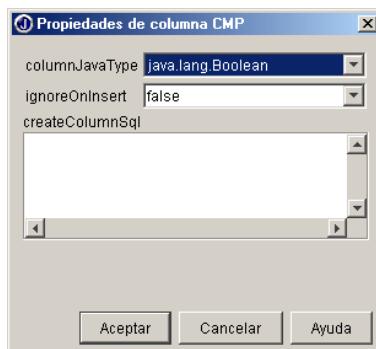
Si desea obtener más información sobre estas propiedades, pulse el botón Ayuda del cuadro de diálogo.

- En los nodos de las columnas, el menú contextual permite modificar las propiedades de la columna y cambiar de nombre y borrar los nodos de columna de la tabla.

Elija Modificar propiedades de columna con el fin de cambiar el tipo de dato de la columna e indicar si se trata de una clave principal. Estas propiedades son exclusivas de los servidores de aplicaciones Borland:



Cuando se elige Modificar propiedades CMP de columna, se abre el cuadro de diálogo Propiedades de columna CMP, en el que se puede especificar la configuración de las propiedades de persistencia gestionada por contenedor de la columna.



Si desea obtener más información sobre estas propiedades, pulse el botón Ayuda del cuadro de diálogo.

## Generación de las clases e interfaces de los beans entidad

Cuando la fuente de datos tiene la configuración deseada, se puede comenzar la generación de las clases e interfaces del bean entidad. En el panel de estructura, pulse con el botón derecho del ratón una tabla que desee utilizar para crear un bean entidad y elija Crear bean entidad con CMP 2.0, si desea crear un bean entidad 2.0 gestionado por contenedor, o Crear bean entidad con BMP, si desea crear un bean entidad gestionado por bean. En el diseñador de EJB se muestra una representación del bean entidad, y en el panel de proyectos aparecen sus clases e interfaces. Éstos son los archivos que se generan:

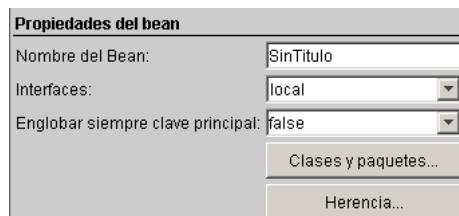
- <Entity>: La interfaz local del bean entidad.
- <Entity>Bean: La clase del bean entidad.
- <Entity>Home: La interfaz base local del bean entidad.

Observe que la convención de nomenclatura de archivos de los beans entidad en el diseñador de EJB no coincide con la de los beans sesión. Las aplicaciones clientes pueden acceder a beans sesión remotos, pero por lo general, sólo los beans sesión pueden acceder a los beans entidad. Por ello, es probable que los beans entidad a los que acceden los beans sesión sean locales. De esta forma, la interfaz que declara los métodos empresariales suele ser la local para los beans entidad y la remota para los beans sesión. La interfaz de los beans entidad que sólo lleva el nombre de la entidad es la interfaz local. Para los beans sesión se trata de la interfaz remota. De igual forma, la interfaz que lleva el sufijo "Home" es la base local para los beans entidad y la base remota para los beans sesión. Para obtener más información, consulte "[Asignación de nombres a los archivos en el diseñador de EJB](#)" en la página 6-21.

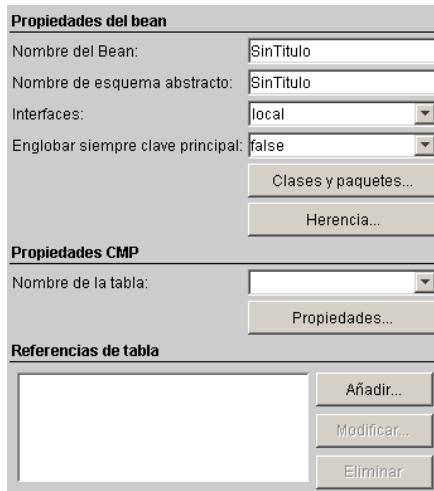
## Modificación de las propiedades de los beans entidad

---

Los inspectores del diseñador de EJB se pueden utilizar para modificar las propiedades de los beans entidad, igual que ocurre con los beans sesión y gestionados por mensajes. Si el inspector no es visible, pulse la representación del bean en la fila superior del diseñador de EJB para que aparezca. Éste es el inspector de un bean entidad BMP:



Éste es el inspector de un bean entidad CMP 2.0:



Los inspectores de los beans BMP y CMP 2.0 permiten cambiar los nombres del bean, así como indicar si el bean tiene interfaces remotas (base y remota), locales (base local y local) o remotas y locales (base, remota, base local y local). El inspector para beans CMP 2.0 también permite modificar el nombre del esquema abstracto, se se utiliza en EJB-QL para métodos de búsqueda y selección.

Por defecto, JBuilder sólo crea una clase de la clave principal integrada cuando la clave principal está compuesta por más de un campo de clave principal o si el único campo es un primitivo de Java. Si desea que JBuilder cree una clave principal englobada en todos los casos, asigne el valor true a la opción Englobar siempre clave principal. Por ejemplo, si dispone de un bean entidad con un campo de clave principal única de tipo `java.lang.Integer`, puede englobarla en una clase de clave principal personalizada que contenga una referencia a este campo.

Pulse Clases y paquetes con el fin de realizar los cambios deseados en los nombres de las clases y paquetes, y pulse Herencia si desea establecer para el bean un antecesor distinto de `java.lang.Object`.

Si desea cambiar el antecesor de la clase del bean a otro distinto de `java.lang.Object`, utilice el botón Herencia.

Pulse el botón Propiedades para abrir el cuadro de diálogo Propiedades CMP de un bean entidad CMP 2.0.



Este cuadro de diálogo del inspector le permite establecer las propiedades de persistencia gestionada por contenedor del bean. Estas propiedades son exclusivas de los servidores de aplicaciones Borland: Determinan a partir de qué esquema de tabla se crea el bean y cómo el contenedor gestiona su persistencia. En la tabla siguiente se explican las propiedades de persistencia gestionada por contenedor de Borland:

- **ejb.invalidateFinderCollectionAtCommit**

Determina si se optimiza la confirmación de transacciones por invalidación de colecciones de buscadores. El valor por defecto es false.

- **ejb.maxBeansInCache**

Determina el número máximo de beans de la memoria caché de la opción A (consulte ejb.transactionCommitMode, a continuación). Si se sobrepasa este límite, las entidades se desplazan a la agrupación de disponibles mediante una llamada a ejbPassivate(). El valor por defecto es 1.000.

- **ejb.maxBeansInPool**

Determina el número máximo de beans de la agrupación de disponibles. Si la agrupación de disponibles excede de este límite, las entidades son eliminadas del contenedor. El valor por defecto es 1.000.

- **ejb.transactionCommitMode**

Indica la disposición de un bean entidad con respecto a una transacción. Los valores son:

A o Exclusive (Exclusivo): acceso exclusivo a la tabla de la base de datos. Se puede presuponer que el estado del bean al finalizar la última transacción enviada es el que tendrá al principio de la próxima transacción. Los beans se almacenan en caché entre transacción y transacción.

B o Shared (Compartido): acceso compartido a la tabla de la base de datos. Sin embargo, por motivos de rendimiento, cada bean permanece

asociado a una clave principal en los intervalos entre transacciones, para evitar las llamadas innecesarias a ejbActivate() y ejbPassivate(). El bean permanece en el búfer de activos. Éste es el valor por defecto.

C o None (Ninguno): acceso compartido a la tabla de la base de datos. Los beans no permanecen asociados a claves principales determinadas entre las transacciones, sino que cada vez que termina una vuelven a la agrupación de disponibles. Normalmente, esta opción no resulta muy útil.

- **ejb.cacheCreate**

Intenta guardar en caché los bean entidad creados hasta que la transacción se confirme. El valor por defecto es true. Asigne false a esta propiedad en las aplicaciones que deban saber inmediatamente si un create tendrá éxito.

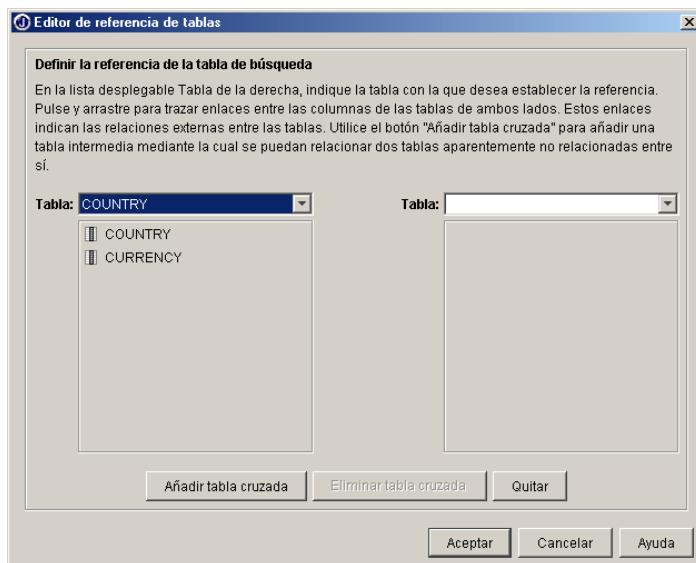
## Referencias a otra tabla

La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

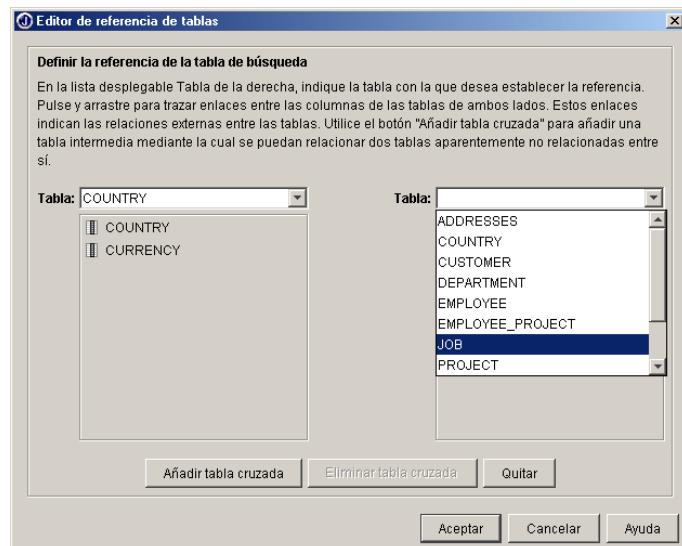
El editor de referencias de tablas le permite hacer referencia a otra tabla y asignar sus columnas a campos del bean entidad. Las referencias a tablas solamente están disponibles para los beans entidad que tienen a Borland Enterprise Server como servidor de destino.

Para crear una referencia de tablas:

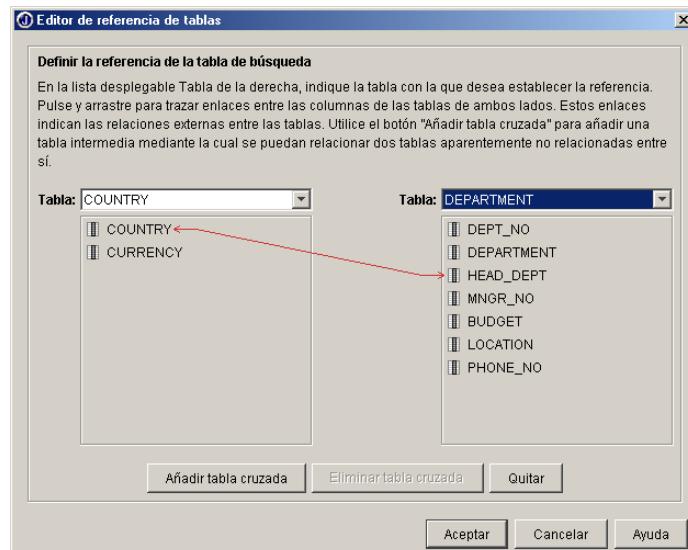
- 1 Pulse el botón Añadir contiguo al cuadro Referencia de tablas en el inspector del bean entidad. Aparece el editor de referencias de tablas:



- 2** Indique la tabla a la que desea hacer referencia en la lista desplegable de la derecha:



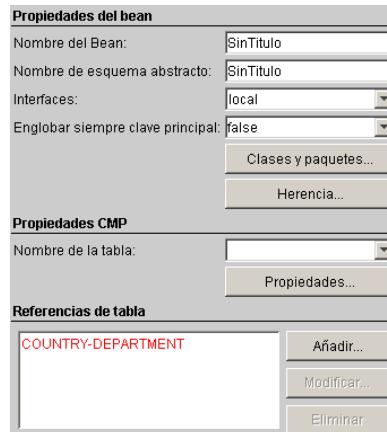
Mantenga pulsado el botón del ratón en una columna de la tabla izquierda que deseé enlazar a una columna de la tabla derecha. Arrastre una línea hasta la columna de la tabla derecha con la que deseé crear el enlace:



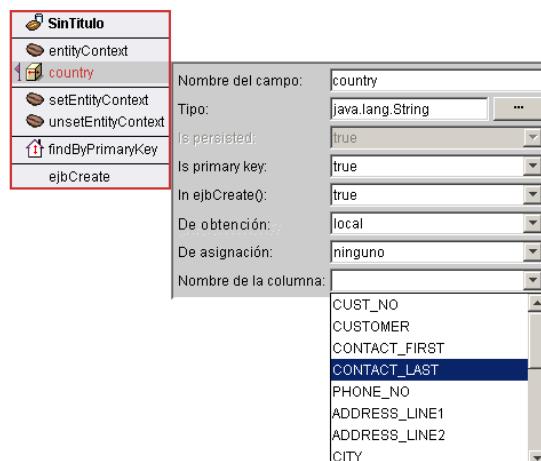
Si desea añadir una tercera tabla para conectar las otras dos, como puede ocurrir en el caso de que sean muchas, pulse Añadir tabla cruzada y elija la tabla cuyos campos se pueden utilizar de enlace. A continuación arrastre el puntero entre las columnas de las tres tablas para completar la referencia.

Si es conveniente, se pueden enlazar varias columnas. Pulse Aceptar cuando haya terminado.

La referencia que se acaba de crear aparece en el cuadro Referencias de tabla:

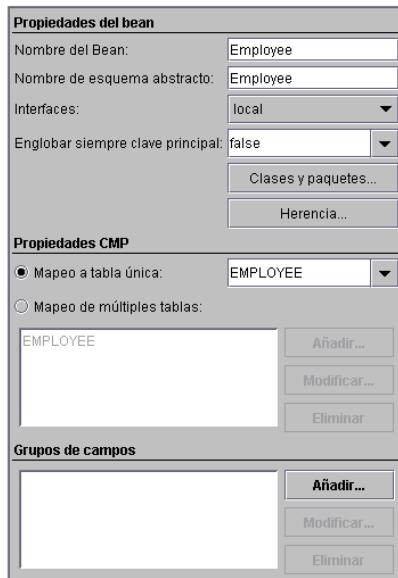


- 3 En la representación del bean entidad, pulse uno de los campos que desea asignar a una columna de la tabla a la que se hace referencia para abrir el inspector.
- 4 Seleccione en la lista desplegable Nombre de la columna a la que desea asignar el campo:



## Cómo añadir referencias de tabla de WebLogic

Si el servidor de aplicaciones de destino es WebLogic Server 7.x o posterior, el inspector del bean entidad tiene el siguiente aspecto:



Las opciones de asignación de tablas sólo se encuentran disponibles para WebLogic Server 7.x. Si utiliza una versión de WebLogic Server 6.x, las opciones de asignación de tablas no aparecerán en el inspector del bean.

El campo Verificar columnas se utiliza para especificar que es necesario comprobar la validez de las columnas justo antes de confirmar una transacción, con el fin de asegurarse de que ninguna otra transacción haya cambiado los datos de la columna. Seleccione Leídas si desea verificar todas las columnas que hayan sido leídas durante la transacción.

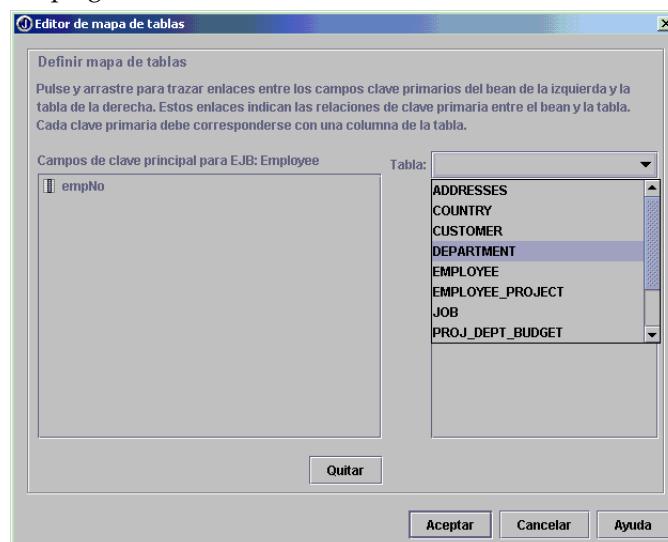
Seleccione Modificadas si sólo desea verificar las columnas que hayan sido actualizadas durante la transacción. Seleccione Versión, o bien Marca de fecha y hora, para especificar que existe una columna en la tabla con datos de versión o de fecha y hora, que se va a utilizar para implementar un esquema de concurrencia optimista.

El campo Columna optimista aparece activado si se seleccionó Versión o Marca de fecha y hora como valor de Verificar columnas. Seleccione la columna en la lista desplegable que contiene el valor de la versión o la marca de fecha y hora. Si desea obtener más información sobre la concurrencia optimista, consulte la documentación de WebLogic.

Puede utilizar el inspector para hacer referencia a otra tabla y para asignar columnas de la tabla a campos del bean entidad a través de la opción Asignación múltiple de tablas . La opción Asignación simple de tablas es la opción por defecto.

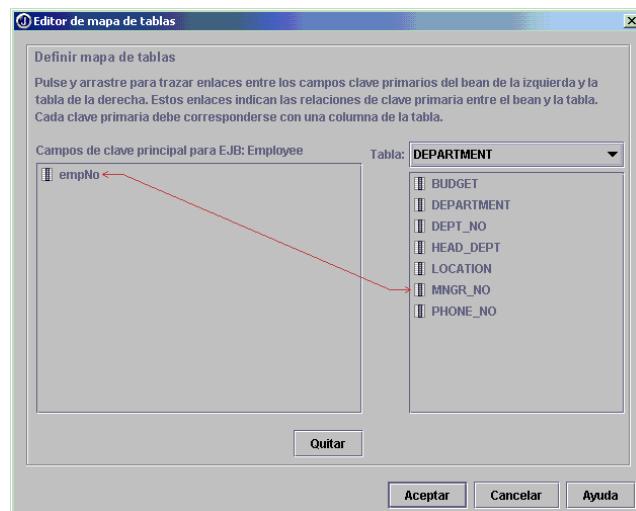
Para hacer referencia a otra tabla y asignar columnas a campos del bean:

- 1 Seleccione la opción Mapeo de múltiples tablas.
- 2 Pulse el botón Añadir para abrir el editor de asignación de tablas.
- 3 Seleccione la tabla a la que desea asignar columnas en la lista desplegable Tabla:



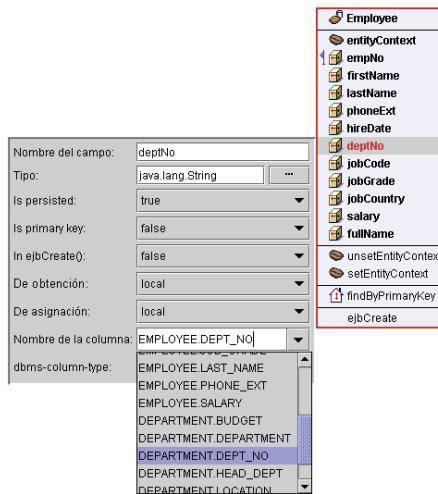
- 4 Pulse y arrastre las claves principales de la tabla, a la izquierda, hacia la columna a la que deseé asignar la columna en la tabla de la derecha.

Aquí se puede ver cómo se ha asignado el campo `empNo` del bean Employee a la columna `MNGR_NO` de la tabla DEPARTMENT.



- 5** Ahora puede volver al bean entidad, añadir nuevos campos y asignarlos a las columnas de la tabla con la que se ha establecido la referencia.

Por ejemplo, aquí se ha añadido un campo denominado department\_no al bean Employee. El inspector muestra cómo se ha asignado el nuevo campo a la columna DEPT\_NO de la tabla DEPARTMENT.



El cuadro Grupos de campos aparece en el inspector de beans entidad de WebLogic. Un elemento grupo-de-campo representa un subconjunto de los campos de persistencia gestionada por contenedor y de los campos de relación gestionada por contenedor de un bean entidad. Si desea más información sobre los grupos de campos y su uso, consulte la documentación de WebLogic.

Para añadir un grupo de campos:

- Pulse el botón Añadir.

Aparece el editor de grupos de campos:



- 2** Escriba un nombre en el campo Nombre del grupo de campos.
- 3** Marque los campos que desea incluir en el grupo.
- 4** Pulse Aceptar.

## Inspectores de campos y métodos de los beans entidad

El inspector de campos de los beans entidad tiene el siguiente aspecto:

Nombre del campo:	mngrNo
Tipo:	java.lang.Short
Is persisted:	true
Is primary key:	false
In ejbCreate():	false
De obtención:	local
De asignación:	local
Nombre de la columna:	MNGR_NO
dbms-column-type:	ninguno

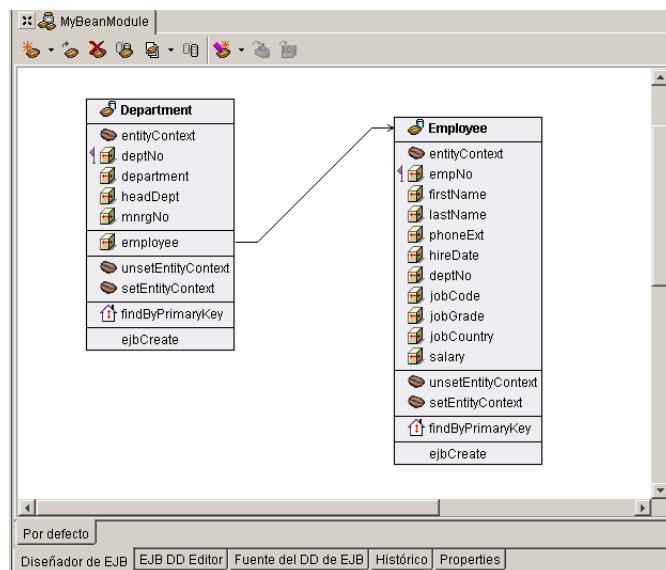
El inspector se puede utilizar para cambiar el nombre del campo a voluntad. También es posible indicar el tipo de campo, determinar si su valor es persistente, si el campo es una clave principal de la tabla, si se desea que su valor se establezca en el método `ejbCreate()` de la clase del bean y dónde se definen los métodos de acceso del campo, así como asignarlo a una columna de la tabla. Si el servidor de destino es WebLogic Server 6.x o 7.x, también se puede especificar el tipo de columna de la base de datos (dbms-column-type); se puede elegir entre OracleClob, OracleBlob y el valor por defecto (ninguno).

El inspector de métodos de los beans entidad es igual que el de los beans sesión y gestionados por mensajes. Si desea más detalles, consulte [“Añadir un nuevo método” en la página 6-13](#).

## Establecimiento de relaciones entre beans entidad

Es posible crear relaciones entre beans entidad EJB 2.0. Por ejemplo, si se tiene una tabla llamada `Department` y otra llamada `Employee` tendría sentido crear una relación que refleje qué empleados pertenecen a qué departamentos. En este ejemplo se debe hacer clic con el botón derecho del ratón en la representación del bean `Department`, en el panel del diseñador de EJB, y elegir `Añadir | Relación`, o pulsar el icono `Añadir` de la barra de herramientas y elegir Relación. En el bean `Department` aparece un nuevo campo. Haga clic en el bean `Employee` y los dos beans se conectarán

mediante una línea. El campo que se ha añadido al bean Department se llama ahora employee con el fin de reflejar la nueva relación.



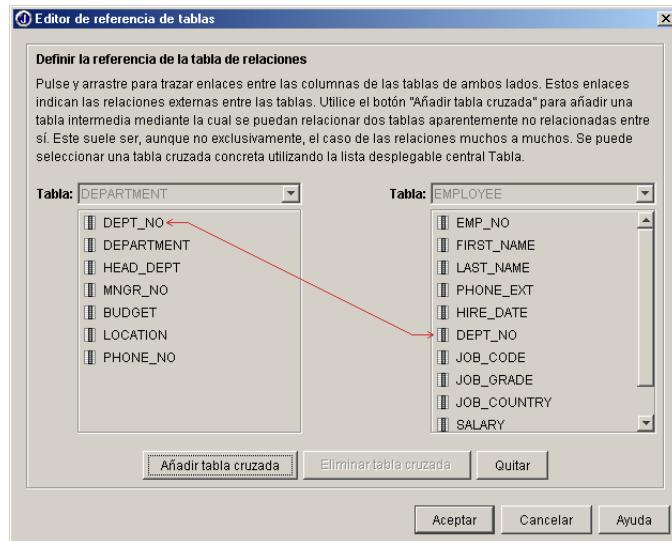
Por defecto, en la creación de relaciones, el diseñador de EJB busca en las dos tablas conectadas columnas o campos con el mismo nombre. Sin embargo, es frecuente que no exista ningún nombre de columna común, por lo que se debe indicar la forma en que se debe crear la relación.

### Establecimiento de relaciones por medio del inspector

Para indicar cómo se debe crear una relación, pulse el campo que se acaba de añadir al bean en el que se acaba de empezar a definir la relación. Se abre un inspector de relaciones:

<b>Propiedades de la relación</b>	
Relación:	department-employee
Multiplicidad:	uno a muchos
Navegabilidad:	unidireccional
Borrar en cascada:	false
<b>Propiedades de campo CMR</b>	
Nombre del campo:	employee
Tipo devuelto:	java.util.Collection
De obtención:	local
De asignación:	local
<a href="#">Modificar referencia de tabla...</a>	

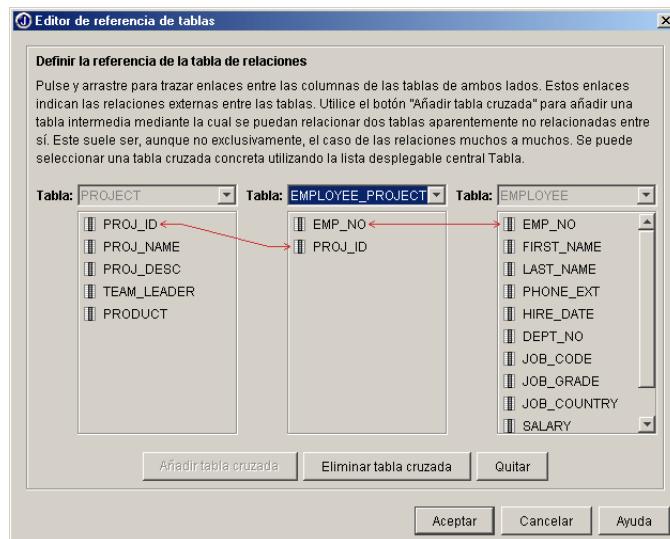
Pulse Modificar referencia de tabla para abrir el editor:



Si las dos tablas tienen columnas con el mismo nombre, el editor de referencia de tablas traza líneas entre ellas. De lo contrario, se debe trazar una línea entre las columnas que desea utilizar para crear la relación. Por ejemplo, si la tabla Department tiene una columna llamada DEPT\_NO y la tabla Employee tiene una columna llamada DEPT, puede hacer clic y arrastrar el puntero desde la columna DEPT\_NO de la tabla Department hasta la columna DEPT de la tabla Employee. Cuando se suelta el botón del ratón, aparece una línea que indica la relación entre las dos columnas.

En las relaciones múltiples se necesita una tercera tabla para establecer la conexión entre las dos primeras. Por ejemplo, puede existir una tabla llamada Employee que contiene una columna llamada EMP\_NO. Además puede existir una tabla llamada Project que contiene una columna llamada PROJ\_ID. Estas dos tablas, por sí mismas, no tienen ninguna columna que se pueda utilizar para establecer una relación directa. Si se tiene una tabla llamada Employee-Project con dos columnas, EMP\_NO y PROJ\_ID, se puede utilizar para crear una relación entre Employee y Project. En este caso, se debe pulsar Añadir tabla cruzada en el editor de referencia de tablas. Seleccione, en la lista desplegable que aparece entre las dos tablas, la que contiene las columnas con las que se establece la relación entre ellas. En este caso se debería seleccionar la tabla Employee-Project. A continuación se arrastra entre las columnas EMP\_NO de las tablas Employee y

Employee-Project, y se hace lo mismo entre las columnas PROJ\_ID de la tabla Project:



Pulse Aceptar cuando termine con el editor Modificar referencia de tabla, para volver al inspector del campo nuevo. En este inspector se puede indicar si se trata de una relación entre dos tablas, entre una y varias o entre varias y varias.

También se puede indicar si la relación es unidireccional o bidireccional. Si se elige la relación bidireccional, en el segundo bean también aparece un campo nuevo. Por ejemplo, si hay un bean entidad llamado Employee y otro llamado Project, puede ser necesario crear una relación que permita que un empleado tenga varios proyectos, y a la vez, realizar un seguimiento de todos los empleados que trabajan en un único proyecto. En este caso, la relación es bidireccional. Por ello, el bean Employee ha de tener un nuevo campo llamado project, y el bean Project ha de tener un nuevo campo llamado employee.

Si desea que se borren las filas relacionadas en la otra tabla de la relación cuando se borre la primera, asigne true al campo Borrar en cascada. Por ejemplo, un proyecto está asignado a varios empleados. Si la tabla Project está relacionada con la tabla Employee por el campo employee y se ha activado Borrar en cascada, cuando se borra un proyecto también se borran todos los empleados correspondientes. El campo Borrar en cascada puede configurar solo para relaciones de uno a uno o de uno a muchos.

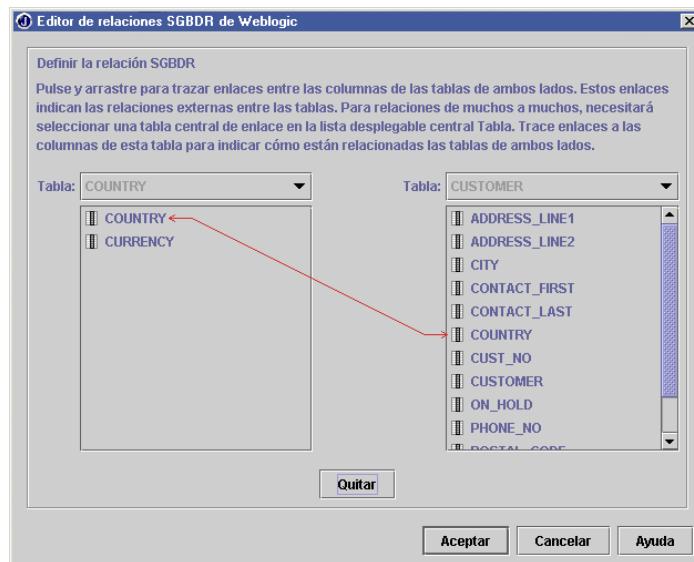
El campo Propiedades de campo CMR del inspector permite indicar en qué interfaces se declaran los métodos de acceso de obtención y definición. En el campo Tipo devuelto se establece el tipo de devolución del método de obtención y el tipo del parámetro que se pasa al método de definición del campo.

## Especificación de una relación WebLogic 6.x o 7.x

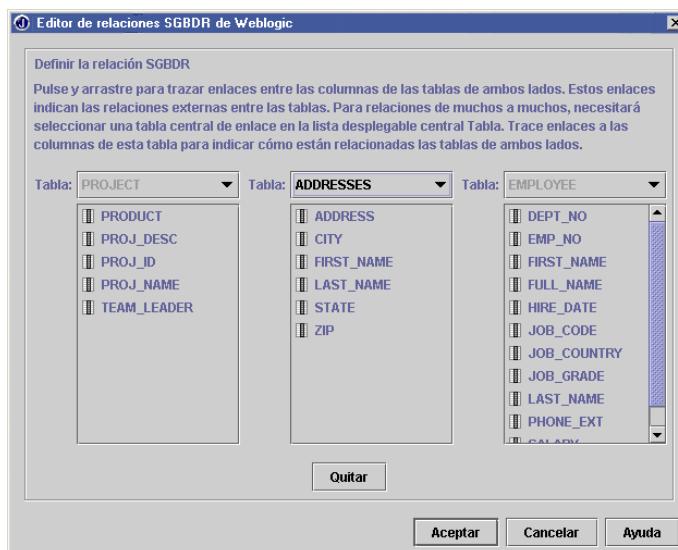
Si el servidor de destino es WebLogic Server 6.x o 7.x, la relación se crea de la forma descrita en “[Establecimiento de relaciones por medio del inspector](#)” en la página 7-17, con algunas diferencias. El inspector de relaciones presentará el siguiente aspecto:

Propiedades de la relación	
Relación	project-employee
Multiplicidad:	uno a muchos
Navegabilidad:	unidireccional
Borrar en cascada:	false
db-cascade-delete	false
Propiedades de campo CMR	
Nombre del campo:	employee
Tipo devuelto:	java.util.Collection
De obtención:	local
De asignación:	local
<a href="#">Modificar relación SGBDR...</a>	

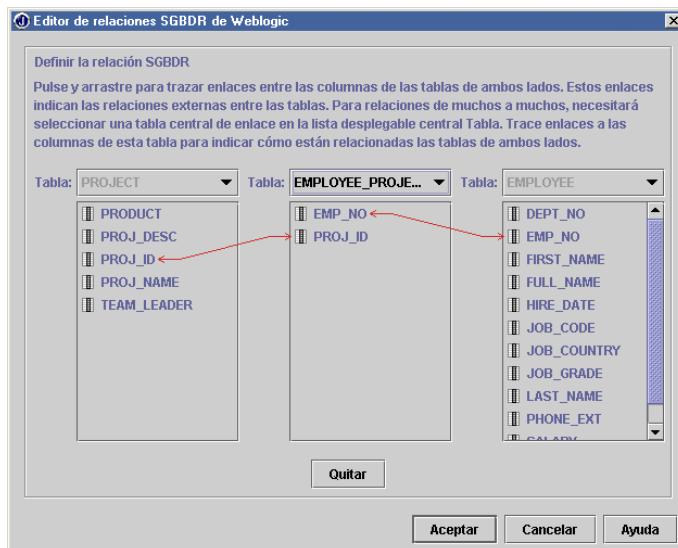
Pulse Modificar relación RDBMS para abrir el editor de relaciones RDBMS deWebLogic. Si se ha definido la multiplicidad como de uno a uno o de uno a muchos en el inspector, el editor presenta el siguiente aspecto:



Si se ha definido la multiplicidad como de muchos a muchos, el editor presenta el siguiente aspecto:



En este caso se selecciona la tabla central, que contiene las columnas que enlazan las otras dos tablas. A continuación se debe arrastrar el puntero entre las columnas que establecen la relación:



## Eliminación de relaciones

Para eliminar una relación entre dos beans entidad, haga clic con el botón derecho del ratón en el campo que se añadió durante el establecimiento de la relación y elija Borrar relación en el menú contextual.

## Adición de métodos de búsqueda

---

Los métodos de búsqueda permiten al cliente acceder al resultado de las consultas en EJB QL. EJB QL es el lenguaje utilizado por los métodos de consulta de persistencia gestionada por contenedor, según se define en la especificación EJB 2.0. Las consultas en EJB QL son cadenas que deben contener las cláusulas SELECT y FROM, y que pueden contener una cláusula WHERE.

Para añadir métodos de búsqueda a beans entidad:

- 1 Pulse la representación del bean en el diseñador de EJB con el botón derecho del ratón y elija Añadir | Buscador en el menú contextual, o pulse el icono Añadir de la barra de herramientas y elija Buscador.  
En la parte inferior del bean aparece un nuevo método de búsqueda, junto con el inspector del buscador.
- 2 Este inspector se puede utilizar para cambiar el nombre del método de búsqueda.
- 3 Indique si el buscador debe devolver una instancia del bean o una java.util.Collection .
- 4 Indique los parámetros que se pasan al buscador. Especifique los tipos de datos y los nombres de los parámetros como valor de Parámetros de entrada. Por ejemplo, java.lang.String lastName.
- 5 En el campo Interfaces base, indique si el buscador se debe declarar en la interfaz base, en la interfaz base local o en las dos.
- 6 Si decide introducir una consulta, escriba una sentencia en EJB QL. Por ejemplo:

```
SELECT DISTINCT OBJECT(o)
  FROM Order AS o, IN(o.lineItems) AS l
 WHERE l.shipped = FALSE
```

Algunos servidores de aplicaciones también permiten utilizar extensiones propias. Si desea más información sobre la definición de consultas, examine la documentación del servidor de aplicaciones.

El inspector del buscador puede tener campos adicionales, según el servidor de aplicaciones de destino. Si necesita ayuda para llenar estos campos adicionales, consulte la documentación del servidor de aplicaciones.

Si desea más información sobre la escritura de consultas con el lenguaje de EJB 2.0, consulte “Enterprise JavaBeans Query Language” en el tutorial para J2EE de la página web de Sun, en [http://java.sun.com/j2ee/tutorial/1\\_3-fcs/doc/EJBQL.html](http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBQL.html).

## Cómo añadir métodos ejbSelect

---

EJB QL se utiliza también para crear métodos ejbSelect() abstractos. Estos métodos permiten utilizar el EJB QL para buscar objetos o valores relacionados con el estado de un bean entidad, sin necesidad de proporcionar el resultado al cliente.

Para añadir un método ejbSelect():

- 1 Pulse con el botón derecho del ratón sobre el nombre del bean en la representación del diseñador de EJB y elija Añadir | Seleccionar en el menú contextual.

En la parte inferior del bean aparece un método de selección, junto con su inspector.

- 2 Este inspector se puede utilizar para cambiar el nombre del método. El diseñador de EJB antepone el prefijo ejbSelect al nombre que se escribe para el método. Por ejemplo, si se especifica el nombre AllLineItems en el inspector, el diseñador de EJB coloca un método abstracto ejbSelectAllLineItems() en la clase del bean.
- 3 Indique si el método debe devolver una instancia del bean, un java.util.Collection o un java.util.Set.
- 4 Indique los parámetros que se pasan al método ejbSelect(). Especifique los tipos de datos y el nombre del parámetro como valor de Parámetros de entrada. Por ejemplo, java.lang.String lastName.
- 5 Seleccione un Mapeo de tipos devueltos de la lista de selección.
- 6 Escriba una sentencia en EJB QL. Las consultas en EJB QL son cadenas que deben contener las cláusulas SELECT y FROM, y que pueden contener una cláusula WHERE.

Algunos inspectores del método de selección puede tener campos adicionales, según el servidor de aplicaciones de destino. Si necesita ayuda para llenar estos campos adicionales, consulte la documentación del servidor de aplicaciones.

Si desea más información sobre la escritura de consultas con el lenguaje de EJB 2.0, consulte “Enterprise JavaBeans Query Language” en el tutorial para J2EE de la página web de Sun, en [http://java.sun.com/j2ee/tutorial/1\\_3-fcs/doc/EJBQL.html](http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBQL.html). Consulte también la documentación de su servidor de aplicaciones.

## Adición de métodos base empresariales

---

Los componentes EJB 2.0 (tanto CMP como BMP) pueden tener métodos empresariales, con la forma `ejbHome`, que se declaran en la interfaz base. Para crear un método base empresarial:

- 1 Haga clic con el botón derecho del ratón en la representación del bean entidad, en el diseñador de EJB, y elija Añadir | Método o bien pulse el icono Añadir de la barra de herramientas y seleccione Método en el menú que aparece.
- 2 Indique en el inspector el nombre del método, su tipo devuelto y sus parámetros, como haría con cualquier otro método.
- 3 En el campo Interfaz, elija el valor Home (base), Local Home (base local) u Home/Local Home (base/base local).

Cuando se especifica una interfaz base, el diseñador de EJB declara el método, con el nombre elegido, en las interfaces base indicadas.

También añade el método a la clase del bean, con el prefijo `ejbHome`.

Como en cualquier otro método empresarial, debe añadir la lógica necesaria para implementar el método `ejbHome` en la clase del bean.

## Creación de esquemas a partir de beans entidad

---

Es posible utilizar un bean entidad en el diseñador de EJB para crear un esquema y añadirlo a una fuente de datos en el panel de estructura.

Para crear esquemas a partir de beans entidad:

- 1 Seleccione los beans entidad que desea utilizar.
- 2 Haga clic con el botón derecho del ratón en el panel del diseñador de EJB y elija Crear esquema a partir de selección o pulse el icono Crear esquema a partir de selección de la barra de herramientas.
- 3 Seleccione la fuente de datos a la que desea añadir el esquema y pulse Aceptar.

## Exportación de fuentes de datos

---

Cuando se modifica una fuente de datos, ésta se puede utilizar para crear tablas de base de datos que reflejen los cambios realizados. Haga clic con el botón derecho del ratón en el nodo de fuente de datos, en el panel de estructura, y elija Exportar esquema a LDD de SQL (LDD equivaldría a lenguaje de definición de datos). Escriba un nombre de archivo en el cuadro de diálogo que aparece. Este archivo se puede utilizar para crear una base de datos que utilice la fuente de datos modificada.

## Creación de beans entidad con persistencia gestionada por bean

---

El diseñador de EJB también puede ayudarle a crear beans entidad con persistencia gestionada por bean (BMP):

- 1 Haga clic con el botón derecho en el panel del diseñador de EJB.
- 2 Elija Crear EJB | Bean entidad BMP o pulse sobre el icono Crear EJB de la barra de herramientas del diseñador de EJB y seleccione Bean entidad BMP en el menú que aparece.

Diseñe el bean BMP de la misma manera que lo haría con cualquier otro bean y añada la lógica necesaria. A diferencia de lo que ocurre con los beans entidad CMP 2.0, debe añadir el código necesario para gestionar la persistencia del bean.



# Creación de componentes EJB 1.x con JBuilder

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

En este capítulo se explica la forma de utilizar JBuilder para crear componentes compatibles con Enterprise JavaBeans 1.1 de Sun Microsystems así como con componentes EJB 1.0 creados con WebSphere 3.5. Si desea información sobre la forma de crear componentes compatibles con la especificación Enterprise JavaBeans 2.0, consulte el Capítulo 6, "Creación de beans sesión y gestionados por mensajes 2.0 con el diseñador de EJB" y el Capítulo 7, "Creación de beans entidad 2.0 con el diseñador de EJB".

## Módulos EJB

Todos los enterprise beans deben pertenecer a un módulo EJB de JBuilder. Un módulo EJB es una agrupación lógica de uno o más beans que se distribuirán en un solo archivo JAR. En las versiones anteriores de JBuilder, los módulos EJB se denominaban "grupos EJB". Los dos términos designan el mismo concepto. Contiene la información necesaria para producir los descriptores de distribución del archivo JAR. Puede modificar el contenido de un módulo EJB mediante el Editor de descriptor de distribución.

Después de crear un módulo EJB y modificarlo a voluntad con el editor del descriptor de distribución, se puede Ejecutar Make o Generar un módulo para crear el archivo JAR. JBuilder utiliza el descriptor de distribución para ayudar en la identificación de los archivos de clase que se incluyen en el archivo.

Los módulos EJB pueden tener dos formatos: XML y binario. Dado que los módulos EJB de formato XML son básicamente archivos de texto, resulta más fácil trabajar con ellos si se utiliza un sistema de control de versiones. Los módulos en formato binario son, en esencia, los descriptores de distribución en un recopilatorio .zip.

Es posible tener varios modulos EJB en un proyecto. Todos los módulos EJB de un proyecto utilizan la misma vía de acceso a clases y el mismo JDK, y están configurados para el mismo servidor de aplicaciones de destino.

Si todavía no lo ha hecho, sigue las instrucciones en el [Capítulo 5, “Configuración del servidor de aplicaciones de destino”](#). Debe seguir los pasos para agregar una biblioteca con los archivos del servidor de aplicaciones en cada proyecto EJB que comience.

## Creación de módulos EJB 1.x

---

Hay dos formas de crear un módulo EJB:

- Si aún no ha creado los enterprise beans, cree un módulo EJB con el asistente.
- Utilice el Asistente para módulos EJB a partir de descriptores con el objeto de crear un EJB a partir de los descriptores de distribución de los enterprise beans con los que cuente.

Si no hay un proyecto abierto cuando se ejecuta un asistente para grupos EJB, JBuilder presenta en primer lugar el asistente para proyectos. Cree un proyecto y aparecerá el asistente para EJB seleccionado.

### Creación de módulos EJB 1.x con el Asistente para grupos EJB vacíos

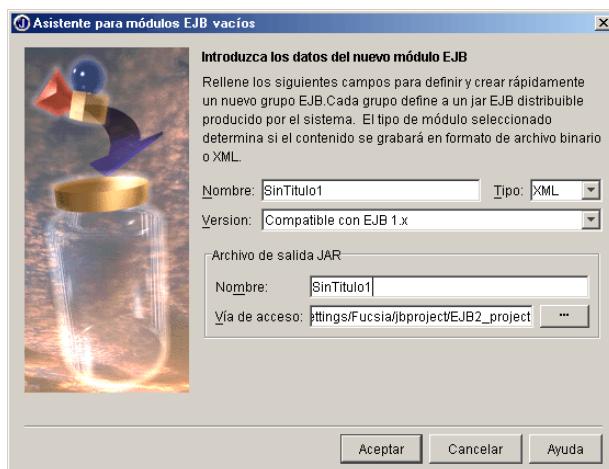
Si aún no ha creado los enterprise beans, empiece por crear un grupo EJB vacío. Para crear un módulo EJB:

- 1 Seleccione Archivo | Nuevo y haga clic en la pestaña Enterprise.

**Nota**

Si los asistentes para EJB de la ficha Enterprise están desactivados, eso significa que no tiene instalada la versión Enterprise de JBuilder, que no ha seleccionado un servidor de aplicaciones configurado para el proyecto o que el servidor de la aplicación seleccionado no está configurado. Si desea información sobre la forma de configurar y seleccionar el servidor de aplicaciones, consulte el [Capítulo 5, “Configuración del servidor de aplicaciones de destino”](#).

- 2** Haga doble clic en el ícono Asistente para módulos EJB y aparecerá el asistente:



- 3** Indique el nombre del módulo EJB.  
**4** Indique el formato del nuevo módulo.

Puede elegir entre binario, que almacena internamente los descriptores de distribución con formato .zip, y que se utilizaba antes de JBuilder 5, o bien XML, que almacena los descriptores de distribución con formato XML. El formato XML permite a los usuarios fusionar los cambios si se están comprobando en un sistema de control de fuentes. Se recomienda utilizar XML a menos que se comparta el archivo con una versión anterior de JBuilder.

- 5** Indique que la versión es compatible con EJB 1.x. Las opciones disponibles dependen del servidor de aplicaciones de destino.  
**6** Indique el nombre del archivo JAR en el que se encontrarán los enterprise beans.

JBuilder ha introducido un nombre por defecto, que es el mismo que el del módulo EJB. Basta con aceptar este nombre o escribir otro. JBuilder también ha introducido una ruta basada en la vía de acceso de su proyecto. Puede modificarla a su gusto o bien aceptar la ruta predeterminada.

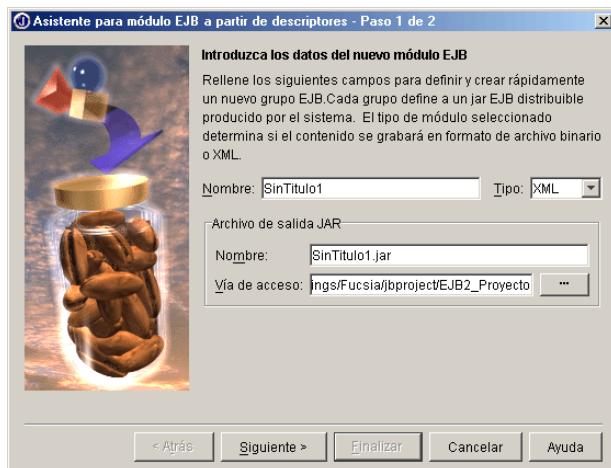
- 7** Pulse Aceptar para crear el módulo EJB.

## Creación de módulos EJB a partir de descriptores de distribución

El Asistente para módulos EJB a partir de descriptores sólo importa el descriptor de distribución ejb-jar.xml y los descriptores de distribución específicos de BEA WebLogic y Borland Enterprise Server a un nuevo módulo EJB.

Para crear módulos EJB a partir de descriptores existentes:

- 1** Seleccione Archivo | Nuevo y haga clic en la pestaña Enterprise.
- 2** Haga doble clic en el Asistente para módulos EJB a partir de descriptores y aparecerá el asistente:



- 3** Indique el nombre del nuevo módulo EJB.

- 4** Indique el formato del nuevo módulo.

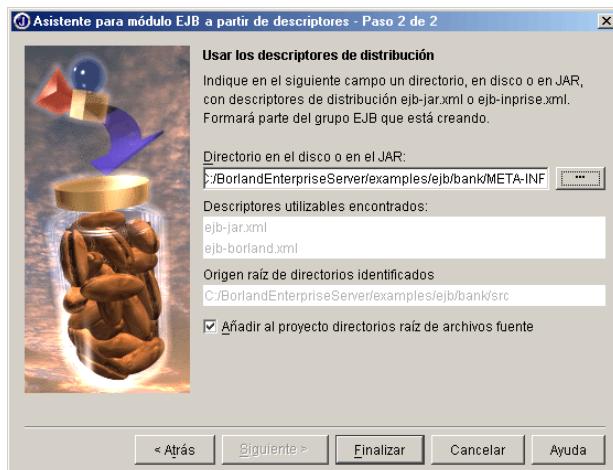
Puede elegir entre binario, que almacena internamente los descriptores de distribución con formato .zip, y que se utilizaba antes de JBuilder 5, o bien XML, que almacena los descriptores de distribución con formato XML. El formato XML permite a los usuarios fusionar los cambios si se están comprobando en un sistema de control de fuentes. Se recomienda utilizar XML a menos que se comparta el archivo con una versión anterior de JBuilder.

- 5** Indique el nombre y la vía de acceso del archivo JAR en el que se encontrará el enterprise bean.

JBuilder ha introducido un nombre por defecto, que es el mismo que el del módulo EJB. Basta con aceptar este nombre o escribir otro.

- 6** Pulse Siguiente e indique el directorio que contiene los descriptores de distribución que desea utilizar para el módulo. (Habitualmente se

encuentra en el directorio META-INF del JAR.) Al hacerlo, el asistente elabora una lista de los descriptores de distribución en el directorio especificado en el campo Descriptores utilizables encontrados. También muestra el directorio raíz de archivos fuente de estos descriptores en el campo Origen raíz de directorios identificados.



- 7 Si desea añadir al proyecto los directorios raíz de archivos fuente, active la casilla de selección Añadir al proyecto directorios raíz de archivos fuente.
- 8 Pulse Finalizar para crear el módulo EJB que contiene los descriptores de distribución de los beans.

## Creación de enterprise beans

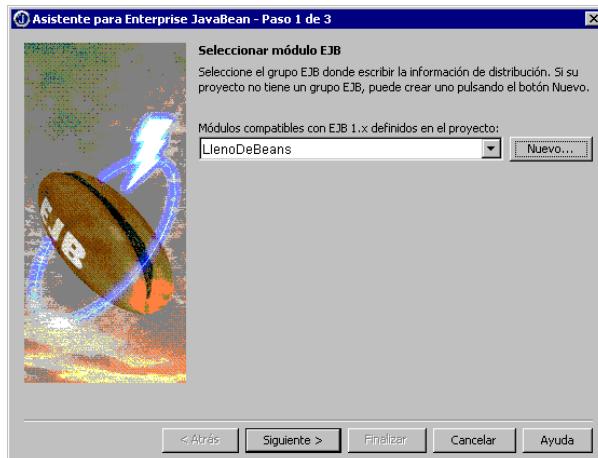
La galería de objetos de JBuilder contiene dos asistentes para la creación de enterprise beans 1.x: el Asistente para Enterprise JavaBean 1.x y el modelador de Bean entidad EJB 1.x. El menú Asistentes cuenta con otro más: el generador de Bean EJB 1.x. En este apartado se trata la creación de enterprise beans con el Asistente para Enterprise JavaBean 1.x. Si desea información sobre el uso del modelador de beans entidad EJB 1.x, consulte el [Capítulo 9, “Creación de beans entidad EJB 1.x a partir de una tabla de base de datos”](#).

El Asistente para Enterprise JavaBean 1.x y el Modelador de Bean entidad EJB 1.x crean las interfaces base y remota a la vez que la clase del bean. Si prefiere empezar el desarrollo de su enterprise bean creando primero la interfaz remota, consulte [“Generación de la clase del bean desde una interfaz remota” en la página 8-13](#) para obtener más información sobre el uso del Generador de bean EJB para generar la clase de bean desde la interfaz remota que haya creado.

Si desea empezar a crear un enterprise bean con la ayuda del Asistente para Enterprise JavaBean 1.x:

- 1 Seleccione Archivo | Nuevo y haga clic en la pestaña Enterprise.
- 2 Haga doble clic en el ícono Enterprise JavaBean 1.x.

Se abre el asistente.



- 3 En la lista desplegable, seleccione el módulo EJB al que desea que pertenezca su enterprise bean. Elija Siguiente para que aparezca la ficha 2 del asistente.

Si no tiene un módulo EJB definido en el momento de iniciar el Asistente para Enterprise JavaBeans, o desea crear otro, pulse Siguiente y se abrirá el Asistente para módulos EJB. Para crear un enterprise bean es necesario tener por lo menos un módulo EJB definido en el proyecto. Después de crear un módulo EJB con el Asistente para módulos EJB, seleccione el nuevo módulo y pulse Siguiente para continuar con el Asistente para Enterprise JavaBean.

- 4 Indique el nombre de la clase del bean, el paquete en el que se encontrará y su clase base.

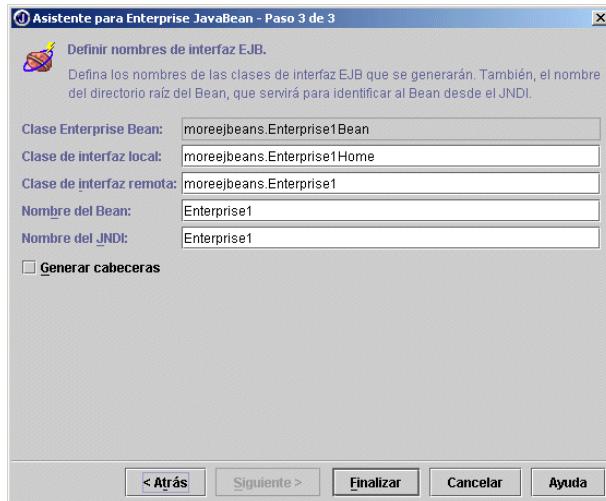
A continuación debe decidir si desea crear un enterprise bean de sesión o entidad.

## Creación de beans sesión

Si va a crear un bean sesión:

- 1 Pulse Bean sesión sin estado o Bean sesión con estado.

- 2** Si selecciona un Bean sesión con estado, puede optar también por implementar la interfaz de sincronización de sesiones (`SessionSynchronization`), activando la casilla de selección correspondiente.
- 3** Pulse Siguiente para pasar a la tercera ficha.



- 4** Elija los nombres que prefiera para las clases de interfaz base y remota y el nombre de la raíz del bean o acepte los que propone JBuilder basándose en el nombre de la clase del bean.
- 5** Pulse el botón Finalizar.

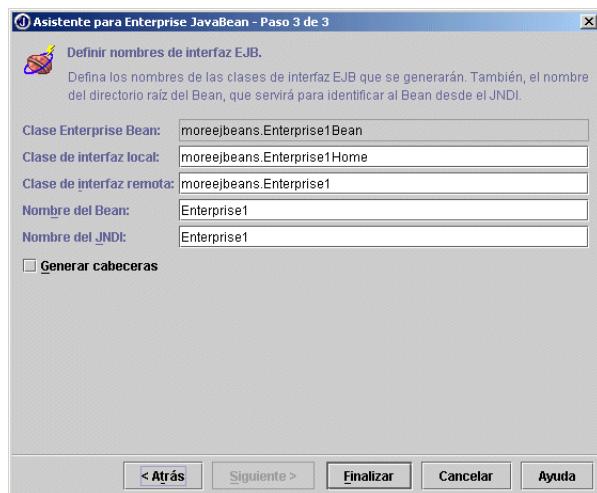
## Creación de beans entidad

---

Si va a crear un bean entidad:

- 1** Elija Bean entidad con persistencia gestionada por Bean o Bean entidad EJB 1.x de persistencia gestionada por contenedor. (Si ha seleccionado el servidor de aplicaciones WebSphere 3.5, la segunda opción es Bean entidad 1.0 de persistencia gestionada por contenedor.)
- 2** Indique una clase de clave principal.

**3** Pulse Siguiente para pasar a la tercera ficha.



**4** Elija los nombres que prefiera para las clases de interfaz base y remota y el nombre de la raíz del bean o acepte los que propone JBuilder basándose en el nombre de la clase del bean.

**5** Pulse el botón Finalizar.

JBuilder crea la clase del bean y sus interfaces local y remota. Los verá aparecer en el panel de proyectos. Examine el código fuente de la clase del bean y podrá ver que implementa la interfaz SessionBean si es un bean de sesión, y EntityBean si es un bean entidad. JBuilder ha añadido métodos con el cuerpo vacío para los métodos que deben implementar todos los enterprise beans. Añádale el código que necesitará el bean cuando se llame a estos métodos.

La interfaz base amplía la interfaz EJBHome y contiene un método create() necesario para crear el bean. La interfaz remota amplía EJBObject pero está vacía porque aún no se han declarado sus métodos de lógica empresarial.

Aunque puede empezar a crear sus beans entidad utilizando el asistente para Enterprise JavaBeans, la mejor forma de crear un bean entidad consiste en utilizar el Modelador de bean entidad EJB 1.x. Los Bean entidad que cree con el Asistente para Enterprise JavaBeans 1.x no podrán pasar la verificación con el editor del descriptor de distribución hasta que termine el bean por completo.

**Nota**

La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

Con WebSphere 4.0 Advanced Edition los descriptores de asignación no se generan en esta etapa. Los genera más adelante EjbDeploy, durante el proceso de generación del bean. Después se puede modificar la asignación y volver a ejecutar el Make para conservar esta asignación en el JAR.

Si desea que JBuilder cree descriptores adicionales en lugar de EjbDeploy, siga estos pasos:

- 1 Si WebSphere 4.0 Advanced Edition es el servidor seleccionado, haga clic con el botón derecho del ratón sobre el nodo del módulo EJB que aparece en el panel de proyecto.
- 2 Seleccione Propiedades en el menú contextual.
- 3 Abra la pestaña WebSphere AE 4.0.
- 4 Active la casilla de selección Generar descriptores CMP.
- 5 Pulse Aceptar.

**Nota** Para WebSphere 4.0 Single Server, los descriptores de asignación no se generan ya que no son necesarios.

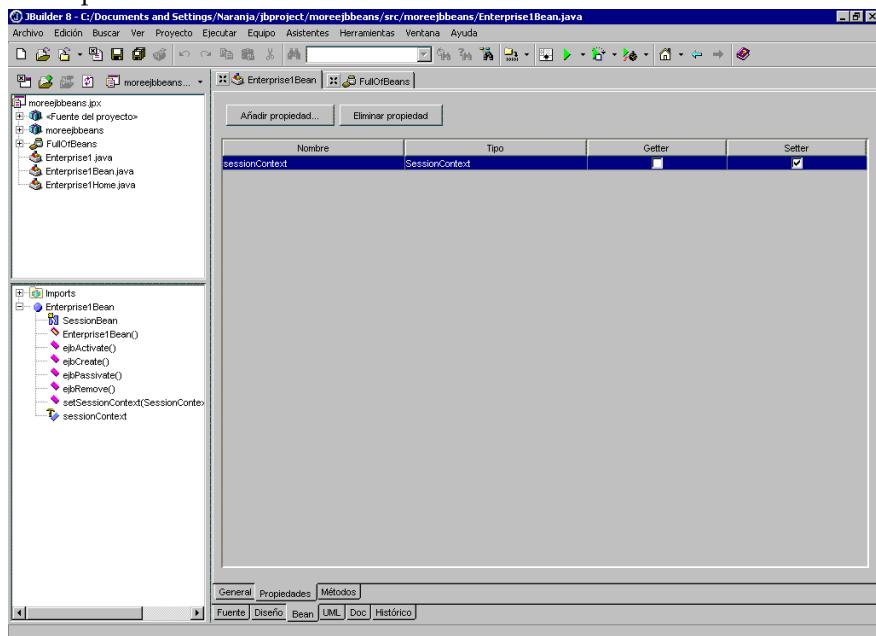
## Cómo añadir lógica empresarial al bean

Defina y escriba en el código fuente de la clase del bean los métodos que implementan la lógica empresarial que necesita el enterprise bean.

Si necesita añadir propiedades al bean, puede hacerlo directamente en el código fuente o utilizar la ficha Propiedades del diseñador de beans.

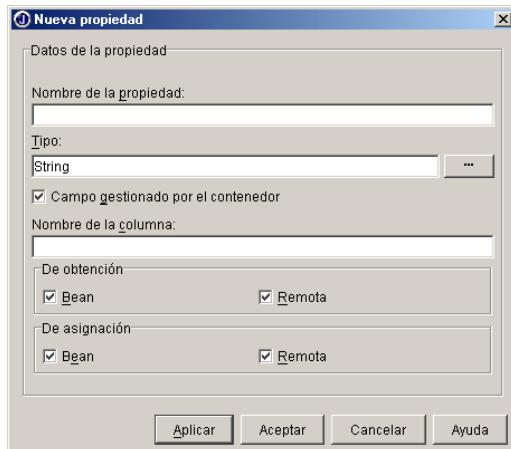
Para cambiar las propiedades con el diseñador de Beans:

- 1 Haga doble clic en la clase del bean, en el panel de proyecto.
- 2 Pulse sobre la pestaña Bean para visualizar el diseñador de Beans.
- 3 Haga clic sobre la pestaña Propiedades para visualizar la ficha Propiedades.



Para añadir una propiedad:

- Pulse el botón Añadir propiedad para mostrar el cuadro de diálogo Nueva propiedad.



- Indique el nombre y el tipo de la propiedad.
- Si se trata de un bean entidad con persistencia gestionada por contenedor, las opciones Campo gestionado por el contenedor y Nombre de la columna están disponibles en el cuadro de diálogo Nueva propiedad. Si la propiedad que está creando es una columna de una tabla de base de datos, active la casilla de selección Campo gestionado por el contenedor e indique el nombre de la columna en la tabla como el valor del campo Nombre de la columna.
- Defina los métodos de acceso con las opciones de obtención y de asignación.  
Si considera que la propiedad requiere un método de acceso por obtención, puede determinar si éste aparece en la clase de bean y/o en la interfaz remota. Y si considera que requiere un método de acceso por asignación, puede también determinar si aparece en la clase de bean y/o en la interfaz remota.
- Puede añadir más propiedades en el cuadro de diálogo. Cuando termine, pulse Aceptar.

Si ha empleado el Asistente para Enterprise JavaBean para iniciar la creación de un bean entidad con persistencia gestionada por contenedor, debe añadir propiedades al bean. Tenga en cuenta que al menos una de las propiedades debe ser la clave principal y que es necesario indicar a qué

campo pertenece en el panel General del editor del descriptor de distribución. De lo contrario, no es posible verificar el descriptor de distribución como válido.

También puede utilizar la ficha Propiedades para modificar la propiedad. Por ejemplo, si no ha definido un método de asignación para la propiedad en el momento de declararla y decide que el bean lo necesita, basta con activar la casilla de verificación De asignación correspondiente a esta propiedad, en la ficha Propiedades, y JBuilder añadirá el método de asignación. También es posible eliminar métodos de obtención y asignación, desactivando la casilla de verificación correspondiente.

Para eliminar una propiedad de un bean desde la ficha Propiedades:

- 1** Seleccione la propiedad en la tabla.
  - 2** Pulse el botón Eliminar propiedad.
- JBuilder pregunta si se desea eliminar la propiedad y su código asociado.
- 3** Pulse Sí.

También puede utilizar la ficha Propiedades para modificar el nombre y el tipo de la propiedad. El diseñador de beans es una herramienta bidireccional: los cambios realizados en la hoja Propiedades se reflejan en el código y viceversa.

## Exposición de métodos empresariales mediante la interfaz remota

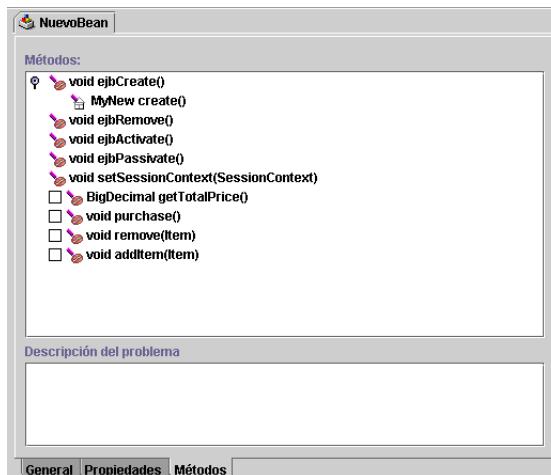
---

Después de declarar los métodos de lógica empresarial en el código fuente del bean es necesario indicar cuáles se desea añadir a la interfaz remota. El cliente sólo puede llamar a los métodos expuestos a través de la interfaz remota del bean.

Para añadir métodos a la interfaz remota:

- 1** Haga doble clic en el enterprise bean, en el panel de proyecto.
- 2** Pulse sobre la pestaña Bean para visualizar el diseñador de Beans.
- 3** Pulse sobre la pestaña Métodos.

- 4 En el cuadro de diálogo Métodos, active la casilla de selección contigua a los métodos que desea exponer en la interfaz remota.



Cuando se eligen elementos del cuadro de diálogo Métodos, éstos se añaden a la interfaz remota.

Para eliminar un método de la interfaz remota, desactive la casilla de selección contigua, en el cuadro de diálogo Métodos.

Para modificar uno de los métodos, haga clic con el botón derecho para abrir el menú contextual y elija Modificar seleccionados. El archivo se abre en el editor de código, con el cursor en el método, ya preparado para su modificación.

El menú contextual tiene otros comandos útiles. Eliminar seleccionados permite eliminar un método de la clase del bean. La opción Activar todo activa todos los métodos, de modo que se añaden a la interfaz remota. Desactivar todo desactiva todos los métodos, de modo que no se añade ninguno.

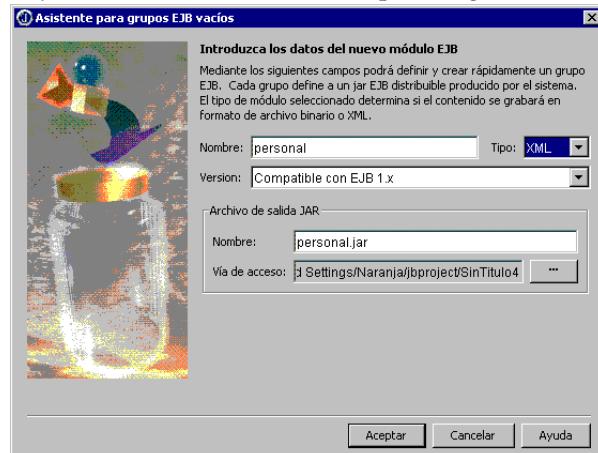
La ficha Métodos permite comprobar que todos los métodos declarados en la clase del bean tienen la misma firma que en las interfaces local y remota. Por ejemplo, supongamos que se añade un parámetro al método ejbCreate() en la clase del bean, pero no se hace lo mismo con el método create() de la interfaz base. El cuadro Métodos muestra ejbCreate() y create() en rojo. Si se hace clic en un método que aparece en rojo, el cuadro Descripción del problema explica el error. Entonces se puede añadir el parámetro necesario al método create(), con lo que las firmas de los métodos coinciden y se resuelve el problema. Si se eliminan métodos de la clase del bean pero se olvida hacer lo mismo en la interfaz remota, el cuadro de diálogo Método los muestra en rojo, para recordar que deben eliminarse de la interfaz.

## Generación de la clase del bean desde una interfaz remota

Algunos desarrolladores prefieren comenzar el desarrollo de un enterprise bean por el diseño de la interfaz remota. Si piensa lo mismo, puede utilizar el generador de Bean EJB 1.x para crear una clase esqueleto del bean desde su interfaz remota.

Si desea generar una clase del bean desde una interfaz remota:

- 1 Abra la interfaz remota en el editor.
- 2 Elija Asistentes | EJB | Asistente para el generador de EJB 1.x:



- 3 Seleccione el módulo EJB al que pertenece el bean y pulse Siguiente.



- 4 Seleccione el tipo de EJB que desea generar y, a continuación, haga clic en Siguiente.

Si eligió una de las opciones del bean sesión, aparece la siguiente ficha.



Especifique las opciones de Bean EJB: Clase del Bean, Nombre del Bean, Clase local y Nombre del JNDI.

Si eligió la opción de bean entidad con persistencia manejada por contenedor, aparece la siguiente pantalla:



Especifique las opciones de Bean EJB: Clase del Bean, Nombre del Bean, Clase local, Nombre del JNDI, Clase de clave principal y qué campos se quiere que sean persistentes.

## 5 Elija Finalizar.

El generador de Bean EJB crea la clase esqueleto del bean especificado, y que incluye los métodos de la interfaz remota. En la nueva clase del bean, estos métodos incluyen un comentario que le recordará rellenar las implementaciones. Debe agregar su código a los métodos para implementarlos como desee.

El generador de Bean EJB 1.x también crea una interfaz base si no existía con anterioridad. Si existía una interfaz base, el generador de Bean EJB 1.x le pregunta si desea sobreescribirla y actúa en consecuencia.

## Creación de las interfaces base y remota para beans existentes

Si ya cuenta con una clase del bean pero no con las interfaces remota y local que necesita, puede utilizar el Asistente para el Generador de interfaces EJB 1.x para crearlas. También puede utilizar este asistente si ha efectuado cambios significativos en el código fuente del bean y desea que se reflejen en las interfaces. El Generador de interfaces EJB 1.x permite regenerar las interfaces a partir del código fuente revisado de la clase del bean.

Para utilizar el Asistente para el Generador de interfaces EJB 1.x:

- 1 Abra el código fuente del bean en el editor de código.
- 2 Elija Asistentes | EJB | Generador de interfaces EJB 1.x.

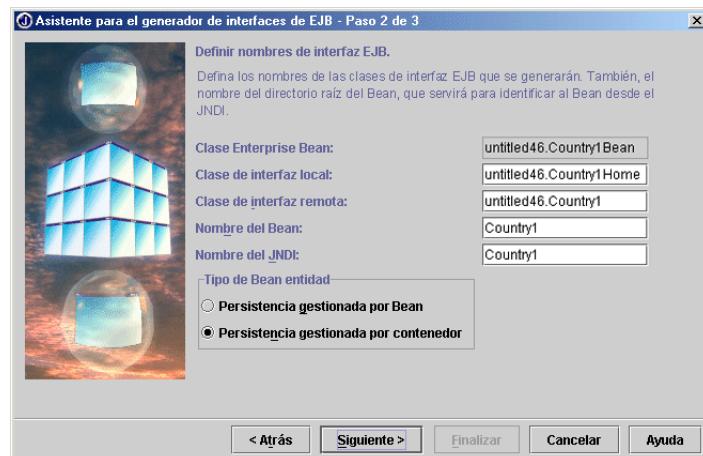


- 3 Seleccione el módulo EJB al que pertenece el bean y pulse Siguiente.

Si el bean es un bean sesión, aparece la siguiente ficha.

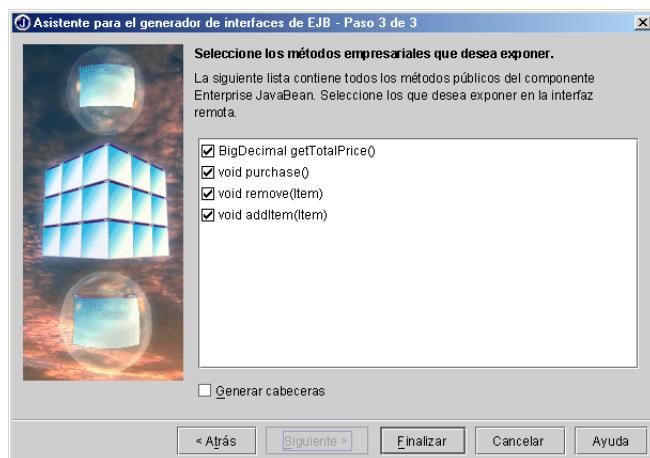


Si el bean es un bean entidad, aparece la siguiente ficha.



- 4 Acepte los nombres por defecto o escriba los que deseé.
- 5 Si el enterprise bean es de sesión, elija Con estado o Sin estado. Si el enterprise bean es un bean entidad, elija Bean entidad con persistencia gestionada por Bean o Bean entidad con persistencia gestionada por contenedor.

- 6 Pulse en Siguiente para que aparezca el Paso 3, que muestra los métodos del bean:



7 Deje activados los métodos que desea exponer en la interfaz remota, y desactive los que no desea que aparezcan en ella.

8 Elija Finalizar.

El siguiente paso consiste en compilar los beans, depurarlos y crear un archivo JAR. Consulte el [Capítulo 10, “Compilación de enterprise beans y creación de archivos JAR”](#).



# Creación de beans entidad EJB 1.x a partir de una tabla de base de datos

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

A menudo, los datos que desea modelar con un bean entidad ya existen en una base de datos. Puede utilizar el modelador de entidad de Jbuilder para crear beans entidad EJB 1.x. Si desea crear beans entidad EJB 2.0 que den forma a los datos de una base de datos, consulte “[Creación de beans entidad CMP 2.0 a partir de una fuente de datos importada](#)” en la página 7-2.

## Creación de beans entidad con el modelador de beans entidad EJB 1.x

El Asistente para modeladores de bean entidad EJB 1.x crea beans entidad a partir de las tablas de cualquier base de datos a la que se pueda acceder mediante JDBC. El asistente se puede utilizar para crear varios beans entidad a la vez, y se pueden establecer las relaciones entre ellos.

Después de utilizar el modelador de beans entidad EJB 1.x para generar el código que constituye los beans entidad, sus claves principales, sus interfaces local y remota y los elementos necesarios en el descriptor de distribución, se puede modificar el resultado mediante otras herramientas de JBuilder, como el diseñador de beans, el editor del descriptor de distribución y el editor de código de JBuilder.

Para abrir el Asistente para modeladores de entidad EJB 1.x, elija Archivo | Nuevo, abra la ficha Enterprise y seleccione Modelador de Bean

entidad EJB 1.x. Si tiene por lo menos un módulo EJB definido en el proyecto aparecerá el modelador de bean entidad EJB 1.x.



Todos los enterprise beans desarrollados con JBuilder deben pertenecer a un módulo EJB. Si no tiene por lo menos un módulo EJB en el proyecto actual, pulse Nuevo y se abrirá el Asistente para módulos EJB. Una vez que haya creado un módulo EJB con el Asistente para módulos EJB, puede continuar con el modelador de bean entidad.

Para crear beans a partir de tablas de base de datos, siga estos pasos:

- 1 Seleccione un módulo EJB en el que colocar el bean y pulse Siguiente para abrir la segunda ficha.

El módulo EJB seleccionado se utiliza para determinar dónde se escribe la información de distribución.



**2** Indique un origen de datos JDBC.

Introduzca la información necesaria para conectarse a una fuente de datos JDBC.

Si la conexión ya está creada, pulse Seleccionar conexión. El resto de la información necesaria para esta ficha se rellena automáticamente, con excepción de la contraseña, que se debe introducir de forma manual si la conexión la requiere.

Si no tiene una conexión o desea crear una, seleccione un controlador de la lista desplegable correspondiente y elija una dirección URL.

Aparecen los controladores configurados por medio de Herramientas | Configurar Enterprise, en la ficha Controladores de bases de datos.

Consulte "[Configuración de los controladores JDBC](#)" en la página 5-11 para obtener más información.

Indique el nombre de usuario de la fuente de datos y escriba la contraseña si es necesaria. Seleccione todas las propiedades ampliadas que necesite. Por último, especifique un nombre JNDI para la fuente de datos.

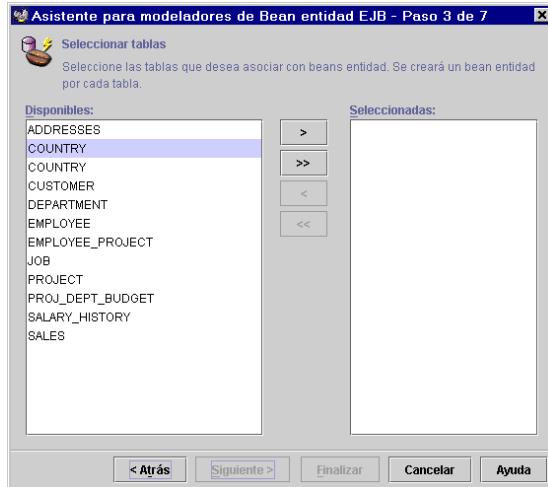
**3** Especifique las opciones Tipos de esquema y tablas que deseé.

Si marcó la opción Todos los esquemas, el modelador de Bean entidad EJB 1.x descargará todos los esquemas a los que el usuario tiene derecho para establecer la conexión. Si deja la opción Todos los esquemas desactivada, sólo se cargan los esquemas con el mismo nombre que el nombre de usuario, reduciendo el tiempo requerido para hacer la conexión y cargar los datos.

Marque la opción Vistas si desea que se descarguen las vistas en el modelador de Bean entidad EJB 1.x. Si no desea descargar las vistas, no marque esta opción.

**4** Pulse Siguiente.

El Asistente para modeladores de Bean entidad EJB 1.x intenta establecer una conexión con la fuente de datos especificada. La ficha siguiente aparece sólo si se establece la conexión.



**5 Seccione las tablas que desea asignar a beans entidad.**

Se creará un bean entidad por cada tabla seleccionada. Elija, en la lista Disponibles, las tablas deseadas y desplácelas a la lista Seleccionadas mediante los botones > y >>. Cuando haya seleccionado todas las tablas, pulse Siguiente.



**6 Seccione las columnas de las tablas para asignarlas a campos de beans entidad e indicar las relaciones que se desea establecer entre las tablas.**

En la sección Tablas y enlaces, verá todas las tablas seleccionadas en el paso anterior. Seleccione las tablas una a una haciendo clic en ellas y desplace las columnas entre las listas Disponibles y Seleccionadas en la sección Columnas de la tabla seleccionada. Por defecto están seleccionadas todas las columnas de todas las tablas.

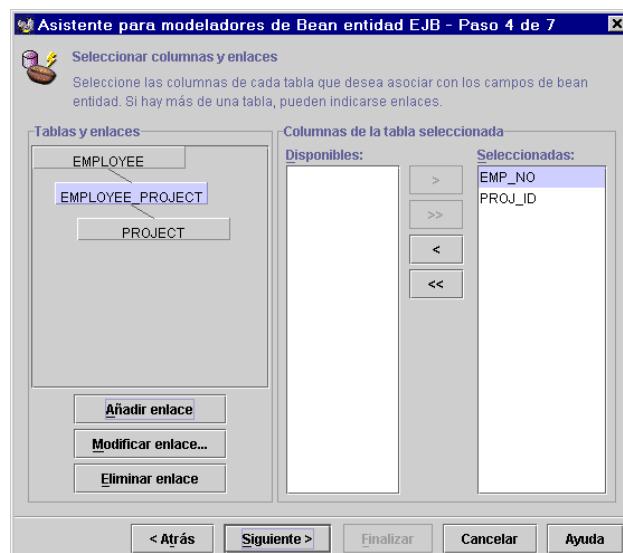
También es posible establecer relaciones entre las tablas arrastrando el puntero entre ellas, en el cuadro de diálogo Tablas y enlaces de la izquierda. Se obtiene el mismo resultado pulsando Añadir enlace. Cuando se utiliza cualquiera de estos métodos aparece un cuadro de diálogo que propone una relación basada en claves ajenas, claves principales, índices irrepetibles, y nombres y tipos de índices de las dos tablas. Puede aceptar la relación propuesta o modificarla para crear la deseada. Para eliminar una asociación entre tablas, pulse Eliminar enlace.

#### Nota

La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

El modelador de Bean entidad EJB 1.x admite relaciones CMP únicamente si el servidor es Borland Enterprise Server 5.0.2 -5.1.x o Borland AppServer 4.51.

He aquí un ejemplo de tres tablas enlazadas.



Después de seleccionar en las tablas todas las columnas que desea asignar a los campos de beans entidad que está creando, pulse Siguiente.



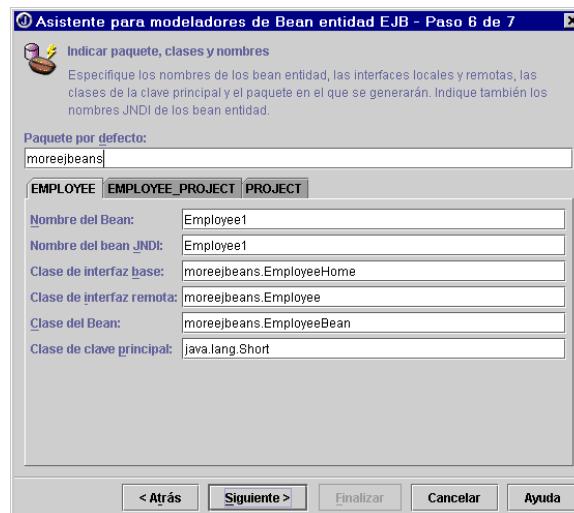
- 7 Seleccione los nombres y tipos de datos de los campos de beans entidad que se asignarán a las columnas de la tabla.

Haga clic en la pestaña adecuada para seleccionar la tabla en la que desea comenzar el proceso de asignación. Aparecen un nombre y un tipo de campo propuestos para cada columna de la tabla. Puede aceptarlos o modificarlos.

Si desea cambiar el tipo de datos de varios campos a un solo tipo, seleccione los campos que desea cambiar y seleccione Actualizar tipo de campo. Aparece un cuadro de diálogo en el que puede escribir el nuevo tipo de campo. Cuando elige Aplicar o Aceptar, el tipo de campo seleccionado para cada campo cambia.

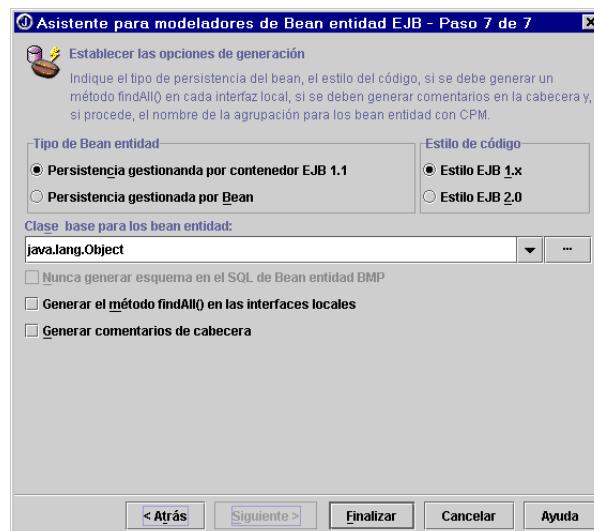
Si la tabla ya tiene una clave principal, este campo o conjunto de campos se selecciona cuando aparece la ficha de asignación de columnas. Si no existe ninguna clave principal, es necesario seleccionar campos para crearla, activando su casilla de selección en la columna Clave principal. Pulse Siguiente cuando haya terminado de asignar

todas las columnas seleccionadas a los nombres y tipos de campos que desee incluir en cada tabla de su bean entidad.



- 8 Indique el paquete, las clases, las interfaces y el nombre JNDI de todos los beans que esté creando.

Para cada tabla, JBuilder sugiere un nombre para el bean entidad, el nombre JNDI, el nombre de las interfaces local y remota, el nombre de la clase del bean y el tipo de la clase de clave principal. Puede especificar un paquete diferente para cada uno de estos nombres; por defecto, se sugiere el paquete del proyecto. Puede aceptar estos valores tal y como están o modificarlos según estime oportuno. Pulse Siguiente cuando haya terminado de especificar la información de cada tabla.



- 9** Decida si la persistencia de los beans entidad debe estar gestionada por éstos o por el contenedor.

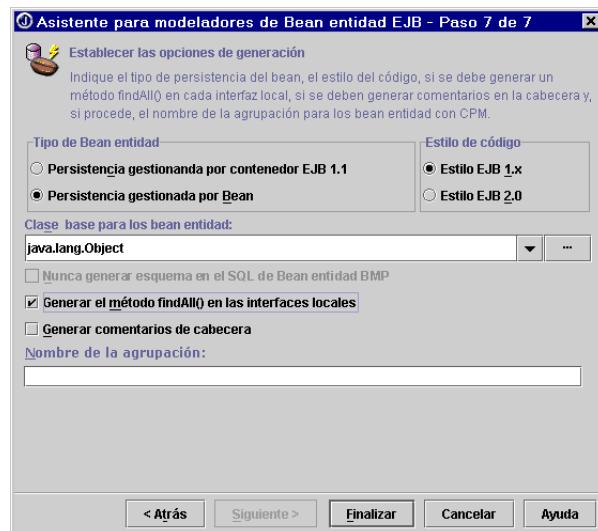
Si desea preparar el proyecto para EJB 2.0 y desea que el código generado siga este estilo, elija la opción EJB 2.0. Si desea más información sobre estas opciones pulse el botón Ayuda del modelador de Bean entidad EJB 1.x.

La clase base por defecto de estos beans es `java.lang.Object`. Si desea basar los beans entidad en otra clase, indíquela en el campo Clase base para los beans entidad.

Si desea que los beans entidad devuelvan todas las filas en un conjunto de datos, marque la opción Generar método `findAll()` en las interfaces locales. El modelador de Bean entidad EJB 1.x coloca un método `findAll()` en las interfaces base de los beans. También puede decidir si desea que los comentarios de cabecera aparezcan en los archivos resultantes.

Las opciones disponibles en esta pantalla dependen de cuál sea su servidor de aplicaciones de destino. Por ejemplo, si su servidor de destino es WebSphere 3.5, no aparecerá ninguna opción de persistencia gestionada por contenedor, ya que este servidor no admite esa opción.

Si el servidor de aplicación de destino es WebLogic Server 5.1 y está creando un bean entidad con persistencia generada por contenedor, esta ficha incluye también un campo llamado Nombre de la agrupación en el que se debe introducir el nombre del búfer de los beans CMP WebLogic:



- 10** Elija Finalizar.

JBuilder crea un bean entidad por cada tabla y para todas las interfaces necesarias. Ahora se puede añadir la lógica empresarial deseada a los beans, definir los métodos de la interfaz remota a los que debe ser capaz de llamar el cliente, compilar los beans y modificar los descriptores de distribución de los beans.

**Nota** Si desea utilizar la asistencia por contenedor de WebSphere 4.0 Advanced Edition, los archivos descriptores de distribución de asignación y esquema (Map.mapxmi y Schema.dbxmi) son descriptores estándar generados por la utilidad EjbDeploy, a la que se llama durante el proceso de generación. Si los nombres de los campos del bean no se corresponden directamente con las columnas de la base de datos, puede hacer que JBuilder cree descriptores de distribución CMP de WebSphere que redefinan el comportamiento por defecto de EjbDeploy :

- 1 Si WebSphere 4.0 es el servidor seleccionado, haga clic con el botón derecho del ratón sobre el nodo del módulo EJB que aparece en el panel de proyecto.
- 2 Seleccione Propiedades en el menú contextual.
- 3 Abra la pestaña WebSphere AE 4.0.
- 4 Active la casilla de selección Generar descriptores CMP.
- 5 Realice los cambios.

La lista Archivos de correspondencia de datos contiene los archivos de correspondencia utilizados para asignar tipos primitivos de Java a tipos de base de datos. Puede modificar, añadir y eliminar archivos de correspondencia. La lista Sustitución de tipo de base de datos en descriptores generados muestra la sustitución que JBuilder realiza si los tipos primitivos de Java no pueden asignarse directamente a un tipo en la base de datos. Puede modificar esta lista. Por ejemplo, podría sustituir otro tipo Java por un tipo original concreto en una base de datos. También puede añadir sustituciones adicionales a la lista.

- 6 Pulse Aceptar.

**Nota** En WebSphere 4.0 Single Server, no se generan descriptores de asignación ya que no son necesarios.



# 10

## Compilación de enterprise beans y creación de archivos JAR

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

Después de crear enterprise beans EJB 1.x o EJB 2.0 con JBuilder es necesario compilarlos y prepararlos para la distribución.

### Compilación de beans

Después de escribir y guardar el enterprise bean, sus interfaces y las clases de apoyo necesarias, se acerca el momento de la compilación. En este capítulo se explica la forma de compilar las clases de beans y crear archivos JAR.

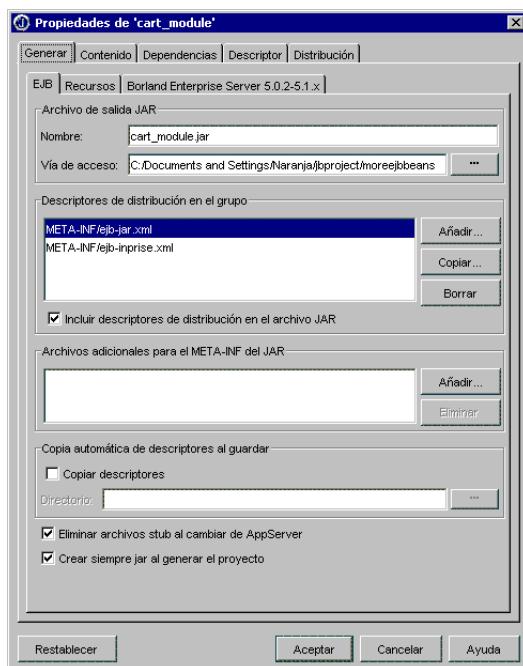
### Cambio de propiedades de generación de los módulos EJB o gruposEAR

Antes de dar comienzo a la compilación, quizás desee modificar las propiedades que determinan la forma en que se genera el archivo JAR, aunque no es necesario.

Para cambiar las propiedades de generación de un módulo EJB o grupo EAR:

- 1 Haga clic con el botón derecho del ratón en el módulo EJB o grupo EAR, en el panel de proyecto, y elija Propiedades.

- 2** Seleccione la pestaña Generar.
- 3** Seleccione la pestaña EJB o EAR.



- 4** Modifique las propiedades de generación como deseé.

Puede cambiar el nombre del archivo JAR o EAR de salida y el lugar donde se genera.

También puede insertar descriptoros de distribución en un módulo EJB o en un grupo EAR. Si pulsa el botón Agregar, aparece el cuadro de diálogo Insertar descriptor de distribución; utilícelo para buscar y seleccionar el archivo que desee agregar al módulo EJB. Si pulsa el botón Aceptar, aparece un cuadro de mensajes que aconseja añadir el archivo al directorio META-INF. Puede aceptar la sugerencia o seleccionar un subdirectorio existente de META-INF.

Puede copiar descriptoros de distribución, incluida su estructura actual de directorios, en una nueva ubicación. También puede eliminar descriptoros de distribución de un módulo EJB o de un grupo EAR.

Si desea indicar los archivos adicionales que deben añadirse al archivo JAR, pulse Añadir y especifique la ubicación. Debe hacer esto si, por ejemplo, ha añadido una clase al proyecto y desea que forme parte del archivo JAR. Si tiene descriptoros de distribución que haya modificado fuera de JBuilder, puede agregarlos en este punto y desactivar la opción Incluir descriptoros de distribución en el archivo JAR. Los descriptoros de distribución que aparecen en la lista Descriptoros de distribución en

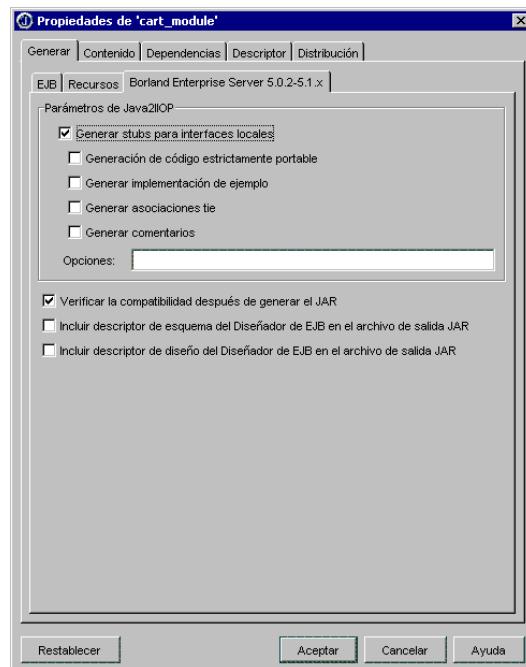
el módulo no se agregan al archivo JAR, pero sí los que especificó en la lista Archivos adicionales para el directorio META-INF del jar.

La opción Copiar automáticamente los descriptores al guardar permite guardar una copia en su estado actual cada vez que se guarda el módulo EJB, el grupo EAR o el proyecto. Para habilitar esta función, active la casilla de selección Copiar descriptores e indique en el campo Directorio el lugar donde se deben guardar. Los desarrolladores que escriben sus propias tareas de generación deberían utilizar esta función.

Si desea seleccionar distintos servidores de aplicaciones, puede activar la opción Eliminar archivos stub al cambiar de AppServer con el fin de eliminar los stubs cliente utilizados por el servidor de aplicaciones antiguo al seleccionar uno nuevo. Esto impide que los stubs destinados a otro servidor de aplicaciones se copien en el archivo JAR recién generado.

La opción Crear siempre jar al generar el proyecto está activa por defecto. Si se desactiva esta opción se evita que se genere un archivo JAR cada vez que se ejecuta el Make o se vuelve a generar un proyecto. Aunque la opción esté desactivada se puede crear el archivo JAR. Para ello, haga clic con el botón derecho del ratón en el módulo EJB y elija Make o Generar.

- 5 Haga clic en la pestaña del servidor de aplicaciones que haya seleccionado. Por ejemplo, esta imagen muestra seleccionada la pestaña del Borland Enterprise Server 5.0.2 -5,1.x:



- 6 Especifique las opciones de generación que desee. Si precisa más información sobre las opciones disponibles, pulse el botón Ayuda.
- 7 Pulse Aceptar cuando haya terminado.

## Cambio de las propiedades de generación de beans

---

Si se utiliza como destino Borland Enterprise Server 5.0.2 – 5.1.x y se va a comprobar el bean en el ordenador local, se deben generar y añadir los stubs clientes a la vía de acceso a clases. Esto se hace modificando las propiedades de generación de la interfaz base antes de compilar.

- 1 Haga clic con el botón derecho en la interfaz base del bean y seleccione Propiedades.
- 2 Haga clic en la pestaña Generar.
- 3 Haga clic en la pestaña VisiBroker.
- 4 Marque la casilla Generar IIOP y seleccione cualquier otra opción Java2IIOP que desee.
- 5 Pulse Aceptar.

La mayoría de los servidores de aplicaciones disponen de una ficha Propiedades de generación específica del servidor que puede utilizar para establecer las opciones de generación del módulo EJB.

## Cambio de las propiedades de generación de módulos EJB

---

Para visualizar la ficha Propiedades de generación del servidor de aplicaciones de destino:

- 1 Haga clic con el botón derecho del ratón en el módulo EJB del panel de proyecto.
- 2 Seleccione Propiedades.
- 3 Abra la pestaña Generar. Una vez abierta, pulse sobre la pestaña con el nombre de su servidor de aplicaciones.

Las opciones que aparecen en la ficha de propiedades de generación son diferentes dependiendo del servidor utilizado. Por ejemplo, la ficha de los servidores WebLogic incluye la configuración del compilador EJBC.

## Compilación

---

Para compilar todas las clases del proyecto, haga clic con el botón derecho del ratón en el archivo del proyecto (<proyecto>.jpx) y elija Ejecutar Make, o simplemente, seleccione Proyecto | Ejecutar Make del proyecto.

Durante el proceso de compilación, JBuilder puede detectar la presencia de un problema en un descriptor de distribución, lo que lo haría no válido. Si esto ocurre, en el panel de mensajes se informará de que se debe verificar el bean en el editor del descriptor de distribución. Si desea más información sobre la verificación del descriptor de distribución, consulte “[Comprobación de la información sobre descriptores](#)” en la página 13-51.

Si ha elegido generar los stubs clientes, verá que, en el panel de proyecto, el nodo de la interfaz local muestra varios archivos debajo si se pulsa su ícono de ampliación. Estos archivos generados son los stubs cliente y las clases auxiliares necesarios, que hacen que funcione los enterprise beans.

El proceso de generación se personaliza para el servidor de aplicaciones de destino y ejecuta las herramientas específicas, además de compilar los archivos .java.

**Nota para usuarios de WebSphere 4.0.**  
La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

Se generan dos descriptores de distribución (Map.mapxmi y Schema.dbxmi para los beans entidad con persistencia gestionada por el contenedor, sólo con la edición Advanced de WebSphere 4.0. Si utiliza WebSphere 4.0 y cambia el servidor de destino a la otra versión, es necesario volver a compilar el proyecto con el fin de garantizar que JBuilder genere los descriptores de distribución correctos.

## El archivo JAR generado

---

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

Para cada enterprise bean que deseé agregar a la especificación EJB 2,0, necesita una entrada de descriptor de distribuciones en formato XML. Este formato no era obligatorio en la especificación EJB 1.1, pero casi todos los fabricantes de servidores de aplicaciones lo adoptaron en sus descriptores de distribución de beans 1.1. Cuando se utilizan los asistentes de JBuilder para crear enterprise beans también se crean descriptores de distribución en formato XML.

Cuando se compila el proyecto, JBuilder crea un archivo JAR a partir del nombre de la configuración y lo muestra como nodo bajo el módulo en el panel de proyecto.

También es posible crear el archivo JAR sin compilar todo el proyecto. Haga clic con el botón derecho del ratón en el nodo del módulo EJB del panel de proyecto y elija Ejecutar Make para compilar el nodo del módulo EJB. Si desea modificar las propiedades de generación antes de seleccionar Ejecutar Make, elija el elemento Propiedades del mismo menú emergente y efectúe las modificaciones deseadas en la pestaña Generar, en el cuadro

de diálogo Propiedades. A continuación, elija Ejecutar Make para generar el archivo JAR.

El archivo JAR contiene los descriptores de distribución para el servidor de aplicaciones de destino. Cada descriptor de distribución es un archivo XML, excepto en el caso de WebSphere 3.5, que utiliza un archivo .ser para cada bean. Cada archivo JAR puede contener uno o varios descriptores de distribución.

JBuilder escogerá como destino un servidor de aplicaciones entre una gran variedad de los mismos. El servidor de aplicaciones que se utiliza como destino determina el número de descriptores de distribución del archivo JAR generado. Todos los archivos JAR tienen un ejb-jar.xml (excepto aquellos que tienen como destino WebSphere 3.5), el cual describe los atributos de distribución de los beans del módulo que comparten todos los servidores de aplicaciones. ejb-jar.xml es el descriptor de distribución acorde con EJB 1.1 o EJB 2.0.

La información de los módulos EJB 2.0 relacionada con el fabricante se almacena en el archivo ejb-borland.xml, aunque el servidor de aplicaciones no sea Borland. Para los módulos EJB 1.1, el archivo se llama ejb-inprise.xml. Al compilar, los archivos XML adicionales del fabricante se generan a partir de esta información. También se generan si hace clic en la pestaña Fuente del Editor de descriptor de distribución. Si el servidor de aplicaciones de destino es WebSphere 3.5, el archivo JAR que se genera contiene un archivo .ser para cada bean.

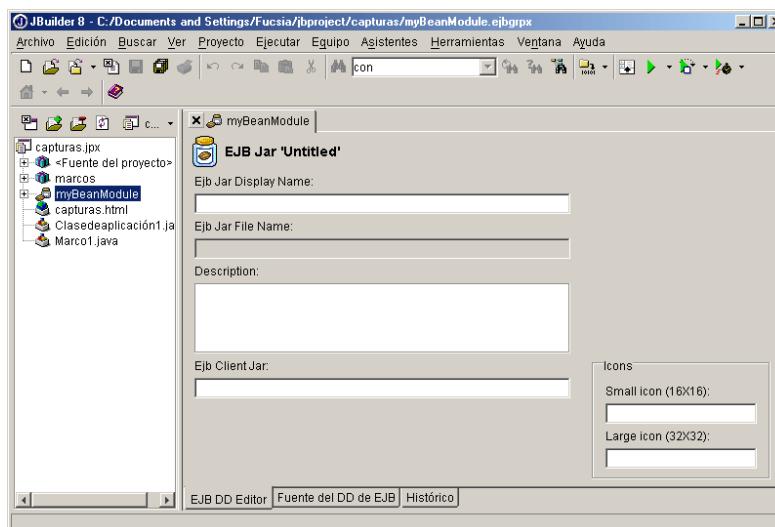
## Modificación del descriptor de distribución

---

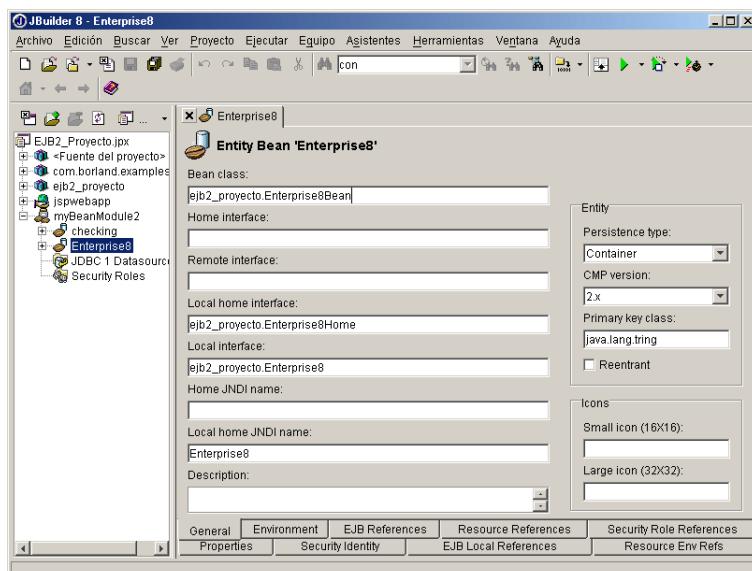
El editor del descriptor de distribución de JBuilder permite modificar los descriptores de distribución. No obstante, se pueden utilizar otras herramientas para modificar el descriptor de distribución.

Para mostrar el Editor de descriptor de distribución haga doble clic en el módulo EJB, en el panel de proyecto, y pulse la pestaña Editor DD de EJB,

en la parte inferior de la ficha de contenido. Aparece el Editor del descriptor de distribución.



Para visualizar información sobre un enterprise bean en el Editor de descriptor de distribución, abra el módulo EJB en el panel de proyecto haciendo clic en el ícono de la izquierda, contiguo a su nombre. Aparece una lista de los beans contenidos en el módulo. Haga doble clic en el nombre del enterprise bean que desea modificar. Cuando hay un bean seleccionado, en el Editor de descriptor de distribución aparecen varias pestañas más. Estas pestañas conducen a paneles en los que se puede modificar la información del descriptor de distribución.



Si desea información más detallada sobre el editor del descriptor de distribución, consulte el [Capítulo 13, “El editor de descriptor de distribución”](#).

## Comprobar descriptores

---

Después de modificar el descriptor puede comprobar el archivo para asegurarse de que la información es correcta, de que existen los archivos de clase de beans necesarios, etc. Dicha verificación ha de llevarse a cabo en los descriptores de distribución `ejb-jar.xml` y `ejb-borland.xml`. Utilicelo sólo si el servidor de aplicaciones de destino es Borland.

Para comprobar la información del descriptor, pulse el módulo con el botón derecho del ratón en el panel de proyecto y seleccione Verificar.

Verify realiza las operaciones siguientes:

- Comprueba que el descriptor cumple la especificación EJB 1.1 ó 2.0, según el tipo del módulo EJB.
- Comprueba que las clases a las que hacen referencia los descriptores de distribución cumplen la especificación EJB 1.1 ó 2.0, según el tipo del módulo EJB.

Si falla la verificación, aparece un panel de registro con mensajes que describen los fallos.

# 11

## Ejecución y prueba de enterprise beans

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

Tras haber creado un enterprise bean, ya puede ejecutarlo. El método más rápido de llevarlo a cabo consiste en pulsar con el botón derecho del ratón sobre el módulo EJB o el archivo JAR que contiene el módulo y, a continuación, seleccionar Ejecutar utilizando la configuración por defecto o Depurar utilizando la configuración por defecto, en el menú contextual. O bien, puede crear una configuración de ejecución para el bean y seleccionar Ejecutar utilizando <configuración de ejecución> o Depurar utilizando <configuración de ejecución> en el menú contextual. Esto hace que se inicie el contenedor para el servidor de aplicaciones actualmente seleccionado utilizando el JAR de este módulo EJB. Tenga paciencia, ya que este proceso es lento.

Puede ver los progresos del proceso de inicio en la ventana de mensajes. Aquí aparecerán todos los errores que se produzcan. Si desea ejecutar varios archivos JAR en el servidor de aplicaciones actual, seleccione varios módulos EJB.

### Nota para usuarios de WebLogic 7.x

Si el servidor de aplicaciones de destino es WebLogic 7.x, al pulsar con el botón secundario del ratón sobre el archivo JAR o el módulo EJB del panel de proyecto, no aparecen elementos de menú contextual para ejecutar o depurar. Durante el inicio, la distribución para WebLogic 7.x no está disponible.

La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

### Nota para usuarios de WebSphere 4.0 Advanced Edition

Debe crear un grupo EAR que contenga los beans antes de distribuir. Si desea obtener más información sobre los grupos de EAR, consulte “[Creación de archivos EAR](#)” en la página 12-6.

### Nota para usuarios de WebSphere

Después de seleccionar Ejecutar o Depurar en el menú contextual, debe seleccionar un comando del menú Distribuir con el objeto de distribuir los beans. Consulte “[Distribución en tiempo real en un servidor de aplicaciones](#)” en la página 12-11 para obtener más información.

La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

### Notas para usuarios de iPlanet

Antes de comenzar a ejecutar o depurar una aplicación, inicie el servidor iPlanet fuera de JBuilder..

Si elige ejecutar o depurar un enterprise bean con iPlanet como servidor de aplicaciones de destino, el proceso de inicio de iPlanet es más complicado. Aparecen más cuadros de mensajes. JBuilder debe detener los procesos KJS de iPlanet que se ejecuten fuera de JBuilder, distribuir el recopilatorio seleccionado, e iniciar un solo proceso KJS dentro de JBuilder. En cada uno de estos pasos aparece un mensaje. Al igual que sucede con otros servidores, puede ver los progresos del proceso de inicio en la ventana de mensajes. No es posible utilizar JBuilder hasta que no aparece el mensaje que indica que el motor está preparado. Una vez que aparece este mensaje, puede retroceder en el panel de mensajes con el fin de comprobar que no se han producido errores.

Si desea detener el servidor iPlanet, pulse el botón rojo de parada de la parte inferior derecha del panel de mensajes. En ese momento, los procesos KJS normales se reiniciarán fuera de JBuilder.

Si se produce un error en iPlanet, debe utilizar la herramienta de administración de iPlanet para apagar y reiniciar el servidor. Puede acceder a ella desde el menú Herramientas de JBuilder, si ha seleccionado la opción Añadir herramientas iPlanet al menú Herramientas al configurar iPlanet para JBuilder con Herramientas | Configurar servidores.

Para lanzar Vista web con iPlanet, pare el servidor web fuera de JBuilder, lance el servidor dentro de JBuilder y reinicie el servidor web una vez que vea el mensaje MOTOR PREPARADO en JBuilder.

Al detener el servidor iPlanet, el archivo JAR que se está ejecutando no se distribuye. Para volver a distribuirlo, reinicie el servidor y distribuya el archivo JAR.

# Comprobación del bean

---

Si desea probar el bean, puede utilizar el Asistente para cliente de prueba EJB con el fin de generar clientes de prueba que realicen llamadas al bean. El Asistente para cliente de prueba EJB puede generar tres tipos diferentes de clientes de prueba:

- Una aplicación cliente de prueba EJB, que llama a los métodos del bean.
- Un test JUnit hace uso del marco de trabajo de JUnit. Se ejecuta como cliente en una máquina virtual diferente del EJB, pasando parámetros por valor.
- Un test Cactus JUnit se ejecuta en el servidor, por lo que puede acceder a los recursos del servidor y hacer llamadas locales al EJB desde una aplicación web.

## Selección del tipo de cliente de prueba

---

Con tres tipos diferentes de clientes de prueba entre los que elegir, puede resultar difícil decidir cuál es el adecuado. A continuación, se ofrece un análisis de las ventajas de cada uno, con el objeto de ayudarle en su decisión.

- El cliente de prueba EJB de tipo Aplicación es el más fácil de instalar. Funciona como una aplicación y se conecta con el servidor. Sin embargo, la validación de los resultados de la prueba se debe realizar manualmente, ya que no se utiliza el entorno de pruebas de JUnit .
- El uso del cliente de prueba de tipo JUnit requiere un conocimiento del entorno JUnit. Este tipo de cliente de prueba está bastante automatizado y se puede ejecutar integrado en un conjunto de tests. El uso del entorno JUnit permite comprobar automáticamente los resultados y disponer de una buena capacidad de generación de informes. No es tan potente como el tipo Cactus, ya que ejecuta el test como un cliente.
- El cliente de prueba de tipo Cactus es el más completo de los tres tipos. Sin embargo, necesita Cactus y el entorno JUnit, lo que hace que su configuración sea más compleja que en los otros tipos. Las pruebas de Cactus se ejecutan en el servidor, de modo que puedan acceder a sus recursos.

## Las aplicaciones cliente de prueba

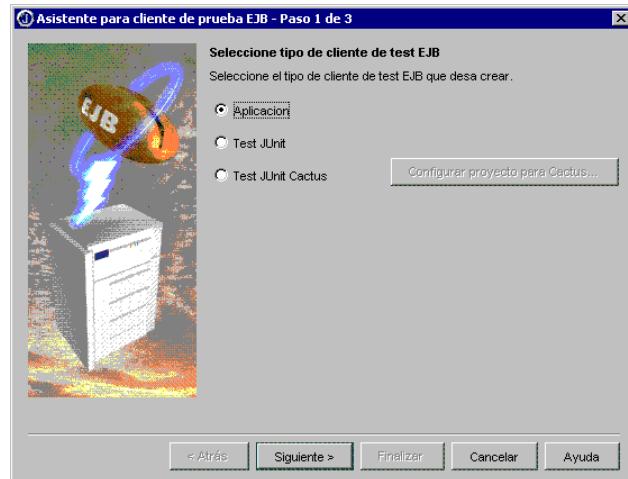
Este apartado explica cómo utilizar el Asistente para cliente de prueba EJB con el fin de crear una aplicación cliente de prueba sencilla, y cómo utilizar esa aplicación para probar el enterprise bean.

### Creación de aplicaciones cliente de prueba

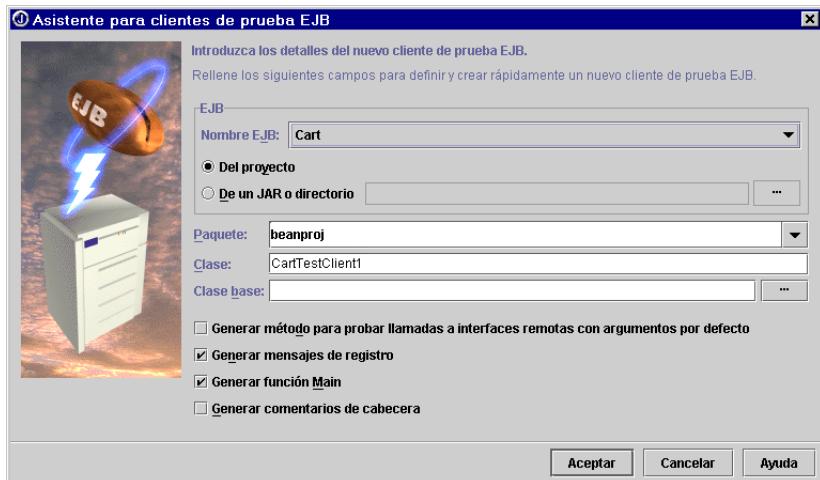
JBuilder puede ayudarle a crear una aplicación cliente de prueba que realice llamadas al bean que ha creado.

Para crear una aplicación cliente de prueba:

- 1 Abra el proyecto que contiene el módulo EJB del enterprise bean.
- 2 Si desea abrir el Asistente para cliente de prueba EJB, seleccione Archivo | Nuevo, haga clic en la pestaña Enterprise y haga doble clic en el ícono del asistente.



**3 Selecione el tipo de cliente de prueba Aplicación y pulse Siguiente.**



**4 Selecione el bean para el que desea crear un cliente mediante una de las opciones Nombre EJB y, a continuación, especifique el bean:**

- Marque la opción Del proyecto, si el bean está en el proyecto actual, y especifique el bean seleccionándolo en la lista desplegable.
- Seleccione De un JAR o directorio, si el bean no está en el proyecto actual pero sí en un archivo JAR o en un directorio. Utilice el botón de puntos suspensivos para buscar la ubicación del archivo JAR y seleccionarlo y, a continuación, utilice la lista desplegable para seleccionar el bean que deseé.

**Nota**

En estas listas sólo hay EJB con interfaces remotas, ya que no se puede acceder a los enterprise beans con interfaces locales desde una aplicación cliente, sólo se puede acceder desde otro bean o un componente web.

**5 Selecione el nombre del paquete de la lista. El valor por defecto es el paquete actual.**

**6 Escriba un nombre para la clase del cliente de prueba o acepte el nombre por defecto.**

**7 Selecione las opciones deseadas:**

- Generar método para probar llamadas a interfaces remotas con argumentos por defecto

Añade un método `testRemoteCallsWithDefaultArguments()` que comprueba las llamadas a una interfaz remota con valores de argumentos por defecto. Por ejemplo, el argumento por defecto para una cadena es "", el argumento por defecto para un int es 0, y así sucesivamente.

- Generar mensajes de registro

Añade código que muestra mensajes que informan sobre el estado del bean mientras se ejecuta el cliente. Por ejemplo, se muestra un mensaje cuando comienza la inicialización del bean y otro cuando termina. Esta opción también genera englobadores para todos los métodos declarados en las interfaces base y remota y en las funciones de inicialización. Por último, los mensajes informan de cuánto tiempo tarda en completarse cada método.

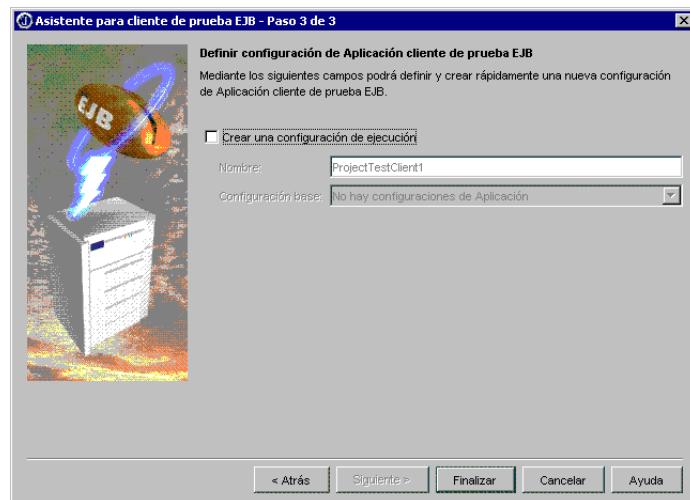
- Generar función Main

Añade la función main al cliente.

- Generar comentarios de cabecera

Añade comentarios de cabecera JavaDoc al cliente que se pueden utilizar para introducir datos tales como título, autor, etc.

**8** Pulse Siguiente.



**9** Marque la casilla de selección Crear una configuración de ejecución y suministre la información necesaria para crear una configuración de ejecución. JBuilder sugiere un nombre por defecto que se puede cambiar por cualquier otro.

**10** Elija Finalizar.

El Asistente para cliente de prueba EJB genera un cliente de prueba que crea una referencia al enterprise bean.

Si se ha seleccionado la opción Generar mensajes de registro, el asistente declara e implementa un método de llamada para cada uno de los métodos declarados en la interfaz remota del bean. Estos métodos

comunican si han conseguido llamar al método remoto y cuánto ha tardado éste en ejecutarse.

Hay muchas maneras de utilizar la aplicación cliente de prueba que se ha generado. Si ha añadido una función `main()` a la aplicación cliente de prueba, puede escribir el código para llamar a los métodos del enterprise bean de la función `main()`. Esto se consigue llamando primero a un método `create` o bien a un método `find`, y, si se devuelve una referencia remota, utilizando esta referencia para llamar a los métodos empresariales del bean. O bien, ya que el asistente ha declarado un objeto `client` en la función `main()`, puede utilizarse el objeto `client` para llamar a los métodos declarados en el cliente de prueba que llama a los métodos remotos del bean.

Si ha seleccionado la opción Generar método para comprobar las llamadas a la interfaz de remota con argumentos por defecto, la clase de cliente cuenta ahora con un método `testRemoteCallsWithDefaultArguments()`. Si ha seleccionado la opción Generar mensajes de registro, este método llama a los contenedores de métodos remotos que se generaron con esta opción. Para comprobar cada método remoto, puede llamar al método `testRemoteCallsWithDefaultArguments()` después de crear una referencia a la interfaz remota en el método `create()` de la clase de cliente o en uno de sus métodos `findByXXX()`.

Si no ha seleccionado esta opción, el método `testRemoteCallsWithDefaultArguments()` requiere que se pase una interfaz remota como un parámetro. Debe crear entonces una referencia a la interfaz remota en el método `create()` de la referencia a la interfaz local o en uno de sus métodos `findByXXX()`. A continuación, añada el código a la clase cliente para llamar al método `testRemoteCallsWithDefaultArguments()`, pasándole la referencia remota como un parámetro.

Si prefiere escribir el código de llamada a los métodos empresariales desde otra clase, puede crear y utilizar una instancia de la aplicación cliente de prueba. Consulte “[Uso de la aplicación cliente de prueba](#)” en la [página 11-7](#).

Compile la aplicación cliente de prueba.

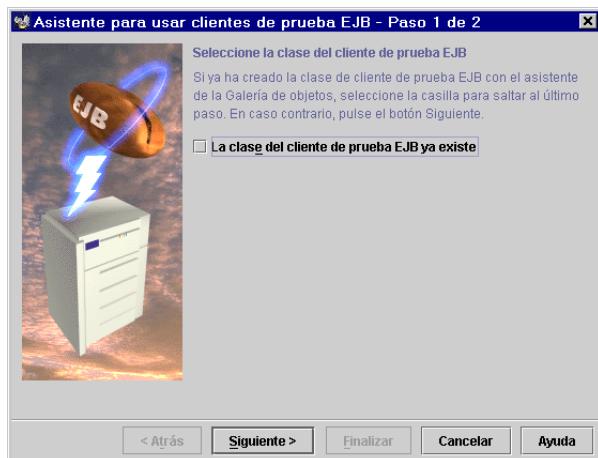
## Uso de la aplicación cliente de prueba

---

Es posible añadir rápidamente una declaración de una clase de cliente de prueba a cualquier clase.

- 1 Abra en el editor la clase en la que desea que aparezca la declaración.

**2 Elija Asistentes | EJB | Usar cliente de prueba EJB.**



- 3** Si ya se ha creado el cliente de prueba, active la opción La clase del cliente de prueba EJB ya existe.  
Si no se ha activado esta opción, cuando se pulsa Siguiente se inicia el Asistente para clientes de prueba EJB. Cuando concluye la creación, se vuelve al Asistente para usar cliente de prueba EJB.

**4** Pulse Siguiente para pasar al Paso 2.



- 5** En el campo Clase, busque la clase de cliente de prueba que desea utilizar.  
**6** En el campo Campo, escriba un nombre para la variable que contendrá una instancia de la clase del cliente de prueba o acepte el valor por defecto que le sugiera el asistente.  
**7** Elija Finalizar.

El asistente añade a la clase una declaración de la aplicación de cliente de prueba especificada como ésta, por ejemplo:

```
EmployeeTestClient1 employeeTestClient1 = new EmployeeTestClient1();
```

Ahora se puede llamar los métodos declarados en la aplicación cliente de prueba.

## Uso de la aplicación cliente de prueba con el enterprise bean

---

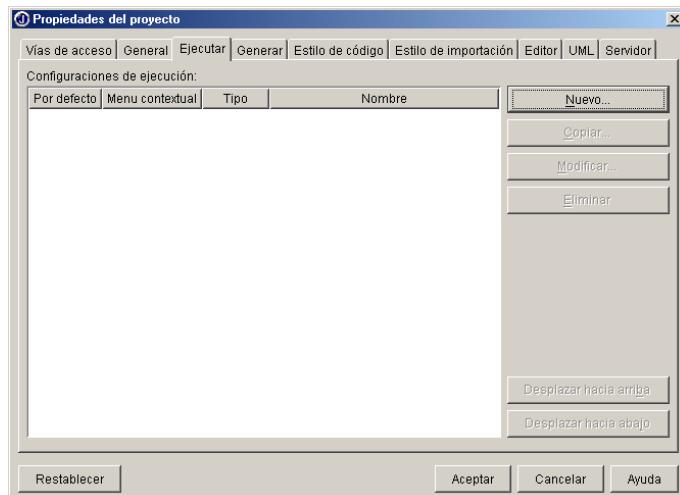
Después de crear la aplicación cliente de prueba se puede iniciar el contenedor y ejecutar la aplicación. Cree dos configuraciones para la ejecución: Servidor y cliente.

### Creación de una configuración de ejecución del servidor

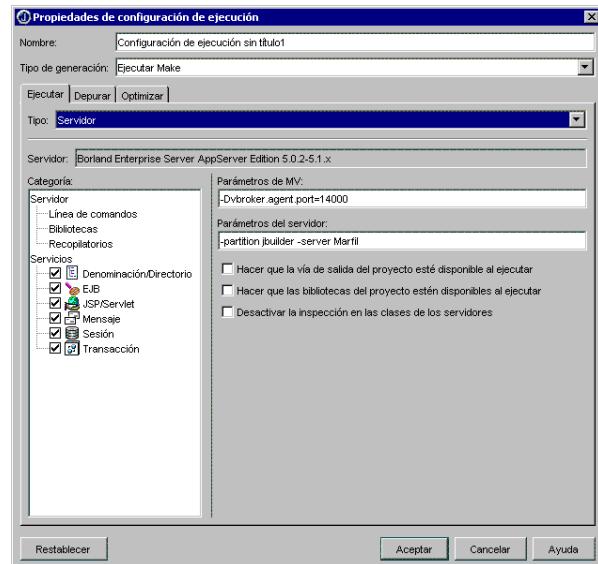
Independientemente del tipo de cliente de prueba EJB que utilice, deberá crear una configuración de ejecución del Servidor.

Para crear una configuración de servidor:

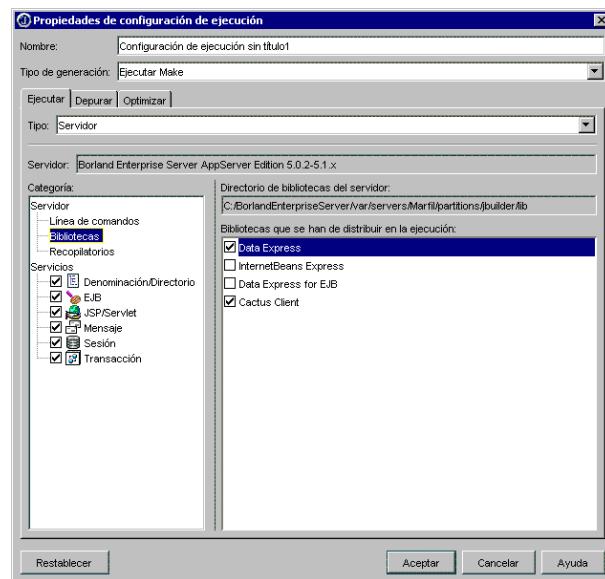
- 1 Seleccione Ejecutar | Configuraciones.



- 2 Pulse el botón Nuevo.
- 3 Seleccione Servidor en la lista desplegable Tipo.

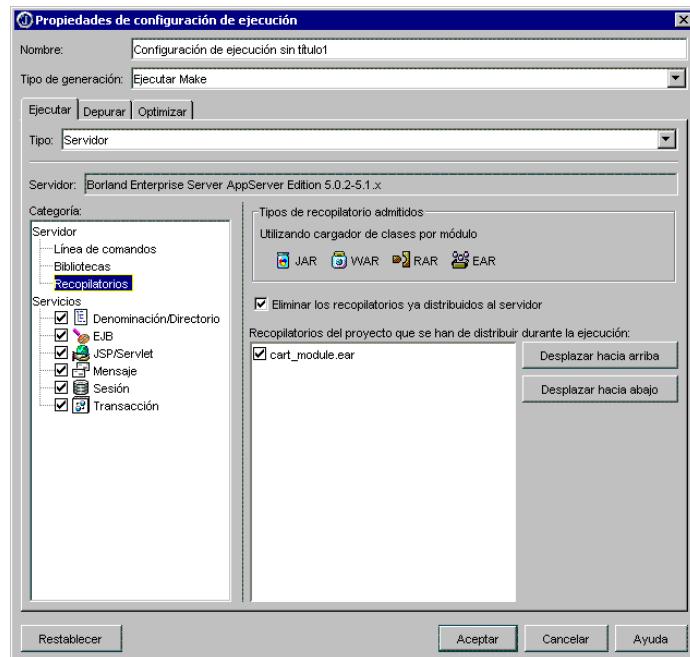


- 4 En el campo Nombre, escriba Server.
- 5 Rellene los campos Parámetros de la MV y Parámetros del servidor, necesarios para ejecutar el servidor. Si ha seleccionado un servidor de aplicaciones de destino tal y como se describe en "[Selección de un servidor](#)" en la página 5-9, valores por defecto ya están en su sitio.
- 6 Si Borland Enterprise Server es el servidor de aplicaciones destino, pulse el nodo Bibliotecas en la lista Categoría. (Esta opción no se encuentra disponible en otros servidores.)



Seleccione las bibliotecas que desea distribuir en la lista Bibliotecas que se han de distribuir durante la ejecución.

- 7 Si Borland Enterprise Server es el servidor de aplicaciones destino, pulse el nodo Recopilatorios en la lista Categoría. (Esta opción no se encuentra disponible en otros servidores.)



- 8 Seleccione en la lista de recopilatorios el archivo JAR que contiene los beans que desea probar. Los recopilatorios enumerados se recuperan de los módulos EJB y de los grupos EAR del proyecto.

La lista de los recopilatos no está disponible en todas las versiones del servidor WebSphere y WebLogic Server 7.x, ya que estos servidores no admiten la distribución una vez que se han iniciado.

En el caso de WebLogic Server 6.x, los JAR se copian en el directorio <inicio WLServer6.x>\config\<nombre dominio>\applications.

Si el cliente de prueba EJB es un cliente Cactus, puede que necesite seleccionar un archivo WAR o un archivo EAR en ese directorio.

Consulte “[Configuración del proyecto para probar un EJB mediante Cactus](#)” en la página 11-17 para obtener más información.

- 9 Haga clic sobre Aceptar dos veces.

## Ejecución de la aplicación cliente de prueba EJB

Para los usuarios de Borland Enterprise Server 5.0.2 -5.1.x: ahora debe iniciar el agente de gestión de Borland Enterprise Server. Seleccione Herramientas | Borland Enterprise Server | Agente de gestión.

Ahora se puede iniciar el contenedor. Seleccione la configuración de ejecución de servidor de la lista desplegable contigua al botón Ejecutar de la barra de herramientas JBuilder.



Se inicia el contenedor. Tenga paciencia, ya que este proceso es lento. Puede ver los progresos del proceso de inicio en la ventana de mensajes. Aquí aparecerán todos los errores que se produzcan.

Si no necesita, distribuya los recopilatorios que desea probar al servidor.

A continuación, elija la configuración de ejecución cliente con el fin de ejecutar la aplicación cliente. El mensaje que aparece en el panel informa del éxito o el fracaso de la ejecución de la aplicación cliente.

Los enterprise beans y los clientes se ejecutan igual que cualquier otro código Java con JBuilder. Para obtener más información acerca de la depuración, consulte “Depuración de programas en Java” en *Creación de aplicaciones con JBuilder*. Si desea más detalles acerca de la depuración remota de aplicaciones de WebSphere 3.5 e iPlanet 6.x+, consulte “Preparación para la depuración remota de aplicaciones WebSphere” en la página 11-26 y “Preparación para la depuración remota de aplicaciones iPlanet” en la página 11-30.

## Los tests JUnit

Este apartado explica cómo utilizar el Asistente para cliente de prueba EJB con el fin de crear un test JUnit y cómo ejecutar la prueba.

### Creación de un test 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. El Asistente

para cliente de prueba EJB puede generar un test JUnit (una clase que extiende `TestCase`) para probar un EJB.

Para crear y ejecutar un test JUnit de un EJB, necesitará lo siguiente:

- Un servidor correctamente configurado que admita servicios EJB.
- Un proyecto con el servidor correcto seleccionado en la ficha Servidor del cuadro de diálogo Propiedades de proyecto.
- Un EJB, incluido en un Grupo EJB vacío.
- Un cliente de prueba EJB para JUnit, el cual se puede crear mediante el Asistente para cliente de prueba EJB.
- Una configuración de ejecución de tipo Servidor que utilice los parámetros correctos del servidor. Consulte “[Creación de una configuración de ejecución del servidor](#)” en la página 11-9.
- Una configuración de ejecución de tipo Test. El método más simple para crear esa configuración consiste en activar la opción Crear una configuración de ejecución, que aparece en el último paso del Asistente para cliente de prueba EJB, cuando se está creando el cliente de prueba EJB para JUnit. De este modo, se garantiza que la configuración de ejecución de tipo Test presente los valores correctos.

**Nota** Si está ejecutando un cliente de prueba EJB para JUnit mediante Borland Enterprise Server, asigne a los parámetros de la MV para la configuración de ejecución de tipo Test el valor `-Dvbroker.agent.port=<puerto de agente inteligente>` (donde `<puerto de agente inteligente>` es el número de puerto de la máquina para el agente inteligente).

### Consulte

- [Capítulo 5, “Configuración del servidor de aplicaciones de destino”](#).
- [“Creación de una configuración de ejecución del servidor” en la página 11-9](#).
- “Test de módulos” en *Creación de aplicaciones con JBuilder*.
- “Definición de las configuraciones de ejecución” en *Creación de aplicaciones con JBuilder*.

## Creación de un test JUnit mediante el Asistente para cliente de prueba EJB

El Asistente para cliente de prueba EJB puede generar tres tipos diferentes de clientes de prueba para un EJB. Uno de ellos es un cliente de test JUnit estándar. Otro tipo de cliente de prueba JUnit es el cliente de test Cactus JUnit, el cual se describe más adelante. Para generar un cliente de test JUnit estándar para un EJB:

- 1 Seleccione Archivo | Nuevo.

- 2 En la ficha Enterprise de la galería de objetos, seleccione Cliente de prueba EJB. Pulse Aceptar. Se abre el Asistente para cliente de prueba EJB.
- 3 Seleccione el botón de radio Test JUnit.
- 4 Haga clic en Siguiente para avanzar en el Asistente para cliente de prueba EJB.
- 5 Seleccione el bean para el que desea crear un cliente mediante una de las opciones Nombre EJB y, a continuación, especifique el bean:
  - Marque la opción Del proyecto, si el bean está en el proyecto actual, y especifique el bean seleccionándolo en la lista desplegable.
  - Seleccione De un JAR o directorio, si el bean no está en el proyecto actual pero sí en un archivo JAR o en un directorio. Utilice el botón de puntos suspensivos para buscar la ubicación del archivo JAR y seleccionarlo y, a continuación, utilice la lista desplegable para seleccionar el bean que deseé.
- 6 Especifique el Paquete, el Nombre de clase y la Clase base del nuevo cliente de prueba JUnit. No necesita cambiar los valores por defecto si no lo desea. No debería cambiar la clase base a menos que haya otra clase que desee extender en lugar de `junit.framework.TestCase`.
- 7 Utilice las siguientes casillas de verificación para indicar al asistente que genere código optativo:
  - Generar método para probar llamadas a interfaces remotas con argumentos por defecto  
Añade un método `testRemoteCallsWithDefaultArguments()` que comprueba las llamadas a una interfaz remota con valores de argumentos por defecto. Por ejemplo, el argumento por defecto para una cadena es "", el argumento por defecto para un int es 0, y así sucesivamente.
  - Generar mensajes de registro  
Añade código que muestra mensajes que informan sobre el estado del bean mientras se ejecuta el cliente. Por ejemplo, se muestra un mensaje cuando comienza la inicialización del bean y otro cuando termina. Esta opción también genera englobadores para todos los métodos declarados en las interfaces base y remota y en las funciones de inicialización. Por último, los mensajes informan de cuánto tiempo tarda en completarse cada método.
  - Generar comentarios de cabecera  
Añade comentarios de cabecera JavaDoc al cliente que se pueden utilizar para introducir datos tales como título, autor, etc.

- 8** Haga clic en Siguiente para avanzar en el Asistente para cliente de prueba EJB.
- 9** Active Crear una configuración de ejecución si desea crear una. Esto es necesario si no dispone aún de una configuración de ejecución de tipo Test para el proyecto. Si va a crear una configuración de ejecución aquí, especifique el Nombre. También puede especificar una Configuración base. De este modo, se copian los valores de la configuración especificada, los cuales se pueden modificar posteriormente mediante el cuadro de diálogo Propiedades de configuración de ejecución.

- 10** Pulse Finalizar para generar el cliente de prueba JUnit.

Ahora que ya ha generado el cliente de prueba JUnit mediante el Asistente para cliente de prueba EJB, debe implementarlo. Deberá hacer lo siguiente:

- 1** Crear una referencia de interfaz remota al EJB mediante una llamada al método `create()` generado por el asistente. Existe un comentario `@todo` en el código generado que indica dónde añadirla (en el método `setUp()`). Puede ser tan simple como añadir la siguiente línea de código (donde `<beanname>` es la variable `private` generada por el asistente para hacer referencia al EJB):

```
<beanname> = create();
```

- 2** Añada cualquier otro método de prueba que desee escribir.

Después de haber creado e implementado el cliente de prueba JUnit, ya se puede ejecutar y probar el EJB.

### Consulte

- “Ejecución del test JUnit” en la página 11-15.
- Capítulo 5, “Configuración del servidor de aplicaciones de destino”.
- “Creación de una configuración de ejecución del servidor” en la página 11-9.
- “Definición de las configuraciones de ejecución” en *Creación de aplicaciones con JBuilder*.

## Ejecución del test JUnit

---

Ejecutar un test JUnit para un EJB desde el IDE de JBuilder es similar a ejecutar otros tipos de pruebas JUnit. Se dispone de los mismo ejecutores de pruebas, con los que es posible interactuar de la misma forma. La diferencia principal entre probar un EJB y ejecutar cualquier otro test JUnit es que para un EJB es necesario iniciar primero el servidor. Para ejecutar el cliente de prueba EJB para JUnit:

- 1** Inicie el servidor con la configuración de ejecución Servidor.

- 2 Si lo necesita, distribuya los recopilatorios que desea probar al servidor.
- 3 Haga clic con el botón derecho del ratón en el archivo test JUnit, del panel de proyecto.
- 4 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.

### Consulte

- [Capítulo 5, “Configuración del servidor de aplicaciones de destino”.](#)
- [“Creación de una configuración de ejecución del servidor” en la página 11-9.](#)
- “Ejecución de tests”, en *Creación de aplicaciones con JBuilder*.
- “Definición de las configuraciones de ejecución” en *Creación de aplicaciones con JBuilder*.

## Los tests Cactus JUnit

---

Este apartado explica cómo configurar el proyecto para probar un enterprise bean con Cactus, cómo utilizar el Asistente para cliente de prueba EJB con el fin de crear una prueba Cactus JUnit y cómo ejecutarla.

### Creación de un test Cactus JUnit

---

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. Por ejemplo, esto le permitirá probar un EJB con interfaces locales. El Asistente para cliente de prueba EJB puede generar un test Cactus JUnit para probar un EJB en el servidor.

Para crear y ejecutar un test Cactus para un EJB, necesitará lo siguiente:

- Un servidor correctamente configurado que admita servicios EJB.
- Un proyecto con el servidor correcto seleccionado en la ficha Servidor del cuadro de diálogo Propiedades de proyecto.
- Una WebApp.
- Un EJB, incluido en un Grupo EJB vacío.
- Un cliente de prueba Cactus para EJB, que se puede crear mediante el Asistente para cliente de prueba EJB.
- Una configuración de ejecución de tipo Servidor que utilice los parámetros correctos del servidor. Consulte [“Creación de una](#)

[“configuración de ejecución del servidor” en la página 11-9](#) para obtener más información.

- Una configuración de ejecución de tipo Test con los recopilatorios correctos seleccionados para su redistribución.
- Una configuración Cactus para el proyecto que especifique las configuraciones correctas de ejecución para los tipos Servidor y Test. Esta configuración se crea mediante el Asistente para la configuración de Cactus.

Si va a ejecutar pruebas de Cactus para cualquier EJB utilizando Borland Enterprise Server o pruebas de Cactus para un EJB con referencias locales utilizando Weblogic, también necesitará:

- Un recopilatorio Enterprise (EAR) que contenga tanto el archivo JAR del EJB como el archivo WAR de la WebApp.
- La configuración de ejecución del tipo Test debe tener el archivo EAR seleccionado como un recopilatorio para redistribuir.
- La configuración de Cactus para el proyecto debe especificar el archivo EAR como un recopilatorio para redistribuir.

### Consulte

- [Capítulo 5, “Configuración del servidor de aplicaciones de destino”.](#)
- [“Creación de una configuración de ejecución del servidor” en la página 11-9.](#)
- “Test de módulos” en *Creación de aplicaciones con JBuilder*.
- “Las WebApps y los archivos WAR” en la *Guía del desarrollador de aplicaciones Web*.
- “Definición de las configuraciones de ejecución” en *Creación de aplicaciones con JBuilder*.

### Configuración del proyecto para probar un EJB mediante Cactus

JBuilder proporciona el Asistente para la configuración de proyectos para Cactus con el fin de ayudar a configurar el proyecto para 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.

### Requisitos previos para la configuración de Cactus

Para poder configurar un proyecto de modo que pueda realizar pruebas de un EJB mediante Cactus, necesitará lo siguiente:

- Un servidor correctamente configurado que admita servicios EJB. Consulte el [Capítulo 5, “Configuración del servidor de aplicaciones de destino”](#).
  - Un proyecto con el servidor correcto seleccionado en la ficha Servidor del cuadro de diálogo Propiedades de proyecto.
  - Una WebApp. Ésta se crea mediante el Asistente para aplicaciones web, disponible en la ficha Web de la galería de objetos (Archivo | Nuevo). Consulte “Las WebApps y los archivos WAR” en la *Guía del desarrollador de aplicaciones Web*
- Sugerencia** Puede ejecutar el Asistente para aplicaciones web desde dentro del Asistente para la configuración de proyectos para Cactus, si pulsa el botón Nuevo situado junto a la lista desplegable WebApp.
- El EJB que deseé probar, incluido en un Grupo EJB vacío.
  - Una configuración de ejecución de tipo Servidor que utilice el servidor correcto. Consulte “[Creación de una configuración de ejecución del servidor](#)” en la página 11-9 para obtener más información.
- Sugerencia** Puede crear una configuración de ejecución de tipo Servidor desde dentro del Asistente para la configuración de proyectos para Cactus, si pulsa el botón Nuevo situado junto a la lista desplegable de configuración de ejecución de servidor.
- Una configuración de ejecución de tipo Test. Consulte “Definición de configuraciones de ejecución” en *Creación de aplicaciones con JBuilder*.
- Sugerencia** Puede crear una configuración de ejecución de tipo Test desde dentro del Asistente para la configuración de proyectos para Cactus, si pulsa el botón Nuevo situado junto a la lista desplegable de configuración de ejecución de test.
- Si va a ejecutar pruebas de Cactus para cualquier EJB utilizando Borland Enterprise Server o pruebas de Cactus para un EJB con referencias locales utilizando Weblogic, antes de configurar el proyecto para Cactus también necesitará lo siguiente:
- Un recopilatorio Enterprise (EAR) que contenga tanto el archivo JAR del EJB como el archivo WAR de la WebApp. Este se crea mediante el Asistente para EAR, disponible en la ficha Enterprise de la galería de objetos (Archivo | Nuevo).
  - La configuración de ejecución del tipo Test debe tener el archivo EAR seleccionado como un recopilatorio para redistribuir.

### Ejecución del Asistente para la configuración de proyectos para Cactus

El Asistente para la configuración de proyectos para Cactus permite configurar el proyecto para ejecutar pruebas de Cactus desde el IDE de JBuilder. Crea el archivo de propiedades requerido por Cactus y se encarga de que los recopilatorios y otros archivos necesarios para las

pruebas se distribuyan correctamente. El Asistente para la configuración de proyectos para Cactus se abre mediante la opción de menú Asistentes | Configuración de Cactus.

Antes de ejecutar el asistente, examine los requisitos previos para la configuración de Cactus que se describen en ["Requisitos previos para la configuración de Cactus"](#) en la página 11-17. Las configuraciones necesarias para ejecución y WebApp se pueden crear desde el propio asistente. Para que el asistente pueda configurar satisfactoriamente el proyecto para pruebas de EJB, se deberán cumplir previamente el resto de los requisitos.

Para ejecutar el Asistente para la configuración de proyectos para Cactus y configurar el proyecto para Cactus:

- 1** Seleccione Asistentes | Configuración de Cactus. Se abre el Asistente para la configuración de proyectos para Cactus.

#### Sugerencia

También puede ejecutar el Asistente para la configuración de proyectos para Cactus desde el Asistente para cliente de prueba EJB.

- 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 redistribuir antes de cada test. 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.

#### Importante

Si utiliza Cactus y Borland Enterprise Server para probar un EJB al que accede una WebApp, asegúrese de que dispone de un archivo EAR que contiene tanto el archivo JAR del EJB como el archivo WAR de la WebApp. Cuando seleccione los recopilatorios para distribuir en el Asistente para la configuración de Cactus, seleccione el archivo EAR y quite los archivos JAR y WAR.

- 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.

Cuando se utiliza Cactus para probar unos EJB que hacen referencia localmente a otros EJB, se requiere una configuración específica del servidor. Los siguientes apartados analizan la configuración necesaria para Borland Enterprise Server y Weblogic (versiones 6.x a 7.0).

### **Acceso a los EJB con interfaces locales mediante Cactus y Borland Enterprise Server**

Para probar un EJB con referencias locales mediante Cactus y Borland Enterprise Server, deberá crear una entrada de referencia local al EJB tanto en el descriptor web estándar (`web.xml`) como en el específico del proveedor (`web-borland.xml`). También deberá actualizar la referencia de nombre JNDI en el cliente de prueba Cactus de modo que utilice la referencia local del EJB.

Para añadir una referencia local al EJB en `web.xml`:

- 1 Amplíe el nodo de la WebApp en el panel de proyecto.
- 2 Amplíe el nodo Descriptores de distribución.
- 3 Haga doble clic en `web.xml` para abrirlo en el Editor DD de WebApp.
- 4 Pulse sobre Referencias de EJB local en el panel de estructura.
- 5 En la ficha Referencias de EJB local, del Editor DD de WebApp, pulse el botón Añadir.
- 6 Especifique un Nombre de referencia apropiado, como `ejb/<nombre de JNDI local>`.
- 7 Seleccione el Tipo. Las opciones posibles son Sesión o Entidad.
- 8 Especifique los nombres de las clases para las interfaces base local y local.

Para añadir una referencia local al EJB en `web-borland.xml`:

- 1 Amplíe el nodo de la WebApp en el panel de proyecto.
- 2 Amplíe el nodo Descriptores de distribución.
- 3 Haga doble clic en `web-borland.xml` con el fin de abrirlo en el editor.
- 4 Añada el siguiente elemento del descriptor al código fuente XML:

```
<ejb-local-ref>
  <ejb-ref-name>ejb/Nombre de referencia</ejb-ref-name>
  <jndi-name>Nombre JNDI del EJB</jndi-name>
</ejb-local-ref>
```

El valor de `ejb-ref-name` debería coincidir con el nombre de la referencia que aparece en `web.xml`; además, el valor de `jndi-name` es el nombre JNDI local del EJB.

Para modificar el cliente de prueba Cactus:

- 1** Haga doble clic en el archivo cliente de prueba Cactus, en el panel de proyecto con el fin de abrirlo en el editor.
- 2** Localice la siguiente sentencia:

```
Object ref = ctx.lookup("<Nombre local JNDI>");
```

- 3** Modique la sentencia como se indica a continuación:

```
Object ref = ctx.lookup("java:comp/env/<Nombre de referencia local EJB>");
```

### **Acceso a los EJB con interfaces locales mediante Cactus y Weblogic 6.x/7.x**

Para probar un EJB con referencias locales mediante Cactus y Weblogic 6.x/7.x, debe asegurarse de que el archivo JAR del EJB y el archivo WAR de la WebApp se encuentran empaquetados en un recopilatorio Enterprise (EAR); éste es el recopilatorio que deberá distribuir.

Para modificar la configuración del servidor:

- 1** Seleccione Proyecto | Propiedades de proyecto.
- 2** Seleccione la ficha Servidor del cuadro de diálogo Propiedades del proyecto.
- 3** Pulse sobre el servicio JSP/Servlet y desactive Mapear las webapps del proyecto durante la ejecución. Esto hará que se desactive el comportamiento por defecto consistente en distribuir la aplicación web en formato desplegado durante el inicio del servidor.
- 4** Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

A continuación, debe crear un EAR que contenga tanto el archivo JAR del EJB como el archivo WAR de la WebApp. Para crear el EAR, utilice el Asistente para EAR, disponible en la ficha Enterprise de la galería de objetos (Archivo | Nuevo). Para obtener más información sobre la creación de EAR, consulte “[Creación de archivos EAR](#)” en la página 12-6 . Una vez creado el EAR, debe volver a generar el proyecto.

Si está utilizando Weblogic 7.x:

- 1** Inicie el servidor.
- 2** Pulse con el botón secundario del ratón sobre el grupo EAR y seleccione Opciones de distribución | Distribuir.

Si está utilizando Weblogic 6.x, deberá crear una configuración de ejecución de tipo Servidor. Para crear la configuración de ejecución:

- 1 Elija Ejecutar | Configuraciones.
- 2 Haga clic en Nuevo. Se abre el cuadro de diálogo Propiedades de configuración de ejecución.
- 3 Seleccione Servidor en la lista desplegable Tipo.
- 4 En el árbol de categorías, pulse sobre el nodo Recopilatorios.
- 5 En la lista Recopilatorios del proyecto que se han de distribuir durante la ejecución, seleccione el EAR y quite la selección de los archivos JAR y WAR.
- 6 Haga clic en Aceptar para cerrar el cuadro de diálogo.
- 7 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

Inicie el servidor con la nueva configuración de ejecución.

**Nota** En el caso de otros servidores, diríjase a la documentación del servidor para obtener instrucciones sobre cómo acceder a los EJB con referencias locales desde una aplicación web.

## Creación de un test Cactus JUnit mediante el Asistente para cliente de prueba EJB

El Asistente para cliente de prueba EJB puede generar tres tipos diferentes de clientes de prueba para un EJB. Uno de ellos es un cliente de prueba Cactus. Cactus es una extensión de JUnit que proporciona funcionalidad para pruebas del código Java en el lado del servidor. Para generar un cliente de prueba Cactus para un EJB:

- 1 Seleccione Archivo | Nuevo.
- 2 En la ficha Enterprise de la galería de objetos, seleccione Cliente de prueba EJB. Pulse Aceptar. Se abre el Asistente para cliente de prueba EJB.
- 3 Seleccione el botón de radio Cactus Test JUnit.
- 4 Si aún no ha ejecutado el Asistente para la configuración de Cactus, pulse Configurar proyecto para Cactus. Al pulsar el botón, se abre el Asistente para la configuración de Cactus. Consulte “[Ejecución del Asistente para la configuración de proyectos para Cactus](#)” en la página 11-18 para obtener más información.
- 5 Haga clic en Siguiente para avanzar en el Asistente para cliente de prueba EJB.
- 6 Seleccione el bean para el que desea crear un cliente mediante una de las opciones Nombre EJB y, a continuación, especifique el bean:

- Marque la opción Del proyecto, si el bean está en el proyecto actual, y especifique el bean seleccionándolo en la lista desplegable.
  - Seleccione De un JAR o directorio, si el bean no está en el proyecto actual pero sí en un archivo JAR o en un directorio. Utilice el botón de puntos suspensivos para buscar la ubicación del archivo JAR y seleccionarlo y, a continuación, utilice la lista desplegable para seleccionar el bean que deseé.
- 7** Especifique el Paquete, el Nombre de clase y la Clase base del nuevo cliente de prueba Cactus. No necesita cambiar los valores por defecto si no lo desea. No debería cambiar la clase base a menos que haya otra clase que desee extender en lugar de `org.apache.cactus.ServletTestCase`.
- 8** Utilice las siguientes casillas de verificación para indicar al asistente que genere código optativo:
- Generar método para probar llamadas a interfaces remotas con argumentos por defecto.  
Añade un método `testRemoteCallsWithDefaultArguments()` que comprueba las llamadas a una interfaz remota con valores de argumentos por defecto. Por ejemplo, el argumento por defecto para una cadena es "", el argumento por defecto para un int es 0, y así sucesivamente.
  - Generar mensajes de registro.  
Añade código que muestra mensajes que informan sobre el estado del bean mientras se ejecuta el cliente. Por ejemplo, se muestra un mensaje cuando comienza la inicialización del bean y otro cuando termina. Esta opción también genera englobadores para todos los métodos declarados en las interfaces base y remota y en las funciones de inicialización. Por último, los mensajes informan de cuánto tarda en completarse cada método.
  - Generar comentarios de cabecera.  
Añade comentarios de cabecera JavaDoc al cliente que se pueden utilizar para introducir datos tales como título, autor, etc.
- 9** Haga clic en Siguiente para avanzar en el Asistente para cliente de prueba EJB.
- 10** Active Crear una configuración de ejecución si desea crear una. Este paso es optativo. Si ya dispone de una configuración de ejecución de tipo Test, especificada al ejecutar el Asistente para la configuración de Cactus, no es probable que necesite crear una. De cualquier forma, si decide crear una configuración de ejecución en este paso, especifique el Nombre. También puede especificar una Configuración base. De este modo, se copian los valores de la configuración especificada, los cuales se pueden modificar posteriormente mediante el cuadro de diálogo Propiedades de configuración de ejecución.

**11** Pulse Finalizar para generar el cliente de prueba Cactus.

Ahora que ha generado el cliente de prueba Cactus mediante el Asistente para cliente de prueba EJB, debe implementarlo. Deberá hacer lo siguiente:

- 1 Crear una referencia de interfaz remota al EJB mediante una llamada al método `create()` generado por el asistente. Existe un comentario `@todo` en el código generado que indica dónde añadirla (en el método `setUp()`). Puede ser tan simple como añadir la siguiente línea de código (donde `<beanname>` es la variable `private` generada por el asistente para hacer referencia al EJB):

```
<beanname> = create();
```

- 2 Añada cualquier otro método de prueba que desee escribir.
- 3 Si está utilizando Borland Enterprise Server como servidor para probar un EJB con interfaces locales, actualice la referencia de nombre JNDI de modo que utilice la referencia local del EJB. Consulte “[Acceso a los EJB con interfaces locales mediante Cactus y Borland Enterprise Server](#)” en la página 11-20 para obtener más información.

Después de haber configurado el proyecto para Cactus y creado e implementado el cliente de prueba Cactus, ya puede ejecutarlo y probar el EJB.

### Consulte

- “[Configuración del proyecto para probar un EJB mediante Cactus](#)” en la página 11-17.
- “[Ejecución del test Cactus JUnit](#)” en la página 11-24.

## Ejecución del test Cactus JUnit

---

Ejecutar un test Cactus JUnit en el IDE de JBuilder es similar a ejecutar otros tipos de pruebas JUnit. Se dispone de los mismo ejecutores de pruebas, con los que es posible interactuar de la misma forma. La diferencia principal entre ejecutar un test Cactus y ejecutar cualquier otra prueba JUnit es que para un test Cactus es necesario iniciar primero el servidor. Para ejecutar un test Cactus:

- 1 Inicie el servidor con la configuración de ejecución Servidor.
- 2 Si lo necesita, distribuya los recopilatorios que desea probar al servidor.
- 3 Haga clic con el botón derecho del ratón en el archivo de prueba Cactus, del panel de proyecto.

- 4** 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.

**Nota** La modificación de test necesita que el la WebApp o el archivo EAR configurado para Cactus se redistribuya.

### Consulte

- “[Configuración del proyecto para probar un EJB mediante Cactus](#)” en la página 11-17.
- “Ejecución de tests”, en *Creación de aplicaciones con JBuilder*.

## Activación de la depuración de WebSphere 4.0 Advanced Edition

---

WebSphere Server 4.0 Advanced Edition 4.0 inicia otra máquina virtual de Java para depurar, por lo que debe seguir estas instrucciones con el objeto de depurar los enterprise beans, JSP o servlets:

- 1** Inicie el servidor WebSphere Server 4.0. Observe qué número de puerto utiliza el servidor.
- 2** Inicie aplicación de consola WebSphere Server 4.0 (que encontrará en <ws4\_ae home>bin\adminclient.bat si lo está ejecutando desde Windows).
- 3** Cuando aparezca la aplicación de consola, amplíe el árbol de la izquierda hasta que pueda ver el nodo Servidores de aplicaciones. Seleccione el nodo.
- 4** En el panel que aparece a la derecha de la consola, abra la pestaña Configuración de la MVJ. Pulse el botón Configuración avanzada. (Quizás necesite desplazar el panel para ver el botón.)
- 5** Ien el cuadro de diálogo que se abre, introduzca las siguientes opciones de MV:
 

```
-Xdebug -Xnoagent -Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,address=5000,suspend=n
```

- 6** Para iniciar el servidor dentro de JBuilder en modo depuración y enlazarlo a él, selección Ejecutar | Configuraciones y pulse el botón Nueva en el cuadro de diálogo que aparece. En el campo Configuración, especifique un nombre para el nuevo servidor de configuración de ejecución para poder depurar en el servidor de aplicaciones.
- 7** En la ficha Ejecutar, seleccione Servidor en la lista desplegable Tipo.
- 8** En el campo Dirección de transporte de depuración, introduzca el número que había anotado anteriormente durante el proceso de inicio del servidor.

- 9 Haga dos veces clic en Aceptar para cerrar los cuadros de diálogo.
- 10 Seleccione la configuración de ejecución que acaba de crear en la lista desplegable del ícono Depurar, en el menú de herramientas de JBuilder. El servidor se inicia el modo depuración y el proceso de depuración se enlaza a él. Ahora puede depurar el enterprise bean o la aplicación web y podrá parar en el punto de interrupción definido en el bean JSP o servlet.

## Preparación para la depuración remota de aplicaciones WebSphere

---

Si el servidor de aplicaciones de destino es un servidor WebSphere, debe llevar a cabo una serie de pasos adicionales para poder activar la depuración remota del enterprise bean.

### WebSphere Server 3.5

---

Si va a utilizar WebSphere Server 3.5, siga estos pasos con el fin de activar la depuración remota en Windows:

- 1 Copie dt\_shem.dll y dt\_socket.dll de `WEBSPHERE_HOME/jdk/jre/bin` a `WEBSPHERE_HOME/jdk/bin`.
- 2 Modifique el script adminserver en `WEBSPHERE_HOME/bin/debug` y añada los siguientes parámetros de depuración remota en la línea de comandos Java:  

```
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdwp:transport=dt_socket,server=y,address=5000,suspend=n
```
- 3 En JBuilder, seleccione Proyecto | Propiedades de proyecto y abra la pestaña Depurar.
- 4 Marque las opciones Activar depuración remota y Acoplarse, y pulse Aceptar.

Cuando esté preparado para depurar, acópelo al proceso remoto seleccionando Ejecutar | Depurar proyecto.

### WebSphere Single Server 4.0

---

Si está utilizando WebSphere Single Server 4.0, siga estas indicaciones para activar la depuración remota:

- 1 Inicie el servidor con la opción `-script`:

```
WEBSPHERE_HOME/bin/startserver -script
```

Este comando escribe un script denominado launch en el directorio WEBSPHERE\_HOME/bin.

- 2** Modifique este script y añada los siguientes parámetros de depuración remota a la línea de comandos Java:

```
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdwp:transport=dt_socket,server=y,address=5000,suspend=n
```

- 3** En JBuilder, seleccione Proyecto | Propiedades de proyecto. Haga clic en la pestaña Depurar.
- 4** Marque las opciones Activar depuración remota y Acoplarse, y pulse Aceptar.

Cuando esté preparado para depurar, acópelo al proceso remoto seleccionando Ejecutar | Depurar proyecto.

## WebSphere Server Advanced Edition 4.0

---

WebSphere Server 4.0 Advanced Edition inicia otra máquina virtual de Java para depurar, por lo que debe seguir estas instrucciones con el objeto de depurar los enterprise beans:

- Nota** No puede depurar JSP remotamente.
- 1** Inicie el servidor WebSphere Server 4.0.
  - 2** Inicie aplicación de consola WebSphere Server 4.0 (que encontrará en <ws4\_ae home>/bin\adminclient.bat si lo está ejecutando desde Windows).
  - 3** Cuando aparezca la aplicación de consola, amplíe el árbol de la izquierda hasta que pueda ver el nodo Servidores de aplicaciones. Seleccione el nodo.
  - 4** En el panel que aparece a la derecha de la consola, abra la pestaña Configuración de la MVJ. Pulse el botón Configuración avanzada. (Quizás necesite desplazar el panel para ver el botón.)
  - 5** Ien el cuadro de diálogo que se abre, introduzca las siguientes opciones de MV:
 

```
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdwp:transport=dt_socket,server=y,address=5000,suspend=n
```
  - 6** Si inicia el servidor fuera de JBuilder, seleccione Ejecutar | Configuraciones y pulse el botón Siguiente en el cuadro de diálogo que aparece. En el campo Configuración, especifique un nombre para el nuevo servidor de configuración de ejecución para poder depurar en el servidor de aplicaciones.
  - 7** Abra la pestaña Depurar y realice los cambios en el panel Depurar:
    - a** Active la casilla de selección Activar depuración remota.

- b** Seleccione la opción Acoplarse.
  - c** En el campo Número de puerto, introduzca el número que había anotado anteriormente durante el proceso de inicio del servidor.
  - d** Haga clic en Aceptar para cerrar todos los cuadros de diálogo.
- 8** Seleccione la configuración de ejecución que acaba de crear en la lista desplegable del ícono Depurar, en el menú de herramientas de JBuilder.

De esta forma, la acoplará al proceso del servidor de aplicaciones. Ahora ya puede distribuir los enterprise beans y detenerse en los puntos de interrupción establecidos en los beans cuando se ejecuta un cliente que llama a esos beans.

Para iniciar el servidor en modo depuración dentro de JBuilder, siga estos pasos:

- Seleccione Ejecutar | Configuraciones.
- Seleccione el nodo Comando e introduzca el número de puerto para depuración remota en el campo Dirección de transporte de depuración.

Ahora, pulsando el botón de depuración de la barra de herramientas se lanza el servidor en modo depuración y se adjunta a él, todo en una única sesión.

## Preparación para depurar las Páginas JavaServer de WebLogic de forma remota

---

Para depurar las JSP remotamente, abra el proyecto que contiene la aplicación web que desea depurar y siga estos pasos:

- 1** Abra el nodo de la aplicación web en el panel de proyecto y abra a su vez el nodo de los descriptores de distribución que se encuentra debajo del nodo de la aplicación web.
- 2** Haga doble clic en `weblogic.xml` y añada los siguientes elementos descriptores:

```
<jsp-descriptor>
  <jsp-param>
    <param-name>keepgenerated</param-name>
    <param-value>true</param-value>
  </jsp-param>
  <jsp-param>
    <param-name>debug</param-name>
    <param-value>true</param-value>
  </jsp-param>
  <jsp-param>
    <param-name>precompile</param-name>
    <param-value>true</param-value>
```

```
</jsp-param>
<jsp-param>
    <param-name>compileFlags</param-name>
    <param-value>-g</param-value>
</jsp-param>
</jsp-descriptor>
```

- 3** Vuelva a generar el proyecto y distribuya la aplicación web (como un WAR).
- 4** Lance el servidor Weblogic con los parámetros de depuración. Para hacer esto, modifique `startWebLogic.sh` en el directorio de dominio y añada los siguientes parámetros a la variable OPCIONES\_JAVA:

```
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=n
```

Estos pasos pueden realizarse en un ordenador local provisto de Weblogic instalado y configurado en JBuilder.

En el ordenador local, siga estos pasos:

- 1** Transfiera el proyecto al ordenador locan en el que desee depurar la JSP y abra el proyecto en JBuilder.
- 2** Defina el servidor destino como Tomcat 4.0 o 3.3.
- 3** Bajo la vía de acceso a archivos fuente del proyecto (normalmente `<RAÍZ_DEL_PROYECTO>/src`), cree un directorio denominado `jsp_servlet`. Si necesita utilizar una vía de acceso a archivos fuente distinta de la vía de acceso por defecto, puede realizar esta configuración en la ficha General del cuadro de diálogo Propiedades de proyecto.
- 4** Copie las JSP que desee depurar en el directorio que acaba de especificar.
- 5** Abra las JSP y defina los puntos de interrupción donde sea necesario.
- 6** Seleccione Ejecutar | Configuraciones.
- 7** Pulse Nuevo, después haga clic en la pestaña Aplicación y a continuación en la pestaña Depurar.
- 8** Marque la opción Activar depuración remota.
- 9** Introduzca el nombre del host del servidor remoto en el campo Nombre del anfitrión y asigne el valor Ninguno a Tipo de generación. Acepte todos los demás valores por defecto.
- 10** Vinculese con el servidor remoto pulsando en el icono Depurar proyecto de la barra de herramientas.
- 11** Cargue su JSP en un navegador web. El depurador se detiene en los puntos de interrupción de la JSP.

## Preparación para la depuración remota de aplicaciones iPlanet

---

Con el fin de prepararse para depurar aplicaciones iPlanet de forma remota, siga los siguientes pasos:

- 1** Seleccione Proyecto | Propiedades de proyecto.
- 2** Haga clic en la pestaña Depurar.
- 3** Marque la opción Activar depuración remota
- 4** Seleccione la opción Acoplarse.
- 5** Pulse Aceptar.

# 12

## Distribución de enterprise beans

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

Normalmente, el proceso de distribución de un enterprise bean en un servidor de aplicaciones incluye los siguientes pasos:

- 1 Creación de un archivo XML de descripción de distribución acorde con la especificación EJB 1.1 o 2.0 de Sun. WebSphere 3.5 constituye una excepción ya que es compatible con la especificación EJB 1.0 y no utiliza descriptores de distribución basados en XML.

Cuando se utiliza el asistente para EJB de JBuilder para crear beans, los descriptores de distribución se crean a la vez, automáticamente.

- 2 Edición de los descriptores de distribución (si es necesario).

Puede modificar los descriptores de distribución que Jbuilder ha creado con la ayuda del Editor del descriptor de distribución.

- 3 Creación de un archivo JAR EJB que contenga el descriptor de distribución y todas las clases necesarias para manejar el enterprise bean (clase del bean, interfaz remota/local, interfaz local/base local, stubs y esqueletos, clase de clave principal si el enterprise bean es un bean entidad y cualquier otra clase asociada).

Cuando se compila el módulo de EJB con el entorno de desarrollo de JBuilder, se crea automáticamente el archivo JAR adecuado.

**Nota** WebSphere 4.0 Advanced Edition permite distribuir únicamente un grupo EAR. Para obtener más información, consulte “[Creación de archivos EAR](#)” en la página 12-6.

- 4 Distribución del EJB en un contenedor de EJB.

JBuilder cuenta con un asistente para distribución de Enterprise que simplifica el proceso de distribución de enterprise beans de Borland. Si

el servidor de aplicaciones de destino no es de Borland, cuando se abre Herramientas | Distribución de Enterprise aparece un cuadro de diálogo Configuración de distribución que es específico del servidor. Una vez que ha rellenado todas las opciones de acuerdo con sus necesidades y ha seleccionado Aceptar para cerrar el cuadro de diálogo, el enterprise bean se distribuye tal y como lo haya especificado.

También es posible acceder a configuraciones de distribución específicas del servidor de la siguiente manera: haga clic con el botón derecho del ratón en el nodo del módulo EJB, elija Propiedades, abra la pestaña Distribución y realice la configuración. Después, haga clic en el módulo EJB con el botón derecho del ratón para distribuirlo, elija Opciones de distribución de <nombre del archivo.jar> y pulse Distribuir o Distribuir de nuevo.

## Creación de un archivo de descriptor de distribución

---

Cuando se utilizan las herramientas de EJB de JBuilder para crear enterprise beans, el programa crea también descriptores. Después puede utilizar el Editor de descriptor de distribución con el fin de añadir la información necesaria y modificar los atributos de los descriptores.

Cada descriptor de distribución compatible con las especificaciones EJB 1.1 ó 2.0 (excluidos los que utiliza WebSphere 3.5):

- Debe estar basado en XML y cumplir las normas de XML.
- Debe ser válido en lo relativo a la DTD de la especificación EJB 1.1 ó 2.0.
- Debe cumplir las normas de sintaxis establecidas en la DTD.
- Debe hacer referencia a la DTD con una de las siguientes sentencias:

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems Inc./DTD Enterprise JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dts/ejb-jar_1_1.dtd">
```

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc./DTD Enterprise JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
```

Cuando se utilizan las herramientas de EJB de JBuilder para crear y modificar los descriptores de distribución, no es necesario preocuparse por aprender XML ni por cumplir las normas de sintaxis establecidas en la DTD de Sun. El editor del descriptor de distribución impone estas reglas en los datos que se introducen y se modifican. Cuando se introduce información mediante este editor, se solicitan los datos necesarios. Las herramientas de JBuilder configuran automáticamente las extensiones específicas de Borland en un archivo ejb-inprise.xml en el caso de los descriptores de distribución 1.1 o en un archivo ejb-borland.xml para los descriptores 2.0. Si desea más información sobre la verificación del

descriptor de distribución, consulte el [Capítulo 13, “El editor de descriptor de distribución”](#).

## Competencias del descriptor de distribución

---

La función del descriptor de distribución consiste en proporcionar información sobre los EJB que deben agruparse y distribuirse en un archivo JAR determinado. Está destinado para su uso por parte del consumidor del archivo JAR EJB. El desarrollador del bean es el responsable de crear el descriptor de distribución.

La información del descriptor de distribución se utiliza para definir los atributos del enterprise bean. Estos atributos establecen el funcionamiento del enterprise bean en un entorno concreto. Por ejemplo: cuando se definen los atributos transaccionales del bean, éstos establecen el comportamiento del bean con respecto a las transacciones. El descriptor de desarrollo recoge la siguiente información:

- Información de tipo, que define los tipos o nombres de las clases de las interfaces local/base local y remota/local y la clase del bean.
- Nombres JNDI, que definen el nombre con el que se ha registrado la interfaz local/base local del enterprise bean.
- Campos que habilitan la persistencia gestionada por el contenedor.
- Directrices transaccionales que rigen el comportamiento transaccional de un bean.
- Atributos de seguridad que rigen el acceso a un enterprise bean.
- Información específica de Borland, como la información del origen de datos que se utiliza para las conexiones a una base de datos.

## Tipos de información del descriptor de distribución

---

La información del descriptor de distribución se puede dividir en dos tipos básicos:

- Información de estructura de los enterprise beans.

Describe la estructura de un enterprise bean y declara sus dependencias externas. Esta información es necesaria. Normalmente no es posible modificar la información de estructura, porque esto podría alterar el funcionamiento del bean.

- Información de ensamblaje de la aplicación.

Describe la forma en que los enterprise beans incluidos en el archivo `ejb-jar.xml` se combinan para formar una unidad más grande de distribución de aplicaciones. Esta información es optativa. La

información de ensamblaje se puede modificar sin alterar el funcionamiento del bean, pero esto puede alterar el funcionamiento de la aplicación ensamblada.

## Información de estructura

El desarrollador del bean debe proporcionar la siguiente información sobre estructura para cada bean del archivo JAR EJB:

### Todos los enterprise beans

- Nombre del enterprise bean, una referencia mnemotécnica que se utiliza para designar el bean en el descriptor de distribución.
- Clase Enterprise Bean.
- Tipo del enterprise bean (sesión, entidad o basado en mensajes)
- Elementos de entorno, si el bean tiene parámetros de configuración.
- Referencias de factoría de recursos.
- Referencias de EJB, si un enterprise bean hace referencia a otro.
- Referencias de seguridad por competencias, si un enterprise bean debe acceder a competencias específicas.
- Referencias de entorno del recurso, si el bean hace referencia a un recurso externo.

### Beans sesión

- Interfaz base o base local del bean sesión.
- Interfaz remota o local del bean sesión.
- Tipo del bean sesión (con o sin estado)
- Tipo de demarcación de transacciones de beans sesión con estado, con devoluciones de llamada de sincronización.

### Beans entidad

- Interfaz base o base local del bean entidad.
- Interfaz remota o local del bean entidad.
- Gestión de persistencia de los beans entidad.
- Clase de clave principal del bean entidad.
- Los campos con persistencia gestionada por contenido de beans gestionados por contenedor y la información sobre relaciones de los componentes EJB 2.0

### Beans gestionado por mensajes

- Tipo de gestión de transacciones del bean basado en mensajes.

- Destino y duración de la suscripción del bean gestionado por mensajes.

## Información de ensamblaje de la aplicación

Se puede introducir la siguiente información sobre el ensamblaje de aplicaciones. Esta información es optativa durante el ensamblaje. Sin embargo, no lo es para el desarrollador.

- Asociación de referencias de enterprise beans
- Competencias de seguridad
- Permisos para los métodos
- Vinculación de referencias de competencias de seguridad
- Identidad de seguridad
- Atributos de transacción

Durante los procesos de ensamblaje y distribución de aplicaciones se puede modificar la siguiente información de estructura:

- Valores de las variables de entorno. El ensamblador de aplicaciones puede cambiar las propiedades de entorno y definir sus valores.
- Campos de descripción. El ensamblador de aplicaciones puede cambiar las descripciones o crear elementos de éstas.

No es posible modificar otros tipos de información estructural. Sin embargo, se puede modificar la información de ensamblaje de las aplicaciones durante la distribución.

## Seguridad

Normalmente, el ensamblador de aplicaciones transmite la siguiente información al descriptor de distribución:

- Competencias de seguridad
- Permisos para los métodos
- Vínculos entre referencias de competencias de seguridad y competencias de seguridad
- Identidad de seguridad

## Competencias de seguridad

Los elementos de competencias de seguridad del descriptor de distribución permiten al desarrollador definir estas competencias. Éstos definen las competencias de seguridad necesarias para los clientes de los enterprise beans.

## Permisos para los métodos

Los elementos de permisos para los métodos del descriptor de distribución permiten al desarrollador definir permisos para métodos. Se trata de relaciones duales entre las competencias de seguridad y los métodos de las interfaces remota/local y base remota/base local del enterprise bean.

**Vinculación de referencias de competencias de seguridad**

Si se definen competencias de seguridad, el desarrollador debe vincularlas con referencias mediante el elemento de vinculación de competencias del descriptor de distribución.

**Propiedades específicas del servidor de aplicaciones**

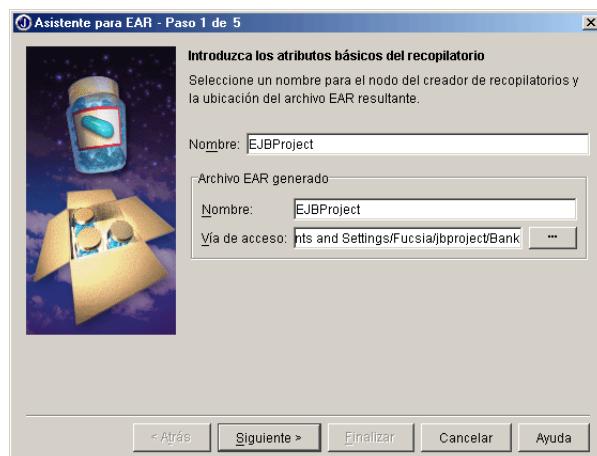
Los descriptores de distribución también pueden incluir propiedades específicas de un servidor de aplicaciones concreto.

## Creación de archivos EAR

Si desea incluir los archivos JAR de EJB en un archivo EAR (Recopilatorio de Enterprise), puede utilizar el Asistente para EAR de JBuilder.

**Nota** Para distribuir los enterprise beans a WebSphere 4.0 Advanced Edition es necesario crear un grupo EAR.

Para acceder al Asistente para EAR, seleccione Archivo | Nuevo, abra la pestaña Enterprise y haga doble clic en el ícono EAR. Se abre el Asistente para EAR:



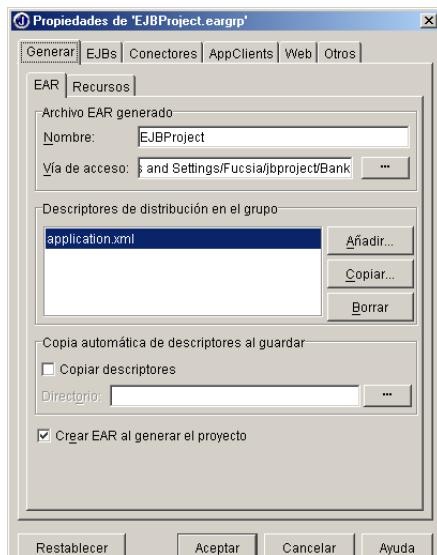
Si desea obtener información sobre la utilización de este asistente, pulse el botón Ayuda.

Cuando se termina de utilizar el asistente, en el panel de proyecto se crea un nodo .eargrp. Cuando se hace doble clic en este nodo, en el panel de contenido aparece una pestaña Fuente del DD del EAR que muestra los descriptores de distribución del archivo.

Las propiedades de generación de .eargrp se pueden modificar, igual que ocurre con los módulos EJB:

- 1 Haga clic con el botón derecho del ratón en el nodo .eargrp del panel de proyecto.
- 2 Seleccione Propiedades.

Aparece el cuadro de diálogo Propiedades:



Si desea más información sobre el uso del cuadro de diálogo Propiedades, pulse el botón Ayuda.

Para crear un archivo EAR a partir del nodo .eargrp, haga clic en él con el botón derecho del ratón y elija Ejecutar Make. Ahora, cuando se amplía el nodo .eargrp aparece el nuevo archivo EAR en el panel de proyecto.

## Distribución en un servidor de aplicaciones

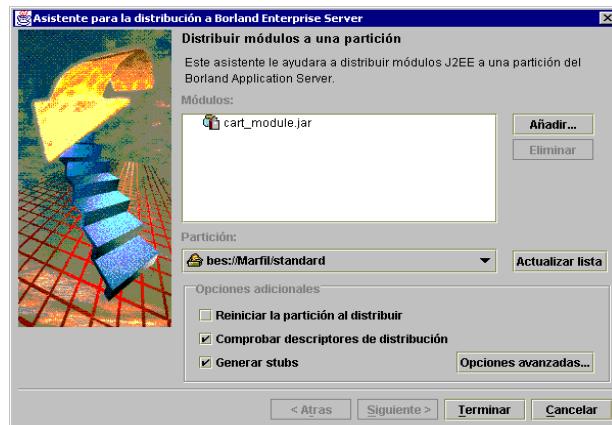
**La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.**

Cuando el bean funciona de forma satisfactoria, y si se selecciona Borland Enterprise Server como servidor de aplicaciones para el proyecto actual, se puede distribuir el bean en Borland Enterprise Server mediante el Asistente para la distribución a Borland Enterprise Server. Si su servidor de aplicaciones de destino no es de Borland, al seleccionar Herramientas | Asistente para distribución de Enterprise, aparece un cuadro de diálogo Configuración de distribución que se puede utilizar para la distribución en estos servidores.

## Distribución de archivos JAR

Para distribuir módulos J2EE en Borland Enterprise Server -5.0.2 – 5.1.x:

- 1 Inicie un agente de gestión, si es que todavía no lo ha hecho, seleccionando Herramientas | Agente de gestión de Borland Enterprise Server. (Este paso es optativo cuando el proceso de inicio del servidor intenta iniciar el Agente de gestión Borland Enterprise si no se ha iniciado ya.)
- 2 Para abrir el Asistente para distribución de Borland Enterprise Server, seleccione Herramientas | Distribución de Enterprise.



- 3 Pulse Añadir, busque el emplazamiento de los módulos J2EE (archivos JAR, WAR, RAR, EAR, ZIP y DAR) que desea distribuir y selecciónelos. Pulse Aceptar.
- 4 Si desea comprobar que los descriptores de distribución y las clases a las que hacen referencia tienen la forma correcta antes de que se distribuyan los módulos, active la opción Verificar descriptores de distribución.
- 5 Si aún no ha generado los stubs de los beans y desea hacerlo, active la opción Generar stubs.
- 6 El botón Opciones avanzadas del asistente permite configurar las opciones adicionales Generador de Stubs y Verificador. Si desea modificar alguna de estas opciones, pulse Opciones avanzadas y realice los cambios en el cuadro de diálogo del mismo nombre. Pulse Aceptar cuando haya terminado.
- 7 Pulse Siguiente para pasar a la otra ficha, donde se especificarán los destinos de distribución.
- 8 Pulse Finalizar para cerrar el asistente e iniciar el proceso de distribución.

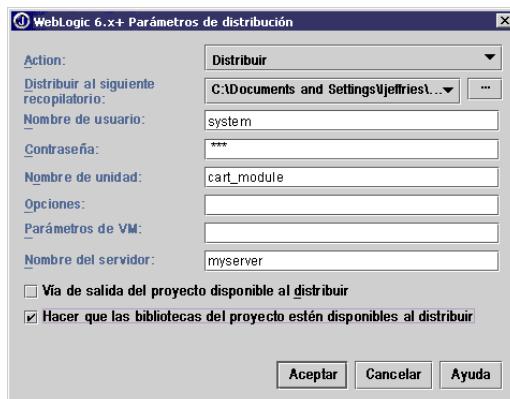
El asistente intenta distribuir los módulos J2EE y comunicar los resultados.

Si desea más información sobre cómo utilizar el Asistente para distribución en Borland Enterprise Server, consulte la documentación de Borland Enterprise Server.

## Distribución a servidores ajenos a Borland

---

Si se desarrolla para otro servidor, como WebLogic, WebSphere o iPlanet, también es posible distribuir los EJB por medio del comando Herramientas | Distribución de Enterprise. Si el servidor de aplicaciones del proyecto es uno de estos servidores, el comando muestra un cuadro de diálogo Configuración de distribución específica para el servidor. Por ejemplo, a continuación se muestra el cuadro de diálogo Configuración de distribución de WebLogic 6.x+:



La herramienta de distribución de WebSphere 4.0 difiere según la versión del servidor utilizada. La herramienta de distribución de WebSphere Single Server es SEAppInstaller. La herramienta de distribución de Advanced Edition es SEAppInstaller. Por ello, el aspecto del cuadro de diálogo Configuración de distribución varía entre las dos ediciones de WebSphere Server 4.0.

En el cuadro de diálogo Configuración de distribución de iPlanet 6.x+, la opción Mostrar distribuciones se puede seleccionar siempre en la lista desplegable Acción, pero sólo funciona cuando la opción Servidor local está seleccionada en Destino.

Rellene los campos que necesite y seleccione Aceptar. Si desea más información pulse el botón Ayuda del cuadro de diálogo Configuración de distribución .

## Definición de las opciones de distribución mediante el cuadro de diálogo Propiedades

Además de usar del comando Herramientas | Distribución de Enterprise para configurar las opciones para la distribución en los nodos distribuibles del proyecto, también puede utilizar el cuadro de diálogo Propiedades. Estas propiedades se guardan en el nodo específico en el que se configuran. Si anteriormente se ha utilizado el cuadro de diálogo Distribución de Enterprise, los valores introducidos en él se convierten en los valores por defecto de los nodos distribuibles del proyecto.

Para configurar opciones de distribución mediante el cuadro de diálogo Propiedades:

- 1 Haga clic con el botón derecho del ratón en el nodo del módulo EJB, en JAR dependiente de este nodo o en un grupo EAR. Se abre un menú contextual. También puede hacer clic con el botón derecho del ratón sobre un nodo EAR o WAR, si considera que se pueden distribuir para el servidor seleccionado.
- 2 Seleccione Propiedades con el fin de que aparezca el cuadro de diálogo homónimo. Si hizo clic con el botón derecho en el nodo del módulo EJB, debe hacer clic en la pestaña Distribución del cuadro de diálogo Propiedades y, a continuación, en la ficha específica de su servidor de aplicaciones (como, por ejemplo, Borland Enterprise Server 5.0.2 -5,1.x).
- 3 Defina sus opciones. Las opciones disponibles dependen del servidor de aplicaciones de destino. Por ejemplo, con Borland Enterprise Server 5.0.2 -5,1.x, se pueden definir el nombre del anfitrión, el contenedor y los parámetros de la MV. Los usuarios de WebLogic pueden definir un nombre de unidad, opciones de distribución, una contraseña y parámetros MV. Los usuarios de WebSphere pueden definir el nombre del nodo principal, el nombre del servidor de aplicaciones, el nombre del contenedor, los parámetros MV y una opción para generar XML. Si se seleccionan varios nodos que tienen opciones de distribución diferentes, se utilizan los valores por defecto. Si desea ayuda para llenar estos campos, consulte la documentación del servidor de aplicaciones.
- 4 Si su servidor de aplicaciones de destino es WebSphere 3.5 o WebSphere 4.0 Advanced Edition y desea que se genere un archivo XML como entrada en la utilidad WebSphere XMLConfig, marque la casilla Generar XML. Si esta opción no está marcada, el archivo no se creará. Si ha introducido modificaciones en el archivo XML que se ha creado (de nombre `deploy_<nodoseleccionado>.xml` y que aparece debajo del nodo del módulo EJB o del grupo EAR), no marque esta opción para asegurarse de que no pierde los cambios. Si se pulsa el módulo EJB con el botón derecho del ratón y elige Opciones de distribución de <nombre del archivo jar>.jar aparecen los comandos de distribución,

con los que se puede generar un archivo XML llamado `deploy.xml`. Este archivo aparece bajo el nodo del proyecto.

## Distribución en tiempo real en un servidor de aplicaciones

---

Durante el ciclo de desarrollo, puede que desee distribuir, redistribuir o deshacer la distribución de forma rápida de los enterprise beans en un contenedor que ya esté en ejecución. Haga clic con el botón derecho del ratón sobre el nodo del módulo EJB (o sobre el nodo EAR o WAR si considera que se pueden distribuir para el servidor) o sobre los nodos dependientes del panel de proyecto y seleccione Opciones de distribución de <nombre del archivo jar>.jar, para ver la lista de comandos de distribución:

- Distribuir – Distribuye un archivo JAR en el contenedor del servidor de aplicaciones del proyecto que está en ejecución. Si la opción Generar destino de la configuración de ejecución actual es Make (elija Ejecutar | Configuraciones, seleccione la configuración de ejecución actual y pulse Modificar. A continuación, pulse Servidor y seleccione Distribución en el árbol de servicios para poder ver el destino de generación), esta opción ejecuta el Make del contenido del archivo JAR antes de distribuirlo en el contenedor.
- Distribuir de nuevo – Distribuye de nuevo el archivo JAR en el contenedor que está en ejecución. Si la opción Generar destino de la configuración de ejecución actual es Make (elija Ejecutar | Configuraciones, seleccione la configuración de ejecución actual, pulse Modificar y elija Servidor; puede ver el destino de generación en la parte inferior del cuadro de diálogo Propiedades de ejecución), esta opción ejecuta el Make del contenido del archivo JAR antes de volver a distribuirlo en el contenedor.
- Deshacer distribución – Deshace la distribución de un archivo JAR ya distribuido en el contenedor que está en ejecución.
- Mostrar distribuciones – Lista todos los archivos JAR distribuidos en el contenedor que está en ejecución.
- Detener contenedor - Detiene el contenedor. Esta opción sólo está disponible para WebSphere 3.5 y WebSphere 4.0 Advanced Edition. Cuando un EJB ya distribuido cambia, el contenedor debe detenerse y reiniciarse para que los cambios queden registrados.
- Iniciar contenedor - Inicia el contenedor. Esta opción sólo está disponible para WebSphere 3.5 y WebSphere 4.0 Advanced Edition.

**Nota** Para ver las Opciones de distribución de la opción de menú <nombre del archivo jar>.jar con WebSphere 4.0 Advanced Edition se debe pulsar con el botón derecho del ratón un nodo de grupo EAR del panel de proyectos

en lugar del módulo EJB. WebSphere Server 4.0 no considera que los JAR de EJB puedan ser distribuidos.

**Nota** La distribución rápida no está disponible con WebSphere 3.5 ni con WebSphere 4.0 Advanced Edition. Para ver los cambios realizados en los enterprise beans es necesario detener e iniciar el contenedor por medio del menú Opciones de distribución, después de volver a realizar la distribución.

# 13

## El editor de descriptor de distribución

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

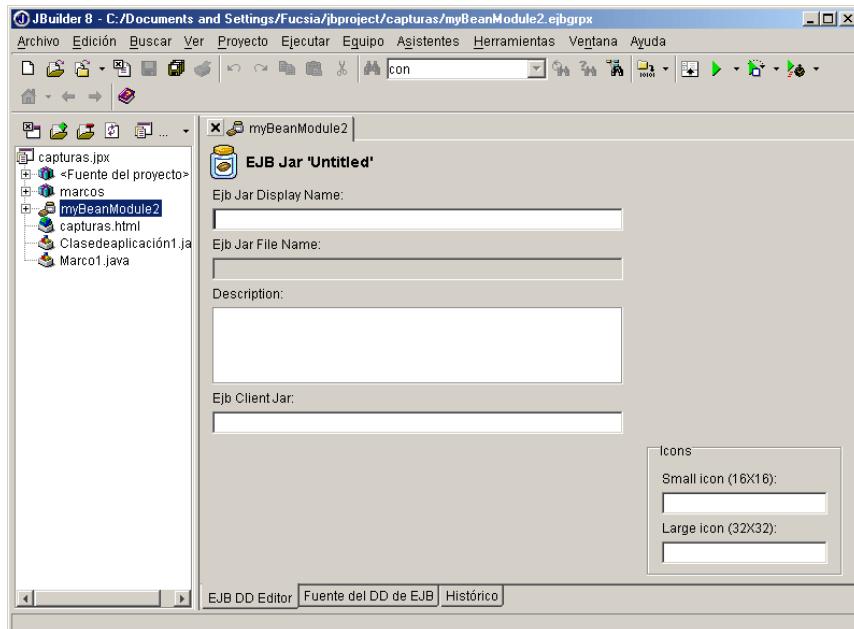
JBuilder incluye un editor del descriptor de distribución que se puede utilizar para modificar la información de distribución (como las normas de transacciones y las competencias de seguridad de los archivos de descripción de distribución EJB). También se puede modificar el método para hacer persistentes los enterprise beans. Si desea información general sobre los descriptores de distribución, consulte el [Capítulo 12, "Distribución de enterprise beans"](#).

También es posible ver y modificar algunas de las propiedades específicas de otros servidores de aplicaciones. Para obtener más información, consulte ["Panel Propiedades específicas del servidor" en la página 13-27](#). También hay varios paneles especiales para WebLogic en el Editor de descriptor de distribución.

Si el servidor de aplicaciones seleccionado es Sybase Enterprise Application Server 4.1, las propiedades específicas de Sybase se examinan y se modifican mediante el panel Editor DD de EA Server, en lugar de con un panel Propiedades específico del servidor. Todas las propiedades específicas de Sybase se encuentran disponibles y modificadas en el nivel del nodo y no en el nivel del bean.

## Presentación del editor de descriptor de distribución

Para mostrar el Editor de descriptor de distribución haga doble clic en el módulo EJB, en el panel de proyecto, y pulse la pestaña Editor DD de EJB , en la parte inferior del panel de contenido.

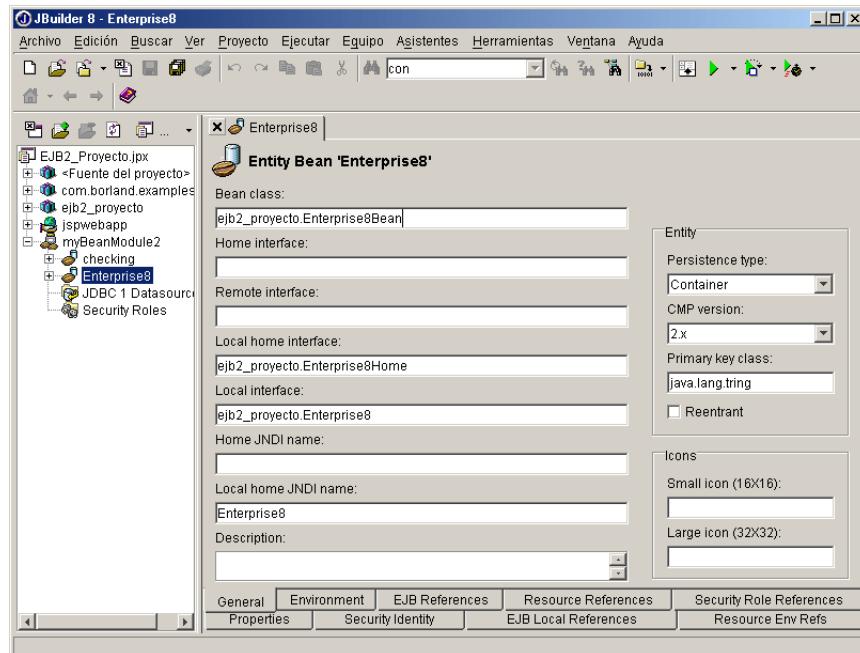


## Visualización del descriptor de distribución de enterprise beans

Para visualizar información sobre un enterprise bean en el Editor de descriptor de distribución:

- 1 Pulse el ícono que se encuentra en el extremo izquierdo del nodo del módulo EJB, para abrirlo.
- 2 Hacer doble clic en el bean del panel de proyecto.

Se abre el panel General del editor de descriptor de distribución.



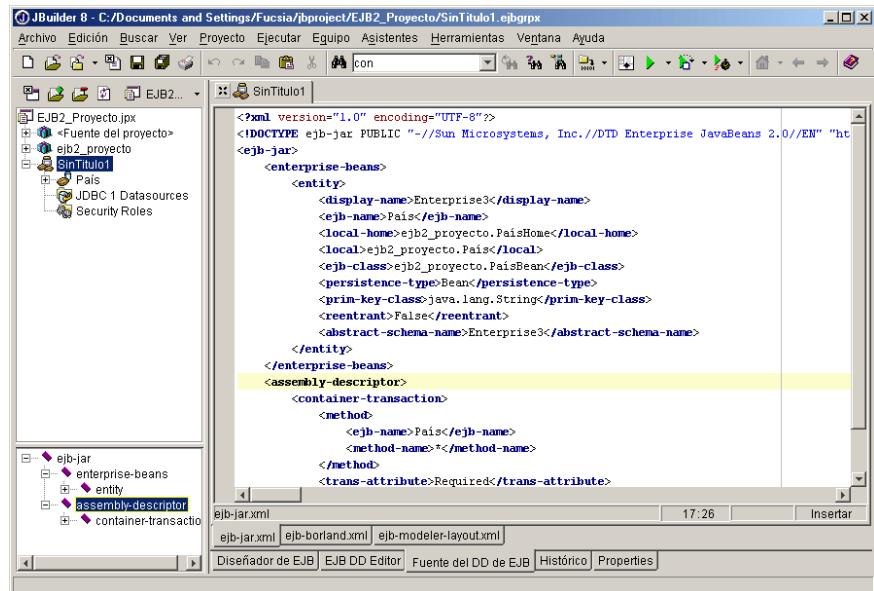
El descriptor de distribución tiene muchas más pestañas. Para ver otros paneles, pulse cualquiera de las pestañas de la parte inferior del Editor de descriptor de distribución. Utilice el editor para efectuar las modificaciones deseadas en la información de distribución del bean.

Abra un nodo de bean en el panel de proyecto si desea información adicional sobre el descriptor de distribución. Cuando se hace doble clic en uno de estos nodos aparecen paneles adicionales. Por ejemplo, si se hace doble clic en el nodo Transacciones de contenedor de un bean, aparece un panel Transacciones de contenedor en el Editor de descriptor de distribución. Los nodos Fuentes de datos JDBC y Competencias de seguridad también se pueden abrir si contienen datos. Los subnodos que aparecen se pueden utilizar para mostrar más información.

Para visualizar el código fuente de un descriptor de distribución, haga doble clic en el módulo EJB, en el panel de proyecto, y pulse la pestaña Fuente del DD de EJB, en la parte inferior del editor. En el módulo EJB aparece una pestaña por descriptor de distribución, con el nombre del archivo. Seleccione la pestaña del archivo que desea ver. Mientras visualiza el código fuente de un descriptor de distribución, puede hacer clic sobre los elementos del panel de estructura y desplazar una barra para

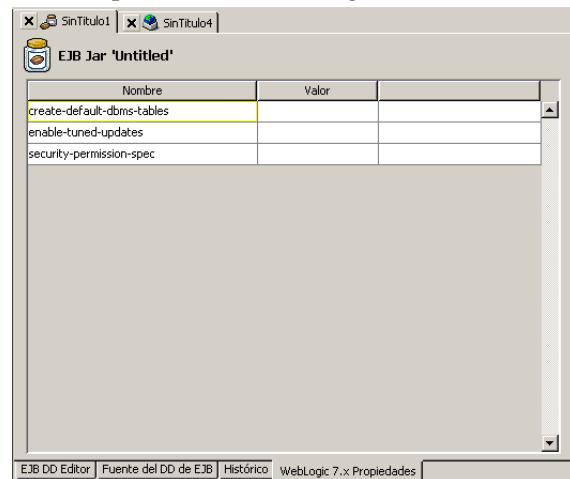
## Visualización del descriptor de distribución de enterprise beans

resaltar el elemento correspondiente en el código fuente. Puede modificar el código fuente directamente.



## Visualización de la ficha Propiedades de módulo EJB de WebLogic 6.x o 7.x

Si el servidor de aplicaciones seleccionado para el proyecto actual es WebLogic Server 6.0 o posterior, cuando el módulo EJB es el nodo actual, el Editor de descriptor de distribución muestra una ficha adicional: la ficha Propiedades de WebLogic 6.x o 7.x.



En esta ficha se pueden definir propiedades específicas de la versión del servidor WebLogic configurada para el proyecto. Configure las propiedades create-default-dbms-tables, database-type y validate-db-schema-with, por medio de las listas desplegables. En security-permission-spec, escriba el valor y pulse *Intro*.

Si desea más información sobre estas propiedades, consulte la documentación de WebLogic.

## Cambio de la información del bean

---

Para modificar la información sobre un bean en un descriptor de distribución:

- 1 Abra o amplíe el nodo del módulo EJB en el panel de proyecto, para ver su contenido.
- 2 Haga doble clic en el bean que le interesa.  
Se abre el panel General del editor de descriptor de distribución.
- 3 Utilice las pestañas de la parte inferior para abrir el panel desde el cual desea modificar la información del nuevo bean.
- 4 Realice los cambios.

Si está familiarizado con el trabajo en XML y con el código fuente del descriptor de distribución, puede pulsar la pestaña Fuente del DD de EJB y efectuar los cambios directamente en el código.

**Advertencia** Aunque en el panel Fuente del DD se pueden modificar las propiedades directamente en el código fuente del descriptor de distribución, para definir las propiedades exclusivas de servidores ajenos a Borland se debe utilizar siempre el panel de Propiedades específicas del servidor, ya que de lo contrario estos cambios se pierden cuando se modifican los valores de otras propiedades con el Editor de descriptor de distribución.

**Advertencia** Toda la información específica de los proveedores se encuentra almacenada en `ejb-borland.xml` (para todos los servidores). Si modifica directamente el código fuente de los descriptores de distribución, asegúrese siempre de que los cambios de este archivo permanecen.

# Información del enterprise bean

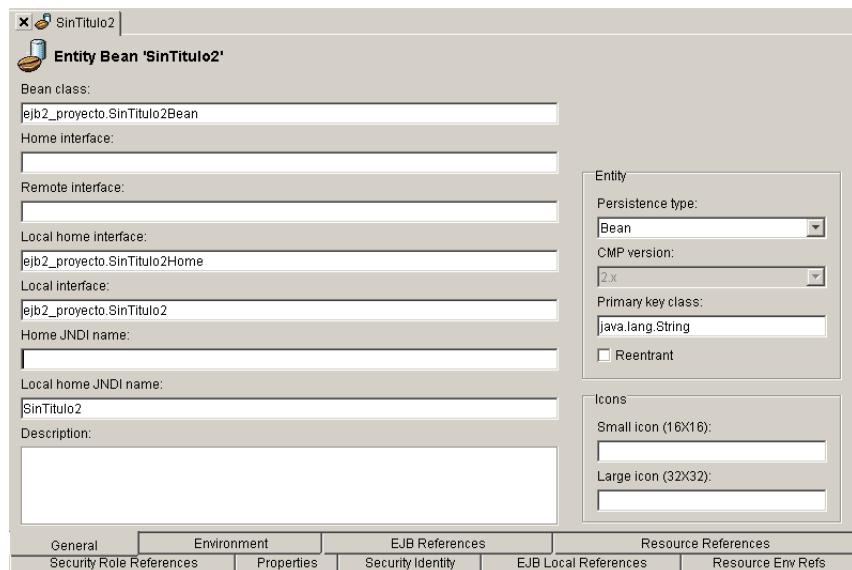
En este apartado se describe el tipo de información que se puede crear y almacenar para enterprise beans en el descriptor de distribución.

## Panel General

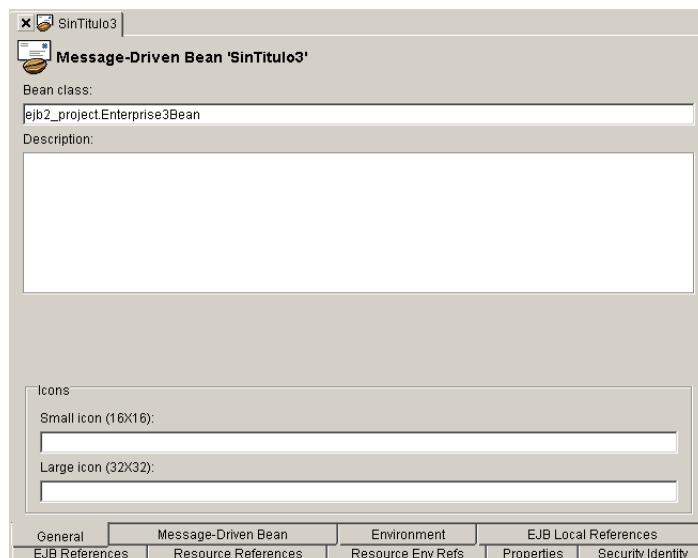
Introduzca o cambie aquí la información general sobre el enterprise bean. Éste es el panel General de un bean sesión:

Bean class:	ejb2_proyecto.SinTituloBean		
Home interface:	ejb2_proyecto.SinTituloHome		
Remote interface:	ejb2_proyecto.SinTitulo		
Local home interface:			
Local interface:			
Home JNDI name:	SinTitulo		
Local home JNDI name:			
Description:			
<b>Session</b> Session type: <input checked="" type="checkbox"/> Stateless Transaction type: <input checked="" type="checkbox"/> Container Timeout (secs): 0			
<b>Icons</b> Small icon (16x16): Large icon (32x32):			
General      Environment      EJB References      Resource References Security Role References      Properties      EJB Local References      Resource Env Refs      Security Identity			

Éste es el panel General de un bean entidad:



Éste es el panel General de un bean gestionado por mensajes:



El panel General incluye la siguiente información:

- **Bean Name (Nombre del bean):** Nombre lógico asignado al enterprise bean por el proveedor de beans. Todos los enterprise beans tienen un nombre lógico. No existe una relación estructurada entre el nombre lógico del bean y el nombre de raíz del bean (JNDI) asignado a éste. El desarrollador puede cambiar los nombres lógicos.

- **Bean Class (Clase del bean):** Nombre completo de la clase Java que implementa los métodos empresariales del bean. Esta información es obligatoria.
- **Home Interface (Interfaz base):** Nombre completo de la interfaz base remota del enterprise bean. Se debe proporcionar esta información si no se especifica una interfaz base local. Un bean puede tener a la vez una interfaz base (base remota) y una interfaz base local.
- **Remote Interface (Interfaz remota):** Nombre completo de la interfaz remota del enterprise bean. Se debe proporcionar esta información si no se especifica una interfaz local. Un bean puede tener a la vez una interfaz base remota y una interfaz base local.
- **Local Home Interface (Interfaz base local):** Nombre completo de la interfaz base local del enterprise bean. Se debe proporcionar esta información si no se especifica una interfaz base. Un bean puede tener a la vez una interfaz base local y una interfaz base remota. Este campo sólo está disponible para los componentes EJB 2.0.
- **Local interface (Interfaz local):** Nombre completo de la interfaz base local del enterprise bean. Se debe proporcionar esta información si no se especifica una interfaz remota. Un bean puede tener a la vez una interfaz local y una interfaz remota. Este campo sólo está disponible para los componentes EJB 2.0.
- **Home JNDI Name (Nombre del JNDI base):** Nombre del JNDI de la interfaz base remota del enterprise bean.
- **Local JNDI Name (Nombre del JNDI local):** Nombre del JNDI de la interfaz base local del enterprise bean. Este campo sólo está disponible para los componentes EJB 2.0.
- **Description (Descripción):** Resumen de la finalidad y la función del bean. Esta información es optativa.
- **Small icon (Icono pequeño):** Nombre de un archivo de ícono de 16 por 16 píxeles que se utiliza para representar al bean.
- **Large icon (Icono grande):** Nombre de un archivo de ícono de 32 por 32 píxeles que se utiliza para representar al bean.

En los beans sesión el panel General también incluye lo siguiente:

- **Session Type (Tipo de sesión):** Indica si el enterprise bean tiene estado.
- **Transaction Type (Tipo de transacción):** Indica si el bean o el contenedor ha establecido métodos de transacción.

En los beans entidad el panel General también incluye lo siguiente:

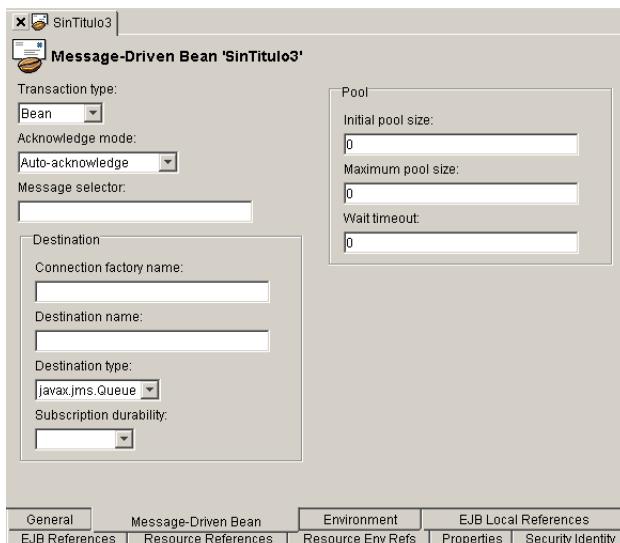
- **Persistence Type (Tipo de persistencia):** Indica si la persistencia del bean está gestionada por el propio bean o por el contenedor.

- **CMP Version (versión CMP):** Indica si se utiliza la versión 1.1 ó 2.0 de la persistencia gestionada por el contenedor. Este campo sólo está disponible para los beans entidad EJB 2.0. Para los beans entidad creados con el diseñador de EJB, la versión CMP debe ser 2.0.
- **Primary Key Class (Clase de clave principal):** Nombre completo de la clase de clave principal del bean entidad. Es necesario indicar la clase de la clave principal.
- **Reentrant (Reentrante):** Indica que el bean es reentrant. Borland recomienda no hacer reentrant los beans.

## Panel de beans gestionados por mensajes

---

Este panel sólo está disponible para los beans gestionados por mensajes.



El panel de beans gestionados por mensajes contiene los siguientes campos:

- **Transaction Type (Tipo de transacción):** Indica el tipo de gestión de transacciones del bean. Elija Bean si el bean gestiona sus propias transacciones; elija Container si se utiliza la gestión de transacciones por contenedor.
- **Acknowledge Mode (Modo de Acuse de recibo):** Este campo aparece únicamente si el valor de Tipo de transacción es Bean. Si se activa la opción Auto-acknowledge (Acuse de recibo automático), el bean envía acuse de recibo de todos los mensajes que le llegan, y se realiza una verificación para impedir que los mensajes duplicados desencadenen acciones.

La opción Dups-ok-acknowledge (Aceptar duplicados) hace que se envíe acuse de recibo de todos los mensajes, incluidos los duplicados.

- **Message Selector (Selector de mensajes):** El selector de mensajes que determina qué mensajes debe recibir el bean gestionado por mensajes. A continuación, se propone un ejemplo.

```
JMSType = 'chair' AND color = 'black' AND fabric = 'leather'
```

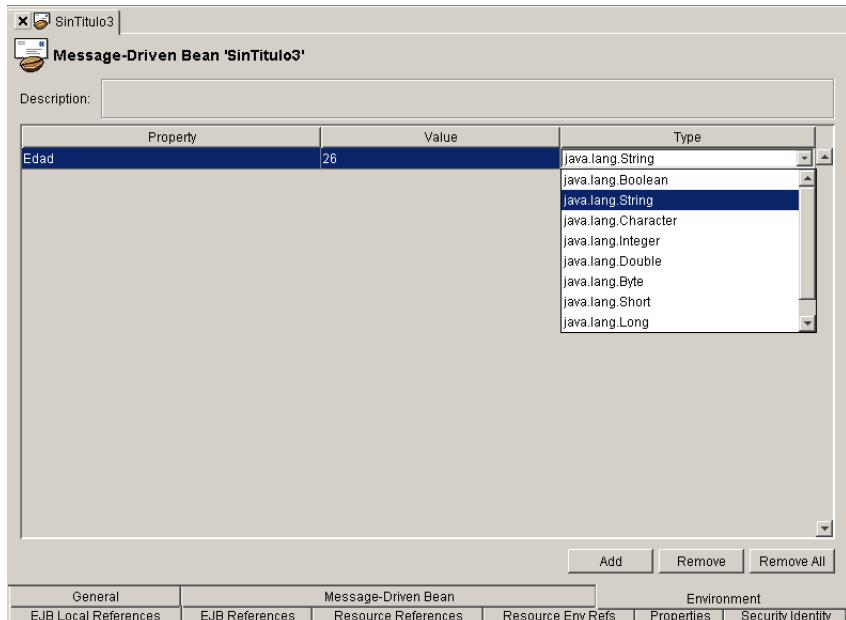
Si desea más información, consulte la especificación JMS en la sede web de Sun Microsystems, en <http://java.sun.com/products/jms/docs.html>.

- **Connecton Factory Name (Nombre de factoría de conexiones):** El nombre JNDI de la factoría de conexiones utilizada para establecer una conexión con el gestor de mensajes.
- **Destination Name (Destino):** El nombre JNDI de la cola o el tema que monitoriza el bean gestionado por mensajes. Éste es el destino JMS desde el que consume mensajes la instancia del bean gestionado por mensajes.
- **Destination Type (Tipo de destino):** Indica si el destino de un bean gestionado por mensajes es Queue (Cola) o Topic (Tema). También se puede elegir Not Specified (No especificado). Si se elige Topic aparece el campo Subscription Durability (Duración de la suscripción).
- **Subscription Durability (Duración de la suscripción):** Indica si la suscripción de un bean a un tema es duradera o temporal. Este campo sólo se muestra si se elige Topic en el campo Message Driven Destination.
- **Initial Pool Size (Tamaño inicial de la agrupación):** El número inicial de instancias del bean gestionado por mensajes que debe crear el contenedor inmediatamente después de la distribución.
- **Maximum Pool Size (Tamaño máximo de la agrupación):** El número máximo de instancias del bean gestionado por mensajes que se pueden crear y mantener en la agrupación de instancias del bean.
- **Wait Timeout (Tiempo de espera superado):** Indica el tiempo expresado en segundos que el mensaje puede hacer cola mientras espera una instancia de bean gestionado por mensajes, para gestionarlo. Por ejemplo, debería limitar la agrupación de instancias de beans gestionados por mensajes a cinco, por razones de rendimiento. Si se satura la cola, debería establecer un límite de espera del mensaje, especialmente en el contexto de una transacción en la que se desea evitar que dicha transacción parezca estar pendiente.

## Panel Environment

El panel Environment (Entorno) enumera todos los elementos de entorno del enterprise bean. Estos elementos permiten personalizar la lógica empresarial del bean cuando éste se ensambla o se distribuye. El entorno permite personalizar el bean sin acceder a su código fuente ni modificarlo.

Cada enterprise bean tiene su propio conjunto de elementos de entorno. Todas las instancias de un enterprise bean comparten los mismos elementos de entorno, pero no pueden modificarlo durante la ejecución.



Para añadir un elemento de entorno:

- 1 Pulse Add (Añadir) para crear un elemento.  
Aparecerá una nueva fila en blanco.
- 2 Introduzca una propiedad en la columna Property y asígnele un valor en la columna Value (Valor).
- 3 Seleccione un tipo de propiedad en la lista desplegable Type (Tipo).
- 4 Si desea ofrecer una descripción de la propiedad de entorno añadida, escríbala en el campo Description (Descripción).
- 5 Añada todos los elementos de entorno deseados.

Para eliminar un elemento de entorno:

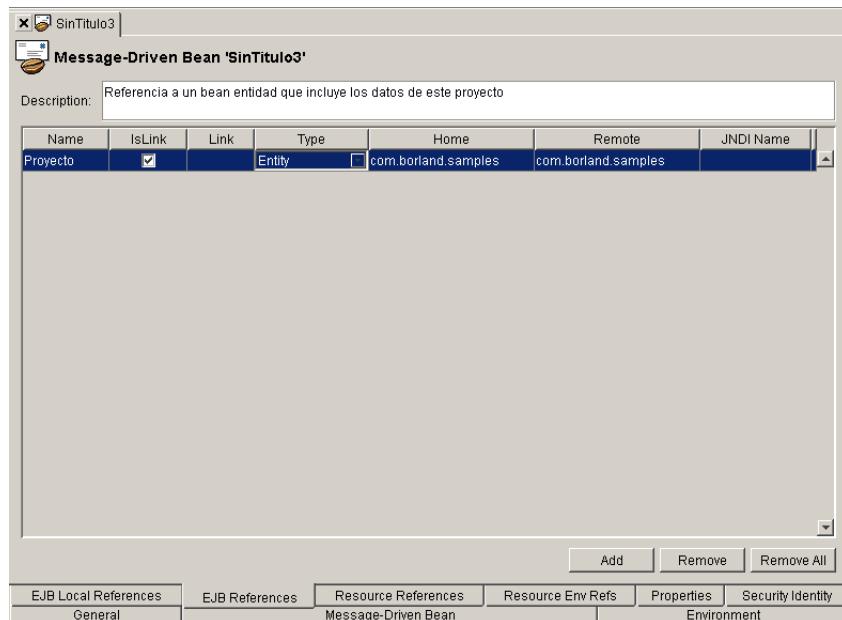
- 1** Seleccione la fila.
- 2** Pulse el botón Remove (Eliminar).

Es necesario tener en cuenta ciertos aspectos relacionados con los elementos de entorno:

- El proveedor de beans debe declarar todos los elementos de entorno a los que se accede desde el código del enterprise bean.
- Si el proveedor de beans incluye un valor para el elemento de entorno, se puede cambiar más adelante, en el ensamblaje o la distribución.
- El ensamblador puede modificar los valores de los elementos de entorno definidos por el desarrollador.
- El desarrollador debe procurar asignar a todos los elementos de entorno valores válidos.

## Panel EJB References

El panel EJB References (de referencias EJB) enumera todas las referencias de enterprise beans a las raíces de otros beans que necesita. Utilice este panel únicamente para los beans que hacen referencia a beans remotos. Para los beans EJB 2.0 que hacen referencia a beans locales, utilice el panel Referencias locales del EJB.



Cada referencia EJB describe los requisitos de interfaz del enterprise bean al que corresponde, para el bean al que se refiere. Se pueden definir referencias entre beans dentro del mismo archivo JAR o desde un enterprise bean externo al archivo JAR, como de un bean sesión a un bean entidad.

Para añadir una referencia EJB:

**1** Pulse Añadir.

**2** En el cuadro de diálogo que aparece, escriba un nombre para la referencia EJB y pulse OK.

Se añade una fila al panel.

**3** Rellene los campos de esta fila con la siguiente información.

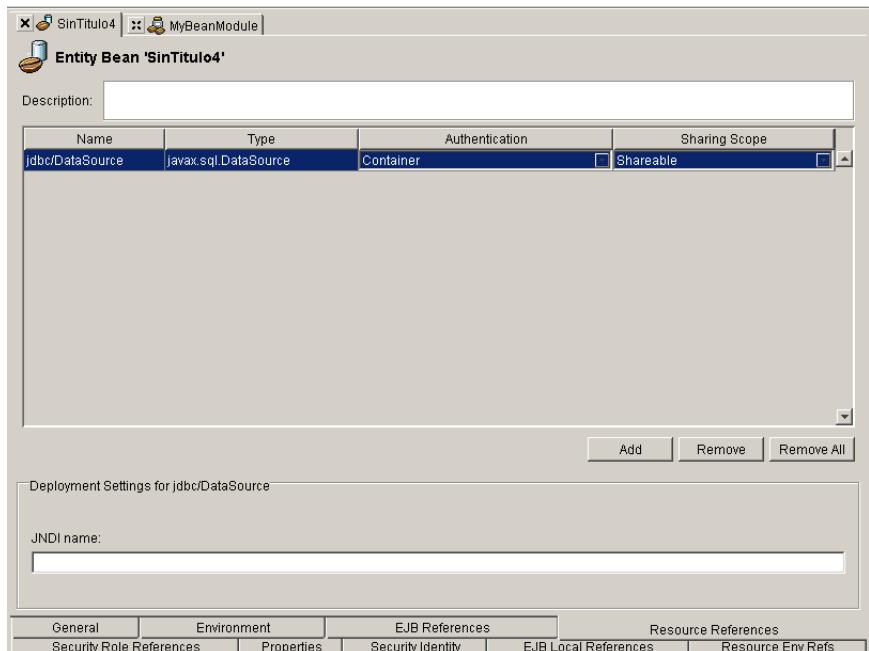
- **Description (Descripción):** Breve descripción del bean al que se hace referencia. Esta información es optativa. Tras escribir la descripción, pulse sobre la fila o, de lo contrario, la perderá.
- **Name (Nombre):** Nombre del bean al que se hace referencia.
- **IsLink:** Cuando se activa IsLink se hace referencia a un bean del JAR, por lo que el valor de Nombre del JNDI es irrelevante. Si no se activa IsLink, se utiliza el nombre del JNDI para buscar el bean. Después de activar esta opción, se debe seleccionar en la lista desplegable Link (Enlace) el bean al que se hace referencia.
- **Link (Enlace):** Crea una asociación entre la referencia EJB y el enterprise bean de destino. El valor de Link (Enlace) es el nombre del bean de destino. Esta información es optativa.
- **Type (Tipo):** Tipo esperado del bean al que se hace referencia.
- **Home (Base):** Tipo Java que se espera que tenga la interfaz local del bean al que se hace referencia. En los componentes EJB 2.0, este campo hace referencia a la interfaz base remota.
- **Remote (Remoto):** Tipo Java que se espera que tenga la interfaz remota del bean al que se hace referencia.
- **JNDI Name (Nombre del JNDI):** Nombre del JNDI del bean al que se hace referencia.

Es importante recordar ciertos aspectos de las referencias EJB:

- El enterprise bean de destino debe ser de un tipo compatible con la referencia EJB declarada.
- Todas las referencias EJB declaradas deben estar asociadas a las raíces de enterprise beans que existen en el entorno operativo.
- Si se define un valor para Link (Enlace), la referencia del enterprise bean debe estar asociada a la raíz del enterprise bean de destino.

## Panel Resource References (Referencias de recursos)

El panel de referencias de recursos enumera todas las referencias de fábrica de recursos del enterprise bean. Esto permite al ensamblador de aplicaciones y al distribuidor de beans localizar todas las referencias que utiliza el enterprise bean.



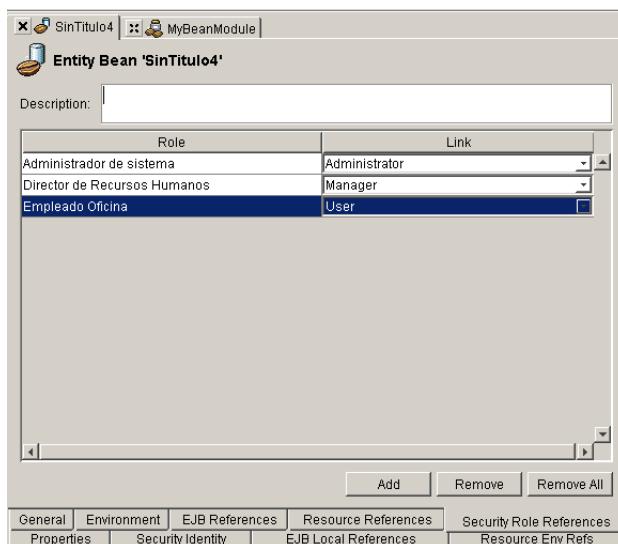
Para añadir una referencia de recurso, pulse Add (Añadir) y complete los siguientes campos:

- **Description (Descripción):** Descripción de la referencia de recurso. Esta información es optativa. Tras escribir la descripción, pulse sobre la fila o, de lo contrario, la perderá.
- **Name (Nombre):** Nombre del elemento de entorno utilizado en el código del enterprise bean.
- **Type (Tipo):** Tipo Java que el código del enterprise bean espera de la fábrica de recursos. (Éste es el tipo Java del recurso *factory*, no el tipo Java del recurso.)
- **Authentication (Autenticación):** La Application authentication (Autenticación de una aplicación) indica que el enterprise bean efectúa el registro en el recurso mediante el código. La Container authentication (Autenticación de contenedor) indica que el contenedor se registra en el recurso basándose en la información de asignación principal suministrada por el distribuidor.

- **Sharing Scope (Ámbito donde compartir):** Determina si se puede compartir el recurso. Se puede elegir Shareable (Sí) o Unshareable (No). Este campo sólo está disponible para los componentes EJB 2.0.
- **JNDI Name (Nombre del JNDI):** Nombre del JNDI de la referencia de recurso.

## Panel Security Role References

El panel de referencias de competencias de seguridad enumera todas las referencias del enterprise bean a las competencias de seguridad. Este panel asocia las referencias a competencias de seguridad que utiliza el desarrollador de beans a competencias de seguridad determinadas definidas por el ensamblador o el distribuidor de la aplicación.



Para poder añadir una referencia de competencia de seguridad es necesario haber definido competencias; de lo contrario, el botón Add (Añadir) de este panel aparecerá desactivado. Si desea más información sobre la creación y la asignación de competencias de seguridad para la distribución de aplicaciones, consulte “[Adición de competencias de seguridad y permisos para métodos](#)” en la página 13-42.

Para añadir una competencia, pulse Add (Añadir) y complete los tres campos.

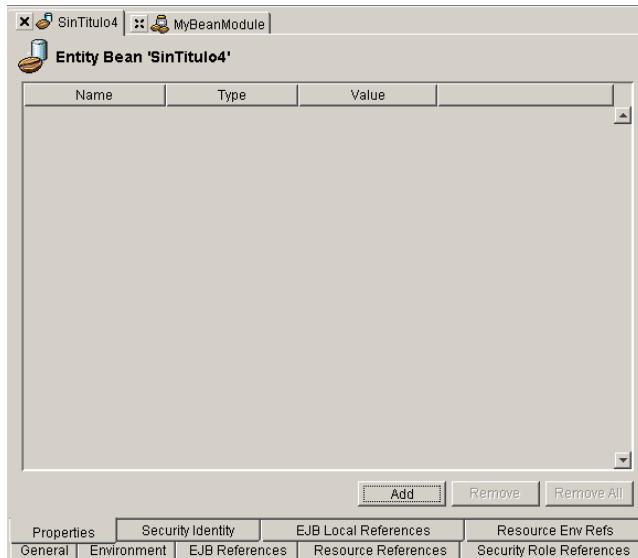
- **Description (Descripción):** Éste es un campo optativo que describe la competencia de seguridad. Tras escribir la descripción, pulse sobre la fila o, de lo contrario, la perderá.
- **Role (Competencia):** Nombre de la competencia de seguridad definida por el desarrollador de beans.

- **Link (Enlace):** Nombre de la competencia de seguridad utilizada en la distribución de la aplicación. Normalmente, el ensamblador o el distribuidor de la aplicación define esta competencia de forma que funcione en un entorno operativo determinado.

## Panel Properties

---

Cuando se hace doble clic en un enterprise bean, en el panel de proyecto, y se pulsa la pestaña Properties del editor del descriptor de distribución, aparece el siguiente panel de propiedades:

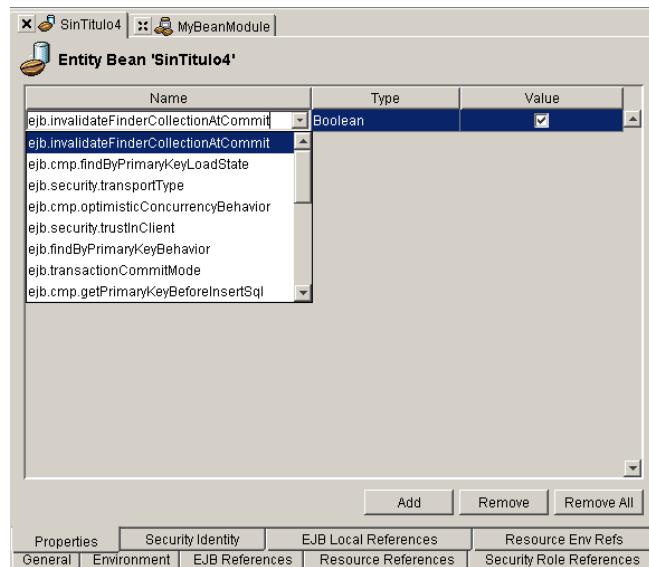


Utilice este panel sólo si el servidor de aplicaciones de destino es Borland.

Para añadir una propiedad a un enterprise bean seleccionado en el panel de proyecto:

- 1 Haga clic en el botón Add (Añadir) para añadir una fila al panel.

- 2** Elija en la lista desplegable Nombre la propiedad que desea añadir.



- 3** Indique un valor en el campo Value (Valor).

Algunos valores se eligen en una lista desplegable; otros, como los de cadena y entero, hay que escribirlos; y otro presenta una casilla de verificación para valores booleanos. Cuando se activa se indica que el valor es true.

A continuación se describen las propiedades de Borland que se pueden elegir y los valores que se les pueden asignar:

- **ejb.cmp.checkExistenceBeforeCreate**

Suprime la comprobación de existencia que tiene lugar antes de la creación de beans entidad. La especificación EJB requiere que el contenedor compruebe la existencia de un bean entidad (esto es, que compruebe que hay una fila en la tabla) y lance una excepción `javax.ejb.DuplicateKeyException` si lo encuentra. Por motivos de rendimiento puede ser aconsejable eliminar estos accesos suplementarios a la base de datos, ya que ésta impide que se introduzcan valores duplicados.

- **ejb.cmp.findByPrimaryKeyLoadState**

Indica si se deben cargar los valores ajenos a la clave principal durante la ejecución del método `findByPrimaryKey()` (por defecto) o si basta con comprobar que el registro de la entidad existe en la base de datos. Este indicador equivale al atributo `<load-state>` de otros buscadores.

- **ejb.cmp.getPrimaryKeyBeforeInsertSql**

Indica la SQL que ejecuta el motor de persistencia gestionada por contenedor para generar una clave principal cuando tiene lugar el siguiente suceso INSERT. El motor de persistencia gestionada por contenedor actualiza el bean entidad con el valor de esta clave principal. Normalmente, esta propiedad se utiliza en combinación con Oracle Sequences. Si desea más información acerca de la generación de claves principales, consulte el ejemplo / BorlandEnterpriseServer/examples/ejb/pkgen.

- **ejb.cmp.getPrimaryKeyAfterInsertSql**

Indica la SQL que ejecuta el motor de persistencia gestionada por contenedor para generar una clave principal después del siguiente suceso INSERT. El motor de persistencia gestionada por contenedor actualiza el bean entidad con el valor de esta clave principal. Cuando se define esta propiedad también se debe definir la propiedad `ejb.cmp.ignoreColumnsOnInsert`. Si desea más información acerca de la generación de claves principales, consulte el ejemplo / BorlandEnterpriseServer/examples/ejb/pkgen.

- **ejb.cmp.ignoreColumnsOnInsert**

Indica el nombre de la columna que no debe definir el CMP durante el suceso INSERT. Esta propiedad se utiliza en combinación con `ejb.cmp.getPrimaryKeyAfterInsertSql`. Si desea más información acerca de la generación de claves principales, consulte el ejemplo / BorlandEnterpriseServer/examples/ejb/pkgen.

- **ejb.cmp.jdbcAccesserFactory**

Especifica la factoría de una instancia implementada por el usuario de la interfaz `com.inprise.ejb.cmp.JdbcAccessor`. Esta interfaz le proporciona una manera de escribir código específico para obtener un valor de un `java.sqlResultSet` o para asignarle un valor a una `java.sql.PreparedStatement`. El valor por defecto es `none`.

- **ejb.cmp.manager**

Especifica el nombre de una clase que implementa la interfaz `com.inprise.ejb.cmp.Manager`. Una instancia de esta clase se utiliza para realizar la persistencia gestionada por contenedor (CMP).

- **ejb.cmp.primaryKeyGenerator**

Indica una clase escrita por el usuario que implementa la interfaz `com.inprise.ejb.cmp.PrimaryKeyGenerator` y genera claves principales. Si desea más información acerca de la generación de claves principales, consulte el ejemplo / BorlandEnterpriseServer/examples/ejb/pkgen.

- **ejb.maxBeansInPool**

Determina el número máximo de beans de la agrupación de disponibles. Si se sobrepasa este límite, las entidades se eliminan del contenedor mediante una llamada a `unsetEntityContext()`. El valor por defecto es 1.000.

- **ejb.maxBeansInCache**

Determina el número máximo de beans de la memoria caché de la opción A (consulte `ejb.transactionCommitMode`, a continuación). Si se sobrepasa este límite, las entidades se desplazan a la agrupación de disponibles mediante una llamada a `ejbPassivate()`. El valor por defecto es 1.000.

- **ejb.maxBeansInTransaction**

Limita el número total de beans entidad que puede haber en el contenedor durante las transacciones. Esta propiedad puede resultar útil cuando se ejecutan transacciones de gran volumen (por lotes) en las que participan muchas entidades. Normalmente, las entidades se mantienen en las transacciones hasta que finalizan. Sin embargo, en las transacciones por lotes, la conservación de todas las entidades asociadas a la transacción puede agotar la memoria de la máquina virtual.

- **ejb.transactionCommitMode**

Indica la disposición de un bean entidad con respecto a una transacción. Los valores son:

- A o Exclusive (Exclusivo): acceso exclusivo a la tabla de la base de datos. Se puede presuponer que el estado del bean al finalizar la última transacción enviada es el que tendrá al principio de la próxima transacción. Los beans se almacenan en caché entre transacción y transacción.
- B o Shared (Compartido): acceso compartido a la tabla de la base de datos. Sin embargo, por motivos de rendimiento, cada bean permanece asociado a una clave principal en los intervalos entre transacciones, para evitar las llamadas innecesarias a `ejbActivate()` y `ejbPassivate()`. El bean permanece en el búfer de activos. Éste es el valor por defecto.
- C o None (Ninguno): acceso compartido a la tabla de la base de datos. Los beans no permanecen asociados a claves principales determinadas entre las transacciones, sino que cada vez que termina una vuelven a la agrupación de disponibles. Normalmente, esta opción no resulta muy útil.

- **ejb.cmp.optimisticConcurrencyBehavior**

Se puede introducir uno de los siguientes valores:

- UpdateAllFields
- UpdateModifiedFields
- VerifyModifiedFields
- VerifyAllFields

**UpdateAllFields:** Actualiza todos los campos, independientemente de que estén modificados. Si un bean de persistencia gestionada por contenedor tiene los tres campos "key", "value1" y "value2", guardados en una tabla llamada "MyTable", al final de cada transacción tiene lugar la siguiente actualización, independientemente de que se haya modificado el bean:

```
UPDATE MyTable SET (value1 = <value1>, value2 = <value2>)
WHERE key = <key>
```

**UpdateModifiedFields:** Es el valor por defecto. Actualiza únicamente los campos modificados, o cancela la actualización si no se ha modificado el bean. Con el bean anterior, si sólo se ha modificado "value1", se transmite la siguiente actualización:

```
UPDATE MyTable SET (value1 = <value1>)
WHERE key = <key>
```

Esto puede aumentar considerablemente el rendimiento por los siguientes motivos:

- 1 Con frecuencia se accede a los datos únicamente para leerlos. En estos casos es mejor no enviar una actualización a la base de datos. En Borland se han observado grandes aumentos del rendimiento gracias a esta optimización.
- 2 Muchas bases de datos escriben registros según las columnas modificadas. Por ejemplo, SQL Server registra la actualización de un campo TEXT o IMAGE, independientemente de que el valor de la columna haya cambiado realmente. Tenga en cuenta que a menudo la base de datos no distingue entre la sustitución de los datos de una columna por los mismos datos (que es lo que ocurre con "UpdateAllFields") y la modificación real de los valores durante la actualización. La supresión de la actualización para el caso en el que el valor no cambia realmente puede tener un efecto significativo sobre el rendimiento cuando se utilizan estos SGBD.
- 3 Provoca menos tráfico de red JDBC en dirección a la base de datos y menos trabajo en dirección al controlador JDBC. El problema de la red no suele ser significativo, pero el del controlador JDBC sí lo es. Las mediciones de rendimiento indican que, en las aplicaciones EJB de grandes proporciones, el controlador JDBC puede llegar a consumir un 70% del tiempo del procesador. Esto se debe a menudo a que muchos controladores JDBC comerciales

no tienen el rendimiento suficientemente ajustado. Incluso en el caso de que los controladores estén bien ajustados es mejor no someterlos a grandes cargas de trabajo.

**VerifyModifiedFields:** En este modo, el motor de persistencia gestionada por el contenedor emite una actualización ajustada mientras comprueba que los campos que está actualizando son coherentes con los valores leídos anteriormente. Así, en el ejemplo anterior, donde sólo se modificó “value1”, se transmite la siguiente actualización:

```
UPDATE MyTable SET (value1 = <value1>
WHERE key = <key> AND value1 = <old-value1>
```

**VerifyAllFields:** Este modo es parecido a VerifyModifiedFields, con la diferencia de que se comprueban todos los campos. Por tanto, la actualización sería:

```
UPDATE MyTable SET (value1 = <value1>
WHERE key = <key> AND value1 = <old-value1> AND value2 =
<old-value2>
```

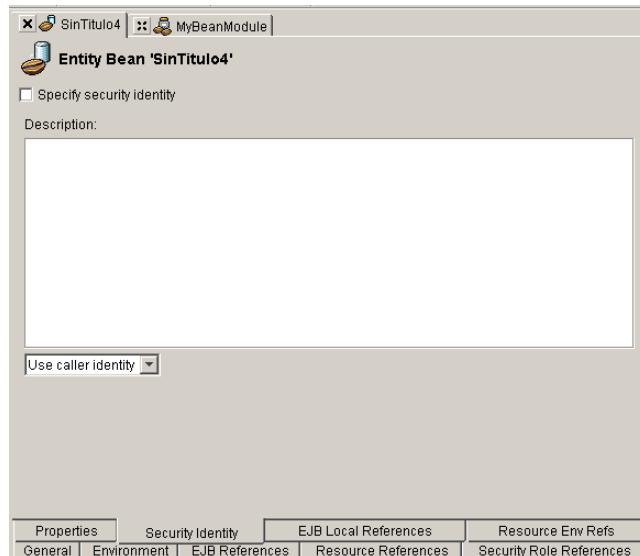
Estos dos ajustes de verificación se pueden utilizar para reproducir el nivel de aislamiento SERIALIZABLE en el contenedor. Es frecuente que las aplicaciones requieran semántica de aislamiento serializable. Sin embargo, si se solicita a la base de datos que implemente esta función se puede reducir considerablemente el rendimiento. En las pruebas se ha demostrado que el uso de SERIALIZABLE con Oracle en lugar de un nivel de aislamiento restringido puede decelerar las aplicaciones en más de un 50%. El motivo principal es que Oracle proporciona una gestión de conflictos optimista, con un modelo de bloqueo por filas. Los dos valores mencionados arriba equivalen a solicitar al motor CMP que implemente la gestión de conflictos optimista con bloqueo de campo. Como ocurre con todos los sistemas concurrentes, la calidad de la gestión de conflictos es inversamente proporcional a las desigualdades del bloqueo.

## Panel Security Identity

---

El panel Security Identify (Identidad de seguridad) permite indicar si se desea utilizar una identidad de seguridad durante la ejecución de los métodos del bean.

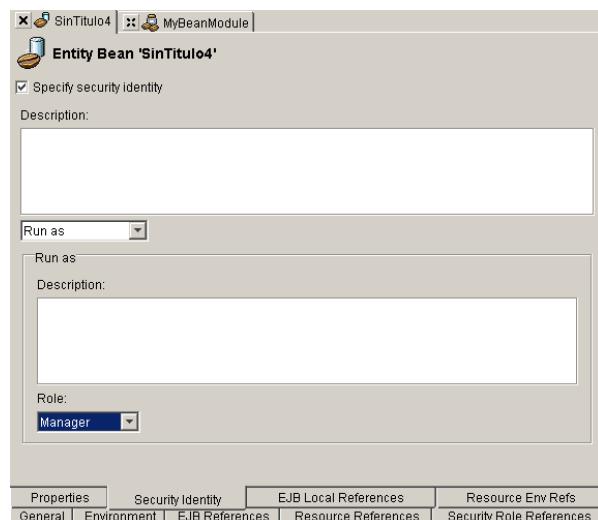
Este panel sólo está disponible para los componentes EJB 2.0.



Para indicar que se debe utilizar una identidad de seguridad cuando se ejecutan métodos del bean, active la casilla de selección Specify Security Identify (Especificar identidad de seguridad) y realice las siguientes acciones:

- 1 Si lo desea, describa la identidad de seguridad en el campo Description (Descripción).
- 2 Indique en la lista desplegable que se encuentra bajo el campo Description si se debe utilizar la identidad de seguridad del llamante (Use caller identity) o una identidad de seguridad de ejecución

determinada (Use run-as identity). Si se elige Ejecutar como, el panel adquiere el siguiente aspecto:



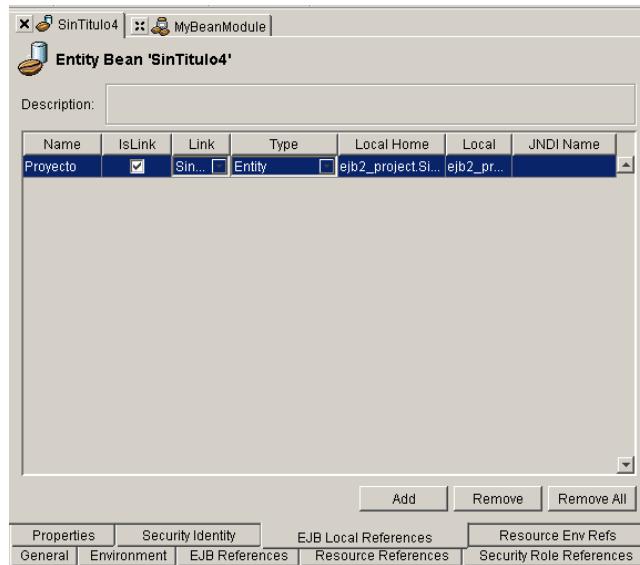
- 3** Si lo desea, escriba una descripción en el campo Description, aunque no es necesario.
- 4** Elija una competencia de la lista desplegable Role (Competencia). En esta lista se muestran todas las competencias de seguridad definidas para el módulo EJB actual. Si desea más información sobre la creación de competencias de seguridad, consulte "[Adición de competencias de seguridad y permisos para métodos](#)" en la página 13-42.

## Panel EJB Local References

---

Cada referencia local del EJB describe los requisitos de interfaz del enterprise bean al que corresponde, para el bean local al que se refiere. En el panel EJB Local References (Referencias locales del EJB) sólo se pueden definir referencias de este tipo. Si desea hacer referencia a un bean remoto, utilice en su lugar el Referencias EJB.

Este panel sólo está disponible para los componentes EJB 2.0.



Para añadir una referencia local del EJB :

- 1 Pulse Añadir.
- 2 En el cuadro de diálogo que aparece, escriba un nombre para la referencia local del EJB y pulse OK.  
Se añade una fila al panel.
- 3 Rellene los campos de esta fila con la siguiente información.
  - **Description (Descripción):** Breve descripción del bean al que se hace referencia. Esta información es optativa. Tras escribir la descripción, pulse sobre la fila o, de lo contrario, la perderá.
  - **Name (Nombre):** Nombre del bean al que se hace referencia.
  - **IsLink:** Cuando se activa IsLink se hace referencia a un bean del JAR, por lo que el valor de Nombre del JNDI es irrelevante. Si no se activa IsLink, se utiliza el nombre del JNDI para buscar el bean. Después de activar esta opción, se debe seleccionar en la lista desplegable Link (Enlace) el bean al que se hace referencia.
  - **Type (Tipo):** Tipo esperado del bean al que se hace referencia.
  - **Local Home (Base local):** Tipo Java que se espera que tenga la interfaz local del bean al que se hace referencia.

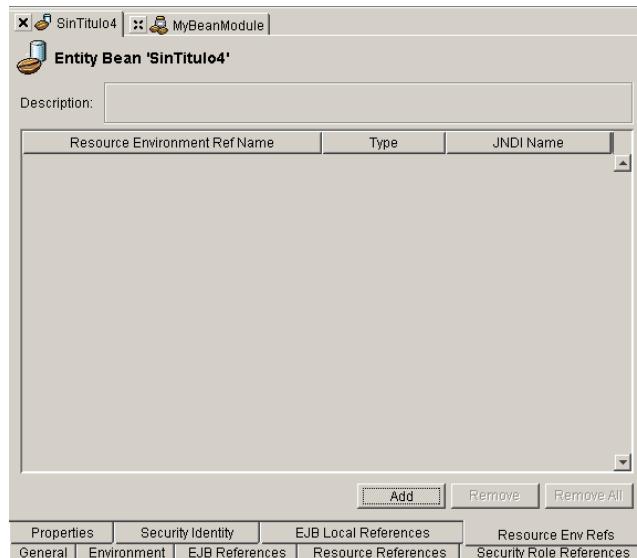
- **Local:** Tipo Java que se espera que tenga la interfaz remota del bean al que se hace referencia.
- **JNDI Name (Nombre del JNDI):** Nombre del JNDI del bean al que se hace referencia.

Es importante recordar ciertos aspectos de las referencias locales del EJB:

- El enterprise bean de destino debe ser de un tipo compatible con la referencia local del EJB declarada.
- Todas las referencias EJB declaradas deben estar asociadas a las raíces locales de enterprise beans que existen en el entorno operativo.
- Si se define un valor para Link (Enlace), la referencia del enterprise bean debe estar asociada a la raíz local del enterprise bean de destino.

## Panel Resource Env Refs

En el panel Resource Env Refs (Referencias de entorno de recurso) se pueden definir las referencias a recursos externos. Sólo está disponible para los componentes EJB 2.0. Éste es el aspecto del panel:



Para añadir una referencia de entorno de recurso:

- 1 Pulse el botón Añadir.
- 2 En la fila que se añade, pulse la columna Resource Environment Ref Name.
- 3 Si lo desea, escriba una descripción en Description, aunque no es necesario.

- 4 Escriba en esta columna el nombre de la referencia de entorno de recurso. Este nombre no puede repetirse en el enterprise bean.
- 5 Indique el tipo de referencia de entorno de recurso en la columna Type (Tipo). Se debe tratar del nombre completo de una clase o interfaz Java.
- 6 Indique el nombre JNDI de la referencia de entorno de recurso en la columna Jndi Name.

## Panel General de WebLogic 6.x o 7.x

Si el servidor de aplicaciones de destino es WebLogic 6.x o 7.x, aparece el panel General correspondiente:

The screenshot shows the 'Entity Bean 'MyCMPBean'' configuration panel in the WebLogic administration interface. The panel is organized into several sections:

- Pool**: Contains fields for 'Initial beans in free pool:' and 'Maximum beans in free pool:'.
- Entity clustering**: Contains fields for 'Home is clusterable:' and 'Home load algorithm:'.
- Automatic key generation**: Contains fields for 'Generator type:' (set to 'Default value'), 'Generator name:', and 'Key cache size:'.
- Data source name:** A field for entering the data source name.
- IIOP Security**: A group of fields for 'Integrity' (set to 'Default value'), 'Confidentiality' (set to 'Default value'), 'Client certificate authentication' (set to 'Default value'), 'Client authentication' (set to 'Default value'), and 'Identity assertion' (set to 'Default value').
- Transaction timeout in seconds:** A field for entering the transaction timeout value.
- Enable call by reference:** A field for enabling call by reference.
- Enable dynamic queries:** A field for enabling dynamic queries.

At the bottom of the panel, there are tabs for 'General', 'Environment', 'EJB References', 'Resource References', 'Security Role References', 'Properties', and 'Security Identity'. Below these tabs, specific links are provided: 'EJB Local References', 'Resource Env Refs', 'WebLogic 6.x+ General', 'WebLogic 6.x+ Properties', and 'WebLogic 6.x+ Cache'.

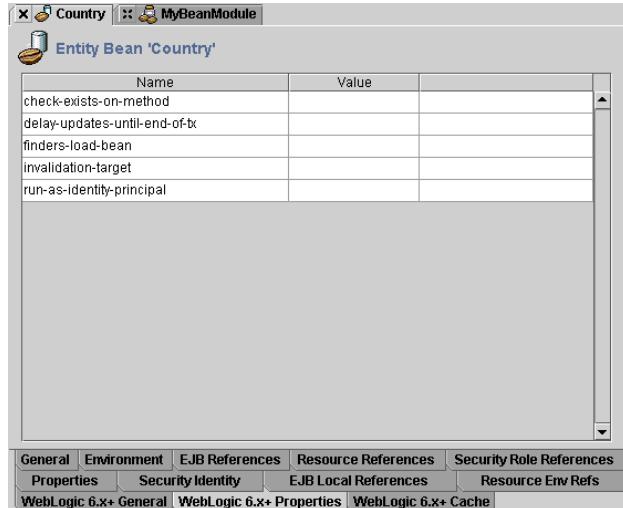
Los campos que aparezcan dependerán del tipo de bean y de cuál servidor WebLogic sea el destino. Por ejemplo, los campos que se muestran arriba aparecen en los beans entidad cuyo servidor de aplicaciones de destino es WebLogic 7.x.

Si desea más información sobre estos campos, consulte la documentación de WebLogic.

## Panel Propiedades específicas del servidor

**La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.**

Si el servidor de aplicaciones de destino no es de Borland o Sybase, cuando se selecciona un bean se abre un panel Properties específico del servidor. Por ejemplo, aquí se ve, en la parte inferior del panel de contenido, la pestaña de propiedades de WebLogic 6.x+:



Este panel permite visualizar y modificar algunos de los elementos propios del fabricante que son exclusivos del servidor de aplicaciones de destino. El panel muestra una tabla de las propiedades específicas del servidor. Utilice la columna de la derecha para introducir valores para las propiedades que deseé modificar. Los valores de las propiedades se almacenan en los descriptores de distribución específicos del servidor.

**Importante**

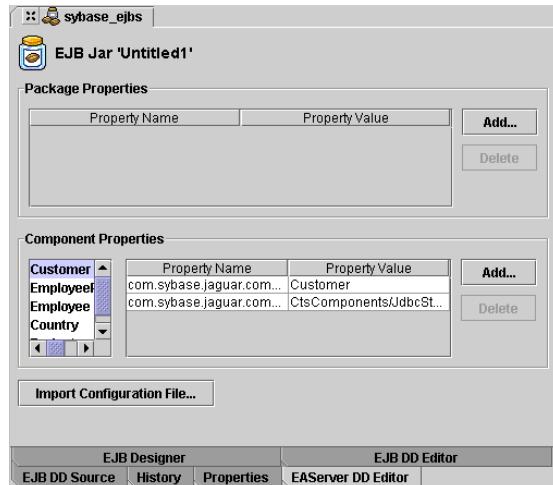
Aunque en el panel de fuente del DD se pueden modificar las propiedades directamente en el código fuente, para asignar los valores de las propiedades específicos del servidor se debe utilizar siempre el panel Properties específico del servidor. Si no lo hace así, es muy posible que se sobrescriban estos valores al modificar otros con el Editor de descriptor de distribución.

Si el servidor seleccionado es Sybase Enterprise Application Server 4.1, utilice en su lugar el panel Editor DD de EAServer para trabajar con las propiedades de Sybase.

## Panel Editor DD de EAServer

**La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.**

El panel Editor DD de EAServer aparece si el servidor seleccionado es Enterprise Application Server 4.1 de Sybase. Utilícelo para modificar en EJB de Sybase, EAR y los descriptores de distribución de la aplicación web. Así es como aparece el panel cuando se selecciona un componente:

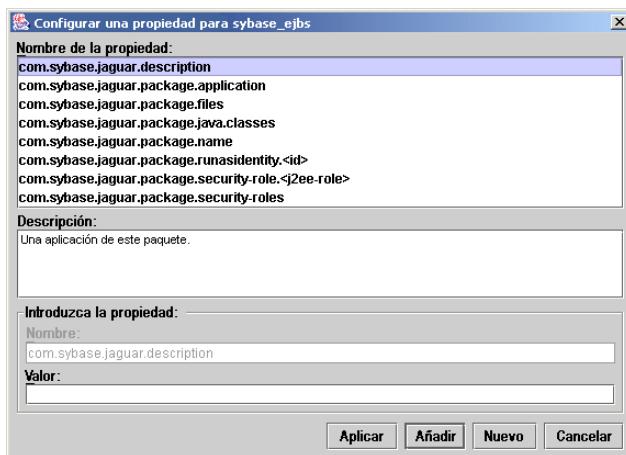


Utilice este panel si desea ver, añadir o modificar propiedades específicas del servidor Sybase Enterprise Application Server. Si desea obtener información específica sobre las propiedades de Sybase, consulte la documentación de Sybase.

### Configuración de propiedades de paquetes de Sybase

Si desea utilizar este panel para añadir propiedades de paquetes al descriptor de distribución `sybase_easerver_config.xml`, pulse el botón

Añadir situado en el cuadro de grupo Propiedades de paquetes. Aparece el siguiente cuadro de diálogo:



Seleccione en la lista de propiedades, la que desea añadir. El campo Descripción proporciona una descripción de cada propiedad, así como instrucciones sobre el tipo de valor que se debe especificar en el campo Valor.

En el caso de algunas propiedades, debe sustituir el nombre y el valor del parámetro. Por ejemplo, la propiedad `com.sybase.jaguar.package.runasidentity.<id>` necesita un parámetro. Indique el nombre del parámetro en el campo Nombre y el valor en el campo Valor.

Después de haber especificado una propiedad y suministrado un valor apropiado, pulse el botón Aplicar. El cuadro de diálogo que contiene la lista de propiedades de paquetes continúa abierto de modo que pueda seguir modificándolas. O si está modificando un única propiedad, pulse Añadir para añadir la propiedad y volver al panel Editor DD de EA Server.

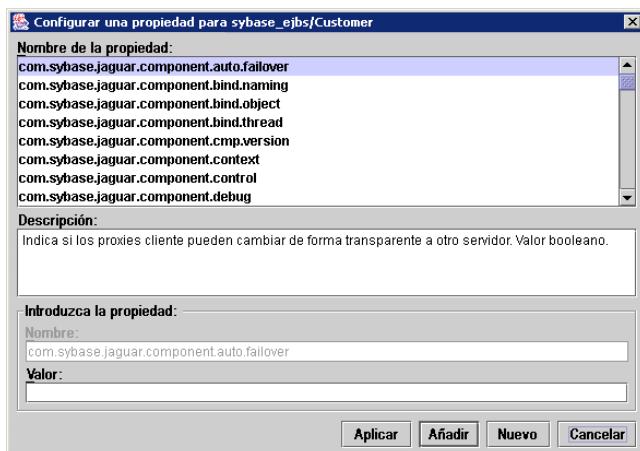
Para añadir una propiedad de paquete, pulse el botón Nuevo y añada el nombre del nuevo paquete a la cadena `com.sybase.jaguar.package.` incluida en el campo Nombre. Especifique un valor en el campo Valor y pulse Aplicar.

Una vez realizadas todas las modificaciones en las propiedades de paquetes, pulse el botón Añadir para volver al panel Editor DD de EA Server. Si ha utilizado el botón Aplicar para aplicar las propiedades como utiliza el cuadro de diálogo, puede también seleccionar Cancelar para volver al Editor DD de EA Server. Podrá ver los nuevos valores en la lista Package Properties (Propiedades de paquetes).

## Definición de las propiedades de los componentes de Sybase

El cuadro de grupo Propiedades del componente contiene una lista de todos los enterprise beans del módulo EJB. Cada componente recibe un nombre JNDI y una asignación de almacenamiento por defecto. Para examinar estos valores, seleccione el componente en el cuadro de lista. Si desea realizar cambios, puede modificar los valores directamente en la columna Property Value (Valor de la propiedad) o pulsar Añadir (Add) y utilizar el cuadro de diálogo que aparece.

Si desea especificar propiedades adicionales para un componente, seleccione el componente en el cuadro de lista y pulse el botón Add (Añadir). Aparece el siguiente cuadro de diálogo:



Seleccione en la lista de propiedades, la que desea añadir. El campo Descripción proporciona una descripción de cada propiedad, así como instrucciones sobre el tipo de valor que se debe especificar en el campo Valor. Para algunas propiedades, también es necesario asignarle un nombre en el campo Nombre, tal como se indica en las instrucciones de la descripción.

En el caso de algunas propiedades, debe sustituir el nombre y el valor del parámetro. Por ejemplo, la propiedad `com.sybase.jaguar.component.security-role.<j2ee-role>` necesita un parámetro. Indique el nombre del parámetro en el campo Nombre y el valor en el campo Valor.

Después de haber especificado una propiedad y suministrado un valor apropiado, pulse el botón Apply (Aplicar). El cuadro de diálogo que contiene la lista de propiedades del componente continúa abierto para que pueda seguir modificando las propiedades del paquete. O si está modificando un única propiedad, pulse Añadir para añadir la propiedad y volver al panel Editor DD de EA Server.

Para añadir una propiedad del elemento, pulse el botón New (Nuevo) y añada el nombre de la nueva propiedad a la cadena com.sybase.jaguar.component. incluida en el campo Nombre. Especifique un valor en el campo Valor y pulse Aplicar.

Una vez realizadas todas las modificaciones en las propiedades del elemento, pulse el botón Añadir para volver al panel Editor DD de EAServer. Si ha utilizado el botón Aplicar para aplicar las propiedades como utiliza el cuadro de diálogo, puede también seleccionar Cancelar para volver al Editor DD de EAServer. Podrá ver los nuevos valores en la lista Component Properties (Propiedades de componentes).

## Importación de un descriptor de distribución de Sybase

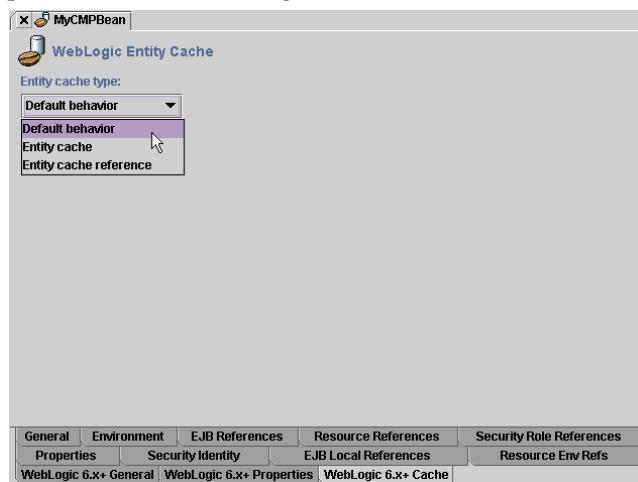
Si desea importar un descriptor de distribución de Sybase, creado previamente, (sybase\_easerver\_config.xml) en el módulo EJB:

- 1 Pulse el botón Import Configuration File (Importar archivo de configuración).
- 2 En el cuadro de diálogo que aparece, desplácese hasta el descriptor de distribución que deseé importar.
- 3 Haga doble clic sobre el descriptor de distribución.
- 4 Pulse Aceptar.

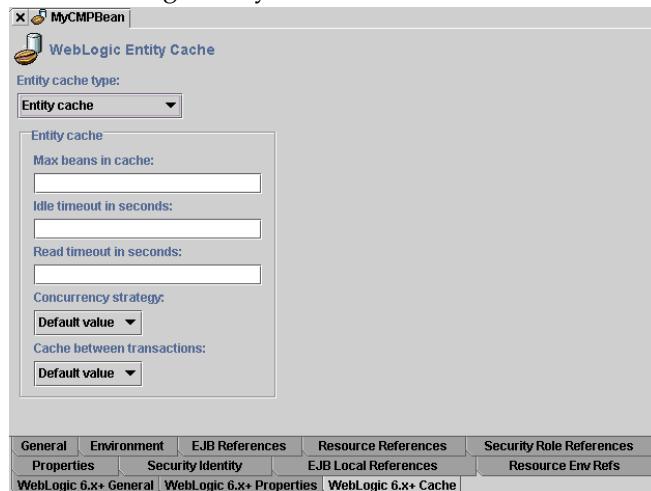
El descriptor de distribución se importa en el módulo EJB.

## Panel Caché de WebLogic 6.x o 7.x

Si el servidor de aplicaciones de destino es WebLogic 6.0 o posterior y se selecciona un bean entidad, se pone a disposición del desarrollador el panel Caché de WebLogic 6.x o 7.x:



Elija en la lista desplegable Entity Cache Type (Tipo de caché de entidad) un valor para el tipo de caché de entidad. Si se selecciona Entity Cache (Caché de entidad) o Entity cache reference (Referencia de caché de entidad) aparecen más campos. Por ejemplo, éste es el aspecto del panel cuando se elige Entity Cache:



Si desea más información sobre cachés de entidad, consulte la documentación de WebLogic.

## Transacciones de contenedor

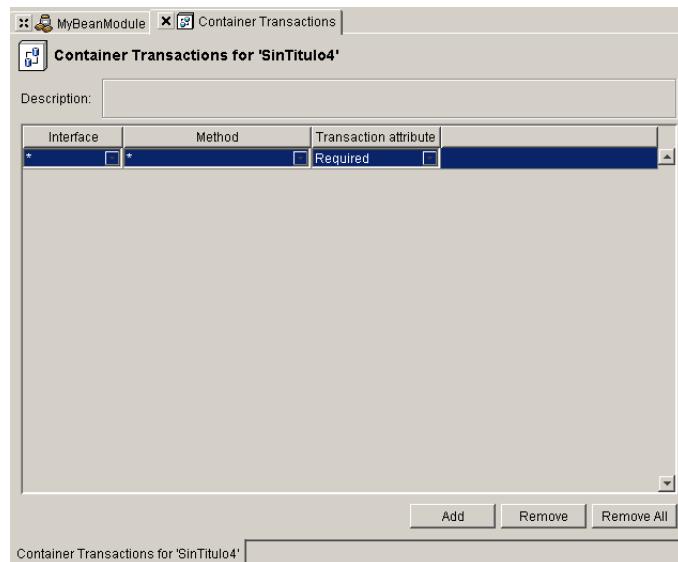
Los enterprise beans que utilizan transacciones gestionadas por contenedor deben tener métodos de transacción establecidos por el contenedor. El editor del descriptor de distribución permite definir estas normas de transacción y asociarlas a métodos de las interfaces base remota, base local, remota y local del enterprise bean.

### Definición de las normas de transacción del contenedor

Para definir una norma de transacción del contenedor para uno o varios métodos:

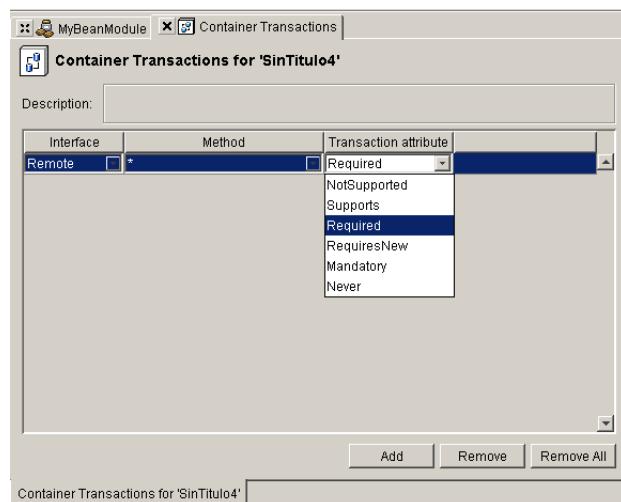
- 1 Haga doble clic en el enterprise bean en el panel de proyecto para ampliar su nodo:

- 2 Haga doble clic en Container Transactions en el panel de proyecto.



- 3 Pulse el botón Add (Añadir) para añadir una fila a la rejilla.
- 4 Seleccione la fila Interface que expone el método o elija \*, que indica todas las interfaces. Lo elegido en este campo determina las opciones del campo Method (Método).
- 5 Seleccione el método para el que desea definir la norma de transacciones en la lista desplegable Methods disponible, o elija \* para seleccionar todos los métodos de las interfaces elegidas en el campo Interface.

- 6** Seleccione el atributo deseado en la lista desplegable Transaction Attributes (Atributos de transacción).



Los enterprise beans gestionados por contenedor tienen atributos de transacción asociados a los métodos o a su totalidad. El valor de atributo indica al contenedor cómo gestionar las transacciones relacionadas con el bean. Existen varios atributos distintos, que el ensamblador o el desarrollador de aplicaciones puede asociar a los distintos métodos de los beans:

- **Required**

Garantiza que el trabajo que efectúa el método asociado se encuentra dentro del contexto de transacción global. Si el método de llamada ya tiene un contexto de transacciones, el contenedor también la utiliza. Si no lo tiene, el contenedor comienza una transacción automáticamente. Este atributo facilita la creación de beans múltiples y la coordinación del trabajo de todos los beans que utilizan la misma transacción global.

- **RequiresNew**

Se utiliza cuando no se desea que el método se asocie a una transacción. Garantiza que el contenedor siempre da comienzo a una transacción.

- **Supports**

Permite que el método no utilice una transacción global. Se debe utilizar este atributo sólo cuando un método de un bean accede sólo a un recurso de transacción o a ninguno y no llama a otro enterprise bean. Dado que este atributo evita el coste asociado a las transacciones globales, su uso optimiza el bean. Si se define este atributo y ya existe una transacción global, el contenedor llama al

método y se une a la transacción. Si no hay transacciones globales, el contenedor inicia una transacción local para el método y se completa al final de éste.

- **NotSupported**

También permite que el bean no utilice una transacción global. Si un cliente llama al método con un contexto de transacciones, el contenedor lo suspende. Al final del método se reanuda la transacción global.

- **Mandatory**

El cliente que llama a un método que tiene este atributo de transacción debe tener ya una transacción asociada. De lo contrario, el contenedor lanza una excepción `javax.transaction.TransactionRequiredException`. Este atributo hace que el bean sea menos flexible, porque se presuponen características de la transacción del método que efectúa la llamada.

- **Never**

El cliente que llama a un método que tiene este atributo de transacción no debe tener contexto de transacción. Si lo tiene, el contenedor lanza una excepción.

- 7 Si lo desea, puede describir la transacción escribiéndola en el campo Description. La descripción es optativa.

## **Panel Aislamiento de transacciones de WebLogic 6.x o 7.x**

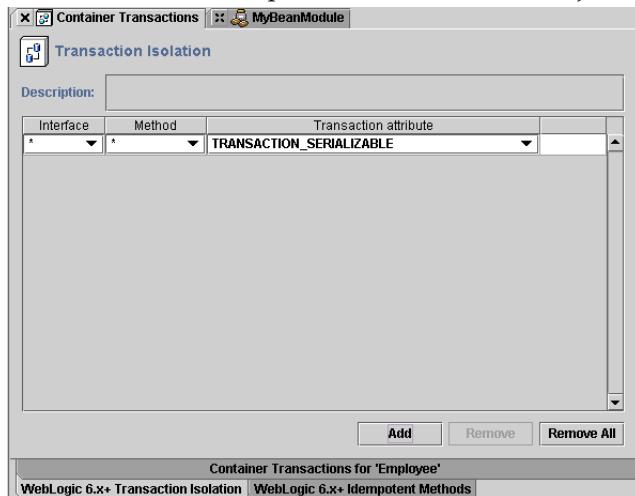
---

Si el servidor de aplicaciones es WebLogic Server 6.x o 7.x, existe un panel de aislamiento de transacciones de WebLogic 6.x o 7.x en el que se pueden configurar los criterios de aislamiento de transacciones de los métodos.

Para definir una norma de aislamiento de transacciones de los métodos en el bean:

- 1 Haga doble clic en el enterprise bean en el panel de proyecto para ampliar su nodo:
- 2 Haga doble clic en Container Transactions en el panel de proyecto.
- 3 Haga clic en la pestaña WebLogic 6.x o 7.x Transaction Isolation del editor de descriptor de distribución.

- 4 Pulse el botón Añadir para añadir una fila a la rejilla.



- 5 Seleccione en la fila la interfaz (Interface) que expone el método: base, base local, remota o local. Elija \* si desea seleccionar todas las interfaces. Lo elegido en este campo determina las opciones del campo Method (Método).
- 6 Seleccione el método para el que desea definir la norma de aislamiento de transacciones en la lista desplegable Methods disponible, o elija \* para seleccionar todos los métodos de las interfaces elegidas en el campo Interface.
- 7 Seleccione en la lista desplegable el atributo de transacción que describa la norma de aislamiento que desea aplicar a los métodos especificados. Si desea más información para realizar la selección, consulte “[Definición de los niveles de aislamiento](#)” en la página 13-39.
- 8 Si lo desea, puede describir la transacción escribiéndola en el campo Description. La descripción es optativa.

Si desea más información sobre la definición de los niveles de aislamiento, consulte la documentación de WebLogic.

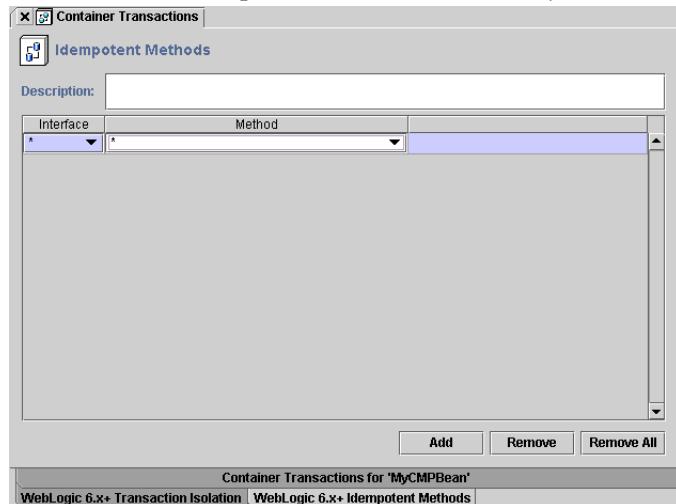
## Panel Métodos idempotentes de WebLogic 6.x o 7.x

Si el servidor de aplicaciones de destino es WebLogic 6.x o 7.x, está disponible el panel Métodos Idempotentes de WebLogic 6.x o 7.x, en el que se pueden definir los métodos que se deben considerar como tales.

Para definir métodos idempotentes en el bean:

- 1 Haga doble clic en el enterprise bean en el panel de proyecto para ampliar su nodo.

- 2** Haga doble clic en Container Transactions en el panel de proyecto.
- 3** Pulse la pestaña WebLogic 6.x o 7.x Idempotent del editor de descriptor de distribución.
- 4** Pulse el botón Add para añadir una fila a la rejilla.



- 5** Seleccione en la fila la interfaz (Interface) que expone el método: base o remota. Elija \* si desea seleccionar todas las interfaces. Lo elegido en este campo determina las opciones del campo Method (Método).
- 6** Seleccione el método que desea definir como idempotente en la lista desplegable Methods disponible, o elija \* para seleccionar todos los métodos de las interfaces elegidas en el campo Interface.
- 7** Si lo desea, puede describir la transacción escribiéndola en el campo Description. La descripción es optativa.

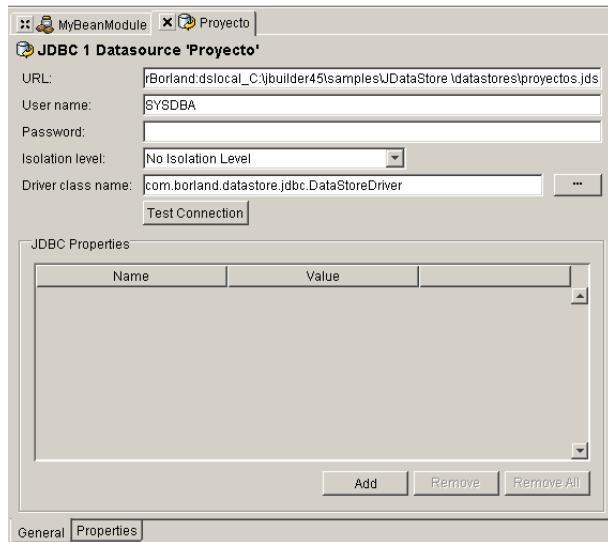
Si desea información sobre los métodos idempotentes, consulte la documentación de WebLogic.

## Las fuentes de datos JDBC 1

---

Si desea ver información sobre una fuente de datos JDBC 1 en el descriptor de distribución, amplíe el nodo DataSources (Fuentes de datos) del panel de proyecto y haga doble clic en una de las fuentes de datos. (El nodo DataSources no está disponible hasta que se añade una fuente de datos.)

Se abre el panel General del editor de descriptor de distribución. Este panel permite modificar la información de la fuente de datos seleccionada.



El Editor de descriptor de distribución permite elegir fuentes de datos para los beans entidad y establecer el nivel de aislamiento para las transacciones de datos.

Para añadir una fuente de datos al descriptor de distribución:

- 1 Haga clic con el botón derecho sobre el nodo JDBC del panel de proyecto y elija New Datasource (Nueva fuente de datos) en el menú que aparece.

Aparece el cuadro de diálogo New DataSource (Nueva fuente de datos).

- 2 Introduzca el nombre para la nueva fuente de datos y pulse OK (Aceptar).

La nueva fuente de datos se añade al árbol del panel de estructura.

- 3 Haga doble clic en la nueva fuente de datos, en el panel de proyecto.
- 4 Introduzca la información necesaria.

La fuente de datos está definida por su nombre, su ubicación URL y, en caso necesario, el nombre de usuario y la contraseña que permiten acceder a ella. El panel incluye también el nombre de clase del controlador JDBC y las propiedades JDBC.

- 5 Después de especificar la conexión de la fuente de datos se puede pulsar el botón Probar conexión.

El Editor de descriptor de distribución intenta establecer una conexión con la fuente de datos indicada. El resultado se recoge en el registro de mensajes.

## Definición de los niveles de aislamiento

El término *nivel de aislamiento* se refiere al grado en el que se evita que las transacciones múltiples, intercaladas, interfieran entre sí en las bases de datos de varios usuarios. Éstas son posibles transgresiones de la transacción:

- **Lectura de datos no confirmados:** La transacción t1 modifica una fila, y a continuación, la transacción t2 la lee. Después, t1 efectúa una cancelación y t2 ha visto una fila que en realidad no ha llegado a existir.
- **Lectura no repetible:** La transacción t1 recupera una fila. La transacción t2 la actualiza y t1 vuelve a recuperarla. Ahora, la transacción t1 ha recuperado dos veces la misma fila y ha visto en ella dos valores distintos.
- **Lecturas fantasma:** La transacción t1 lee un conjunto de filas que cumplen determinadas condiciones de búsqueda. Después, la transacción t2 inserta filas que cumplen la misma condición. Si la transacción t1 repite la lectura, detectará filas que no existían anteriormente. Estas filas se denominan fantasmas.

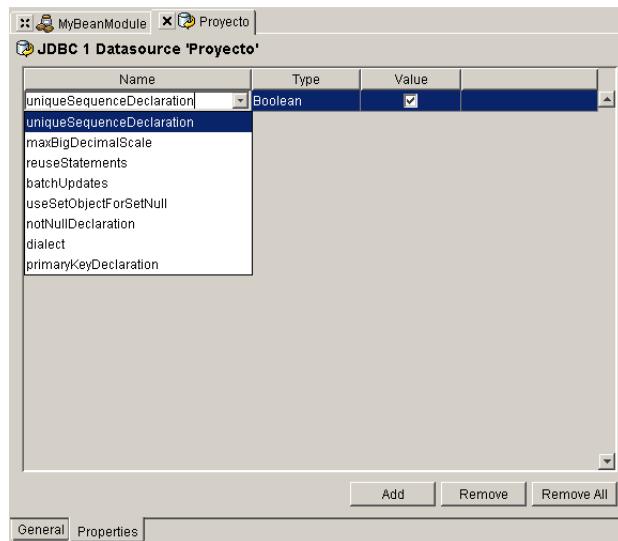
Si desea definir o cambiar el nivel de aislamiento de transacciones para una fuente de datos, seleccione uno de estos niveles de aislamiento en la lista desplegable Nivel de aislamiento:

Atributo	Sintaxis	Descripción
Uncommitted	TRANSACTION_READ_UNCOMMITTED	Permite las tres transgresiones.
Committed	TRANSACTION_READ_COMMITTED	Permite lecturas no repetibles y fantasmas, pero no lecturas modificadas.
Repeatable	TRANSACTION_REPEATABLE_READ	Permite las filas fantasma, pero no las otras dos transgresiones.
Serializable	TRANSACTION_SERIALIZABLE	No acepta ninguna de las tres transgresiones.

## Definición de las propiedades de fuente de datos

Cuando se selecciona una fuente de datos en el Editor de descriptor de distribución aparece, además de la pestaña General, otra llamada Properties (Propiedades). El panel de propiedades permite definir

propiedades que afectan al motor de persistencia gestionada por contenedor (CMP) de Borland.



Para modificar las propiedades de una fuente de datos:

- 1 Haga doble clic en la fuente de datos del panel de proyecto.
- 2 Haga clic en la pestaña Properties.
- 3 Elija la propiedad que desea definir en la lista desplegable Name (Nombre) del panel de propiedades.  
El valor Type (Tipo) se define automáticamente, según lo seleccionado.
- 4 Elija un valor para la propiedad en la columna Value.
- 5 Añada propiedades pulsando el botón Add (Añadir) para agregar filas, y seleccione los elementos Name (Nombre) y Value (Valor) para ellas.

Éstas son las propiedades posibles:

- maxBigDecimalScale

Si utiliza JDBC 1.0, el valor de esta propiedad determina la escala que se debe utilizar en las llamadas al método:

```
java.sql.BigDecimal java.sql.ResultSet.getBigDecimal(int columnIndex, int scale);
```

En JDBC 2.0 no se utiliza ningún valor de escala cuando se llama a getBigDecimal(int columnIndex) .

- **uniqueSequence**

Determina si el motor de persistencia gestionada por contenedor debe declarar una sola secuencia para las columnas de la clave principal. Normalmente esto se consigue declarando las columnas adecuadas como claves principales (consulte “[primaryKeyDeclaration](#)” en la [página 13-42](#)).

- **batchUpdates**

Indica si el motor de persistencia gestionada por contenedor debe enviar lotes de actualizaciones a la base de datos. Esto puede suponer una ventaja significativa en lo tocante al rendimiento de las transacciones que actualizan varias entidades, y se debe utilizar si el controlador lo acepta. Desafortunadamente, la mayoría no acepta las actualizaciones todavía. El valor por defecto es false.

- **useSetObjectForSetNull**

Normalmente, cuando se asigna el valor null a una columna SQL se emplea este método:

```
void java.sql.PreparedStatement.setNull(int parameterIndex, int sqlType);
```

Dado que algunos controladores de JDBC no lo aceptan, se puede utilizar este indicador para hacer que el motor CMP utilice en su lugar el siguiente método:

```
void java.sql.PreparedStatement.setObject(int parameterIndex, Object x);
```

“null” se convierte en el valor de x.

- **reuseStatements**

Determina si el motor de persistencia gestionada por contenedor debe volver a utilizar las sentencias preparadas de unas transacciones a otras. La reutilización de las sentencias preparadas tiene un efecto significativo sobre el rendimiento, y se debe emplear excepto si se reutilizan los objetos del controlador JDBC. El valor por defecto es true.

- **notNullDeclaration**

Determina si los campos de Java cuyo valor no puede ser null (como `int` y `float`) deben asignarse a columnas cuyo valor no sea null. El valor por defecto es true.

- **dialect**

Determina el tipo de la fuente de datos (JDataStore, Oracle, Informix, etc.). Seleccione el valor de dialect en la lista desplegable Value (Valor). Si no se define este campo, el motor de persistencia gestionada por contenedor sólo crea tablas para JDataStore. El valor por defecto es none.

- primaryKeyDeclaration

Determina si el motor de persistencia gestionada por contenedor declara las columnas de clave principal de la tabla como claves principales. Algunas bases de datos no aceptan las declaraciones de clave principal. El valor por defecto es true.

## Adición de competencias de seguridad y permisos para métodos

---

El Editor de descriptor de distribución permite crear y modificar las competencias de seguridad del descriptor de distribución. Después de crear las competencias, se les pueden asociar métodos de las interfaces base, remota, local y base local del enterprise bean, con lo que se define la vista de seguridad de la aplicación.

En este apartado se describe la forma de utilizar el Editor de descriptor de distribución para crear las competencias de seguridad y asignarles métodos de enterprise beans. En “[Panel Security Role References](#)” en la [página 13-15](#) se describe la forma de utilizar el panel de competencias para asignar a éstas grupos y cuentas de usuarios.

La definición de competencias de seguridad en el descriptor de distribución es optativa.

### Creación de competencias de seguridad

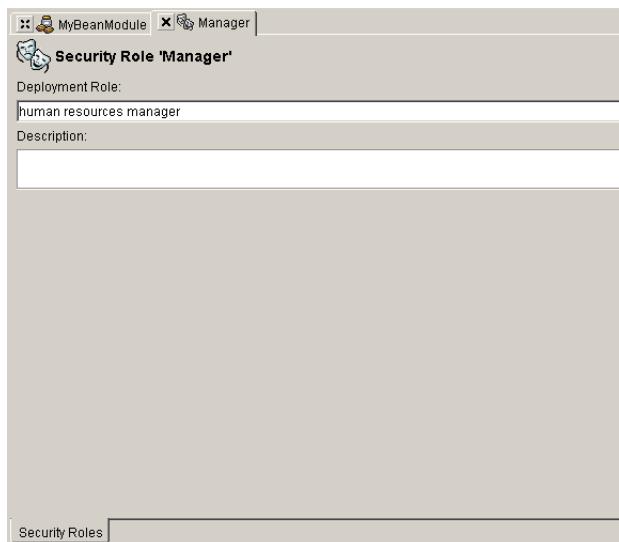
---

Para crear una competencia de seguridad en el descriptor de distribución:

- 1 Haga clic con el botón derecho del ratón en el nodo de la carpeta Security Roles del panel de proyecto. Elija New Role (Nueva competencia) en el menú contextual.
- 2 Escriba en el cuadro de diálogo que aparece el nombre de la competencia de seguridad y pulse OK (Aceptar).

La nueva competencia aparece en el nodo Security Roles (Competencias de seguridad) del panel de proyecto. Amplíe el nodo Security Roles para

verlo. Haga doble clic en la nueva competencia, en el panel de proyecto, para abrir el panel Security Roles.



Se puede introducir una descripción de la competencia en el panel Security Roles, si se desea.

Si el servidor de aplicaciones seleccionado es WebLogic Server 5.1 o posterior, en su lugar aparece el panel Security Roles de WebLogic 5.1, de WebLogic 6.x o de WebLogic 7.x+ (Competencias de seguridad de WebLogic 5.1, de WebLogic 6.x o de WebLogic 7.x). Por ejemplo, éste es el panel WebLogic 7.x Security Roles:



Introduzca los nombres principales, separados por comas, en el campo Principal Names. El campo Description es optativo.

- WebLogic 7.0 con service pack 1** El panel también muestra la casilla de selección Global. Cuando se marca esta casilla se puede declarar el nombre de la competencia modificado basado en las competencias de seguridad del reino/asignación principal.

## Asignación de permisos para métodos

Después de definir las competencias de seguridad se pueden indicar los métodos de las interfaces del enterprise bean a los que la competencia puede llamar.

No es necesario asociar las competencias de seguridad con métodos de las interfaces de los beans. En tal caso, ninguna de las competencias de seguridad definidas en el descriptor de distribución puede llamar a estos métodos.

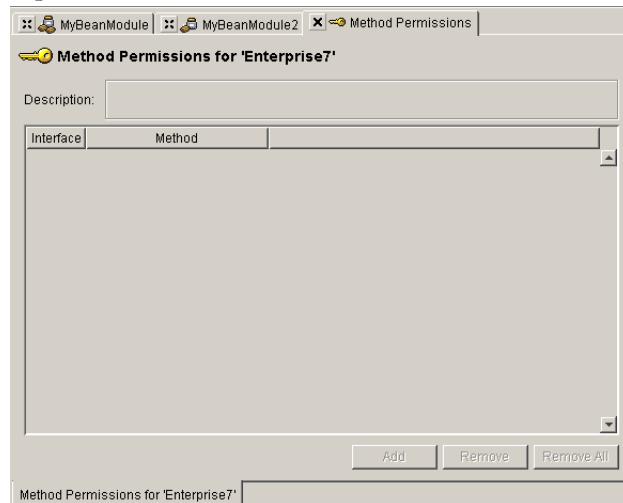
Para asignar permisos para los métodos:

- 1 Amplíe el nodo del bean en el panel de proyecto para mostrar el subnodo Permisos para los métodos.
- 2 Haga doble clic en el nodo Method Permission para que se abra el panel.

Cada competencia de seguridad definida aparece como una cabecera de columna.

- 3 Haga clic en el botón Add (Añadir) para añadir una fila al panel.

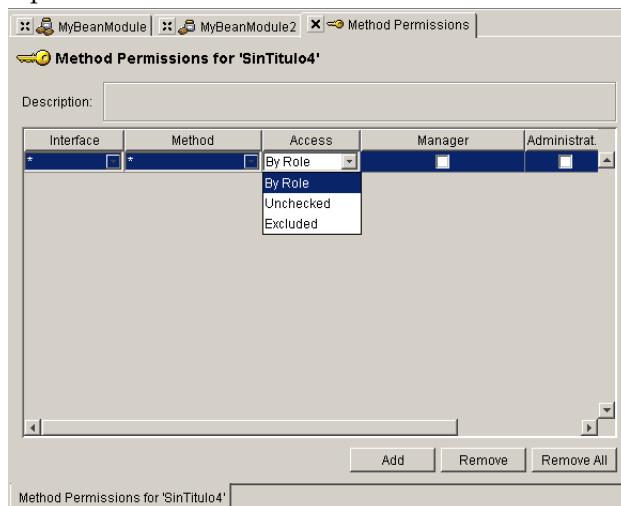
Si el bean es EJB 1,1, el panel Method Permissions tiene el siguiente aspecto:



Para los beans 1.1, siga estas instrucciones:

- 1 Elija Home (Base) o Remote (Remota) en la nueva fila, para indicar con qué interfaz se trabaja. Elija \* para seleccionar las dos interfaces. Lo elegido en la columna Interface (Interfaz) determina los métodos disponibles en la columna Method (Método).
- 2 Seleccione en la lista desplegable Method el método al que desea que esté permitido efectuar llamadas, o elija \* para autorizar las llamadas a todos los métodos.
- 3 Active las casillas de selección de las distintas competencias de seguridad para conceder permiso para llamar a los métodos especificados.
- 4 Por último, escriba si lo desea una descripción en el campo Description, para indicar los permisos que define la fila.

Si el bean es EJB 2.0, el panel Method Permissions tiene el siguiente aspecto:



Para los beans 2.0, siga estas instrucciones:

- 1 Elija Home (Base), Remota (Remote), Local o LocalHome (Base local) en la nueva fila, para indicar con qué interfaz se trabaja. Elija \* para seleccionar todas las interfaces. Lo elegido en la columna Interface (Interfaz) determina los métodos disponibles en la columna Method (Método).
- 2 Seleccione en la lista desplegable Method el método al que desea que esté permitido efectuar llamadas, o elija \* para autorizar las llamadas a todos los métodos.
- 3 Seleccione en la lista desplegable Access (Acceso) la forma en que desea establecer el acceso al método.

- By Role (Por competencia): Se les concede permiso para acceder al método a todas las competencias de seguridad verificadas.
  - Unchecked (Desactivado): Indica que los principales de cualquier competencia pueden acceder al método.
  - Excluded (Excluido): Ninguna competencia puede llamar al método.
- 4 Si selecciona By Role, active las casillas de selección de las distintas competencias de seguridad para conceder permiso para llamar a los métodos especificados.
- 5 Por último, escriba si lo desea una descripción en el campo Description, para indicar los permisos que define la fila.

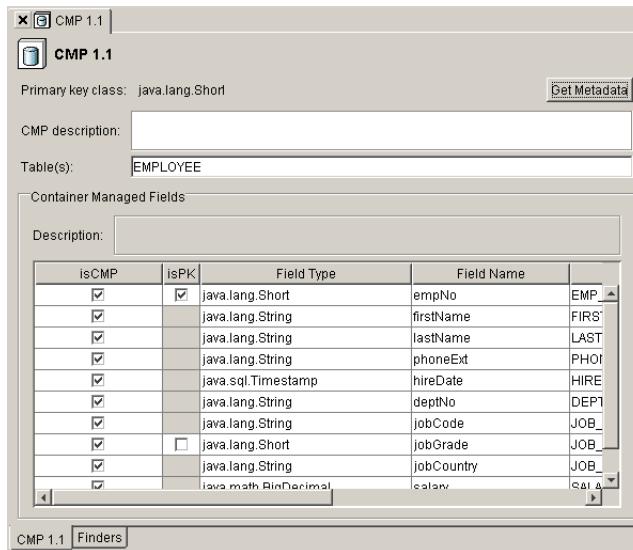
## Cómo añadir persistencia gestionada por contenedor para componentes EJB 1.1

CMP 1.1 especifica la forma en que el contenedor gestiona la persistencia de los beans entidad. El panel permite asignar campos del bean a columnas de la base de datos.

Para abrir el panel CMP 1.1:

- 1 Amplíe un nodo de bean entidad EJB 1.1 en el panel de proyecto.
- 2 Haga doble clic en el nodo CMP 1.1 en el panel de proyecto.

Aparece el panel CMP 1.1:



El panel CMP 1.1 incluye los siguientes campos:

- **Primary Key Class (Clase de clave principal):** Nombre completo de la clase de clave principal del bean entidad. La clave principal de los beans entidad se define en el panel General del editor del descriptor de distribución.
- **Botón Get Metadata (Obtener metadatos):** Al hacer clic en este botón, se recuperan los metadatos de la tabla y se rellena una lista desplegable para cada celda Nombre de la columna. Cada elemento de la lista desplegable es una pareja nombre de la columna/tipo de la columna. Al seleccionar de la lista desplegable se llenan las celdas nombre de la columna y tipo de la columna. La lista desplegable sólo incluye los nombres de las columnas que todavía no se han utilizado en el panel.
- **CMP Description (Descripción CMP):** Descripción de la persistencia gestionada por contenedor definida en este panel. Este campo es optativo. Tras escribir la descripción, pulse sobre la fila o, de lo contrario, la perderá.
- **Table(s) (Tablas):** Nombre de las tablas de base de datos a las que hace referencia el bean.
- **Description (Descripción):** Descripción de una fila seleccionada de la tabla. Este campo se activa únicamente cuando hay una fila seleccionada. Este campo es optativo.
- **isCMP:** Una marca de verificación indica que el campo está gestionado por el contenedor.
- **isPK:** Una marca de verificación indica que el campo es la clave principal.
- **Field Type (Tipo del campo):** Tipo de datos del campo.
- **Field Name (Nombre del campo):** El nombre del campo. La columna Field Name (Nombre de campo) enumera todos los campos del bean entidad.
- **Column Name(s) (Nombre(s) de la columna):** Los campos compuestos del bean (por ejemplo, location.street) se pueden asignar a columnas de la base de datos. Es posible asignar el campo de raíz (por ejemplo, location) o los subcampos (por ejemplo, location.street), pero no los dos.
- **Column Type (Tipo de columna):** Tipo de datos del campo.
- **EJB Reference (Referencia EJB):** Si el tipo de campo es una clase EJB aparece un menú con una lista de referencias EJB entre las que se puede elegir. Estas referencias se definen en el panel de referencias EJB.

El panel muestra el nombre de la clase de clave principal, que no se puede cambiar. En el campo CMP Description (Descripción de CMP) se puede introducir una descripción del bean.

Todos los campos del bean entidad gestionado por contenedor tienen activado el campo CMP. Se debe introducir en cada campo el nombre de la columna a la que corresponde. Si se ha utilizando el modelador de bean entidad EJB 1.x, JBuilder ya ha asignado las columnas. Si se desea, es posible modificar Column Name y Column Type (el nombre y el tipo de columna). En el campo Description (Descripción) se puede escribir texto para describir los campos, pero no es necesario.

El editor del descriptor de distribución utiliza JDBC para obtener metadatos sobre las tablas. Es posible asociar beans entidad a tablas, después de haber creado unos y otras. Por ejemplo, se puede comprar un enterprise bean de terceros con el fin de utilizarlo con una tabla de la base de datos.

Para llenar los campos Column Name (Nombre de la columna) y Column Type (Tipo de la columna), pulse el botón Get Meta Data (Obtener metadatos).

## Panel Finders

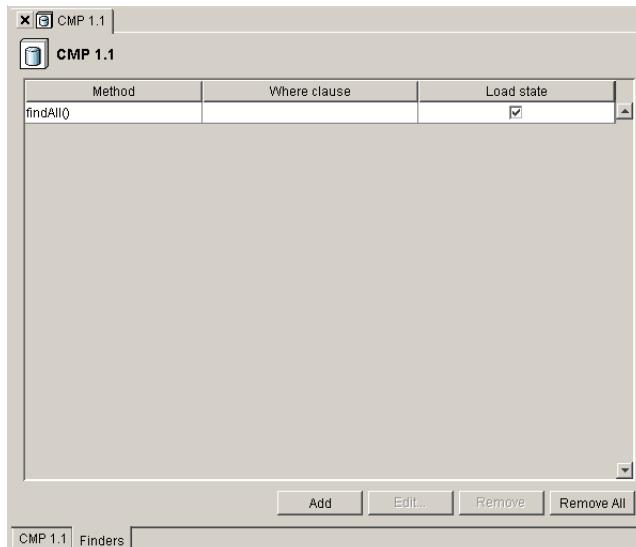
---

El panel Finders (de buscadores) indica las cláusulas "where" que utiliza el bean gestionado por contenedor para ejecutar métodos de búsqueda definidos por el bean. Este panel está disponible para los beans entidad EJB 1.1 con persistencia gestionada por contenedor.

Para abrir el panel Finders:

- 1 Amplíe un nodo de bean entidad EJB 1.1 en el panel de proyecto.
- 2 Haga doble clic en el nodo CMP 1.1 en el panel de proyecto.
- 3 Haga clic en la pestaña Finders de la parte inferior del Editor de descriptor de distribución.

Aparece el panel Finders:



Si ha utilizado el modelador de beans entidad EJB 1.x para crear el bean entidad, debe activar Generate findAll() Method in Home Interface (Generar el método findAll() en la interfaz base) en la última ficha del asistente o, de lo contrario, el botón Add (Añadir) estará desactivado en el panel Finders (Buscadores).

En el panel de buscadores se puede encontrar la siguiente información:

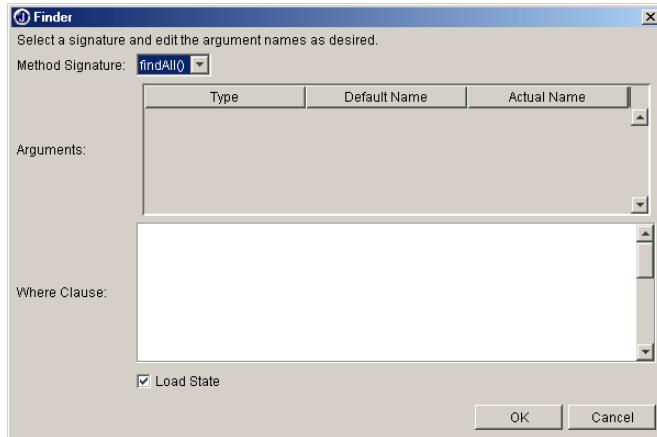
- **Method (Método):** Nombre del método de búsqueda y lista de todos sus parámetros.
- **Where clause (Cláusula Where):** Indica una cláusula SQL “where”, que el contenedor utiliza para recuperar registros de la base de datos. Tenga en cuenta que no es posible convertir todas las sentencias SQL a la lógica de consultas WebLogic.
- **Load state (Cargar estado):** Este atributo permite al contenedor cargar previamente todos los campos que gestiona cuando ocurre una operación de búsqueda.

Para definir un método de búsqueda:

- 1 Pulse el botón Añadir. Este botón Add sólo está disponible si se ha definido un método de buscador en el bean.  
Aparece el cuadro de diálogo Finder (Buscador).
- 2 Seleccione en la lista desplegable la firma del método de búsqueda deseado.

- 3 Modifique los nombres de los argumentos si lo desea, e indique la cláusula Where adecuada para la operación de búsqueda.

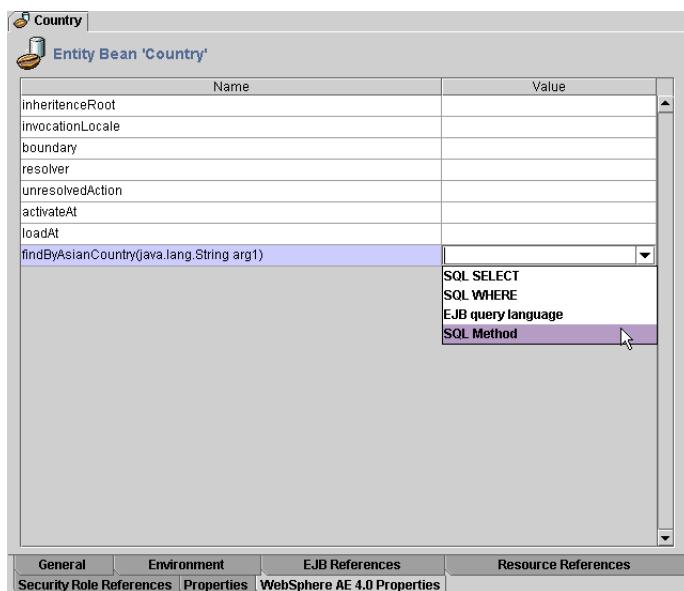
A continuación, se propone un ejemplo.



### Definición de buscadores WebSphere 4.0

La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

En el panel WebSphere 4.0 Properties (Propiedades del WebSphere 4.0), los buscadores del bean aparecen al final de la lista de propiedades. Es posible elegir el tipo de consulta para la que se desea utilizar el buscador: SQL SELECT, SQL WHERE, EJB query language (Lenguaje de consultas EJB) o SQL Method (Método SQL):



Seleccione el tipo de buscador deseado. El valor por defecto es una cláusula WHERE (SQL WHERE). El tipo de buscador seleccionado se añade al descriptor de distribución propio de WebSphere `ibm-ejb-jar-ext.xml`. A continuación, se puede volver al panel Finders (Buscadores) y definir la búsqueda utilizando la consulta del tipo seleccionado en el panel WebSphere 4.0 Properties (Propiedades de WebSphere 4.0) como valor de la cláusula Where, aunque el tipo de consulta no sea una cláusula WHERE.

## Comprobación de la información sobre descriptores

---

Después de modificar el archivo de descriptores puede comprobar que la información se encuentra en el formato correcto, que existen los archivos de clase de beans necesarios, etc.

Para verificar la información del descriptor, pulse el módulo EJB con el botón derecho del ratón y elija Verify (Verificar) en el menú contextual.

Verify realiza las operaciones siguientes:

- Comprueba que el descriptor cumple la especificación EJB.
- Comprueba que las clases a las que hacen referencia los descriptores de distribución cumplen la especificación EJB.

Verify sólo debe usarse en los servidores de aplicaciones de Borland.



# 14

## Uso de DataExpress para componentes EJB

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

JBuilder cuenta con varios componentes que permiten recuperar datos de beans entidad en los DataSets de DataExpress (suministrar), y guardar datos de DataSets de DataExpress en los beans entidad (almacenar). Estos DataExpress para componentes EJB facilitan la implementación de una pauta de diseño Session Bean wrap Entity Bean o Session Facade. Si se utiliza esta pauta de diseño, los clientes no acceden directamente a los beans entidad, sino que acceden a ellos utilizando beans sesión. Los beans sesión, que se colocan junto a los beans entidad, realizan todas las llamadas a éstos mediante una única transacción y devuelven todos los datos de una sola vez. Los DataSets ofrecen un modo de transportar los datos desde el bean sesión al cliente y viceversa. Así, el rendimiento mejora, gracias a que los datos se envían a través del cable una sola vez para proporcionárselos al cliente y, a continuación, también una sola vez para guardar los cambios en los beans entidad del servidor.

Estos componentes también le facilitan generar aplicaciones clientes con la ayuda de componentes visuales enlazados a DataExpress, tales como dbSwing o InternetBeans Express. Para una descripción completa de DataExpress, consulte la *Guía del desarrollador de aplicaciones de bases de datos*.

Este capítulo explica cómo utilizar los componentes para transmitir datos de beans entidad distribuidos en un servidor a la aplicación de su cliente y viceversa. Este código se parece mucho al del proyecto de ejemplo / <jbuilder>/samples/Ejb/Ejb11/EjbDx.jpx. Los datos a los que se tiene acceso con el ejemplo están almacenados en un almacén de datos Employee. El ejemplo crea un bean entidad para mantener los datos Employee. También crea un bean sesión Personnel que recupera datos del bean Employee y, a

continuación, los envía al cliente. El cliente devuelve los datos al bean Personnel, que los almacena en las instancias del bean entidad Employee.

- Nota** Si utiliza el JDK 1.4, no podrá ver datos activos con los componentes DataExpress para EJB del Diseñador de interfaces de usuario cuando utilice Borland Enterprise Server.

## Los componentes EJB DataExpress

---

Seis de los componentes EJB DataExpress se encuentran en la ficha EJB de la paleta de componentes. Puede trabajar con estos componentes en el diseñador de interfaces, y configurar las propiedades y sucesos con la ayuda del Inspector. Existen más clases a las que puede llamar su código con las que no se trabaja de forma visual. Si desea más información acerca de todas las clases, consulte el apartado Referencia de la API.

### Componentes para el servidor

---

Los dos componentes de la ficha EJB de la paleta de componentes que utiliza el bean sesión distribuido en el servidor son el EntityBeanProvider y el EntityBeanResolver. EntityBeanProvider proporciona datos de los beans entidad distribuidos en el servidor, mientras que EntityBeanResolver almacena datos en esos beans entidad. Estos componentes se añaden al bean sesión que haya creado para que éste pueda suministrar y almacenar información en los beans entidad.

Si crea enterprise beans que se ejecutan en un contenedor EJB 1.x debe seguir utilizando los componentes EntityBeanProvider y EntityBeanResolver. Si los beans se van a ejecutar en un contenedor EJB 2.0 se deben utilizar en su lugar los componentes LocalEntityBeanProvider y LocalEntityBeanResolver. Estos componentes tienen la propiedad ejbLocalHome en vez de ejbHome. También tienen la propiedad ejbLocal, que toma la clase de la interfaz del bean entidad que implementa EJBLocalObject. Todos los sucesos y monitores de los componentes EntityBeanProvider y EntityBeanResolver tienen versiones locales correspondientes en LocalEntityBeanProvider y LocalEntityBeanResolver.

### Componentes para el cliente

---

Dos de los componentes de la ficha EJB se utilizan en el ordenador cliente: EjbClientDataSet y SessionBeanConnection. El componente EjbClientDataSet suministra datos y guarda los cambios en el bean sesión al que se refiere el componente SessionBeanConnection. Un componente SessionBeanConnection mantiene la referencia al bean sesión en el servidor e incluye los nombres de los métodos para suministrar conjuntos de datos y guardarlos en ese bean sesión.

## Creación de beans entidad

---

Comience por utilizar el modelador de bean entidad EJB 1.x para crear los beans entidad que dan acceso a los datos que le interesan. El proyecto modelo crea los beans entidad Employee y Department, aunque este capítulo sólo se refiere al bean Employee.

Si desea más información sobre la creación de beans entidad EJB 1.1, consulte el [Capítulo 9, “Creación de beans entidad EJB 1.x a partir de una tabla de base de datos”](#). Si desea más información sobre la creación de beans entidad EJB 2.0, consulte el [Capítulo 7, “Creación de beans entidad 2.0 con el diseñador de EJB”](#).

## Creación del bean sesión en el servidor

---

Cree el bean sesión que se albergará en el servidor. Para crear beans sesión sin estado se puede utilizar el Asistente para Enterprise JavaBean 1.x o el diseñador de EJB. Más adelante en la siguiente sección se añaden a este bean las clases EntityBeanProvider y EntityBeanResolver. Ya que estas clases no son serializables, resulta fácil colocarlas en un bean sesión sin estado que, en el caso del BAS, nunca se desactivan. Si necesita un bean sesión con estado para su aplicación, debe hacer que éste haga referencia a un bean sesión sin estado, o bien reiniciar las clases EntityBeanProvider y EntityBeanResolver cada vez que se active el bean sesión con estado.

Así es como quedaría la clase resultante del bean llamada PersonnelBean:

```
public class PersonnelBean implements SessionBean {
    private SessionContext sessionContext;
    public void ejbCreate() {
    }
    public void ejbRemove() throws RemoteException {
    }
    public void ejbActivate() throws RemoteException {
    }
    public void ejbPassivate() throws RemoteException {
    }
    public void setSessionContext(SessionContext sessionContext) throws
        RemoteException {
        this.sessionContext = sessionContext;
    }
}
```

Pulse sobre la pestaña Diseño para visualizar el diseñador de interfaces de usuario.

## Adición de componentes proveedores y almacenadores al bean sesión

---

En la ficha EJB de la paleta de componentes, añada un EntityBeanProvider y un EntityBeanResolver al bean sesión. Si se trata de un enterprise bean 2.0, añada en su lugar un LocalEntityBeanProvider y un LocalEntityBeanResolver. También debe añadir un componente dataset para mantener los datos recogidos de los beans entidad antes de que se envíen al cliente y los datos que devuelve el cliente. En la ficha DataExpress de la paleta de componentes, añada un componente TableDataSet y cambie el nombre de TableDataSet por otro más adecuado.

Así es como quedaría la parte superior de PersonnelBean. A TableDataSet se le ha cambiado el nombre por el de employeeDataSet:

```
public class PersonnelBean implements SessionBean {
    private SessionContext sessionContext;
    EntityBeanProvider entityBeanProvider = new EntityBeanProvider();
    EntityBeanResolver entityBeanResolver = new EntityBeanResolver();
    TableDataSet employeeDataSet = new TableDataSet();
    ...
}
```

Con la ayuda del Inspector, asigne a las propiedades provider y resolver del TableDataSet a los nuevos componentes añadidos EntityBeanProvider y EntityBeanResolver, respectivamente. Como resultado, hay dos nuevos métodos en el método jbInit():

```
employeeDataSet.setProvider(entityBeanProvider);
employeeDataSet.setResolver(entityBeanResolver);
```

El proyecto de ejemplo muestra estos métodos en el método setSessionContext(). Si prefiere copiar exactamente el proyecto de ejemplo, puede añadir las llamadas a los métodos en setSessionContext(). Cualquiera de las dos propuestas es válida.

Para los miembros de esta clase, añada una referencia a la interfaz base del bean entidad que contenga los datos a los que desea tener acceso. En este ejemplo, la referencia se hace a la interfaz base del bean entidad Employee, tal y como se muestra en negrita.

```
public class PersonnelBean implements SessionBean {
    private SessionContext sessionContext;
    EntityBeanProvider entityBeanProvider = new EntityBeanProvider();
    EntityBeanResolver entityBeanResolver = new EntityBeanResolver();
    TableDataSet employeeDataSet = new TableDataSet();
    EmployeeHome employeeHome;
    ...
}
```

## Escritura del método setSessionContext()

---

Añada un bloque try en el método sessionContext() del bean sesión. Modifique el método para que quede del siguiente modo:

```
public void setSessionContext(SessionContext sessionContext)
    throws RemoteException {
    this.sessionContext = sessionContext;
    try {
        Context context = new InitialContext();
        Object object = context.lookup("java:comp/env/ejb/Employee");
        employeeHome = (EmployeeHome) PortableRemoteObject.narrow(object,
            EmployeeHome.class);
        entityBeanProvider.setEjbHome(employeeHome);
        entityBeanResolver.setEjbHome(employeeHome);
    }
    catch (Exception ex) {
        throw new EJBException(ex);
    }
}
```

Observe que el método setSessionContext() asigna al valor de las propiedades ejbHome de los componentes EntityBeanProvider y EntityBeanResolver el nombre de la interfaz base del bean entidad Employee.

### Cómo añadir referencias EJB o referencias EJB locales al descriptor de distribución

Debe añadir una referencia EJB a Personnel en el descriptor de distribución con el fin de que funcione la búsqueda. En el caso de los beans entidad con interfaces locales, debe añadir una referencia EJB local en su lugar. Puede utilizar el Editor de descriptor de distribución:

- 1 En el panel de proyecto, haga doble clic en el nodo del módulo EJB. En el proyecto de ejemplo, éste es personnel.ejbgrpx.
- 2 Haga doble clic en el bean Personnel, en el panel de proyecto. Aparece el Editor de descriptor de distribución.
- 3 Pulse sobre la pestaña Referencias EJB (o en la pestaña Referencias EJB locales ) en el editor del descriptor de distribución.
- 4 Haga clic en el botón Añadir para añadir una referencia al bean entidad que contenga los datos que le interesan.
- 5 Escriba el nombre de la referencia. En el proyecto de ejemplo, el nombre es ejb/Employee.
- 6 Marque la casilla de selección IsLink.
- 7 Especifique el bean entidad en la lista desplegable Enlace. Los demás datos se deberían llenar automáticamente.

## Adición de los métodos de suministro y almacenamiento

---

Debe añadir dos métodos al bean sesión, un método de suministro y otro de almacenamiento. Los nombres de estos métodos utilizan el valor que especificó como valor de la propiedad `methodName` en el componente `EjbClientDataSet`. De este modo, el suministrador para `PersonnelBean` se convierte en `provideEmployee()` y el almacenador en `resolveEmployee()`.

El suministrador debe llamar al método de una clase `EntityBeanConnection` que suministre los datos de un bean entidad a un conjunto de datos que se puedan enviar por cable. El método `provideEmployee()` debe quedar como sigue:

```
public DataSetData [] provideEmployee(RowData [] parameterArray,
    RowData [] masterArray) {
    return EntityBeanConnection.provideDataSets(new StorageDataSet []
        {employeeDataSet}, parameterArray, masterArray);
}
```

El almacenador debe llamar al método de una clase `EntityBeanConnection` que almacene las actualizaciones en los beans entidad. Así debe quedar `resolveEmployee()`:

```
public DataSetData [] resolveEmployee(DataSetData[] dataSetdataArray) {
    return EntityBeanConnection.saveChanges(dataSetdataArray,
        new DataSet [] {employeeDataSet});
}
```

A continuación, añada estos métodos a la interfaz remota. La forma más simple de llevarlo a cabo para los beans EJB 1.x sería a través de BeansExpress. Con el archivo fuente del bean abierto en el editor, haga clic en la pestaña Bean, a continuación en la pestaña Métodos y, por último, marque las casillas de selección de los nombres de los dos métodos que acaba de añadir. Para beans EJB 2.0, utilice el diseñador de EJB. Pulse sobre el método que acaba de añadir en la representación del bean del diseñador de EJB para que aparezca su inspector. Una vez abierto el inspector, seleccione Remota o Local en la lista desplegable Interfaces.

Ahora puede activar la interfaz remota del bean sesión (o la interfaz local en el caso de un bean 2.0 si esa es la interfaz que ha elegido para definir los métodos) para verificar que se han definido los dos métodos. Si decide definir los métodos en la interfaz remota, así es como quedaría:

```
public interface Personnel extends EJBObject {
    public com.borland.dx.dataset.DataSetData[]
        providePersonnel(com.borland.dx.ejb.RowData[] parameterArray,
            com.borland.dx.ejb.RowData[] masterArray) throws RemoteException;
    public com.borland.dx.dataset.DataSetData[]
        resolvePersonnel(com.borland.dx.dataset.DataSetData[] dataSetdataArray)
            throws RemoteException;
}
```

## Llamadas a métodos de búsqueda

---

Debe indicar a EntityBeanProvider qué beans entidad debe suministrar. Para ello, añada un suceso a EntityBeanProvider:

- 1 Si se encuentra en el diseñador de interfaces de usuario, seleccione EntityBeanProvider en el panel de estructura.
- 2 Haga clic en la pestaña sucesos del Inspector y, a continuación, haga clic dos veces en la columna vacía junto al suceso findEntityBeans. Se añade un suceso.

El suceso resultante es:

```
entityBeanProvider1.addEntityBeanFindListener(new
    com.borland.dx.ejb.EntityBeanFindListener() {
        public void findEntityBeans(EntityBeanFindEvent e) {
            entityBeanProvider1_findEntityBeans(e);
        }
    });
    ...
void entityBeanProvider_findEntityBeans(EntityBeanFindEvent e) {
```

- 3 Para el manejador del nuevo suceso, añada un método de búsqueda con el fin de que devuelva los beans entidad que desee. En este caso, el código añadido aparece en negrita:

```
void entityBeanProvider_findEntityBeans(EntityBeanFindEvent e) {
    try {
        e.setEntityBeanCollection(employeeHome.findAll());
    }
    catch (Exception ex) {
        throw new EJBException(ex);
    }
}
```

En este ejemplo, el manejador de sucesos llama a un método findAll() para que devuelva todos los beans entidad. Puede llamar a cualquier método de búsqueda. También puede utilizar la propiedad parameterRow de EntityBeanProvider para determinar de forma dinámica a qué método de búsqueda hay que llamar y/o qué parámetros hay que pasar.

Para almacenar, EntityBeanResolver puede decidir de forma predeterminada cómo hay que aplicar las actualizaciones y las eliminaciones. Sin embargo, no puede determinar de forma automática cómo crear beans entidad nuevos porque no hay forma de que pueda determinar a qué método create() llamar y qué parámetros pasarse. Por lo tanto, si desea añadir una fila a la fuente de datos, debe añadir el suceso create y suministrar la lógica necesaria. Puede utilizar el Inspector para añadir el código esqueleto del suceso create a su bean sesión. Puede ver

un ejemplo de un suceso `create` en el proyecto de ejemplo `EjbDx.jpx`. También puede utilizar los otros sucesos disponibles en `EntityBeanResolver` para redefinir el comportamiento predeterminado, si lo desea.

Distribuya los beans entidad y los beans sesión en el servidor de aplicaciones. Para obtener más información sobre los tipos de bean sesión, consulte “[Distribución en un servidor de aplicaciones](#)” en la página 12-7.

## Generación del lado cliente

---

Una vez que ha creado los beans entidad y el bean sesión que da acceso a ellos y los ha distribuido en su servidor de aplicaciones de destino, puede comenzar a generar el cliente.

Siga estos pasos:

- 1 Creación de un módulo de datos. Seleccione Archivo | Nuevo | Módulo de datos.
- 2 En la paleta de componentes, seleccione `EjbClientDataSet` y añádalo al módulo de datos.
- 3 En la paleta de componentes, seleccione `SessionConnectionBean` y añádalo al módulo de datos.
- 4 En el Inspector, asigne a la propiedad `sessionBeanConnection` de `EjbClientDataSet` el nombre del componente `SessionBeanConnection`.
- 5 En el Inspector, especifique un nombre para la propiedad `methodName` del componente `EjbClientDataSet`.

La propiedad `methodName` determina cómo se nombra a los métodos que suministran y almacenan datos. Por ejemplo, si especifica un valor de `Employee` para `methodName`, los métodos del bean sesión para suministrar y almacenar datos se convierten en `provideEmployee()` y `resolveEmployee()`. Más adelante necesitará añadir estos métodos al bean sesión que cree.

- 6 En el Inspector o directamente en el código fuente, configure la propiedad `jndiName` del componente `SessionBeanConnection`. También puede especificar el nombre de la interfaz remota del bean sesión que cree en lugar del valor de la propiedad `sessionBeanRemote`.

Puede utilizar el Inspector para añadir un suceso `creating` a su `SessionBeanConnection`. El código que añada al manejador de sucesos puede controlar la creación de un bean sesión una vez que ha tenido lugar la búsqueda JNDI. Normalmente, debe añadir un suceso `creating` si desea llamar a un método `create()` de la interfaz base que requiera parámetros. Por ejemplo, examine este código:

```
import com.borland.dx.dataset.*;
import com.borland.dx.ejb.*;
```

```

public class PersonnelDataModule implements DataModule {
    private static PersonnelDataModule myDM;
    SessionBeanConnection sessionBeanConnection = new SessionBeanConnection();
    EjbClientDataSet personnelDataSet = new EjbClientDataSet();

    public PersonnelDataModule() {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception{
        try {
            sessionBeanConnection.setJndiName("Personnel");
            sessionBeanConnection.addCreateSessionBeanListener(new
                com.borland.dx.ejb.CreateSessionBeanListener() {
                    public void creating(CreateSessionBeanEvent e) {
                        sessionBeanConnection_creating(e);
                    }
                });
            personnelDataSet.setSessionBeanConnection(sessionBeanConnection);
            personnelDataSet.setMethodName("Personnel");
        }
        catch (Exception ex) {

        }
    }

    public static PersonnelDataModule getDataModule() {
        if (myDM == null) {
            myDM = new PersonnelDataModule();
        }
        return myDM;
    }

    public com.borland.dx.ejb.SessionBeanConnection getSessionBeanConnection() {
        return sessionBeanConnection;
    }

    public com.borland.dx.ejb.EjbClientDataSet getPersonnelDataSet() {
        return personnelDataSet;
    }

    void sessionBeanConnection_creating(CreateSessionBeanEvent e) {

    }
}

```

## Tratamiento de relaciones

---

El EntityBeanProvider aplana automáticamente las relaciones. Por ejemplo, si tiene un bean entidad `Employee` con un método `getDept()` que devuelve un `Dept`, siendo `Dept` un bean entidad remoto, se crea un `DataSet` que tiene todos los campos en el bean entidad `Employee`, además de todos los campos en el bean entidad `Dept`, incluidas las columnas ocultas con las claves principales de cada uno de los beans entidad. Excepto en el caso de `Dept.ejbPrimaryKey`, los demás campos de `Dept` serán de sólo lectura.

Si desea almacenar datos en el caso de una relación uno-a-uno, debe añadir un monitor de sucesos al `EntityBeanProvider`, ya que no se puede determinar de forma dinámica el directorio raíz del bean entidad relacionado. El proyecto de ejemplo `EjbDx.jpx` no manifiesta el tratamiento de las relaciones.

## El proyecto de ejemplo

---

Hasta aquí ha visto cómo transferir datos de forma eficaz desde y hacia el cliente y el servidor. El proyecto de ejemplo `/<jbuilder>/samples/Ejb/Ejb11/EjbDx.jpx` muestra cómo utilizar las técnicas descritas con un cliente Java que utiliza controles dbSwing y con un cliente Web que utiliza tecnología JSP combinada con InternetBeans Express. Podrá trabajar con datos activos. Busque en la página `EjbDx.html` del proyecto las instrucciones completas para ejecutar el proyecto de ejemplo.

En el proyecto `/<jbuilder>/samples/Ejb/Ejb11/EjbDxWL/EjbDxWL.jpx` se muestra la forma de ejecutar el mismo proyecto con un servidor de aplicaciones distinto. En este caso se emplea BEA WebLogic Server 6.x. En el archivo `EjbDxWL.html` se describen las diferencias entre los dos proyectos y lo que se debe hacer para ejecutar el proyecto con WebLogic 6.x.

# 15

## Desarrollo de beans sesión

Las herramientas para EJB de JBuilder pueden simplificar enormemente la creación de enterprise beans y sus interfaces. Para poder modificar los archivos creados por JBuilder y entender lo que hace el programa, es necesario comprender los requisitos de estas clases e interfaces. Esto es lo que se explica en los siguientes capítulos.

Normalmente, los beans sesión existen mientras dura una sola sesión con el cliente. Sus métodos efectúan un conjunto de tareas o procesos para el cliente que utiliza el bean. Los beans sesión existen solamente durante la conexión con el cliente. En cierto modo, los beans sesión representan al cliente en el servidor EJB. Suelen proporcionar un servicio al cliente. Si no se necesita trabajar con datos persistentes de un almacén, normalmente se utilizan beans sesión.

### Tipos de beans sesión

Existen dos tipos de beans sesión: los que pueden mantener información sobre el estado entre llamadas a métodos, que se denominan beans *con estado* y los que no pueden, que se denominan beans *sin estado*.

#### Beans sesión con estado

Los beans sesión con estado son objetos que utilizan un solo cliente, y mantienen su estado con los datos correspondientes de éste. Por ejemplo, consideremos un bean sesión de carro de la compra. A medida que el comprador de la tienda en línea selecciona los artículos, va añadiéndolos al carro de la compra; esto es, los elementos seleccionados se almacenan en una lista dentro del objeto de bean sesión de carro de la compra.

Cuando el comprador está listo para adquirir los objetos se utiliza la lista para calcular el precio total.

## Beans sesión sin estado

---

Estos beans sesión no mantienen el estado para ningún cliente específico, por lo que pueden utilizarlos muchos clientes. Por ejemplo, consideremos un bean sesión para ordenar, que contiene un método empresarial `sortList()`. El cliente llamaría a `sortList()` y le pasaría una lista de elementos sin ordenar. `sortList()` pasaría al cliente la lista ordenada.

## Escritura de la clase del bean sesión

---

Para crear una clase de bean sesión:

- Cree una clase que sea una extensión de la interfaz `javax.ejb.SessionBean`.
- Implemente uno o varios métodos `ejbCreate()`. Si está creando un bean sesión sin estado, la clase implementa solamente un método `ejbCreate()` sin parámetros. Si ya ha creado la interfaz base remota o base local del bean, éste tendrá un método `ejbCreate()` con la misma firma para todos los métodos `create()` de la interfaz base remota/base local.
- Defina e implemente los métodos empresariales que desea tenga el bean. Si ya ha creado la interfaz remota o local del bean, los métodos deben definirse tal y como están en la interfaz remota/local.

Las herramientas EJB de JBuilder pueden iniciar automáticamente estas tareas, incluida la creación de las interfaces local y remota. Crean una clase que es una ampliación de la interfaz `SessionBean` y escriben implementaciones vacías de los métodos `SessionBean`. La implementación se rellena si el bean la requiere. En el apartado siguiente se explica qué son estos métodos y cómo se utilizan.

## Implementación de la interfaz SessionBean

---

La interfaz `SessionBean` define los métodos que deben implementar todos los beans sesión. Es una ampliación de la interfaz `EnterpriseBean`.

```
package javax.ejb;
public interface SessionBean extends EnterpriseBean {
    void setSessionContext(SessionContext sessionContext)
        throws EJBException, RemoteException;
    void ejbRemove() throws EJBException, RemoteException;
    void ejbActivate() throws EJBException, RemoteException;
    void ejbPassivate() throws EJBException, RemoteException;
}
```

Los métodos de la interfaz SessionBean están estrechamente relacionados con el ciclo vital de un bean sesión. En esta tabla se explica su finalidad:

Método	Descripción
setSessionContext()	Define un contexto de sesión. El contenedor del bean llama a este método para asociar una instancia de un bean sesión a su contexto. La interfaz de contexto de la sesión proporciona métodos para acceder a las propiedades de ejecución del contexto en que se ejecuta una sesión. Normalmente, los beans sesión retienen el contexto en un campo de datos.
ejbRemove()	Indica que un objeto de sesión está a punto de eliminarse. El contenedor llama a este método cuando elimina un bean sesión con estado como resultado de una llamada del cliente al método <code>remove()</code> de la interfaz remota/local o base remota/local.
ejbActivate()	Indica que un objeto de sesión con estado se ha activado.
ejbPassivate()	Indica que el contenedor está a punto de desactivar un objeto de sesión con estado.

Los métodos `ejbActivate()` y `ejbPassivate()` permiten a los beans sesión con estado gestionar recursos. Para obtener más información, consulte “[Beans con estado](#)” en la página 15-7.

## Escritura de métodos empresariales

Dentro de la clase del enterprise bean, escriba implementaciones completas de los métodos empresariales que necesita el bean mediante el editor de código de JBuilder. Para que estos métodos estén a disposición del cliente también es necesario declararlos en la interfaz remota del bean, exactamente de la misma forma que en la clase del bean. Puede utilizar las herramientas EJB de JBuilder para realizar esta tarea. Si utiliza el diseñador de EJB para crear componentes EJB 2.0, los métodos se declaran de forma correcta en la interfaz remota/local si se utiliza el inspector del bean para especificar dónde deben declararse los métodos. Consulte “[Creación de beans sesión](#)” en la página 6-9. Si va a crear componentes EJB 1.1, utilice el diseñador de beans de JBuilder con el fin de asegurarse de que los métodos se declaran correctamente en el método remoto del bean. Consulte “[Exposición de métodos empresariales mediante la interfaz remota](#)” en la página 8-11.

## Adición de métodos ejbCreate()

Si utiliza las herramientas EJB de JBuilder para crear el enterprise bean, se añade a la clase del bean un método `ejbCreate()` sin parámetros. Puede añadir otros métodos `ejbCreate()` con parámetros. Aunque los beans sesión sin estado no necesitan más que un método `ejbCreate()` sin parámetros, los que tienen estado necesitan a menudo varios métodos

ejbCreate() con parámetros. Tenga presentes estas reglas al escribir métodos ejbCreate() con parámetros:

- Los métodos ejbCreate() deben declararse como públicos.
- Ninguno de ellos debe devolver nada.
- Los parámetros del método ejbCreate() deben ser del mismo número y tipo que los del método create() correspondiente de la interfaz remota del bean. En el caso de beans sesión sin estado, sólo puede haber un método ejbCreate() sin parámetros.

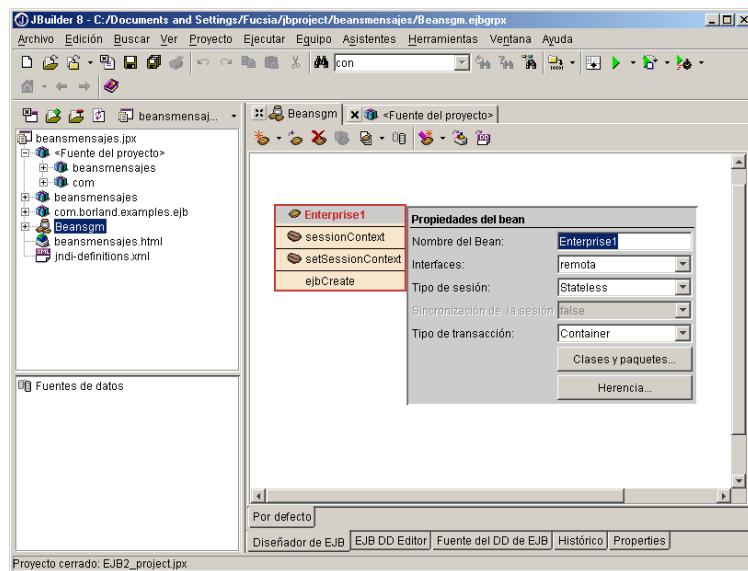
Éste es el rasgo distintivo de todos los métodos ejbCreate() de un bean sesión:

```
public void ejbCreate( <ceros o más parámetros> ) {  
    // implementación  
}
```

El método ejbCreate() no necesita lanzar excepciones, aunque puede lanzar excepciones de aplicaciones y de otro tipo, como, por ejemplo, javax.ejb.CreateException. Las herramientas EJB de JBuilder generan un método ejbCreate() que lanza la excepción javax.ejb.CreateException.

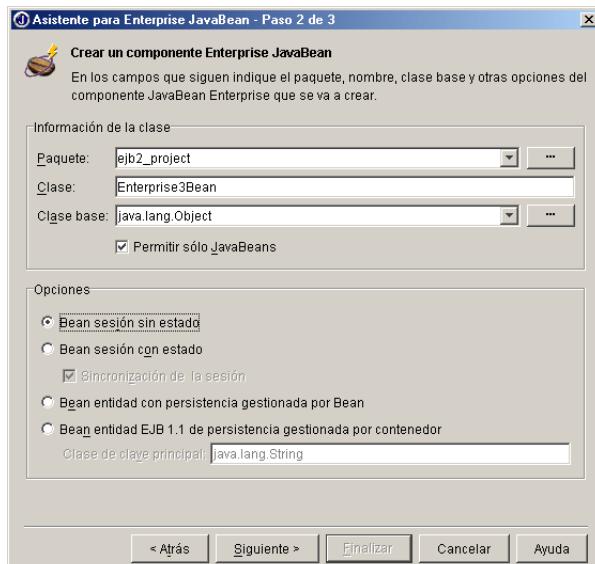
## JBuilder en la creación de beans sesión

Si va a crear un bean sesión EJB 2.0, comience por pulsar con el botón derecho del ratón en el panel Diseñador de EJB, y seleccione en el menú contextual Crear EJB | Bean sesión. También puede pulsar el ícono Crear EJB de la barra de herramientas del diseñador de EJB y, a continuación, seleccionar Bean sesión. Aparece una representación del bean sesión en el diseñador de EJB con un inspector para modificar sus atributos:



Utilice el inspector para seleccionar si el bean sesión tiene o no estado, y para configurar otros atributos. Si desea más información acerca de la creación de beans sesión con el diseñador de EJB 2.0, consulte “[Creación de beans sesión](#)” en la página 6-9.

En el Asistente para Enterprise JavaBean de JBuilder se puede dar comienzo a la creación de beans sesión eligiendo la opción Bean sesión sin estado o Bean sesión con estado en la segunda ficha del asistente.



Las herramientas EJB de JBuilder no solamente crean la clase del enterprise bean, sino que también crean las interfaces base y remota/local del bean mientras crean la clase. De esta forma se garantiza que el método `create()` de la interfaz base devuelva la interfaz remota/local, y que el método `ejbCreate()` nunca devuelva nada.

Al escribir los métodos empresariales en la clase del bean EJB 2.0, utilice el inspector para especificar en qué interfaz se declaran los métodos. El diseñador de EJB añade la declaración correcta en la interfaz adecuada. Si especifica una interfaz base, el diseñador de EJB declara el método como método empresarial base. El diseñador añade el prefijo `ejbHome` al nombre del método en la clase del bean, y declara el método sin el prefijo en la interfaz base.

Después de escribir los métodos empresariales en la clase de bean EJB 1.x, utilice el diseñador de beans para indicar los que desea definir en la interfaz remota del bean sesión EJB 1.1. Las aplicaciones clientes sólo pueden acceder a los métodos definidos en la interfaz remota. Después de que escriba los métodos a los que desea que pueda llamar el cliente, el diseñador de beans definirá los métodos de la interfaz remota.

Si ya tiene una clase de enterprise bean completa pero no tiene interfaces local y remota para ella, puede crearlas con el Asistente para interfaces EJB 1.x de JBuilder. Las firmas de los métodos cumplen las normas de EJB 1.1 en las interfaces base y remota, sin necesidad de que haya que corregirlas de forma manual.

Para obtener información acerca del uso de las herramientas EJB de JBuilder para crear session beans, consulte “[Creación de beans sesión](#)” en la página 8-6. Para obtener información acerca del uso de las herramientas EJB de JBuilder para crear session beans, consulte el Capítulo 6, “[Creación de beans sesión y gestionados por mensajes 2.0 con el diseñador de EJB](#)”.

## Ciclo de vida de los beans sesión

---

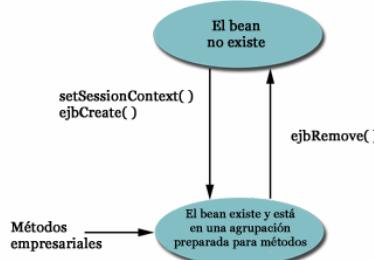
Los beans sesión con y sin estado tienen ciclos de vida distintos. Es necesario comprender lo que ocurre en cada uno.

### Beans sin estado

---

La vida de un bean sesión sin estado comienza cuando el cliente llama al método `create()` de la interfaz local del bean. El contenedor crea una instancia del bean sesión y devuelve una referencia de objeto al cliente.

**Figura 15.1** Ciclo de vida de un bean sesión sin estado



Durante el proceso de creación, el contenedor llama al método `setSessionContext()` de la interfaz `SessionBean`, y al método `ejbCreate()` de la implementación del bean sesión. El nuevo objeto de bean se une a un búfer de beans sin estado listos para que los usen los clientes. Dado que los objetos de sesión sin estado no mantienen un estado relacionado con el cliente, el contenedor puede asignar a cualquiera de ellos la gestión de las llamadas a métodos que se reciban. Cuando el contenedor elimina un objeto del búfer de beans sesión, llama al método `ejbRemove()` del objeto de bean.

Tenga en cuenta que cuando se llama al método `create()` o `remove()` de las interfaces base y remota/local no se añaden ni se eliminan objetos de bean.

sesión del búfer. El contenedor controla el ciclo de vida de los beans sin estado.

## Beans con estado

La vida de un bean sesión con estado comienza cuando el cliente llama al método `create()` de la interfaz local del bean. El contenedor crea una instancia del bean sesión, la inicializa y devuelve una referencia de objeto al cliente.

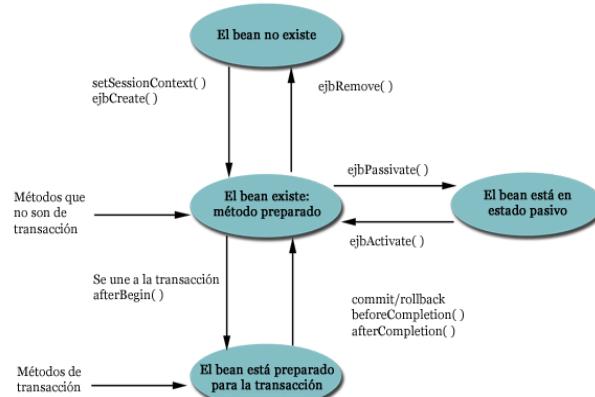
Durante el proceso de creación, el contenedor llama al método `setSessionContext()` de la interfaz `SessionBean`, y al método `ejbCreate()` de la implementación del bean sesión. El proveedor de los beans puede utilizar estos métodos para inicializar el bean sesión.

Ahora, el bean sesión está preparado para recibir métodos, lo que significa que puede efectuar operaciones no transaccionales o incluirse en transacciones para operaciones transaccionales. El bean permanece en este estado hasta que se da una de las siguientes circunstancias:

- El bean entra en una transacción.
- El bean se elimina.
- El bean se desactiva.

Cuando un cliente llama al método `remove()` de la interfaz remota/local o base, el contenedor llama al método `ejbRemove()` correspondiente del objeto de bean sesión. El proveedor de beans puede colocar en este método cualquier código de depuración específico de una aplicación. Cuando se ejecuta `ejbRemove()` ya no es posible utilizar el objeto bean. Si el cliente intenta llamar a un método del objeto de bean, el contenedor lanza la excepción `java.rmi.NoSuchObjectException`.

**Figura 15.2** Ciclo de vida de un bean sesión con estado



El contenedor puede desactivar la instancia del bean sesión. Esto ocurre normalmente por motivos de gestión de recursos, como cuando un objeto de sesión permanece inactivo durante cierto tiempo o si el contenedor requiere más memoria. El contenedor desactiva el bean llamando a su método `ejbPassivate()`. Cuando se desactiva una instancia de un bean, el contenedor guarda en el disco información de referencia y el estado del objeto de sesión, y libera la memoria asignada al bean. Se puede añadir código a `ejbPassivate()` si se desea ejecutar alguna tarea antes de desactivar el bean.

El contenedor activa el bean llamando a su método `ejbActivate()`. Esto ocurre cuando el cliente llama a un método del bean sesión desactivado. Durante la activación el contenedor recrea el objeto de sesión en la memoria y restablece su estado. Si desea que ocurra algo inmediatamente después de que el bean vuelva a activarse, añada el código al método `ejbActivate()`.

### Estado preparado para métodos en transacciones

Cuando un cliente llama a un método de un objeto de bean sesión en un entorno transaccional, el contenedor inicia una transacción o incluye el objeto de bean en una creada. El bean entra en el estado preparado para métodos. En el ciclo de vida de una transacción hay puntos de sincronización en los que se puede informar a un objeto de bean sesión de los sucesos de transacción próximos, para que pueda efectuar de antemano las acciones necesarias.

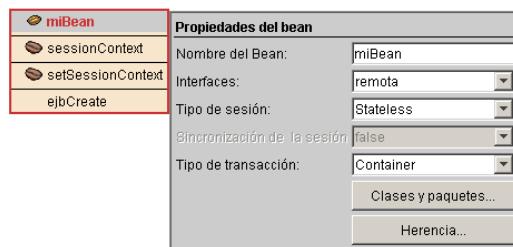
### La interfaz SessionSynchronization

Un bean sesión puede implementar la interfaz `SessionSynchronization` si se desea que reciba información sobre el estado de una transacción en la que tome parte. Solamente los beans sesión con estado que utilizan transacciones gestionadas por contenedor pueden implementar `SessionSynchronization`. Su utilización es optativa. Los métodos de `SessionSynchronization` son devoluciones de llamada que efectúa el contenedor al bean, y marcan puntos dentro de la transacción. Ésta es la interfaz `SessionSynchronization`:

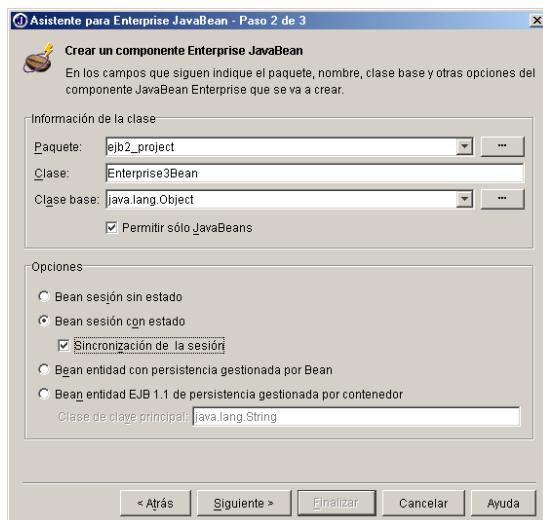
```
public interface javax.ejb.SessionSynchronization
{
    public abstract void afterBegin() throws RemoteException;
    public abstract void beforeCompletion() throws RemoteException;
    public abstract void afterCompletion(boolean completionStatus) throws
        RemoteException;
}
```

En el caso de los beans sesión EJB 2.0, utilice el inspector de beans del diseñador EJB para asignar el valor true al atributo `Session`

Synchronization, y para añadir los tres métodos de la interfaz SessionSyncronization a la clase del bean:



Para los beans sesión EJB 1.1, el asistente para Enterprise JavaBean 1.1 puede añadir estos métodos a la clase del bean. Active la casilla de selección Sincronización de la sesión y el asistente declarará los tres métodos vacíos en la clase del bean:



La siguiente tabla describe brevemente los métodos:

Método	Descripción
afterBegin()	Indica a la instancia del bean que una transacción está a punto de ser utilizado. El código que se escribe dentro de <code>afterBegin()</code> se ejecuta mientras dura la transacción.
beforeCompletion()	Indica a la instancia del bean que la transacción está a punto de enviarse. Si el bean tiene valores en caché, utilice <code>beforeCompletion()</code> para escribirlos en la base de datos. En caso necesario, un bean sesión puede utilizar el método <code>beforeCompletion()</code> para forzar la retirada de la transacción llamando al método <code>setRollbackOnly()</code> de la interfaz <code>SessionContext</code> .
afterCompletion()	Indica a la instancia del bean que la transacción ha concluido. Si se ha enviado la transacción, el parámetro <code>completionStatus</code> adquiere el valor <code>true</code> . Si se ha cancelado la transacción, este parámetro se establece en <code>false</code> .

Ésta es la forma en que se utilizan los métodos `SessionSynchronization`: El cliente llama a un método empresarial transaccional definido en la interfaz remota, que coloca el objeto bean en estado listo para transacciones. El contenedor llama al método `afterBegin()` del objeto de bean. Más adelante, si se envía la transacción, el contenedor llama a `beforeCompletion()`, y si el envío es correcto, a `afterCompletion(true)`. Si la transacción se cancela o no se envía por otro motivo, el contenedor llama a `afterCompletion(false)`. El objeto de bean sesión está de nuevo en el estado preparado para métodos.

Para más información acerca de la utilización de session beans en transacciones, consulte el [Capítulo 20, “Gestión de las transacciones”](#).

# 16

## Desarrollo de beans entidad

Los beans entidad representan directamente los datos que se encuentran en almacenes persistentes, como una base de datos. Se asignan filas a una o más tablas de bases de datos relacionales o a objetos de entidad de bases de datos orientadas a objetos. También se pueden asignar a filas de varias tablas. En las bases de datos, la clave principal identifica de forma única las filas de las tablas. De igual manera, identifica instancias determinadas de beans entidad. Cada columna de la base de datos relacional corresponde a una variable de instancia del bean entidad.

Dado que los beans entidad representan normalmente los datos almacenados en una base de datos, tienen la misma duración que los datos. El contenedor no elimina el bean entidad del almacenamiento persistente, independientemente del tiempo durante el cual permanezca inactivo.

Los beans entidad sólo se pueden eliminar de forma explícita. Para eliminar un bean entidad se llama a su método `remove()`, que borra los datos subyacentes de la base de datos. También se pueden eliminar los datos con una aplicación enterprise.

### La persistencia y los beans entidad

Todos los enterprise beans entidad son persistentes, lo que significa que su estado se almacena entre sesiones y clientes. El proveedor de los beans puede elegir la forma de implementar la persistencia.

La persistencia del bean se puede implementar directamente en su clase, con lo que el bean entidad pasa a ser responsable de mantener su persistencia. Esto se denomina *persistencia gestionada por bean*.

También es posible delegar la gestión de la persistencia del bean entidad al contenedor EJB. Esto se denomina *persistencia gestionada por contenedor*.

## Persistencia gestionada por bean

---

Los beans entidad que gestionan su persistencia contienen el código necesario para acceder a la base de datos y actualizarla, por lo que el proveedor debe escribir las llamadas a la base de datos directamente en el bean o en sus clases asociadas. Esas llamadas se suelen escribir con JDBC.

Las llamadas de acceso a la base de datos pueden aparecer en los métodos empresariales del bean o en uno de los métodos de su interfaz. (Más adelante se facilitan explicaciones detalladas sobre la interfaz de los beans entidad.)

Normalmente resulta más difícil programar la persistencia gestionada por bean, porque es necesario escribir el código adicional de acceso a datos. Como se puede incrustar el acceso a datos en los métodos del bean, también puede resultar más difícil adaptar el bean entidad a bases de datos o esquemas distintos.

## Persistencia gestionada por contenedor

---

En los beans entidad con persistencia gestionada por contenedor no es necesario escribir el código de acceso y actualización de las bases de datos. El contenedor se encarga de acceder a la base de datos y actualizarla.

La persistencia gestionada por contenedor tiene muchas ventajas respecto a la gestionada por bean:

- Resulta más fácil escribir el código.
- Se pueden cambiar los detalles de la persistencia sin necesidad de modificar y volver a compilar todo el bean. Basta con que el distribuidor o el ensamblador de aplicaciones modifique el descriptor de distribución.
- Se reduce la complejidad del código, y por ende, la posibilidad de error.
- El proveedor de beans puede concentrarse en la lógica empresarial del bean y olvidar los asuntos relacionados con el sistema subyacente.

Sin embargo, la persistencia gestionada por contenedor tiene ciertas limitaciones. Por ejemplo, el contenedor puede cargar todo el estado del objeto de bean entidad en los campos de instancia del bean antes de llamar al método `ejbLoad()`. Si el bean tiene muchos campos, esto puede ocasionar problemas de rendimiento.

## Claves principales de los beans entidad

---

Todas las instancias de los beans entidad deben tener una clave principal. La clave principal es un valor o una combinación de valores que identifica la instancia de forma irrepetible. Por ejemplo, en una tabla de base de datos que contiene registros de empleados se pueden utilizar los números de seguridad social como clave principal. El bean entidad que conforma esta tabla de empleados también utilizará el número de seguridad social como clave principal.

En los enterprise beans, la clave principal está representada por un tipo String o Integer o una clase Java que contiene los datos irrepetibles. La clase de la clave principal puede ser cualquiera mientras que sea de un tipo de valor legal en RMI-IIOP. Esto significa que la clase debe extender la interfaz `java.io.Serializable` y debe implementar los métodos `Object.equals(Other other)` y `Object.hashCode()`, heredados por todas las clases Java.

La clase de la clave principal puede ser exclusiva de una clase de bean entidad determinada, es decir, cada bean entidad puede definir su propia clase de clave principal, o varios pueden compartir la misma clase.

## Escritura de la clase del bean entidad

---

Para crear una clase de bean entidad:

- Cree una clase que implemente la interfaz `javax.ejb.EntityBean`.
- Implemente uno o varios métodos `ejbCreate()`. Si ya ha creado la interfaz base o base local del bean, éste tendrá un método `ejbCreate()` con la misma firma para todos los métodos `create()` de la interfaz. Si un método `ejbCreate()` tiene un parámetro, también se debe declarar e implementar el método `ejbPostCreate()`.
- Defina e implemente los métodos empresariales que desea tenga el bean. Si ya ha creado la interfaz remota o local del bean, los métodos deben definirse tal y como están en la interfaz remota o local.
- Implemente métodos de búsqueda en los beans entidad que gestionen su persistencia.

Las herramientas EJB de JBuilder pueden iniciar estas tareas. Crean una clase que es una ampliación de la interfaz `EntityBean` y escriben implementaciones vacías de los métodos `EntityBean`. La implementación se rellena si el bean la requiere. En el apartado siguiente se explica qué son estos métodos y cómo se utilizan.

Si desea generar beans entidad EJB 2.0 a partir de tablas de base de datos, importe el esquema al diseñador de EJB y utilícelo en la creación de los

beans entidad. Para obtener más información, consulte “[Creación de beans entidad CMP 2.0 a partir de una fuente de datos importada](#)” en la página 7-2.

Si desea utilizar tablas de base de datos para crear beans entidad EJB 1.x, utilice el modelador de beans entidad EJB 1.x. Para obtener más información, consulte “[Creación de beans entidad con el modelador de beans entidad EJB 1.x](#)” en la página 9-1.

## Implementación de la interfaz EntityBean

La interfaz EntityBean define los métodos que deben implementar todos los beans entidad. Es una ampliación de la interfaz EnterpriseBean.

```
public void EntityBean extends EnterpriseBean {
    public void setEntityContext(EntityContext ctx) throws EJBException,
        RemoteException;
    public void unsetEntitycontext() throws EJBException, RemoteException;
    void ejbRemove() throws RemoveException, EJBException, RemoteException;
    void ejbActivate() throws EJBException, RemoteException;
    void ejbPassivate() throws EJBException, RemoteException;
    void ejbLoad() throws EJBException, RemoteException;
    public void ejbStore() throws EJBException, RemoteException;
}
```

Los métodos de la interfaz EntityBean están estrechamente relacionados con el ciclo vital de un bean entidad. En esta tabla se explica su finalidad:

Método	Descripción
setEntityContext()	Define un contexto de entidad. El contenedor utiliza este método para pasar a la instancia del bean una referencia a la interfaz EntityContext. La interfaz EntityContext proporciona métodos para acceder a las propiedades de ejecución del contexto del bean entidad. Las instancias de beans entidad que utilizan este contexto deben almacenarlo en variables de instancia.
unsetEntityContext()	Libera los recursos asignados durante la llamada al método setEntityContext(). El contenedor llama a este método antes de poner fin a la instancia actual del bean entidad.
ejbRemove()	Elimina los elementos de base de datos asociados al bean entidad. El contenedor llama a este método cuando un cliente llama a remove().
ejbActivate	Indica que un bean entidad se ha activado. El contenedor llama a este método de la instancia seleccionada en el búfer de instancias disponibles y asignada a la identidad de un objeto de entidad determinado. Cuando se activa la instancia del bean tiene la oportunidad de adquirir los recursos adicionales que necesite.

Método	Descripción
ejbPassivate()	Indica que un bean entidad está a punto de desactivarse, es decir, que la asociación de la instancia con la identidad de un objeto de entidad está a punto de interrumpirse, con lo que la instancia volverá al búfer de disponibles. A continuación, la instancia puede liberar los recursos asignados al método <code>ejbActivate()</code> que no es necesario conservar en el búfer.
ejbLoad()	Actualiza los datos de la base de datos que representa el objeto entidad. El contenedor llama a este método de la instancia del bean entidad para que ésta sincronice el estado de entidad de sus variables con el estado de la entidad de la base de datos.
ejbStore()	Almacena los datos de la base de datos que representa el objeto entidad. El contenedor llama a este método de la instancia del bean entidad para que ésta sincronice el estado de entidad de la base de datos con el estado de la entidad de las variables de la instancia.

## Declaración e implementación de los métodos del bean entidad

Los beans entidad pueden tener tres tipos de métodos:

- Métodos de creación.
- Métodos de búsqueda.
- Métodos empresariales.

### Creación de métodos de creación

Si utiliza las herramientas EJB de JBuilder para dar comienzo a la creación del bean, verá que se añade un método `ejbCreate()` y otro `ejbPostCreate()`, sin parámetros, a la clase del bean. Si el bean requiere más métodos de creación, se pueden escribir.

Los beans entidad no necesitan métodos de creación. Cuando se llama a un método de creación de un bean entidad se añaden datos a la base de datos. Se pueden tener beans entidad sin método de creación si sólo se añaden instancias de objetos de entidad a la base de datos mediante actualizaciones del SGBD o desde aplicaciones anteriores.

### Método `ejbCreate()`

Tenga presentes estas reglas si decide añadir métodos `ejbCreate()` que incluyan parámetros:

- Los métodos `ejbCreate()` deben declararse como públicos.
- En los beans entidad gestionados por contenedor el método `ejbCreate()` debe devolver el valor null.

El contenedor es el único encargado de crear los beans entidad que gestiona.

- Si los beans entidad se gestionan a sí mismos, el método `ejbCreate()` debe devolver una instancia de la clase de clave principal del nuevo objeto entidad.

El contenedor utiliza esta clave principal para crear la referencia a la entidad.

- Los parámetros del método `ejbCreate()` deben ser del mismo número y tipo que los del método `create()` correspondiente de la interfaz remota del bean.
- Cada método `ejbCreate()` debe tener un método `ejbPostCreate()` que coincida con `ejbCreate()` en el número de parámetros.

La firma de los métodos `ejbCreate()` es la misma, independientemente de que la persistencia de los beans esté gestionada por ellos mismos o por los contenedores. Ésta es la firma de todos los métodos `ejbCreate()` de los beans entidad:

```
public <PrimaryKeyClass> ejbCreate( <zero or more parameters> )
    // implementación
}
```

Cuando el cliente llama al método `create()`, el contenedor responde ejecutando el método `ejbCreate()` e inserta en la base de datos un registro que representa el objeto de entidad. Normalmente, los métodos de `ejbCreate()` restablecen un estado de entidad, por lo que a menudo tienen uno o varios parámetros, y sus implementaciones incluyen código que define el estado de la entidad en los valores de los parámetros. Por ejemplo, la aplicación de banca que se trata más adelante en este capítulo contiene un bean entidad de cuenta corriente cuyo método `ejbCreate()` tiene dos parámetros: una cadena y un valor flotante. El método restablece el nombre de la cuenta en el valor de la cadena y el balance de la cuenta en el valor de coma flotante:

```
public AccountPK ejbCreate(String name, float balance) {
    this.name = name;
    this.balance = balance;
    return null;
}
```

### Método `ejbPostCreate()`

Cuando concluye la ejecución de un método `ejbCreate()` el contenedor llama al método `ejbPostCreate()` correspondiente para permitir que la instancia termine de inicializarse. Los parámetros de `ejbPostCreate()` coinciden con los de `ejbCreate()`, pero devuelve un valor vacío:

```
public void ejbPostCreate( <zero or more parameters> )
    // implementación
}
```

Siga estas reglas para definir el método `ejbPostCreate()`:

- Se debe declarar como público.
- No se puede declarar como final o estático.
- Su tipo de retorno debe estar vacío.
- Su lista de parámetros debe coincidir con la del método `ejbCreate()` correspondiente.

Utilice `ejbPostCreate()` para efectuar cualquier proceso especial que necesite el bean antes de quedar a disposición del cliente. Si el bean no requiere proceso especial, deje vacío el cuerpo del método, pero no olvide incluir un `ejbPostCreate()` por cada `ejbCreate()` en los beans entidad que gestionan su propia persistencia.

### Creación de métodos de búsqueda

Los beans entidad deben tener métodos de búsqueda. Los clientes utilizan estos métodos para buscar los beans. Los beans entidad que gestionan su persistencia deben tener un método `ejbFindByPrimaryKey()` con su correspondiente `findByPrimaryKey()` en la interfaz local. Ésta es la firma del método `ejbFindByPrimaryKey()`:

```
public <PrimaryKeyClass> ejbFindByPrimaryKey(<PrimaryKeyClass primaryKey>)
    // implementación
}
```

Es posible definir más métodos de búsqueda para el bean. Por ejemplo, se puede crear un método `ejbFindByLastName()`. Los métodos de búsqueda deben seguir estas reglas:

- Se debe declarar como público.
- Su nombre debe empezar por el prefijo **ejbFind**.
- No se puede declarar como estático ni como final.
- En el caso de los beans EJB 1.1, se debe devolver una clave principal, una colección de claves principales o una enumeración de claves principales. En el caso de los beans EJB 2.0, se debe devolver una colección `java.util.Collection` de `EJBObjects` (ya sea la interfaz remota o local) o un único objeto `EJBObject`.
- Los parámetros y el tipo de devolución del método deben ser de tipos Java RMI válidos.

Para los beans entidad que gestionan su persistencia, cada método de búsqueda declarado en la clase debe corresponder a un método de búsqueda de la interfaz base o base local que tenga los mismos parámetros y devuelva la interfaz remota/local del bean. El cliente busca el bean entidad llamando al método de búsqueda de la interfaz base, y el contenedor llama al método correspondiente de la clase del bean.

Consulte “[Métodos de búsqueda en beans entidad](#)” en la página 18-6.

Los métodos de búsqueda de los beans entidad EJB 2.0 con persistencia gestionada por contenedor deben tener una consulta en el lenguaje de consultas EJB definido en el descriptor de distribución. Añada un método de búsqueda con el diseñador de EJB y utilice el inspector del método para definir la consulta. Si desea información sobre la escritura de consultas en EJB QL, consulte “Enterprise JavaBeans Query Language” en la página web de Sun, en [http://java.sun.com/j2ee/tutorial/1\\_3-fcs/doc/EJBQL.html](http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBQL.html).

### Escritura de métodos empresariales

Dentro de la clase del enterprise bean, escriba implementaciones completas de los métodos empresariales que necesita el bean. Para que estos métodos estén a disposición del cliente también es necesario declararlos en la interfaz remota o local del bean, exactamente con la misma firma.

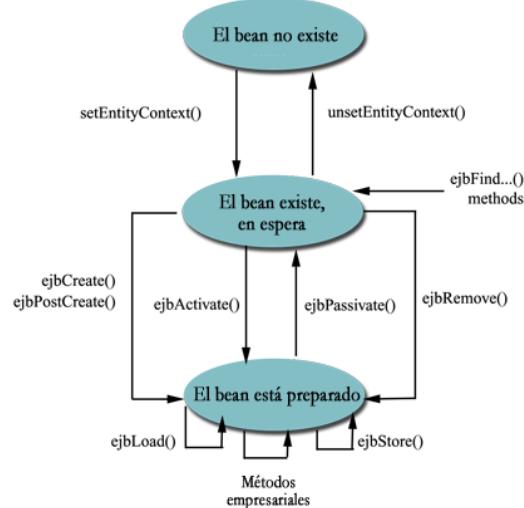
## Ciclo de vida de los beans entidad

Los enterprise beans entidad tienen tres etapas:

- Inexistente
- En espera
- Activado

En el siguiente diagrama se describe el ciclo de vida de las instancias de los beans entidad:

**Figura 16.1** Ciclo de vida de un bean entidad



## Estado de inexistencia

---

Al principio, la instancia no existe. El contenedor EJB crea una instancia de un bean entidad y llama al método `setEntityContext()` para pasar a la instancia una referencia a su contexto, esto es, a la interfaz `EntityContext`. La interfaz `EntityContext` proporciona a la instancia acceso a los servicios proporcionados por el contenedor y le permite obtener información sobre sus clientes. Ahora, el bean entidad se encuentra en el búfer.

## Estado de espera

---

Todos los tipos de bean entidad tienen su propio búfer. Ninguna de las instancias del búfer está asociada a datos. Como no se ha definido ninguna de sus variables, las instancias no tienen identidad y son equivalentes. El contenedor puede asignar libremente cualquiera de ellas a los clientes que soliciten beans entidad de este tipo.

Cuando una aplicación cliente llama a uno de los métodos de búsqueda del bean entidad, el contenedor ejecuta el método `ejbFind()` correspondiente en una instancia arbitraria, en el búfer. La instancia permanece en espera durante la ejecución del método de búsqueda.

Cuando el contenedor selecciona una instancia para cumplir la solicitud de un cliente de un objeto de entidad, esta instancia pasa del estado de espera al estado de activada. Es posible cambiar el estado de las instancias de dos formas:

- Mediante los métodos `ejbCreate()` y `ejbPostCreate()`.
- Mediante el método `ejbActivate()`.

El contenedor selecciona la instancia para gestionar la solicitud `create()` del cliente en la interfaz local del bean. Como respuesta a la llamada a `create()`, el contenedor crea un objeto de entidad y llama a los métodos `ejbCreate()` y `ejbPostCreate()` cuando se asigna la instancia al objeto.

El contenedor llama al método `ejbActivate()` para que active una instancia, de forma que pueda responder a una llamada a un objeto de entidad existente. Normalmente, el contenedor llama a `ejbActivate()` cuando no hay instancias adecuadas preparadas para gestionar las llamadas del cliente.

## Estado de activación

---

Cuando se asocia a una clave principal concreta, la instancia se activa. Los clientes llaman a los métodos específicos de la aplicación del bean entidad. El contenedor llama a los métodos `ejbLoad()` y `ejbStore()` para indicar al bean que debe cargar y almacenar sus datos. También permiten que la

instancia del bean sincronice su estado con el de la entidad de datos subyacente.

## Vuelta al estado de espera

---

Cuando una instancia de un bean entidad vuelve al estado de espera, se separa de los datos que representa. El contenedor puede asignar ahora la instancia de cualquier objeto de entidad en la misma base del bean entidad. Es posible cambiar el estado de las instancias de dos formas:

- El contenedor llama al método `ejbPassivate()` para separar la instancia de su clave principal sin eliminar el objeto de entidad subyacente.
- El contenedor llama al método `ejbRemove()` para que elimine el objeto de entidad. Llama a `ejbRemove()` cuando la aplicación cliente llama al método `remove()` de la interfaz local o remota del bean.

Para eliminar del búfer una instancia sin asociar, el contenedor llama al método `unsetEntityContext()` de la instancia.

## Ejemplo de bean entidad de banca

---

En el ejemplo de banca se explica el uso de los beans entidad. Incluye dos implementaciones de la misma interfaz remota `Account`. Una de ellas utiliza la persistencia gestionada por bean, y la otra, la persistencia gestionada por contenedor.

El bean entidad `Savings`, que gestiona su propia persistencia, configura las cuentas de ahorro. Si examina el código podrá ver que incluye llamadas directas JDBC.

El bean entidad `Checking`, de persistencia gestionada por contenedor, configura las cuentas corrientes. Su persistencia está implementada por el contenedor, y no por el desarrollador de los beans.

Un tercer enterprise bean, llamado `Teller`, transfiere fondos de una cuenta a la otra. Se trata de un bean sesión sin estado que demuestra la forma en que se pueden agrupar las llamadas a varios beans entidad dentro de una sola transacción gestionada por contenedor. Si la retirada tiene lugar antes que el ingreso en la operación de transferencia, el contenedor cancela la transacción si no tiene lugar el ingreso, y no ocurre ninguna de las dos acciones.

El código completo de este ejemplo se encuentra en el directorio /  
`BorlandEnterpriseServer/examples/ejb/bank`.

## Interfaz base del bean entidad

---

Varios beans entidad pueden compartir las interfaces base y local/remota, aunque tengan distintos métodos de gestión de la persistencia. Los beans entidad Savings y Checking comparten la interfaz base, AccountHome. También utilizan la misma interfaz remota, Account.

La interfaz base de los beans entidad se parece mucho a la de los beans sesión. Amplían la misma interfaz javax.ejb.EJBHome o javax.ejb.EJBLocalHome. La interfaz base de los beans entidad debe incluir por lo menos un método de búsqueda. El método `create()` es optativo.

Éste es el código de la interfaz AccountHome:

```
public interface AccountHome extends javax.ejb.EJBHome {
    Account create(String name, float balance)
        throws java.rmi.RemoteException, javax.ejb.CreateException;
    Account findByPrimaryKey(AccountPK primaryKey)
        throws java.rmi.RemoteException, javax.ejb.FinderException;
    java.util.Enumeration findAccountsLargerThan(float balance)
        throws java.rmi.RemoteException, javax.ejb.FinderException;
}
```

La interfaz base AccountHome implementa tres métodos. Aunque el método `create()` no es necesario en los beans entidad, el ejemplo de banca lo implementa. El método `create()` inserta un objeto de bean entidad en la base de datos subyacente. La creación de los objetos de entidad se puede delegar en el SGBD o en otra aplicación, en cuyo caso no se define el método `create()`.

`create()` requiere dos parámetros: el nombre de la cuenta (una cadena) y el saldo (un valor numérico). La implementación de este método en la clase del bean entidad utiliza los valores de estos dos parámetros para restablecer el estado del objeto entidad (el nombre de cuenta y el saldo inicial) al crear el objeto. La cláusula `throws` del método `create()` debe incluir las excepciones `java.rmi.RemoteException` y `java.ejb.CreateException`. También puede incluir otras excepciones propias de la aplicación.

Los beans entidad deben tener el método `findByPrimaryKey()`, por lo que la interfaz AccountHome lo declara. Tiene un parámetro, la clase de clave principal AccountPK, y devuelve una referencia a la interfaz remota Account. Este método busca una entidad de cuenta y devuelve una referencia a ella.

Aunque no es necesario, la interfaz local también declara un segundo método de búsqueda, `findAccountsLargerThan()`. Este método devuelve una enumeración de Java que contiene todas las cuentas cuyo saldo supera cierta cantidad.

## La interfaz remota del bean entidad

---

Es posible utilizar la misma interfaz remota/local para varios beans entidad, aunque sus métodos de gestión de persistencia sean distintos. Los dos beans entidad del ejemplo de banca utilizan la misma interfaz remota, Account.

La interfaz remota amplía la interfaz javax.ejb.EJBObject y pone los métodos empresariales a disposición de los clientes. Éste es el código:

```
public interface Account extends javax.ejb.EJBObject {
    public float getBalance() throws java.rmi.RemoteException;
    public void credit(float amount) throws java.rmi.RemoteException;
    public void debit(float amount) throws java.rmi.RemoteException;
}
```

## Bean entidad con persistencia gestionada por contenedor

---

En el ejemplo de banca se implementa un bean entidad Checking que ilustra los fundamentos de la persistencia gestionada por contenedor 1.1. En muchos aspectos se parece a la implementación de un bean sesión. Sin embargo, se deben observar ciertos aspectos clave en la implementación de los beans entidad con persistencia gestionada por contenedor:

- El bean entidad no tiene implementaciones para los métodos de búsqueda. El contenedor EJB proporciona las implementaciones de método de búsqueda de los beans entidad con persistencia gestionada por contenedor. En lugar de proporcionar la implementación de los métodos de búsqueda de la clase del bean, el descriptor de distribución contiene información que permite al contenedor implementar estos métodos de búsqueda.
- El bean entidad declara públicos todos los campos que gestiona el contenedor. El bean Checking declara los campos name y balance como públicos.
- La clase del bean entidad implementa los distintos métodos declarados en la interfaz de EntityBean: ejbActivate(), ejbPassivate(), ejbLoad(), ejbStore(), ejbRemove(), setEntityContext() y unsetEntityContext(). Basta con que el bean entidad proporcione las implementaciones de esqueleto de estos métodos; sin embargo, puede añadir código específico de la aplicación, si procede. El bean CheckingAccount guarda el contexto que devuelve setEntityContext() y libera la referencia de unsetEntityContext(). No añade más código a los métodos de la interfaz EntityBean.
- Checking incluye una implementación del método ejbCreate() porque este enterprise bean permite crear cuentas corrientes a los objetos que lo llaman. La implementación también restablece el valor de las dos variables de la instancia, name y balance, en los valores de los

parámetros. ejbCreate() devuelve un valor null porque, con la persistencia gestionada por contenedor, éste crea la referencia adecuada para devolver al cliente.

- La clase de bean CheckingAccount proporciona la implementación mínima del método ejbPostCreate(), aunque en caso necesario podría haber efectuado más tareas de inicialización. En los beans con persistencia gestionada por contenedor se necesita únicamente una implementación mínima de ejbPostCreate(), porque cumple las funciones de una devolución de llamada de notificación.

```

import javax.ejb.*;
import java.rmi.RemoteException;

public class CheckingAccount implements EntityBean {
    private javax.ejb.EntityContext _context;
    public String name;
    public float balance;

    public float getBalance() {
        return balance;
    }

    public void debit(float amount) {
        if(amount > balance) {
            // imponer la cancelación de la transacción ...
            _context.setRollbackOnly();
        }
        else {
            balance = balance - amount;
        }
    }

    public void credit(float amount) {
        balance = balance + amount;
    }

    public AccountPK ejbCreate(String name, float balance) {
        this.name = name;
        this.balance = balance;
        return null;
    }

    public void ejbPostCreate(String name, float balance) {}
    public void ejbRemove() {}
    public void setEntityContext(EntityContext context) {
        _context = context;
    }

    public void unsetEntityContext() {
        context = null;
    }

    public void ejbActivate() {}

```

```
public void ejbPassivate() {}  
public void ejbLoad() {}  
public void ejbStore() {}  
public String toString() {  
    return "CheckingAccount[name=" + name + ",balance=" + balance + "]";  
}  
}
```

## Bean entidad con persistencia gestionada por bean

---

En el ejemplo de banca se implementa también `Savings`, un bean entidad que gestiona su propia persistencia. El bean `Savings` accede a una tabla de cuenta distinta de la de `Checking`. Aunque estos dos beans entidad utilizan distintos sistemas de gestión de persistencia, pueden compartir las interfaces local y remota. Sin embargo, existen diferencias entre las dos implementaciones.

La implementación de los beans entidad que gestionan su persistencia hace lo siguiente:

- Puede declarar sus variables de instancia como privadas en lugar de públicas.

El bean incluye el código de acceso a estas variables, la carga de los valores de la base de datos en ellas y el almacenamiento de los cambios en la base de datos. El bean puede limitar el acceso a estas variables en los momentos oportunos. Esto difiere de los beans con persistencia gestionada por contenedor, que deben declarar públicas todas las variables gestionadas por el contenedor para que éste pueda acceder a ellas.

- El método `ejbCreate()` devuelve la clase de la clave principal.

En la clase del bean `SavingsAccount` la clase es `AccountPK`. El contenedor toma la clase de clave principal devuelta y la utiliza para construir una referencia remota a la instancia del bean entidad.

- Igual que ocurre con los beans de persistencia gestionada por contenedor, los que gestionan su propia persistencia pueden proporcionar una implementación del método `ejbCreate()` que no esté vacía.

El bean `SavingsAccount` no necesita incluir código de inicialización adicional en este método.

- Tiene implementaciones para los métodos `ejbLoad()` y `ejbStore()`.

Normalmente, los beans con persistencia gestionada por contenedor sólo proporcionan una implementación vacía de estos métodos. Los beans entidad que gestionan su propia persistencia deben contar con su propio código de lectura de los valores de base de datos de las variables

de instancia, en el método `ejbLoad()`, y de escritura de los valores cambiados en la base de datos, en el método `ejbStore()`.

- Tiene implementaciones para todos los métodos de búsqueda.

El bean entidad `SavingsAccount` implementa dos métodos de búsqueda: `ejbFindByPrimaryKey()`, que es necesario, y `ejbFindAccountsLargerThan()`, que es optativo.

- Los beans entidad que gestionan su persistencia deben implementar el método `ejbRemove()`.

Dado que el bean gestiona el objeto de entidad de la base de datos subyacente, debe implementar este método para poder eliminar el objeto de la base de datos. Los beans con persistencia gestionada por contenedor omiten la implementación de este método porque el contenedor es el responsable de la gestión de la base de datos.

- Todos los métodos que acceden al objeto de base de datos deben incluir el código de acceso correcto.

Estos métodos son `ejbCreate()`, `ejbRemove()`, `ejbLoad()`, `ejbStore()`, `ejbFindByPrimaryKey()`, todos los demás métodos de búsqueda y los métodos empresariales. Contienen código de conexión a la base de datos, seguido de código para construir y ejecutar sentencias SQL que cumplen las funciones del método. Cuando se ejecutan las sentencias SQL, el método cierra las sentencias y la conexión de base de datos antes de volver.

El siguiente ejemplo de código muestra las partes interesantes de la clase de implementación de `SavingsAccount`. En el ejemplo se ha eliminado el código que sea una simple implementación vacía de los métodos de las interfaces `EntityBean`, como `ejbActivate()`, `ejbPassivate()`, etc.

Fíjese en primer lugar en el método `ejbLoad()`, que accede al objeto de base de datos, para observar la forma en que la persistencia gestionada por bean implementa el acceso a la base de datos. Todos los métodos implementados en la clase `SavingsAccount` son parecidos a `ejbLoad()`. El método `ejbLoad()` empieza por establecer una conexión con la base de datos. Llama al método `getConnection()` interno, que utiliza un objeto `DataSource` para obtener una conexión JDBC a la base de datos desde un búfer de conexión JDBC. Cuando se establece la conexión, `ejbLoad()` crea un objeto `PreparedStatement` y construye su sentencia SQL de acceso a la base de datos. Dado que `ejbLoad()` lee los valores del objeto de entidad de las variables de instancia del bean entidad, crea una sentencia SQL SELECT para la consulta que selecciona el valor del saldo de la cuenta de ahorro cuyo nombre coincide con una pauta. A continuación, el método ejecuta la consulta y si devuelve un resultado, extrae el importe del saldo. Al final, el método `ejbLoad()` cierra los objetos `PreparedStatement` y la conexión con la base de datos. Tenga presente que el método `ejbLoad()` no cierra la conexión realmente. Simplemente, la devuelve al búfer de conexión.

## Ejemplo de bean entidad de banca

```
import java.sql.*;
import javax.ejb.*;
import java.util.*;
import java.rmi.RemoteException;

public class SavingsAccount implements EntityBean {
    private entitycontext _constext;
    private String _name;
    private float _balance;

    public float getBalance() {
        return _balance;
    }

    public void debit(float amount) {
        if(amount > balance) {
            // imponer la cancelación de la transacción...
            _context.setRollbackOnly();
        } else {
            _balance = _balance - amount;
        }
    }

    public void credit(float amount) {
        _balance = _balance + amount;
    }

    // setEntitycontext(), unsetEntityContext(), ejbActivate(), ejbPassivate(),
    // Las implementaciones de esqueleto ejbPostCreate() no se muestran aquí
    ...

    public AccountPK ejbCreate(String name, float balance)
        throws RemoteException, CreateException {
        _name = name;
        _balance = balance;
        try {
            Connection connection = getConnection();
            PreparedStatement statement = connection.prepareStatement
                ("INSERT INTO Savings_Accounts (name, balance) VALUES (?,?)");
            statement.setString(1, _name);
            statement.setFloat(2, _balance);
            if(statement.executeUpdate() != 1) {
                throw new CreateException("Could not create: " + name);
            }
            statement.close();
            connection.close();
            return new AccountPK(name);
        } catch(SQLException e) {
            throw new RemoteException("Could not create: " + name, 3);
        }
    }

    ...
    public void ejbRemote() throws RemoteException, RemoveException {
        try {
            Connection connection = getConnection();
            PreparedStatement statement = connection.prepareStatement
                ("DELETE FROM Savings_Account WHERE name = ?");
        }
    }
}
```

```

        statement.setString(1, _name);
        if(statement.executeUpdate() != 1) {
            throw new RemoteException("Could not remove: " + _name, e);
        }
        statement.close();
        connection.close();
    } catch(SQLException e) {
        throw new RemoteException("Could not remove: " + _name, e);
    }
}
public AccountPK ejbFindByPrimaryKey(AccountPK key) throws RemoteException,
    FinderException {
try {
    Connection connection = getConnection();
    PreparedStatement statement = connection.prepareStatement
        ("SELECT name FROM Savings_Accounts WHERE name = ?");
    statement.setString(1, key.name);
    ResultSet resultSet = statement.executeQuery();
    if(!resultSet.next()) {
        throw new FinderException("Could not find: " + key);
    }
    statement.close();
    connection.close();
    return key;
} catch(SQLException e) {
    throw new RemoteException("Could not find: " + key, e);
}
}

public java.util.Enumeration ejbFindAccountsLargerThan(float balance)
    throws RemoteException, FinderException {
try {
    Connection connection = getConnection();
    PreparedStatement statement = connection.prepareStatement
        ("SELECT name FROM Savings_Account WHERE balance > ?");
    statement.setFloat(1, balance);
    ResultSet resultSet = statement.executeQuery();
    Vector keys = new Vector();
    while(resultSet.next()) {
        String name = resultSet.getString(1);
        keys.addElement(new AccountPK(name));
    }
    statement.close();
    connection.close();
    return keys.elements();
} catch(SQLException e) {
    throw new RemoteException("Could not findAccountsLargerThan: " +
        balance, e);
}
}

public void ejbLoad() throws RemoteException {
    // obtener el nombre de la clave principal
    _name = (AccountPK) _context.getPrimaryKey().name;
    try {

```

## Ejemplo de bean entidad de banca

```
Connection connection = getConnection();
PreparedStatement statement = connection.prepareStatement
    ("SELECT balance FROM Savings_Account WHERE name = ?");
statement.setString(1, _name);
ResultSet resultSet = statement.executeQuery();
if(!resultSet.next()) {
    throw new RemoteException("Account not found: " + _name);
}
_balance = resultSet.getFloat(1);
statement.close();

connection.close();
} catch(SQLException e) {
    throw new RemoteException("Could not load: " + _name, e);
}
}

public void ejbStore() throw RemoteException {
try {
    Connection connection = getConnection();
    PreparedStatement statement = connection.prepareStatement
        ("UPDATE Savings_Accounts SET balance = ? WHERE name = ?");
    statement.setFloat(1, _balance);
    statement.setString(2, _name);
    statement.executeUpdate();
    statement.close();
    connection.close();
} catch(SQLException e) {
    throw new RemoteException("Could not store: " + _name, e);
}
}

private Connection getConnection() throws SQLException {
Properties properties = _context.getEnvironment();
String url = properties.getProperty("db.url");
String username = properties.getProperty("db.username");
String password = properties.getProperty("db.password");
if(username != null) {
    return DriverManager.getConnection(url, username, password);
} else {
    return DriverManager.getConnection(url);
}
}

public String toString() {
    return "SavingsAccount[name=" + _name + ",balance=" + _balance + "]";
}
}
```

## La clase de la clave principal

---

Las clases de bean `CheckingAccount` y `SavingsAccount` utilizan el mismo campo para identificar de forma irrepetible un registro de cuenta determinado. En este caso, los dos utilizan la misma clase de clave principal, `AccountPK`, para representar el identificador único de cualquiera de los dos tipos de cuenta:

```
public class AccountPK implements java.io.Serializable {
    public String name;
    public AccountPK() {}
    public AccountPK(String name) {
        this.name = name;
    }
}
```

## El descriptor de distribución

---

El descriptor de distribución del ejemplo de banca distribuye beans de tres tipos: el bean sesión `Teller`, el bean entidad con persistencia gestionada por contenedor `Checking` y el bean entidad con persistencia gestionada por sí mismo `Savings`.

Las propiedades del descriptor de distribución permiten especificar información sobre las interfaces del bean entidad, los atributos de transacción, etc., igual que se hace con los beans sesión, pero también cuentan con información adicional exclusiva.

El ejemplo de código XML gestionado por bean muestra las etiquetas típicas de descriptor de distribución de un bean entidad que gestiona su persistencia. El ejemplo de código XML gestionado por contenedor muestra las etiquetas típicas de descriptor de distribución de un bean entidad con persistencia gestionada por contenedor. Si se comparan las etiquetas de descriptor de los dos tipos de beans entidad, se observa que el descriptor de distribución de los beans entidad con persistencia gestionada por contenedor es más complejo.

El tipo de descriptor de distribución del bean se establece en `<entity>`.  
LLLAs primeras etiquetas de la sección `<enterprise-beans>` de los dos ejemplos de código indican que se trata de beans entidad.

El descriptor de distribución del bean entidad transmite la siguiente información:

- Nombre de las interfaces relacionadas (local y remota) y clase de implementación del bean.

Los enterprise beans indican la interfaz base por medio de la etiqueta `<home>`, la interfaz remota por medio de la etiqueta `<remote>` y el nombre de clase de implementación por medio de la etiqueta `<ejb-class>`.

- Los nombres JNDI con los que se ha registrado el bean entidad y mediante los cuales los clientes localizan este bean.
- Los atributos de transacción del bean y su nivel de aislamiento de transacciones.

Normalmente aparece en la sección <assembly-descriptor> del descriptor de distribución.

- Nombre de la clase de clave principal del bean entidad.

En este ejemplo, la clase de la clave principal es `AccountPK` y aparece dentro de la etiqueta <prim-key-class>.

- La persistencia utilizada por el bean.

El bean `CheckingAccount` utiliza persistencia gestionada por contenedor, por lo que el descriptor de distribución define la etiqueta <persistence-type> como Container.

- Si la clase del bean es reentrante.

Ni `SavingsAccount` ni `CheckingAccount` es reentrante, por lo que la etiqueta <reentrant> tiene el valor False en ambos casos.

- Los campos gestionados por el contenedor, si el bean utiliza persistencia gestionada por contenedor.

Si el bean utiliza persistencia gestionada por contenedor no se especifican estos campos. Por tanto, el descriptor de distribución del bean `SavingsAccount` no indica los campos gestionados por el contenedor. Los beans con persistencia gestionada por contenedor deben indicar el nombre de sus variables de campo o instancia gestionadas por el contenedor. Para esto se utiliza una combinación de las etiquetas <cmp field> y <field name>. La primera etiqueta, <cmp field>, indica que el campo está gestionado por el contenedor. Dentro de esta etiqueta <fields name> indica el nombre del campo. Por ejemplo, el descriptor de distribución del bean `CheckingAccount` indica de la siguiente forma que el campo de saldo está gestionado por el contenedor:

```
<cmp field><field name>balance</field name></cmp field>
```

Información sobre los campos gestionados por el contenedor para beans gestionados por contenedor. El contenedor utiliza esta información para generar los métodos de búsqueda de estos campos.

## Descriptor de distribución de un bean entidad con persistencia gestionada por bean

El siguiente ejemplo de código muestra partes más importantes del descriptor de distribución de un bean entidad que gestiona su persistencia. Dado que es el bean y no el contenedor el que gestiona las lecturas de los valores de entidad de la base de datos y sus

actualizaciones, el descriptor no indica los campos que debe gestionar el contenedor ni le indica la forma de implementar los métodos de búsqueda, ya que la implementación del bean los proporciona.

```
<enterprise beans>
<entity>
    <description>Este bean entidad es un ejemplo de bean que gestiona su consistencia
    persistence</description>
    <ejb-name>savings</ejb-name>
    <home>AccountHome</home>
    <remote>Account</remote>
    <ejb-class>SavingsAccount</ejb-class>
    <persistence-type>Bean</persistence-type>
    <prim-key-class>AccountPK</prim-key-class>
    <reentrant>False</reentrant>
</entity>
...
</enterprise-beans>
<assembly-descriptor>
    <container-transaction>
        <method>
            <ejb-name>savings</ejb-name>
            <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
    </container-transaction>
</assembly-descriptor>
```

## **Descriptor de distribución de un bean entidad con persistencia gestionada por contenedor**

El siguiente ejemplo de código muestra partes más importantes del descriptor de distribución de un bean entidad con persistencia gestionada por contenedor. Dado que el bean permite que el contenedor gestione la carga de los valores de entidad de la base de datos y sus actualizaciones, el descriptor indica los campos que debe gestionar el contenedor.

```
<enterprise beans>
<entity>
    <description>Este bean entidad es un ejemplo de bean que gestiona su persistencia por contenedor
    persistence</description>
    <ejb-name>checking</ejb-name>
    <home>AccountHome</home>
    <remote>Account</remote>
    <ejb-class>chkingAccount</ejb-class>
    <persistence-type>Container</persistence-type>
    <prim-key-class>AccountPK</prim-key-class>
    <reentrant>False</reentrant>
    <cmp-field>
        <field-name>name</field-name>
    <cmp-field>
</entity>
```

## Ejemplo de bean entidad de banca

```
</enterprise-beans>
<assembly-descriptor>
    <container-transaction>
        <method>
            <ejb-name>checking</ejb-name>
            <method-name>*</method-name>
        </method>
        <trans-attribute-transaction>
    </container-transaction>
</assembly-descriptor>
```

## Desarrollo de beans gestionados por mensajes

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

Los beans gestionados por mensajes, incorporados en la especificación EJB 2.0, simplifican en gran medida la programación gestionada por mensajes. La primordial tarea del proveedor es implementar un método `onMessage()` que contiene la lógica que responde al mensaje. El contenedor EJB se encarga del resto de las tareas de mensajería. Los beans gestionados por mensajes utilizan un proveedor JMS, como SonicMQ Message Broker, que se suministra con JBuilder Enterprise.

Un bean gestionado por mensajes es un enterprise bean capaz de procesar mensajes JMS. Esos mensajes pueden venir de cualquier componente capaz de enviar mensajes JMS, como, por ejemplo, otro enterprise bean, una cliente de la aplicación, un componente web o un sistema heredado. Un bean gestionado por mensajes es un monitor de mensajes JMS, que responde cuando detecta un tipo particular de mensaje. Al igual que los beans sesión, los beans gestionados por mensajes no están asociados a un solo cliente y no cuentan con un estado conversacional.

Los beans gestionados por mensajes controlan y responden a mensajes *asíncronos*. Si el emisor envía un mensaje asíncrono, no espera a que el receptor lo reciba y lo procese antes de continuar con su propia tarea. Por el contrario, el emisor de un mensaje *síncrono* espera a que el receptor procese el mensaje antes de que el control del programa vuelva al emisor. Los mensajes asíncronos permiten hacer más fluida la conexión entre el emisor del mensaje (el productor) y el receptor (el consumidor, el bean gestionado por mensajes). Esto hace que los beans gestionados por mensajes sean de especial utilidad en las interacciones entre empresas y en la integración de un sistema EJB con sistemas heredados.

## Funcionamiento de los beans gestionados por mensajes

---

El distribuidor de beans gestionados por mensajes establece el destino JMS que monitoriza el bean en el descriptor de distribución. El descriptor de distribución también puede incluir un filtro de selección de mensajes que asocie al bean gestionado por mensajes con otro tipo de mensajes. Consulte “[Atributos del descriptor de distribución de los beans generados por mensajes](#)” en la página 17-6 para obtener más información.

El contenedor activa una instancia del tipo adecuado de bean gestionado por mensajes. La instancia del bean consume el mensaje que se envía a su destino JMS asociado. Éste responde al mensaje con la lógica que se encuentra en el método `onMessage()`.

Al contrario que los beans sesión y beans entidad, los beans gestionado por mensajes no cuentan con interfaces base/base local ni remota/local. Por ello, un cliente nunca puede tener acceso directo a un bean gestionado por mensajes. En su lugar, un cliente envía un mensaje a un destino JMS, que bien puede ser una cola o un tema.

El modelo de mensajes punto a punto utiliza una cola. La mayoría de los beans gestionados por mensajes pueden recibir mensajes de la misma cola, pero solamente un bean puede recibir cada mensaje. Los consumidores *extraen* mensajes de la cola en la que no se recibe ninguno hasta que lo solicita un consumidor.

El modelo publicar y suscribir utiliza temas. Un productor de mensajes puede enviar un mensaje a varios consumidores utilizando un tema. Los consumidores, al igual que los beans gestionados por mensajes, se suscriben al tema. Los mensajes enviados a un tema se envían a todos los suscriptores de ese tema. Por lo tanto, cada suscriptor recibe una copia del mensaje. Los mensajes se *mandan* al consumidor.

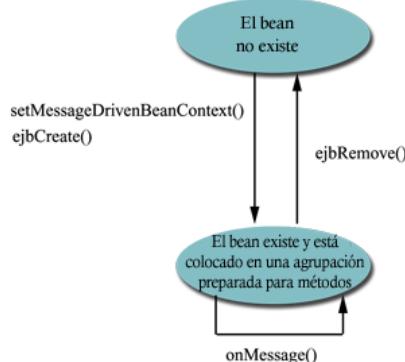
## Ciclo de vida de la instancia de un bean gestionado por mensajes

---

El ciclo de vida de un bean gestionado por mensajes es muy sencillo. Un cliente envía un mensaje a un destino JMS que monitoriza el bean. El contenedor EJB crea una nueva instancia de la clase del bean. A continuación, llama a los métodos `setMessageDrivenContext()` y `ejbCreate()` de la instancia del bean, por ese orden. A partir de ese momento, la instancia del bean gestionado por mensajes puede consumir mensajes enviados al destino del bean. El ciclo de vida de la instancia del bean termina cuando el contenedor EJB llama al método `ejbRemove()`. En el

siguiente diagrama se describe el ciclo de vida de las instancias de los beans gestionados por mensajes:

**Figura 17.1 Ciclo de vida de un bean gestionado por mensajes**



## Cómo escribir clases del bean gestionado por mensajes

---

La tarea de escribir beans gestionados por mensajes es mucho más simple, porque solamente se crea la clase del bean; los beans gestionados por mensajes no cuentan con interfaz base/base local ni remota/local.

Para crear una clase de bean gestionado por mensajes:

- 1 Cree una clase que sea una extensión de la interfaz `javax.ejb.MessageDrivenBean`. La clase debe definirse como pública y no como final ni abstracta.
- 2 En la clase del bean, implemente la interfaz `javax.jms.MessageListener`.
- 3 Incluya un constructor público sin argumentos. El contenedor llama a este constructor para crear instancias de la clase del bean gestionado por mensajes.
- 4 Implemente el método `ejbCreate()` sin argumentos. El método debe declararse como público y no como final ni estático. Su tipo devuelto debe ser `void` y no debe definir ninguna excepción de aplicaciones.

## Implementación de la interfaz Message DrivenBean

---

La interfaz MessageDrivenBean define dos métodos que deben implementar todos los beans gestionados por mensajes. Es una ampliación de la interfaz EnterpriseBean:

```
package javax.ejb;  
  
public interface MessageDrivenBean extends javax.ejb.EnterpriseBean {  
    public void setMessageDrivenContext(MessageDrivenContext context)  
        throws EJBException;  
    public void ejbRemove() throws EJBException;  
}
```

Los métodos de la interfaz MessageDrivenBean están estrechamente asociados al ciclo de vida del bean:

- `setMessageDrivenContext()`: El contenedor llama al `setMessageDrivenContext()` para proporcionar la instancia del bean gestionado por mensajes con una referencia a su método `MessageDrivenContext`, que se pasa al método. El contenedor llama a este método al principio del ciclo de vida del bean.
- `ejbRemove()` : El contenedor lo llama al final del ciclo de vida del bean. Puede utilizar este método para liberar los recursos del método `ejbCreate()`.

## Implementación de la interfaz MessageListener

---

Solamente los beans gestionados por mensajes pueden implementar la interfaz MessageListener. Esta interfaz sólo define un método, `onMessage()`:

```
package javax.jms;  
  
public interface MessageListener {  
    public void onMessage(Message message);  
}
```

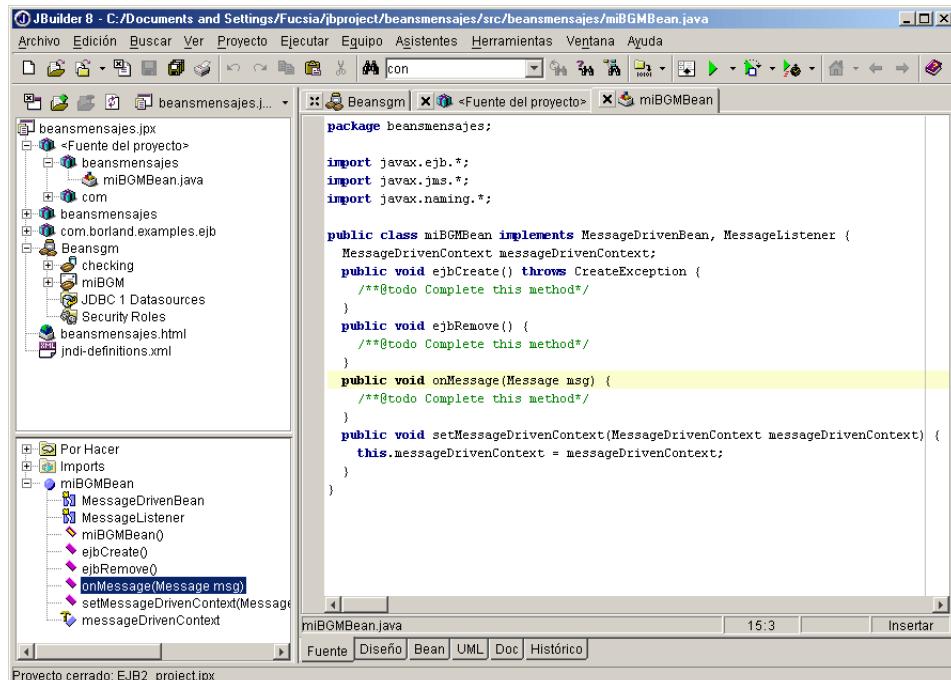
### Escritura del método `onMessage()`

El núcleo de un bean gestionado por mensajes es su método `onMessage()`. Coloque toda la lógica que gestiona un mensaje entrante en el método `onMessage()`. El mensaje se pasa a `onMessage()` en un único argumento. La lógica puede gestionar el mensaje entrante, pasarlo a otro bean o enviarlo a otro destino JMS. Es posible implementar métodos de ayuda en la clase del bean que el método `onMessage()` puede llamar.

## JBuilder en la creación de beans gestionados por mensajes

Para comenzar a crear beans gestionados por mensajes, pulse con el botón derecho del ratón en el panel del diseñador de EJB y seleccione en el menú contextual Nuevo bean mensaje. Aparece una representación del bean gestionado por mensajes en el diseñador de EJB. Pulse con el botón derecho del ratón en el nombre del bean de la representación, y configure las propiedades del bean en el inspector que aparece.

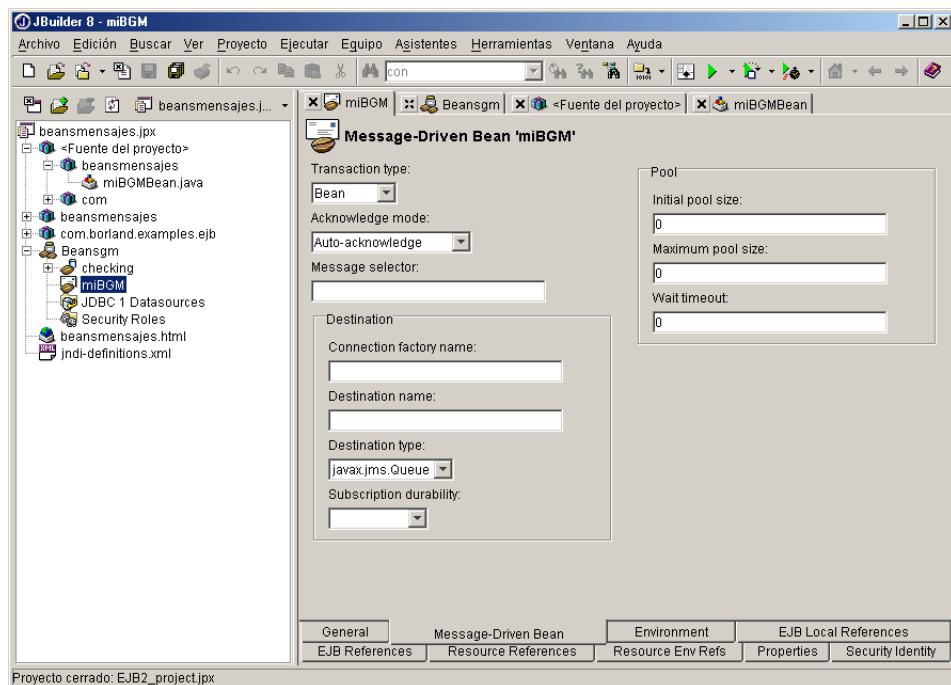
Haga doble clic sobre el nodo del código fuente del bean en el panel de proyecto de JBuilder para ver el código fuente generado:



En el código fuente de la clase del bean, busque el método `onMessage()` y escriba la lógica que ha de procesar los mensajes entrantes. Puede añadir a la clase del bean otros métodos a los que llama el método `onMessage()`.

Se puede utilizar el editor del descriptor de distribución de JBuilder para modificar el descriptor de distribución del bean. Haga doble clic sobre el nodo de la clase del bean gestionado por mensajes en el panel de proyecto y abra la pestaña Editor DD de la parte inferior del panel de contenido. Cuando aparezca el editor del descriptor de distribución, pulse la pestaña

Bean gestionado por mensajes para que se abra el panel Bean gestionado por mensajes:



A continuación, se detallan algunos de los atributos de distribución que puede encontrar en este panel.

## Atributos del descriptor de distribución de los beans generados por mensajes

A cada bean gestionado por mensajes se le debe asignar un destino JMS desde el que el bean monitoriza y consume mensajes. Esta tarea corresponde al distribuidor, pero el proveedor de beans puede introducir un destino JMS en el descriptor de distribución como información para el distribuidor. El destino puede ser una cola o un tema. Si se trata de una cola, ésta debe tener un solo bean gestionado por mensajes como consumidor: no asigne un destino a más de un bean.

Si el destino es un tema, el atributo de duración de suscripción debe declararse duradero o no duradero. Si, por algún motivo, se pierde la conexión entre el contenedor EJB y el proveedor JMS para una suscripción duradera, los mensajes no se pierden. El proveedor JMS guarda los mensajes que pierde el bean suscriptor y los envía cuando se restablece la conexión. Las suscripciones no duraderas conllevan la pérdida de los mensajes si la conexión entre el contenedor EJB y el proveedor JMS se

rompe. La principal ventaja de una suscripción no duradera es que mejora el funcionamiento, aunque su utilización hace que los beans gestionados por mensajes resulten menos fiables.

Cuando un bean gestionado por mensajes ejecuta transacciones gestionadas por bean, se pone en funcionamiento el atributo de acuse de recibo. Puede tener dos posibles valores: acuse de recibo automático (auto-acknowledge) o aceptar duplicados (dups-ok-acknowledge). Si se selecciona el acuse de recibo automático, el contenedor EJB envía un acuse de recibo al proveedor JMS inmediatamente después de que el mensaje llegue a una instancia del bean gestionado por mensajes. El valor dups-ok-acknowledge (aceptar duplicados) permite al contenedor EJB retrasar el acuse de recibo. Algunas veces, el proveedor JMS cree que el mensaje no se ha enviado, y lo vuelve a mandar, dando lugar a un mensaje duplicado. Si los beans gestionados por mensajes utilizan el valor dups-ok-acknowledge (aceptar duplicados), debe estar preparado para gestionar mensajes duplicados.

Un descriptor de distribución de beans gestionados por mensajes puede incluir un selector de mensajes que permita al bean ser más selectivo con los mensajes que recibe de una cola o un tema. Un selector de mensajes consiste en una expresión que utiliza lógica booleana. Utiliza un subconjunto de la sintaxis de expresiones condicionales SQL-92. A continuación, se propone un ejemplo.

```
Department = "547" AND Salary BETWEEN 56000.00 AND 85000.00
```

Cuando especifique un selector de mensajes, recuerde que esta información se guarda en el descriptor de distribución del bean, que es un archivo XML. En los archivos XML, los símbolos < and > y otros caracteres especiales tienen un significado diferente. Si se incluyen en el selector de mensajes, el archivo XML resultante crea errores de análisis. Por lo tanto, la lógica que utiliza estos símbolos debe colocarse en una sección CDATA. Por ejemplo:

```
<message-selector>
<![CDATA[
Total > 100.00
]]>
</message-selector>
```

Si desea más información, consulte la especificación JMS en la sede web de Sun Microsystems, en <http://java.sun.com/products/jms/docs.html>.

# Utilización de SonicMQ Message Broker con beans gestionados por mensajes

---

SonicMQ Message Broker es un proveedor JMS incluido en JBuilder Enterprise. Si desea utilizar SonicMQ con el bean gestionado por mensajes que ha creado, siga las siguientes indicaciones:

- 1** Pulse la parte superior del bean en el diseñador de EJB para abrir su inspector.
- 2** Si el bean gestionado por mensajes monitoriza una cola, siga estos pasos:
  - a** En la lista desplegable, seleccione el Tipo de transacción como Bean o Contenedor.
  - b** Especifique Modo de acuse de recibo como Acuse de recibo automático o como Aceptar duplicados.
  - c** Especifique el selector de mensajes.
  - d** Especifique el Destino.
  - e** En la lista desplegable, seleccione javax.jms.Queue como valor del Tipo de destino.
  - f** Indique el Nombre de la factoría de conexiones.
  - g** Configure el Tamaño inicial de la agrupación. Por ejemplo, puede configurarlo en 2.
  - h** Establezca el Tamaño máximo de la agrupación. Por ejemplo, puede configurarlo en 20.
  - i** Indique el Nombre de la factoría de conexiones.
- 3** Si el bean gestionado por mensajes se suscribe a un tema, siga las siguientes indicaciones:
  - a** En la lista desplegable, seleccione el Tipo de transacción como Bean o Contenedor.
  - b** Especifique el Destino.
  - c** Especifique el selector de mensajes.
  - d** En la lista desplegable, seleccione javax.jms.Topic como Tipo de destino.
  - e** Especifique la Duración del mensaje como Duradero o No duradero en la lista desplegable.
  - f** Configure el Tamaño inicial de la agrupación. Por ejemplo, puede configurarlo en 2.
  - g** Establezca el Tamaño máximo de la agrupación. Por ejemplo, puede configurarlo en 20.

**h** Indique el Nombre de la factoría de conexiones.

Los nombres que defina para el nombre de la factoría de conexiones y el nombre de destino deben ser nombres únicos en el árbol JNDI. Para más información acerca de la manera de llenar estos campos en el inspector del bean, consulte “Panel de beans gestionados por mensajes” en el [Capítulo 13, “El editor de descriptor de distribución”](#).

Si desea más detalles acerca de SonicMQ, consulte la documentación de SonicMQ.

También puede utilizar el panel Message Driven Bean (Bean gestionado por mensajes) del editor del descriptor de distribución para personalizar la configuración en lugar de hacerlo en el inspector del bean.

Compile el proyecto. Para obtener más información sobre los sucesos, consulte el [Capítulo 10, “Compilación de enterprise beans y creación de archivos JAR”](#).

Una vez que ha personalizado la configuración mediante el inspector del bean o el editor del descriptor de distribución y lo ha compilado, inicie SonicMQ seleccionando Herramientas | Sonic MQ Broker. También puede iniciar el agente de gestión de Borland Enterprise Server seleccionando Herramientas | Agente de gestión de Borland Enterprise Server. Ahora está preparado para ejecutar el bean. El método más rápido de llevarlo a cabo consiste en pulsar con el botón derecho del ratón sobre el módulo EJB o el archivo JAR que contiene el módulo y, a continuación, seleccionar Ejecutar utilizando la configuración por defecto o Depurar utilizando la configuración por defecto, en el menú contextual. El contenedor EJB se inicia y ejecuta el bean.



# 18

## Creación de interfaces base y remota/local

El proveedor de enterprise beans debe crear, al menos, dos interfaces para cada bean sesión y entidad: Para los beans EJB 1.x se debe crear una interfaz base y una interfaz remota. Los beans EJB 2.0 pueden tener una interfaz base remota y otra remota, así como una base local y otra local, como sustitución o adición a las interfaces anteriormente mencionadas. Las interfaces base remota y remota proporcionan al cliente una vista remota del bean, mientras que la interfaz base local y la local proporcionan al cliente una vista local. Las interfaces remota y base local definen los métodos que emplea la aplicación cliente para crear, localizar y destruir instancias de un enterprise bean. La interfaz remota/local define los métodos empresariales que se implementan en el bean. Los clientes acceden a estos métodos a través de la interfaz remota/local.

### Creación de la interfaz base

La interfaz base de los enterprise beans, ya sea remota o local, controlan su ciclo de vida. Contiene la definición de los métodos necesarios para crear, localizar y destruir instancias de un enterprise bean.

El proveedor de los beans debe definir la interfaz local, pero no implementarla. Esto lo hace el contenedor EJB, que genera un objeto base que devuelve una referencia al bean.

Los clientes del enterprise bean tienen una vista local o remota de éste. Los beans remotos tienen una interfaz base que amplía la interfaz `EJBHome`. Los beans locales tienen una interfaz base que amplía la interfaz `EJBLocalHome`.

## La interfaz EJBHome

---

Todas las interfaces base remotas amplían la interfaz `javax.ejb.EJBHome`. Ésta es la definición completa de `EJBHome`:

```
package javax.ejb
public interface EJBHome extends java.rmi.Remote {
    void remove(Handle handle) throws java.rmi.RemoteException,
        RemoveException;
    void remove(Object primaryKey) throws java.rmi.RemoteException,
        RemoveException;
    EJBMetaData getEJBMetaData() throws RemoteException;
    HomeHandle getHomeHandle() throws RemoteException;
}
```

`EJBHome` tiene dos métodos `remove()` que sirven para eliminar instancias de enterprise beans. El primer método `remove()` identifica la instancia mediante un identificador, y el segundo por una clave principal.

El identificador del objeto de bean serializable tiene la misma duración que el objeto de enterprise bean al que hace referencia. Para los beans sesión, el identificador sólo tiene la duración de la sesión. El identificador de un bean entidad puede ser persistente, y un cliente puede utilizar un identificador serializable para restablecer una referencia al objeto de entidad que identifica.

El cliente utiliza el segundo método `remove()` para eliminar una instancia de un bean entidad por medio de su clave principal.

El método `getEJBMetaData()` devuelve la interfaz `EJBMetaData` del objeto de enterprise bean. Esta interfaz permite al cliente obtener información de metadatos sobre el bean. Los utilizan herramientas de desarrollo que generan aplicaciones que utilizan enterprise beans distribuidos.

La interfaz `EJBHome` no tiene métodos para la creación ni la localización de instancias de enterprise beans. Se deben añadir estos métodos a las interfaces base desarrolladas para los beans. Dado que los beans sesión y los beans entidad tienen ciclos de vida distintos, los métodos definidos en sus interfaces base son distintos.

## La interfaz LocalHome

---

Todas las interfaces base locales implementan `javax.ejb.EJBLocalHome`. La interfaz base local sólo tiene un método `remove()`:

```
package javax.ejb
public interface EJBLocalHome {
    void remove(Object primaryKey) throws RemoteException, EJBException;
}
```

## Creación de interfaces base o base locales para beans sesión

---

En la mayoría de los casos (excepto en algunos beans sin estado), los beans sesión tienen un solo cliente. Cuando un cliente crea un bean sesión, la instancia existe sólo para el uso de este cliente.

Para crear la interfaz base remota de un bean sesión:

- Declare una interfaz base que amplíe javax.ejb.EJBHome.
- Añada una firma de método `create()`, que coincida exactamente en el número y el tipo de los argumentos, para cada método `ejbCreate()` del bean.

Para crear la interfaz base local de un bean sesión:

- Declare una interfaz base local que amplíe javax.ejb.EJBLocalHome.
- Añada una firma de método `create()`, que coincida exactamente en el número y el tipo de los argumentos, para cada método `ejbCreate()` del bean.

Cuando se utiliza las herramientas EJB de JBuilder, se crea una interfaz base con un método `create()` definido, a la vez que crea la clase del enterprise bean. Después se deben añadir más métodos `create()` a la interfaz base si se añaden más métodos `ejbCreate()` al bean.

Si se utiliza el diseñador de EJB para crear enterprise beans EJB 2.0 se añade otro método `create()` cuando se añade un método `ejbCreate()` a la clase del bean. No es necesario realizar más acciones.

Si la clase de enterprise bean 1.x ya está creada, el asistente para interfaces EJB 1.x de JBuilder crea interfaces base y remota con firmas que coinciden con las de la clase. Para obtener más información, consulte el [Capítulo 8, “Creación de componentes EJB 1.x con JBuilder”](#).

Si elige comenzar el desarrollo de EJB para beans 1.x creando primero la interfaz remota, puede utilizar el generador de Bean EJB 1.x con el fin de crear una clase esqueleto del bean y la interfaz base. Para más información acerca de la utilización de session beans en transacciones, consulte [“Generación de la clase del bean desde una interfaz remota” en la página 8-13](#).

### Métodos `create()` en beans sesión

Las interfaces base remota o base local de los beans sesión funcionan como fábricas de beans sesión, porque deben definir métodos `create()`. Cuando el cliente llama a `create()` se crea una instancia del bean. Según la especificación EJB, todos los métodos `create()` definidos en la interfaz base remota o base local deben:

- Devolver el tipo de interfaz remota del bean si se está creando una interfaz base y devolver el tipo de interfaz local del bean si se está creando una interfaz base local.
- Tener el nombre `create()`.
- Coincidir con un método `ejbCreate()` de la clase del bean sesión. El número y los tipos de argumentos de los métodos `create()` deben coincidir con sus métodos `ejbCreate()` correspondientes de la clase del bean sesión.
- Lanzar la excepción `java.rmi.RemoteException` si se utiliza una interfaz base. En las interfaces base locales, el método `create()` **no** debe lanzar `RemoteException`.
- Lanzar la excepción `javax.ejb.CreateException`.
- Utilizar sus parámetros, si los hay, para inicializar el objeto del bean sesión.

Se pueden utilizar las herramientas EJB de JBuilder para garantizar el cumplimiento de estas normas.

En el siguiente ejemplo de código se muestran dos métodos `create()` posibles para una interfaz base de sesión. Las partes en negrita son obligatorias:

```
public interface AtmHome extends javax.ejb.EJBHome {
    Atm create()
    throws java.rmi.RemoteException, javax.ejb.CreateException;
    Atm create(Profile preferredProfile)
    throws java.rmi.RemoteException, javax.ejb.CreateException;
}
```

## Creación de interfaces base locales o base remotas para beans entidad

---

Los beans entidad están configurados para servir a varios clientes. Cuando un cliente crea una instancia de un bean, los demás clientes también pueden usarla.

Para crear una interfaz base remota de un bean entidad:

- Declare una interfaz que amplíe `javax.ejb.EJBHome`.
- Añada una firma de método `create()`, que coincida exactamente, para cada método `ejbCreate()` del bean.
- Añada una firma de método de búsqueda, que coincida exactamente, para cada método de búsqueda del bean.

Para crear una interfaz base local de un bean entidad:

- Declare una interfaz que amplíe javax.ejb.EJBLocalHome.
- Añada una firma de método `create()`, que coincida exactamente, para cada método `ejbCreate()` del bean.
- Añada una firma de método de búsqueda, que coincida exactamente, para cada método de búsqueda del bean.

Cuando se utilizan las herramientas para EJB, JBuilder crea una interfaz base remota y base local con un método `create()` definido, a la vez que crea la clase del enterprise bean. Después se pueden añadir más métodos `create()` a la interfaz base y base local si se añaden más métodos `ejbCreate()` al bean. Si se utiliza el diseñador de EJB para crear beans entidad 2.0, cuando se añade un método `ejbCreate()` se añade automáticamente un método `create()` configurado correctamente en la interfaz base o base local.

Si la clase de enterprise bean 1.x ya está creada, el asistente para interfaces EJB 1.x de JBuilder crea interfaces base y remota con firmas que coinciden con las de la clase. Para obtener más información, consulte el [Capítulo 8, “Creación de componentes EJB 1.x con JBuilder”](#).

Si elige comenzar el desarrollo de EJB para beans 1.x creando primero la interfaz remota, puede utilizar el generador de Bean EJB 1.x con el fin de crear una clase esqueleto del bean y la interfaz base. Para más información acerca de la utilización de session beans en transacciones, consulte [“Generación de la clase del bean desde una interfaz remota” en la página 8-13](#).

### Métodos `create()` en beans entidad

Igual que ocurre con las interfaces base o base local de beans sesión, las interfaces base o base local de los bean entidad deben definir uno o varios métodos `create()`. Según la especificación EJB, todos los métodos `create()` definidos deben:

- Lanzar la excepción `java.rmi.RemoteException` para una interfaz base. En las interfaces base locales, el método `create()` **no** debe lanzar `RemoteException`.
- Lanzar la excepción `javax.ejb.CreateException`.
- Devolver el tipo de interfaz remota del bean entidad si se está creando una interfaz base y devolver el tipo de interfaz local del bean si se está creando una interfaz base local.
- Tener el nombre `create()`.
- Coincidir con un método `ejbCreate()` de la clase del bean sesión. El número y los tipos de argumentos de los métodos `create()` deben

coincidir con sus métodos ejbCreate() correspondientes de la clase del bean sesión.

- Incluir en la cláusula throws todas las excepciones lanzadas por los métodos ejbCreate() y ejbPostCreate() correspondientes en la clase del bean entidad, es decir, el conjunto de excepciones del método create() debe incluir las excepciones de los métodos ejbCreate() y ejbPostCreate(). El tipo de devolución del método ejbCreate() es la clase de la clave principal.
- Utilizar sus parámetros, si los hay, para inicializar el objeto del bean entidad.

## Métodos de búsqueda en beans entidad

Dado que los beans entidad suelen tener una larga duración y pueden utilizarlos varios clientes, es probable que ya exista una instancia cuando la necesite una aplicación cliente. En este caso, el cliente no necesita crear la instancia, sino localizar la que ya está creada. Éste es el motivo por el que la interfaz base y base local de los beans entidad definen uno o varios métodos de búsqueda.

Los beans sesión no necesitan métodos de búsqueda porque sólo sirven a un cliente, la aplicación que los ha creado. El cliente no necesita localizar la instancia del bean sesión; ya sabe dónde está.

Cada interfaz base remota o base local de bean entidad debe definir el método de búsqueda por defecto, findByPrimaryKey(). que permite a los clientes localizar los objetos de entidad mediante una clave principal. Éste es el método findByPrimaryKey() de una interfaz base:

```
<interfaz remota del bean entidad> findByPrimaryKey(<tipo de clave principal>
key)
throws java.rmi.RemoteException, FinderException;
```

Éste es el método findByPrimaryKey() de una interfaz base local:

```
<interfaz remota del bean entidad> findByPrimaryKey(<tipo de clave principal>
key)
throws FinderException;
```

findByPrimaryKey() tiene un solo argumento, la clave principal. Su tipo de devolución es la interfaz remota del bean entidad. En el descriptor de distribución del bean se indica al contenedor el tipo de la clave principal. findByPrimaryKey() devuelve siempre un solo objeto de entidad.

Es posible definir más métodos de búsqueda para las interfaces base remota y local. Si la persistencia está gestionada por bean, cada método de búsqueda debe tener la implementación correspondiente en la clase del bean entidad. Si la persistencia está gestionada por contenedor, es éste el que implementa los métodos de búsqueda. Los métodos de búsqueda deben cumplir estas convenciones:

- En una interfaz base remota, el tipo devuelto es el de la interfaz remota, y si el método de búsqueda devuelve varios objetos de entidad, se devuelve un tipo de conjunto cuyo tipo de contenido es el de la interfaz remota. En una interfaz base local, el tipo devuelto es el de la interfaz local, y si el método de búsqueda devuelve varios objetos de entidad, se devuelve un tipo de conjunto cuyo tipo de contenido es el de la interfaz local. Los tipos de conjunto válidos de Java son `java.util.Enumeration` y, para los beans EJB 2.0, `java.util.Collection`.
- El método de búsqueda siempre comienza con el prefijo **find**. El método de búsqueda correspondiente de la clase del bean entidad con persistencia gestionada por bean comienza con el prefijo **ejbFind**.
- El método debe lanzar la excepción `java.rmi.RemoteException` si se define en una interfaz base. Los buscadores de las interfaces base locales **no** deben lanzar `RemoteException`.
- El método debe lanzar la excepción `java.ejb.FinderException`.
- La cláusula `throws` del método de búsqueda de la interfaz base remota/base local debe coincidir con la misma cláusula del método `ejbFind<xxx>` correspondiente de la clase del bean entidad.

El siguiente ejemplo de interfaz local contiene dos métodos `create()` y dos métodos de búsqueda. Las partes en negrita son obligatorias:

```
public interface AccountHome extends javax.ejb.EJBHome {

    Account create(String accountID)
    throws java.rmi.RemoteException, javax.ejb.CreateException;

    Account create(String accountID, float initialBalance)
    throws java.rmi.RemoteException, javax.ejb.CreateException;

    Account findByPrimaryKey(String key)
    throws java.rmi.RemoteException, javax.ejb.FinderException;

    Account findBySocialSecurity(String socialSecurityNumber)
    throws java.rmi.RemoteException, javax.ejb.FinderException;
}
```

Tenga en cuenta que los componentes EJB 2.0 también pueden incluir métodos base empresariales. Para obtener más información, consulte “[Adición de métodos base empresariales](#)” en la página 7-24.

## Creación de la interfaz remota o local

---

La interfaz remota o local creada para el enterprise bean describe los métodos empresariales a los que puede llamar una aplicación cliente. Mientras se definen los métodos en la interfaz remota o local, estos mismos métodos se implementan en la clase del enterprise bean. El cliente del enterprise bean nunca accede a éste directamente; accede a sus métodos a través de la interfaz remota o local.

Para crear una interfaz remota:

- Declare una interfaz que amplíe `javax.ejb.EJBObject`.
- Declare en la interfaz remota todos los métodos empresariales del enterprise bean a los que desea que pueda llamar la aplicación, y haga que las firmas coincidan exactamente con las de la clase del bean.

Para crear una interfaz local:

- Declare una interfaz que amplíe `javax.ejb.EJBLocalObject`.
- Declare en la interfaz local todos los métodos empresariales del enterprise bean a los que desea que pueda llamar la aplicación, y haga que las firmas coincidan exactamente con las de la clase del bean.

Cuando se utilizan los asistentes para EJB, JBuilder crea una interfaz remota que amplía `EJBObject`.

Todos los métodos definidos en la interfaz remota o local deben seguir estas normas, que son las mismas para los beans sesión y los beans entidad:

- Debe ser público.
- Debe lanzar la excepción `java.rmi.RemoteException` si es un método de una interfaz remota. No debe lanzar `RemoteException` si es un método de una interfaz local.
- En la interfaz remota debe existir un método por cada método de la clase del enterprise bean al que se desea que pueda llamar un cliente. Los métodos de la interfaz remota/local y el bean deben tener el mismo nombre, el mismo número y tipos de argumentos y el mismo tipo de devolución, y deben lanzar las mismas excepciones o un subconjunto de las excepciones de los métodos de la interfaz remota.

A continuación se muestra el código de una interfaz remota de ejemplo llamada `Atm`, correspondiente a un bean sesión llamado `ATM`. La interfaz remota `Atm` define un método empresarial llamado `transfer()`. Las partes en negrita son obligatorias:

```
public interface Atm extends javax.ejb.EJBObject{
    public void transfer(String source, String target, float amount)
```

```

        throws java.rmi.RemoteException, InsufficientFundsException;
    }
}

```

El método `transfer()` declarado en la interfaz `Atm` lanza dos excepciones: `java.rmi.RemoteException`, que es necesaria, y `InsufficientFundsException`, que es específica de una aplicación.

## Las interfaces EJBObject y EJBLocal Object

---

La interfaz remota amplía la interfaz `javax.ejb.EJBObject`. Éste es el código fuente de `EJBObject`:

```

package javax.ejb;
public interface EJBObject extends java.rmi.Remote {
    public EJBHome getEJBHome() throws java.rmi.RemoteException;
    public Object getPrimaryKey() throws java.rmi.RemoteException;
    public void remove() throws java.rmi.RemoteException, RemoveException;
    public Handle getHandle() throws java.rmi.RemoteException;
    boolean isIdentical (EJBObject other) throws java.rmi.RemoteException;
}

```

El método `getEJBHome()` permite a una aplicación obtener la interfaz local del bean. Si se trata de un bean entidad, el método `getPrimaryKey()` devuelve su clave principal. El método `remove()` borra el enterprise bean. `getHandle()` devuelve un identificador persistente de la instancia del bean. `isIdentical()` permite comparar dos enterprise beans.

El contenedor de EJB crea una interfaz `EJBObject` para el enterprise bean. Dado que la interfaz remota amplía `EJBObject`, la interfaz `EJBObject` creada por el contenedor incluye implementaciones para todos los métodos de `EJBObject` y los métodos empresariales definidos en la interfaz remota. La interfaz `EJBObject` de la que se crea una instancia se puede ver en toda la red y hace las veces de proxy para el bean. Tiene un stub y una estructura. El bean no se puede ver en toda la red.

Ésta es la interfaz `EJBLocalObject` completa:

```

package javax.ejb;
public interface EJBLocalObject {
    public EJBHome getEJBLocalHome() throws EJBException;
    public Object getPrimaryKey() throws EJBException;
    public void ejbStore() throws EJBException, RemoteException;
    boolean isIdentical (EJBLocalObject other) throws EJBException;
}

```



# 19

## Desarrollo de clientes de enterprise beans

El cliente de un enterprise bean es una aplicación, una aplicación autónoma, un servlet, un applet u otro enterprise bean. En cualquiera de los casos, el cliente debe hacer lo siguiente para utilizar un enterprise bean:

- Localizar la interfaz local del bean. La especificación EJB estipula que el cliente debería utilizar la API JNDI (Java Naming and Directory Interface) para encontrar las interfaces base.
- Obtener una referencia a la interfaz remota de un objeto de enterprise bean. Para esto hay que utilizar métodos definidos en la interfaz local del bean. Estos métodos permiten crear un bean sesión y crear o buscar un bean entidad.
- Llamar a uno o varios métodos definidos por el enterprise bean. El cliente no llama directamente a los métodos definidos por el enterprise bean, sino que llama a los métodos de la interfaz remota o local del objeto de enterprise bean. Los métodos definidos en la interfaz remota o local son los que el enterprise bean pone a disposición de los clientes.

En los apartados siguientes se describe la aplicación clientes SortClient.java, que llama al bean sesión de ejemplo SortBean. SortBean es un bean sesión sin estado que implementa un algoritmo de fusión y ordenación. Éste es el código de SortClient:

```
// SortClient.java  
...  
public class SortClient {  
    ...  
    public static void main(String[] args) throws Exception {
```

```
javax.naming.Context context;
{ // obtener un contexto JNDI utilizando un servicio de denominación
    context = new javax.naming.InitialContext();
}
Object objref = context.lookup("sort");
SortHome home = (SortHome) javax.rmi.PortableRemoteObject.narrow(objref,
    SortHome.class);
Sort sort = home.create();
... // realice la clasificación y fusione el trabajo
sort.remove();
}
}
```

## Localización de la interfaz base

---

SortClient importa las clases JNDI necesarias y las interfaces local y remota de SortBean. El cliente utiliza la API de JNDI para buscar la interfaz local del enterprise bean. En primer lugar, el cliente debe obtener un contexto de denominación inicial JNDI. El código de SortClient crea una instancia de un nuevo objeto javax.naming.Context denominada InitialContext. Después, el cliente utiliza el método de contexto lookup() para resolver el nombre en una interfaz base.

El método lookup() del contexto devuelve un objeto del tipo java.lang.Object. El código debe convertir este objeto devuelto en el tipo esperado. El código de SortClient muestra una parte del código del cliente para el ejemplo de ordenación. La rutina main() empieza por utilizar el servicio de denominación JNDI y su método de contexto lookup() para buscar la interfaz base. El nombre de la interfaz remota, que en este caso es sort, se pasa al método context.lookup(). Observe que, al final, el programa clasifica el resultado del método context.lookup() como SortHome, el tipo de la interfaz base.

## Obtención de la interfaz remota/local

---

Después de obtener la interfaz base de un enterprise bean se necesita una referencia a su interfaz remota o local. Para ello se utilizan los métodos de creación y de búsqueda de la interfaz local. El método adecuado depende del tipo del enterprise bean y de los métodos que haya definido su proveedor en la interfaz base.

### Beans sesión

---

Si el enterprise bean es un bean sesión, el cliente utiliza un método de creación para devolver la interfaz remota o local. Los beans sesión no tienen métodos de búsqueda. Si el bean sesión no tiene estado, sólo tendrá

un método `create()`, al que el cliente debe llamar para obtener la interfaz remota o local. El método `create()` por defecto no tiene parámetros. Por tanto, en el ejemplo de código `SortClient`, la llamada a la interfaz remota o local tiene el siguiente aspecto:

```
Sort sort = home.create();
```

El ejemplo del carro que se trata en el [Capítulo 21, “Tutorial: Desarrollo de beans sesión con el diseñador de EJB”](#), utiliza un bean sesión con estado. El método `create()` tiene tres parámetros, que en conjunto identifican al comprador del contenido del carro, y devuelve una referencia a la interfaz remota de `Cart`. `CartClient` define los valores de tres parámetros: `cardHolderName`, `creditCardNumber` y `expirationDate`. A continuación llama al método `create()`. Éste es el código:

```
...
String cardHolderName = "Suzy Programmer";
String creditCardNumber = "1234-5678-9012-3456";
Date expirationDate = new GregorianCalendar(2004, Calendar.JULY, 30).getTime();

Cart cart = null;

try {
    cart = home.create(cardHolderName, creditCardNumber, expirationDate);
} catch (Exception e) {
    System.out.println("Could not create Cart session bean\n" + e);
}
```

## Beans entidad

---

Si se trata de un bean entidad, se debe utilizar un método de creación o de búsqueda con el fin de obtener la interfaz remota o local. Dado que los objetos de entidad representan datos subyacentes, persistentes, almacenados en una base de datos, la duración de los beans entidad suele ser prolongada. Por ello, el cliente necesita a menudo buscar el bean entidad que representa estos datos en lugar de crear un objeto de entidad, que crearía y guardaría nuevos datos en la base de datos subyacente.

El cliente utiliza una operación de búsqueda para localizar un objeto de entidad, como una fila de una tabla de una base de datos relacional. Esto significa que las operaciones de búsqueda localizan entidades de datos insertadas previamente en el sistema de almacenamiento. Los datos pueden haberse añadido al almacén de datos mediante un bean entidad o fuera del contexto de EJB. Por ejemplo, se pueden añadir directamente desde el sistema de gestión de bases de datos (SGBD). En el caso de los sistemas anteriores, es posible que los datos existieran antes de la instalación del contenedor EJB.

El cliente utiliza el método `create()` de un objeto de bean entidad para crear una entidad de datos que se almacena en la base de datos subyacente. El método `create()` del bean entidad inserta el estado de la

entidad en la base de datos y devuelve a las variables los valores de los parámetros del método `create()`.

Todas las instancias de los beans entidad deben tener una clave principal que las identifique de forma única. También pueden tener claves secundarias, que se pueden utilizar para buscar objetos de entidad concretos.

## Los métodos de búsqueda y la clase de la clave principal

El método de búsqueda por defecto de los beans entidad es `findByPrimaryKey()`, que busca el objeto entidad utilizando su valor de clave principal. Ésta es su firma:

```
public <interfaz remota> findByPrimaryKey(<tipo clave> primaryKey)
```

Todos los beans entidad deben implementar un método `findByPrimaryKey()`. El parámetro `primaryKey` es una clase de clave principal independiente, que se define en el descriptor de distribución. El tipo de clave es el tipo de la clave principal, y debe tener un tipo de valor válido de RMI-IIOP. La clase de clave principal puede ser cualquier clase, como una clase Java o una clase de creación propia.

Por ejemplo, supongamos que se ha definido la clase de clave principal `AccountPK` del bean entidad `Account`. `AccountPK`, del tipo `String` guarda el identificador del bean `Account`. Para obtener una referencia a una instancia determinada del bean entidad `Account`, asigne a `AccountPK` el identificador del bean y llame al método `findByPrimaryKey()` como se muestra aquí:

```
AccountPK accountPK = new AccountPK("1234-56-789");
Account source = accountHome.findByPrimaryKey(accountPK);
```

Los proveedores de beans pueden definir más métodos de búsqueda para uso del cliente.

Si desea más información sobre el uso del diseñador de EJB para añadir métodos de buscador a beans entidad EJB 2.0, consulte “[Adición de métodos de búsqueda](#)” en la página 7-22.

## Métodos de creación y eliminación

Los clientes también pueden crear beans entidad con los métodos de creación definidos en la interfaz base. Cuando un cliente llama a un método de creación de un bean entidad, la nueva instancia del objeto de entidad se guarda en el almacén de datos. Los objetos de entidad creados siempre están identificados mediante un valor de clave principal. Su estado se puede restablecer con valores pasados como parámetros al método de creación.

Se debe tener en cuenta que los beans entidad existen mientras los datos permanecen en la base de datos. La vida del bean entidad no está asociada a la sesión del cliente. Es posible eliminarlo llamando a uno de sus métodos de eliminación. Estos métodos eliminan de la base de datos el

bean y la representación subyacente de los datos de entidad. También es posible borrar objetos enteros, como cuando se borra un registro de base de datos por medio del SGBD o una aplicación distinta.

## Llamadas a métodos

---

Después de obtener una referencia a la interfaz remota o local del bean, el cliente puede llamar a los métodos definidos en ella. El cliente está más interesado en los métodos que forman la lógica empresarial del bean.

Por ejemplo, el código siguiente es de un cliente que accede al bean sesión `Cart`. El fragmento reproducido empieza en el momento en que ha creado una instancia de bean sesión para el poseedor de una tarjeta y ha recuperado una referencia `Cart` en la interfaz remota. El cliente está preparado para llamar a los métodos del bean:

```
...
Cart cart = null;

// obtener una referencia a la interfaz local de un identificador
try {
    cart = home.create(cardHolderName, creditCardNumber, expirationDate);
} catch (Exception e) {
    System.out.println("Could not create Cart session bean\n" + e);
}

if (cart != null) {

    Item kassemBook = new Item("J2EE Blueprints", 39.99f, "Book");

    try {
        cart.addItem(kassemBook);
    } catch (Exception e) {
        System.out.println("Could not add the book to the cart\n" + e);
    }
    ...

    ...
    // muestra una lista de los elementos del carro
    summarize(cart);
    cart.removeItem(kassemBook);
    ...
}
```

En primer lugar, el cliente crea un objeto de elemento y define sus parámetros `title`, `price` y `type`. A continuación llama al método empresarial `addItem()` del enterprise bean para añadir el objeto del elemento al carro de la compra. El bean sesión `Cart` define el método `addItem()` y la interfaz remota `Cart` lo hace público. El cliente añade otros elementos, que no se muestran aquí, y llama a su propio método `summarize()` para enumerar los artículos que contiene el carro. A

continuación llama al método `remove()` con el objeto de eliminar la instancia del bean. El cliente llama a los métodos del enterprise bean del mismo modo que a cualquier otro método, como su propio método `summarize()`.

Si desea seguir un tutorial para la creación de esta aplicación cliente que accede al bean sesión `Cart`, consulte el [Capítulo 22, “Tutorial: Creación de aplicaciones cliente de prueba”](#).

## Eliminación de instancias de beans

---

El método `remove()` funciona de forma distinta en los beans sesión y en los de entidad. Dado que los objetos de sesión existen para un cliente y no son persistentes, el cliente del bean sesión debe llamar al método `remove()` tras terminar con el objeto de sesión. El cliente dispone de dos métodos `remove()`: el cliente puede eliminar el objeto de sesión con el método `javax.ejb.EJBObject.remove()`, o eliminar el identificador de sesión con el método `javax.ejb.EJBHome.remove(Handle handle)`. Para obtener más información sobre los paquetes, consulte [“Referencia a beans mediante identificadores” en la página 19-6](#).

Aunque no es necesario que el cliente elimine el objeto de sesión, resulta conveniente. Si el cliente no elimina un objeto de bean sesión con estado, el contenedor lo elimina cuando transcurre el tiempo establecido mediante el valor de tiempo de espera. Este valor es una propiedad de distribución. Sin embargo, el cliente también puede conservar un identificador de la sesión para hacer referencias en lo sucesivo.

Los clientes de los beans entidad no tienen este problema, ya que la asociación sólo se establece mientras dura la transacción y el contenedor gestiona su ciclo de vida, lo que incluye la activación y la desactivación. El cliente del bean entidad llama al método `remove()` del bean sólo cuando se va a eliminar el objeto de entidad de la base de datos subyacente.

## Referencia a beans mediante identificadores

---

Otra forma de hacer referencias a los enterprise beans consiste en utilizar identificadores. Un identificador es una referencia serializable a un bean. Se puede obtener un identificador de la interfaz base del bean. Después de obtener el identificador se puede escribir en un archivo u otro sistema de almacenamiento persistente. Más adelante se puede recuperar para restablecer una referencia al enterprise bean.

Sin embargo, el identificador de la interfaz remota/local sólo se puede utilizar para reproducir la referencia al bean. No es posible utilizarlo para reproducir el propio bean. Si otro proceso ha eliminado el bean, o si el sistema ha eliminado su instancia, se lanza una excepción cuando el

cliente intenta utilizar el identificador para restablecer la referencia al bean.

Si no está seguro de que la instancia del bean siga existiendo, en lugar de utilizar un identificador de la interfaz remota se puede almacenar el identificador base del bean y volver a crear el objeto de bean más adelante, llamando a su método de creación o de búsqueda.

Después de crear una instancia de bean, el cliente puede utilizar el método `getHandle()` con el fin de obtener un identificador de esta instancia. Este bean se puede escribir en un archivo serializado. Más adelante, el programa cliente puede leer el archivo y atribuir el objeto a un tipo `Handle`. A continuación, llama al método `getEJBObject()` del identificador para obtener la referencia al bean y restringe el resultado de `getEJBObject()` al tipo correcto para el bean.

Por ejemplo, el programa `CartClient` puede hacer lo siguiente para utilizar un identificador del bean sesión `Cart`:

```

import java.io;
import javax.ejb.Handle;
...
Cart cart;
...
cart = home.create(cardHolderName, creditCardNumber, expirationDate);

// llamar a getHandle() en el carro para obtener su identificador
cartHandle = cart.getHandle();

// escribir el identificador al archivo serializado
FileOutputStream f = new FileOutputStream ("carthandle.ser");
ObjectOutputStream o = new ObjectOutputStream(f);
o.writeObject(myHandle);
o.flush();
o.close();
...

// leer el identificador del archivo más tarde
FileInputStream fi = new FileInputStream ("carthandle.ser");
ObjectInputStream oi = new ObjectInputStream(fi);

//leer el objeto desde el archivo y convertir en Identificador
cartHandle = (Handle)oi.readObject();
oi.close();
...

// Utilizar el identificador para referenciar la instancia del bean
Cart cart = (Cart) javax.rmi.PortableRemoteObject.narrow(cartHandle,
                                                               getEJBObject(),
                                                               Cart.class);
...

```

Cuando termina con el identificador del bean sesión, el cliente puede eliminarlo mediante una llamada al método `javax.ejb.EJBHome.remove(Handle handle)`.

## Gestión de las transacciones

---

Los programas cliente también pueden gestionar sus propias transacciones en vez de delegar su gestión al enterprise bean o al contenedor. Los clientes gestionan sus propias transacciones exactamente de la misma forma que los beans sesión.

Cuando un cliente gestiona sus propias transacciones es responsable de definir las limitaciones. Esto significa que debe iniciar las transacciones expresamente, y después ponerles fin mediante el envío o la cancelación.

Los clientes utilizan la interfaz `javax.transaction.UserTransaction` para gestionar sus propias transacciones. En primer lugar deben obtener una referencia a la interfaz `UserTransaction`, utilizando para ello JNDI. Cuando tiene el contexto `UserTransaction`, el cliente utiliza el método `UserTransaction.begin()` para dar comienzo a la transacción. Después, para ponerle fin, la envía con el método `UserTransaction.commit()`, o la cancela con `UserTransaction.rollback()`. Mientras tanto, el cliente accede a objetos EJB y de otros tipos.

En este código se muestra la forma en que un cliente gestiona sus propias transacciones. El código relacionado específicamente con las transacciones gestionadas por el cliente se resalta en negrita:

```
...
import javax.naming.InitialContext;
import javax.transaction.UserTransaction;
...
public class clientTransaction {
    public static void main(String[] args) {
        InitialContext initContext = new InitialContext();
UserTransaction ut = null;ut =
        (UserTransaction)initContext.lookup("java:comp/UserTransaction");

        // comenzar una transacción
ut.begin();

        // hacer transacción
        ...

        // confirmar o cancelar la transacción
        ut.commit(); // or ut.rollback();
        ...
    }
}
```

Para obtener más información sobre los puntos de interrupción, consulte el [Capítulo 20, “Gestión de las transacciones”](#).

## Detección de información sobre el bean

---

La información sobre los enterprise beans se conoce como “metadatos”. Un cliente puede obtener metadatos sobre un bean mediante el método `getMetaData()` de la interfaz local del enterprise bean.

El método `getMetaData()` se utiliza a menudo en entornos de desarrollo y constructores de herramientas, donde es necesario detectar información sobre un enterprise bean, por ejemplo, para asociar beans ya instalados. Los clientes de guiones también pueden necesitar metadatos sobre el bean.

Después de recuperar la referencia de la interfaz base, el cliente puede llamar a su método `getEJBMetaData()`. A continuación puede llamar a los métodos de la interfaz `EJBMetaData` para extraer información como la siguiente:

- Interfaz local `EJBHome` del bean, con el método `EJBMetaData.getEJBHome()`.
- Objeto de clase de la interfaz local del bean, incluidas sus interfaces, clases, campos y métodos, con el método `EJBMetaData.getHomeInterfaceClass()`.
- Objeto de clase de la interfaz remota del bean, que incluye toda la información sobre la clase, con el método `EJBMetaData.getRemoteInterfaceClass()`.
- Objeto de clase de la clave principal del bean, con el método `EJBMetaData.getPrimaryKeyClass()`.
- Determinar si se trata de un bean sesión o de un bean entidad, con el método `EJBMetaData.isSession()`. El método devuelve `true` si el bean es de sesión.
- Determinar si un bean sesión tiene estado, con el método `EJBMetaData.isStatelessSession()`. El método devuelve `true` si el bean sesión no tiene estado.

Ésta es la interfaz `EJBMetaData` completa:

```
package javax.ejb;

public interface EJBMetaData {
    EJBHome getEJBHome();
    Class getHomeInterfaceClass();
    Class getRemoteInterfaceClass();
    Class getPrimaryKeyClass();
    boolean isSession();
    boolean isStatelessSession();
}
```

## Creación de clientes con JBuilder

---

Se puede utilizar JBuilder para iniciar la creación del cliente. JBuilder tiene un Asistente para clientes de prueba EJB que sirve para crear una sencilla aplicación cliente con la que probar el enterprise bean. También se puede utilizar como base para la creación de la aplicación cliente real. Indique al asistente del nombre de uno de los enterprise beans a los que accede el cliente, y se escribirá el código de obtención de contexto de denominación, se localizará la interfaz base del bean y se asegurará la referencia a la interfaz remota/local.

Sin embargo, es probable que el cliente llame a varios beans, por lo que será necesario escribir estos pasos en el código para cada bean. También será necesario añadir al código del cliente, de forma manual, las llamadas que acceden a la lógica empresarial de los enterprise beans.

Para más información acerca del uso del Asistente para cliente de prueba EJB, consulte “[Creación de aplicaciones cliente de prueba](#)” en la [página 11-4](#).

# 20

## Gestión de las transacciones

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

Resulta muy cómodo desarrollar aplicaciones en plataformas como Java 2 Enterprise Edition (J2EE) que aceptan las transacciones. Los sistemas basados en transacciones simplifican el desarrollo de aplicaciones porque evitan al desarrollador tener que llevar a cabo las tediosas tareas de recuperación de fallos y programación para varios usuarios. Las transacciones no están limitadas a una base de datos o una página. Las transacciones distribuidas pueden actualizar simultáneamente varias bases de datos en varias páginas.

Normalmente, los desarrolladores dividen el trabajo total de la aplicación en varias unidades. Cada unidad de trabajo constituye una transacción. A medida que avanza la aplicación, el sistema subyacente garantiza que todas las unidades de trabajo (todas las transacciones) se completan totalmente sin interferencia de otros procesos. Si no es así, el sistema cancela la transacción y deshace por completo los cambios que haya efectuado, dejando la aplicación en el mismo estado que antes de que comenzara la transacción.

### Características de las transacciones

Normalmente, las transacciones se utilizan para acceder a una base de datos. Todos los accesos a la base de datos ocurren dentro de una transacción. Todas estas transacciones comparten estas características, conocidas con las siglas ACID (en inglés):

- Atomicidad

Normalmente, las transacciones constan de más de una operación. La atomicidad requiere que todas las operaciones de una transacción se lleven a cabo correctamente para considerar que la transacción está

completa. Si no es posible efectuar todas las operaciones, no se permite la realización de ninguna.

- Consistencia

Se trata de la coherencia de la base de datos. Una transacción debe llevar la base de datos de un estado consistente a otro, y debe conservar su integridad semántica y física.

- Aislamiento

El aislamiento requiere que todas las transacciones parezcan ser las únicas que manipulan actualmente los datos de la base de datos. Aunque es posible ejecutar otras transacciones a la vez, ninguna de ellas debe detectar estas manipulaciones hasta que terminen correctamente y envíen su trabajo. A causa de las interdependencias de las actualizaciones, una transacción podría obtener una vista incoherente de los datos si detectara un conjunto de las actualizaciones de otra transacción. El aislamiento protege las transacciones de este tipo de inconsistencia.

El aislamiento está relacionado con la concurrencia de las transacciones. Existen distintos niveles de aislamiento. Los mayores limitan el alcance de la concurrencia. Se da el mayor nivel de aislamiento cuando se pueden serializar todas las transacciones, es decir, cuando el contenido de la base de datos presenta el mismo aspecto que si cada transacción se completara antes de que empezara la siguiente. Sin embargo, algunas aplicaciones pueden tolerar un nivel reducido de aislamiento para permitir más concurrencia. Normalmente, estas aplicaciones ejecutan muchas transacciones concurrentes, aunque lean datos que pueden actualizarse parcialmente, con lo que se producirá una incoherencia.

- Persistencia

La persistencia significa que las actualizaciones efectuadas por las transacciones enviadas permanecen en la base de datos independientemente de las condiciones de fallo. Esto garantiza que las actualizaciones no se desharán aunque ocurra un fallo después de la operación de envío, y que las bases de datos se pueden recuperar después de un fallo del sistema o del soporte.

## Aceptación de transacciones en el contenedor

---

Los contenedores EJB aceptan las transacciones lineales, pero no las anidadas. También propaga las transacciones de forma implícita. Esto significa que no es necesario pasar explícitamente el contexto de la transacción como parámetro, porque el contenedor gestiona esta tarea por el cliente de forma transparente.

Es necesario tener en cuenta que, aunque las JSP y los servlets pueden hacer de clientes, su cometido no es el de ser componentes transaccionales. Para las tareas transaccionales es conveniente utilizar enterprise beans. Cuando se llama a un enterprise bean para que efectúe una tarea transaccional, el bean y el contenedor configuran la transacción correctamente.

## Enterprise beans y transacciones

---

Los enterprise beans y el contenedor EJB simplifican enormemente la gestión de transacciones. Gracias a los enterprise beans, las aplicaciones pueden actualizar en una sola transacción los datos de varias bases de datos, residentes en distintos servidores EJB.

Hasta ahora, las aplicaciones responsables de gestionar las transacciones debían realizar estas tareas:

- Creación del objeto de transacción.
- Inicio explícito de la transacción.
- Seguimiento del contexto de la transacción.
- Envío de la transacción al completarse todas las actualizaciones.

Para crear aplicaciones de estas características era necesario contar con amplios conocimientos de programación y los errores no eran infrecuentes.

Con los enterprise beans, el contenedor gestiona la mayoría de los aspectos de la transacción, o incluso todos ellos. Inicia y finaliza la transacción, y mantiene su contexto mientras dura el objeto. La responsabilidad del desarrollador se reduce enormemente, sobre todo en las transacciones de entornos distribuidos.

Los atributos de transacción de los enterprise beans se declaran durante la distribución. Estos atributos de transacción indican si el contenedor gestiona las transacciones del bean o el bean gestiona sus propias transacciones, y hasta qué punto.

## Transacciones gestionadas por contenedor y gestionadas por bean

---

Cuando un enterprise bean efectúa su propia demarcación de transacciones como parte de los métodos empresariales, se considera que el bean gestiona sus transacciones. (Demarcar una transacción es indicar dónde empieza y dónde termina.) Cuando un bean delega la demarcación de todas las transacciones en su contenedor EJB, éste demarca las transacciones basándose en las instrucciones de distribución del

ensamblador de aplicaciones. En este caso se habla de transacciones gestionadas por contenedor.

Los beans sesión, independientemente de que tengan estado, pueden utilizar los dos tipos de transacciones. Sin embargo, no pueden utilizar los dos tipos de gestión de transacciones a la vez. El proveedor de los beans decide qué tipo de beans sesión se van a utilizar. Los beans entidad sólo pueden utilizar las transacciones gestionadas por contenedor.

Puede ser conveniente que los beans gestionen sus transacciones si se desea que éstas comiencen como parte de una operación y terminen como parte de otra. Sin embargo, puede haber problemas si una operación llama al método de inicio de la transacción pero ninguna llama al de finalización.

Siempre que sea posible se deben escribir enterprise beans con transacciones gestionadas por contenedor. Requieren menos trabajo por parte del desarrollador y son menos propensos a los errores. Además resulta más fácil personalizar beans con transacciones gestionadas por contenedor y utilizarlos para crear otros beans.

## Transacciones locales y globales

---

Cuando existe una sola conexión a una base de datos, el enterprise bean puede controlar directamente la transacción mediante una llamada al método `commit()` o `rollback()` de la conexión. Este tipo de transacción es local. Las transacciones globales permiten a todas las conexiones a la base de datos registrarse en el servicio de transacción global, que la gestiona. En las transacciones globales, el enterprise bean nunca efectúa por sí mismo llamadas directamente a conexiones a bases de datos.

Un bean que gestiona sus propias demarcaciones de transacción utiliza la interfaz `javax.transaction.UserTransaction` para identificar los límites de una transacción global. Cuando un bean utiliza la demarcación gestionada por contenedor, éste interrumpe las llamadas del cliente para controlar la demarcación de transacciones, con el atributo que define el ensamblador de aplicaciones en el descriptor de distribución del bean. El atributo de transacción también determina si la transacción es local o global.

Para las transacciones gestionadas por contenedor, éste sigue ciertas normas para determinar si debe efectuar una transacción local o global. Normalmente, el contenedor llama al método dentro de la transacción local después de comprobar que no existe ninguna transacción global. También comprueba que no se espera que inicie una transacción global y que se definan los atributos de las transacciones gestionadas por contenedor. El contenedor engloba automáticamente una llamada a un método dentro de una transacción local si se da una de las siguientes condiciones:

- Se asigna a la transacción el atributo NotSupported, y el contenedor detecta que se ha accedido a los recursos de la base de datos.
- Se asigna a la transacción el atributo Supported, y el contenedor detecta que no se ha llamado al método desde dentro de una transacción global.
- Se asigna a la transacción el atributo Never, y el contenedor detecta que se accede a los recursos de la base de datos.

## API de transacciones

---

Todas las transacciones utilizan el API de transacciones de Java (JTA). Cuando las transacciones están gestionadas por contenedor, la plataforma gestiona la demarcación de límites de las transacciones, y el contenedor utiliza el API JTA. No es necesario utilizar nunca esta API en el código del bean.

Si el bean gestiona sus propias transacciones, debe utilizar la interfaz JTA `javax.transaction.UserTransaction`. Esta interfaz permite que un cliente o componente pueda demarcar los límites de la transacción. Los enterprise que utilizan transacciones gestionadas por contenedor deben usar el método `EJBContext.getUserTransaction()`.

Además, todos los clientes transaccionales utilizan JNDI para consultar la interfaz `UserTransaction`. Esto se consigue construyendo una JNDI `InitialContext` mediante el servicio de denominación JNDI, como se muestra aquí:

```
javax.naming.Context context = new javax.naming.InitialContext();
```

Cuando el bean tiene `InitialContext`, puede utilizar la operación `lookup()` de JNDI para obtener la interfaz `UserTransaction`:

```
javax.transaction.UserTransaction utx = (javax.transaction.UserTransaction)
context.lookup("java:comp/UserTransaction")
```

No olvide que los enterprise beans pueden obtener referencias a la transacción `UserTransaction` desde el objeto `EJBContext`. El bean puede utilizar el método `EJBContext.getUserTransaction()` en lugar de tener que obtener un objeto `InitialContext` y utilizar después el método JNDI `lookup()`. Sin embargo, los clientes de transacciones que no sean enterprise beans deben utilizar la consulta con JNDI.

Cuando el bean o el cliente tiene la referencia a la interfaz `UserTransaction`, puede iniciar sus propias transacciones y gestionarlas, es decir, se pueden utilizar los métodos de la interfaz `UserTransaction` para iniciar y enviar o cancelar transacciones. Se utiliza el método `begin()` para iniciar la transacción, y después `commit()` para enviar los cambios a la base de datos. También se puede utilizar el método `rollback()` para cancelar todos los cambios efectuados dentro de la transacción y restablecer la base de datos.

en el estado anterior al comienzo de la transacción. Se debe incluir código para efectuar la lógica empresarial de la transacción, entre los métodos `begin()` y `commit()`. A continuación, se propone un ejemplo.

```
public class NewSessionBean implements SessionBean {  
    EJBContext ejbContext;  
  
    public void doSomething(...) {  
        javax.transaction.UserTransaction utx;  
        javax.sql.DataSource dataSource1;  
        javax.sql.DataSource dataSource2;  
        java.sql.Connection firstConnection;  
        java.sql.Connection secondConnection;  
        java.sql.Statement firstStatement;  
        java.sql.Statement secondStatement;  
  
        java.naming.Context context = new javax.naming.InitialContext();  
  
        dataSource1 = (javax.sql.DataSource)context.lookup("java:comp/env/jdbcDatabase1");  
        firstConnection = dataSource1.getConnection();  
  
        firstStatement = firstConnection.createStatement();  
  
        dataSource2 = (javax.sql.DataSource)context.lookup("java:comp/env/jdbcDatabase2");  
        secondConnection = dataSource2.getConnection();  
  
        secondStatement = secondConnection.createStatement();  
  
        utx = ejbContext.getUserTransaction();  
  
        utx.begin();  
  
        firstStatement.executeQuery(...);  
        firstStatement.executeUpdate(...);  
        secondStatement.executeQuery(...);  
        secondStatement.executeUpdate(...);  
  
        utx.commit();  
  
        firstStatement.close;  
        secondStatement.close  
        firstConnection.close();  
        secondConnection.close();  
    }  
    ...  
}
```

# Gestión de excepciones de transacción

---

Los enterprise beans pueden lanzar excepciones en la aplicación y el sistema si se detectan errores durante la gestión de las transacciones. Las excepciones de aplicación están derivadas de errores en la lógica empresarial. La aplicación que efectúa la llamada debe gestionarlas. Las excepciones de sistema, como los errores de ejecución, llegan más allá de la aplicación y pueden estar gestionadas por ésta, por el enterprise bean o por el contenedor del bean.

El enterprise bean declara las excepciones de aplicación y sistema en las cláusulas `throws` de sus interfaces base y remota/local. Es necesario buscar excepciones comprobadas en el bloque `try` o `catch` al llamar a los métodos de los beans.

## Excepciones de sistema

---

Los enterprise beans lanzan una excepción de sistema (normalmente `java.ejb.EJBException`, aunque también puede ser `java.rmi.RemoteException`) para indicar los errores de sistema inesperados. Por ejemplo, se lanza una excepción si no se puede abrir una conexión de base de datos. `java.ejb.EJBException` es una excepción de ejecución y no es necesario incluirla en la cláusula `throws` de los métodos empresariales del bean.

Normalmente, las excepciones de sistema requieren la cancelación de la transacción. A menudo, el contenedor que gestiona el bean efectúa la cancelación, pero en ocasiones debe efectuarla el cliente, sobre todo si las transacciones están gestionadas por bean.

## Excepciones de aplicación

---

Los beans lanzan excepciones de aplicación para indicar condiciones de error propias de la aplicación. Se trata de errores de la lógica empresarial y no de problemas del sistema. Las excepciones de aplicación son distintas de la excepción `java.ejb.EJBException`. Las excepciones de aplicación son excepciones comprobadas, es decir, hay que comprobarlas cuando se llama a un método que pueda lanzar esta excepción.

Los métodos empresariales del bean utilizan excepciones de aplicación para informar de condiciones anómalas, como la introducción de valores inaceptables o cantidades que sobrepasan los límites permitidos. Por ejemplo, un método de un bean que anota el débito del saldo de una cuenta puede lanzar una excepción de aplicación para comunicar que el saldo no es suficiente para efectuar una operación de débito concreta. A menudo, los clientes pueden recuperarse de estos errores de aplicación sin necesidad de cancelar la totalidad de la transacción.

La aplicación o el programa que efectúa la llamada obtiene la excepción lanzada, lo que le permite conocer la naturaleza exacta del problema. Cuando se produce una excepción de aplicación, la instancia del enterprise bean no cancela automáticamente la transacción del cliente. Así, el cliente tiene datos para evaluar el mensaje de error y la oportunidad de efectuar las acciones necesarias para corregir la situación y recuperar la transacción o cancelarla.

## Gestión de excepciones de aplicación

---

Dado que las excepciones de aplicación informan sobre los errores de lógica empresarial, los clientes deben gestionar estas excepciones. Aunque estas excepciones pueden requerir la retirada de las transacciones, no la imponen. El cliente puede volver a intentar realizar la transacción, aunque a menudo es necesario interrumpirla y cancelarla.

El proveedor de beans debe asignarles un estado que haga que, si el cliente continúa con la transacción, no se pierda la integridad de los datos. Si no se puede garantizar esto, se debe imponer la cancelación de la transacción.

### Cancelación de transacciones

Cuando el cliente obtiene una excepción de aplicación, se debe comprobar en primer lugar si la transacción actual está marcada sólo para cancelación. Por ejemplo, un cliente puede recibir una excepción `javax.transaction.TransactionRolledbackException`. Esta excepción indica que el bean enterprise ha fallado y que la transacción se ha cancelado o se ha marcado como sólo para cancelación. Normalmente, el cliente no conoce el contexto de transacciones dentro del cual funcionaba el enterprise bean. El bean puede haber operado en su propio contexto de transacciones independiente o en el correspondiente al programa que ha efectuado la llamada.

Si el enterprise bean operaba en el mismo contexto de transacción que el programa que hace la llamada, el bean (o su contenedor) ya ha impuesto la cancelación de la transacción. Cuando un contenedor EJB marca una transacción para que se cancele, el cliente debe detener todo el trabajo que se efectúa con ella. Normalmente el cliente que utiliza transacciones declarativas obtiene una excepción apropiada, como `javax.transaction.TransactionRolledbackException`. Las transacciones declarativas son aquellas en las que el contenedor gestiona los detalles de la transacción.

Si el cliente es un enterprise bean, debe llamar al método `javax.ejb.EJBContext.getRollbackOnly()` para determinar si su transacción va a cancelarse.

El bean cliente que gestiona las transacciones debe cancelarlas mediante una llamada al método `rollback()` de la interfaz `java.transaction.userTransaction`.

## Opciones para la continuación de transacciones

Cuando una transacción no está marcada para la cancelación, el cliente tiene las siguientes opciones:

- Cancelar la transacción.  
Cuando un cliente recibe una excepción comprobada para una transacción que no está marcada para la cancelación, lo más seguro es cancelarla. Para ello, el cliente marca la transacción como sólo cancelación, o si la ha iniciado, la cancela llamando al método `rollback()`.
- Para transmitir la responsabilidad, se lanza una excepción comprobada o se vuelve a lanzar la original.

El cliente también puede lanzar su propia excepción comprobada o volver a lanzar la original. Al lanzar una excepción, el cliente deja que otros programas más avanzados en la cadena de la transacción decidan si debe interrumpirse. Sin embargo, es preferible que el código o el programa más cercano a la aparición del problema tome la decisión sobre si debe continuar.

- Volver a intentar y continuar la transacción. Para esto puede ser necesario intentar volver a ejecutar partes de la transacción.

El cliente puede continuar con la transacción. Puede evaluar el mensaje de excepción y decidir si es probable que se ejecute correctamente una segunda llamada al método con distintos parámetros. Sin embargo, se debe recordar que intentar volver a ejecutar una transacción puede ser peligroso, ya que el código no detecta si el enterprise bean ha limpiado su estado adecuadamente.

Los clientes que llaman a beans sesión sin estado pueden volver a intentar la realización de las transacciones si pueden determinar el problema a partir de la excepción lanzada. Dado que el bean llamado no tiene estado, no hay que preocuparse por la posibilidad de que sea incorrecto.

**La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.**

Si está utilizando Borland Enterprise Server, consulte la documentación del Borland Enterprise Server, AppServer Edition 5.0.2 -5.1.x, con el fin de obtener información adicional acerca de las transacciones y del contenedor de Borland.



# Tutorial: Desarrollo de beans sesión con el diseñador de EJB

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

En este tutorial se crea un bean sesión “carro de la compra”, que contiene los artículos que compra un cliente en una tienda virtual. En este ejemplo, los artículos a la venta son libros y compact discs. El bean que se crea siguiendo las instrucciones de este tutorial es muy sencillo. El objetivo que se persigue es mostrar cómo se utilizan las características de JBuilder para crear beans sesión.

En este tutorial se desarrolla el bean sesión y se ejecuta en un contenedor local. En otro tutorial posterior, puede crear una aplicación cliente que llama a los métodos del bean, para probar su lógica.

Si desea ver el código creado por este tutorial, consulte “[Código para el bean sesión de carro de la compra](#)” en la página 21-20.

Si desea examinar un bean de carro de compra en línea más complejo, que utiliza además enterprise beans adicionales, páginas JavaServer, servlets y DataExpress, consulte el ejemplo `JBuilder/samples/Ejb/Ejb11/ESite/ESite.jpx` basado en beans EJB 1.1 y Borland Enterprise Server 5.0.2 -5.1.x. También puede encontrar este mismo ejemplo escrito para WebLogic Server 6.1 y beans EJB 2.0 en `JBuilder/samples/Ejb/Ejb20/ESite/esite.jpx`.

El apartado Accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder por parte de 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 1-2.

# Creación de proyectos

Para crear un nuevo proyecto para el bean sesión que va a desarrollar:

- 1 Seleccione Archivo | Nuevo proyecto.

Se abre el asistente para proyectos.

- 2 Defina el nombre del proyecto como `cart_session` y pulse Finalizar.

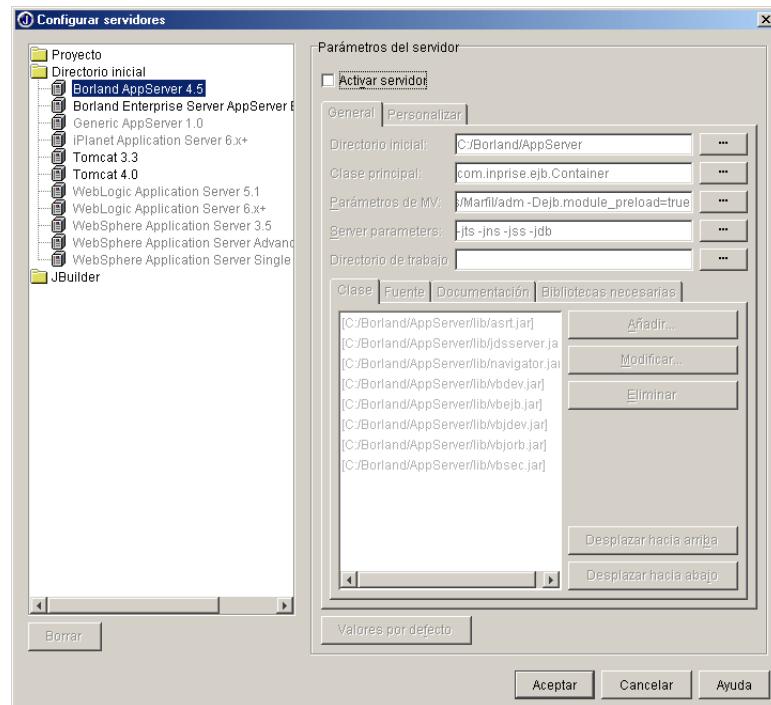
Aparece un nodo `cart_session.jpx` en el panel de proyecto.

## Definición del servidor de aplicaciones de destino

**La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.**

El tutorial crea un bean sesión que se va a distribuir en Borland Enterprise Server. Por lo tanto, si no lo ha hecho todavía, debe instalar JBuilder para seleccionar ese servidor. Siga estos pasos:

- 1 Seleccione Herramientas | Configurar servidores para que aparezca el cuadro de diálogo Configurar servidores:



- 2** Seleccione Borland Enterprise Server, AppServer Edition 5.0.2 -5,1.x, en la lista de servidores de la parte izquierda del cuadro de diálogo.
- 3** Marque la casilla Activar servidores.  
Utilice el botón de puntos suspensivos (...) situado junto al campo Directorio inicial para especificar la ubicación de Borland Enterprise Server AppServer Edition 5.0.2 -5,1.x. Los demás campos contienen valores por defecto que deberían resultar apropiados para las necesidades actuales. Puede realizar cambios en estos valores por defecto si lo necesita.
- 4** Si todavía no está definido, pulse el botón Personalizar y especifique la ubicación del JDK utilizado por Borland Enterprise Server en el directorio de instalación de JDK .
- 5** Pulse Aceptar.
- 6** Cierre y reinicie JBuilder.

Si ya ha instalado y configurado más de un servidor de aplicaciones mediante el cuadro de diálogo Configurar Enterprise, debe asegurarse de que ha seleccionado el servidor de aplicaciones correcto para el proyecto:

- 1** Seleccione Proyecto | Propiedades de proyecto.
- 2** Pulse la pestaña Servidor.
- 3** Seleccione la opción Servidor único para todos los servicios del proyecto y, a continuación, Borland Enterprise Server, AppServer Edition 5.0.2 -5,1.x de la lista desplegable.
- 4** Pulse Aceptar.

Podrá encontrar más detalles acerca de la configuración y selección de un servidor de aplicaciones de destino en el [Capítulo 5, “Configuración del servidor de aplicaciones de destino”](#).

## Creación de módulos EJB

---

Cada bean enterprise debe pertenecer a un módulo EJB. Si desea crear un nodo de módulos EJB y abrir el diseñador de EJB, siga los pasos siguientes:

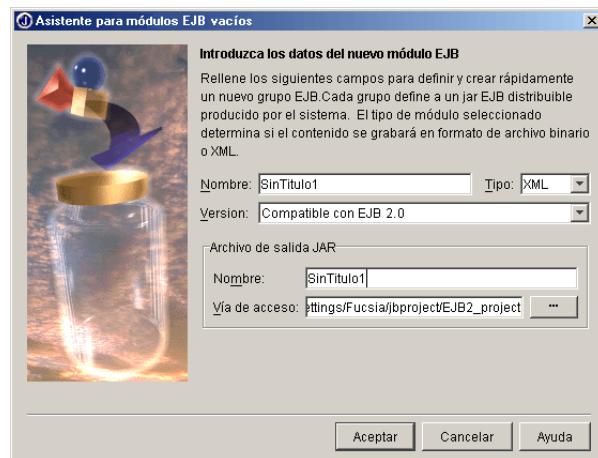
- 1** Seleccione Archivo | Nuevo; se abre la galería de objetos. Pulse la pestaña Enterprise.

- 2** Haga doble clic en el ícono Diseñador de EJB 2.0. Aparece el asistente Diseñador de beans EJB 2.0.



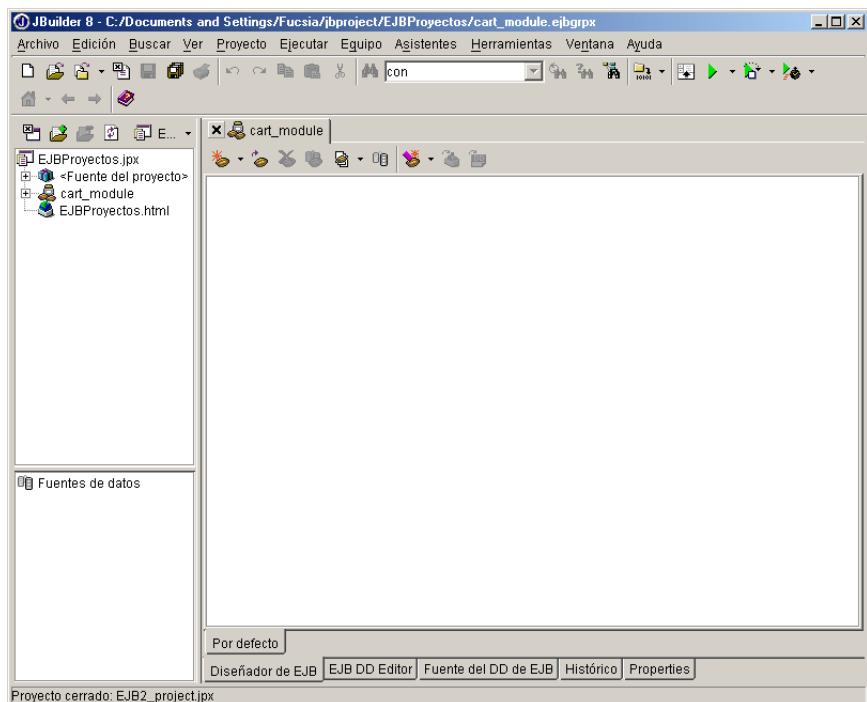
No aparece en la lista ningún módulo EJB 2.0 compatible, ya que todavía no se han creado.

- 3** Pulse en el botón Nuevo para que se abra el Asistente para módulos EJB vacíos.



- 4** En el campo Nombre, escriba `cart_module`. Puede comprobar que, cuando escribe el nombre del módulo EJB, el campo Nombre del cuadro Archivo de salida JAR se rellena automáticamente con `cart_module.jar`.
- 5** Acepte los demás valores por defecto y pulse Aceptar.

- 6** Pulse de nuevo Aceptar para que vuelva a aparecer el diseñador de EJB:



Ahora aparece en el panel de proyecto un nodo de módulos EJB con el título `cart_module`. Si pulsa el icono del extremo izquierdo del nodo para abrirlo, se ve que ya contiene otros nodos, aunque todavía no contengan nada.

## Creación del bean

---

Para crear un bean sesión de carro de la compra, pulse con el botón derecho del ratón en el panel Diseñador de EJB y seleccione en el menú Crear EJB | Bean sesión. También puede pulsar el ícono Crear EJB de la barra de herramientas del diseñador de EJB y, a continuación, seleccionar Bean sesión.

En el diseñador de EJB aparece una representación del bean con el nombre `Enterprise1`, y se abre un inspector de beans.

## Configuración de las propiedades del bean

---

El inspector de beans se utiliza para asignar valores de la propiedad al bean. (Si se cierra el inspector, se puede abrir de nuevo pulsando el nombre del bean.)

Asigne los valores de la propiedad que afecten a todo el bean:

- 1 Escriba `Cart` en el campo Nombre del bean.
- 2 Seleccione Remota en la lista desplegable Interfaces. (Probablemente se selecciona por defecto.) Esta opción determina qué interfaces va a crear el diseñador de EJB: remota, local o ambas.
- 3 Seleccione Con estado de la lista desplegable Tipo de sesión. El objetivo que se persigue es diseñar el bean para que el comprador pueda, si lo desea, detener una sesión de compra, volver más tarde a la tienda virtual y seguir teniendo acceso a los mismos artículos de su carro de la compra. Ya que el bean debe poder mantener el estado entre las sesiones, debe declararse como bean con estado. (El bean que se desarrolla aquí no tiene esta característica.)
- 4 No modifique los valores de los demás campos.

Ahora, en el panel de proyecto, aparece un nodo de paquetes con el nombre `cart_session`. Abra ese nodo para ver los archivos que contiene. Entre esos archivos se encuentran estos tres:

- `Cart.java`: La interfaz remota del bean
- `CartBean.java`: La clase del bean
- `CartHome.java` : La interfaz base del bean

Si pulsa dos veces en los nombres de los archivos, podrá ver el código fuente y lo que el diseñador de EJB ha generado hasta el momento. A medida que se trabaja en el diseñador de EJB, se generan líneas adicionales de código. Si desea volver al diseñador, pulse dos veces el nodo `cart_module` del panel de proyecto, o bien, pulse sobre la pestaña `cart_module` de la parte superior del panel de contenido.

## Adición de campos al bean Cart

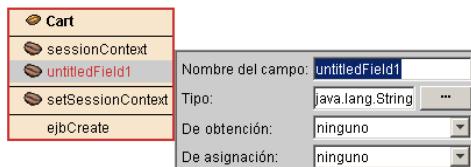
---

El bean de carro de la compra debe asociarse a un usuario en concreto. Puede contener el nombre del usuario, su número de tarjeta de crédito y la fecha de caducidad de esa tarjeta. También debe tener un modo de conservar la lista de artículos que desea comprar ese usuario. Siga los pasos siguientes para agregar al bean los campos necesarios:

- 1 Pulse con el botón derecho del ratón la representación del bean `Cart` del diseñador de EJB para que aparezca el menú contextual.

**2 Selecione Añadir | Campo.**

Aparece un nuevo campo en la representación del bean y se abre un inspector asociado.



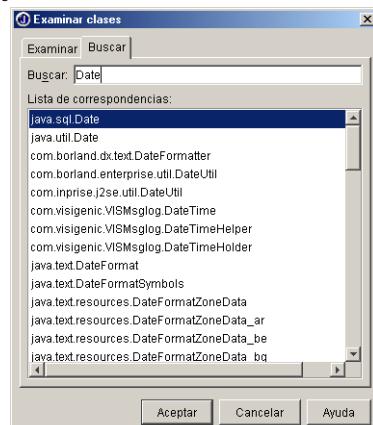
**3 En el inspector, escriba \_cardHolderName en Nombre del campo.**

**4 Asigne a Tipo el valor por defecto java.lang.String, y De obtención y De asignación el valor Ninguno.**

En cualquier momento, puede modificar el valor de la propiedad de un campo pulsando sobre la representación del bean.

Añada los tres campos restantes al bean Cart:

- 1 Pulse sobre el bean con el botón derecho del ratón y seleccione Añadir | Campo.**
- 2 En el inspector, escriba \_creditCardNumber en Nombre del campo.**
- 3 Asigne a Tipo el valor por defecto java.lang.String, y De obtención y De asignación el valor Ninguno.**
- 4 Pulse sobre el bean con el botón derecho del ratón y seleccione Añadir | Campo.**
- 5 En el inspector, escriba \_expirationDate en Nombre del campo.**
- 6 Pulse en el botón de puntos suspensivos situado junto al campo Tipo para que se abra el cuadro de diálogo Examinar clases. Pulse sobre la pestaña Buscar y escriba java.util.Date en el campo Buscar. De este modo, podrá ver la lista de todas las posibles coincidencias. Seleccione java.util.Date.**



En el campo Lista de correspondencias, seleccione `java.util.Date` y pulse Aceptar. También puede escribir `java.util.Date` en el campo Tipo del inspector.

- 7 Pulse sobre el bean con el botón derecho del ratón y seleccione Añadir | Campo.
- 8 En el inspector, escriba `_items` en Nombre del campo.
- 9 Pulse el botón de puntos suspensivos situado junto al campo Tipo y vuelva a utilizar el cuadro de diálogo Examinar clases para especificar la clase `java.util.List`, o bien, escriba `java.util.List` en el campo Tipo del inspector.

Puede ver los nuevos campos en el código fuente del bean. Para ir rápidamente desde el diseñador de EJB al código fuente del bean, pulse con el botón derecho del ratón en la representación del bean que aparece en el diseñador de EJB y seleccione Ver código fuente del bean. Hasta aquí, la clase del bean debe tener la siguiente apariencia:

```
package cart_session;

import javax.ejb.*;

public class CartBean implements SessionBean {
    SessionContext sessionContext;
    java.lang.String _cardHolderName;
    java.lang.String _creditCardNumber;
    java.util.Date _expirationDate;
    java.util.List _items;
    public void ejbCreate() throws CreateException {
        /**@todo: complete this method*/
    }
    public void ejbRemove() {
        /**@todo: complete this method*/
    }
    public void ejbActivate() {
        /**@todo: complete this method*/
    }
    public void ejbPassivate() {
        /**@todo: complete this method*/
    }
    public void setSessionContext(SessionContext sessionContext) {
        this.sessionContext = sessionContext;
    }
}
```

La clase del bean sesión debe definirse como pública. No se puede definir como final o abstracta. La clase del bean debe implementar la interfaz `SessionBean`. La clase del bean sesión generado cumple todos estos requisitos.

El código creado también incluye cuatro métodos definidos por la interfaz `SessionBean`. El contenedor EJB llama a estos métodos de la instancia de

bean en momentos específicos del ciclo de vida del bean sesión. El bean debe incluir estos métodos, aunque el cuerpo de los método pueda estar vacío:

```
public void ejbRemove() {}
public void ejbActivate() {}
public void ejbPassivate() {}
public void setSessionContext(SessionContext context) {}
```

Como puede comprobar, el diseñador de EJB ha añadido métodos requeridos a la clase del bean de forma correcta. En el método `setSessionContext()`, el diseñador de EJB asigna el valor del parámetro `context` a la variable de instancia `sessionContext`.

El contenedor llama al método `setSessionContext()` para asociar la instancia del bean a su contexto. Se puede hacer que el bean retenga esta referencia de contexto de sesión como parte de su estado de comunicación, pero no es necesario. El bean sesión puede utilizar el contexto de sesión para obtener información sobre sí mismo, como las variables de entorno y la interfaz base.

El contenedor llama al método `ejbPassivate()` de la instancia del bean cuando necesita desactivarla. Además, escribe el estado actual del bean en el almacén secundario. Más adelante, cuando activa el bean, restablece este estado. Dado que el contenedor llama al método `ejbPassivate()` antes de desactivar la instancia del bean, el proveedor de beans puede añadir a este método el código necesario para guardar en caché las variables especiales deseadas. De igual modo, el contenedor llama al método `ejbActivate()` de la instancia del bean antes de activarla. Cuando activa el bean restablece todos los valores de estado persistentes. Si lo desea, puede añadir código al método `ejbActivate()`. `CartBean` deja estas implementaciones en blanco.

Aunque no es necesario que un bean sesión implemente un constructor, debe implementar por lo menos un método `ejbCreate()`, que hace de constructor para crear una instancia de un bean. Los bean sesión con estado pueden implementar más de un método `ejbCreate()`. Muchos de los métodos `ejbCreate()` difieren sólo en los parámetros. Más adelante se pueden añadir parámetros al método `ejbCreate()` de `CartBean`.

El contenedor llama al método `ejbRemove()` antes de eliminar la instancia del bean. Se puede añadir código específico de la aplicación que se ejecute antes de la eliminación del bean, pero no es necesario. En el ejemplo `CartBean`, se deja en blanco el método `ejbRemove()`.

## Adición de métodos empresariales al bean Cart

El bean carro de la compra debe contener métodos que añadan o eliminen artículos del carro mientras el cliente navega por la tienda virtual. Cuando el cliente está listo para comprar esos artículos, el bean necesita un método que pueda enumerar el contenido del carro y otro que calcule el precio total. También puede necesitar uno o más métodos que terminen la operación de compra.

Cada uno de los métodos empresariales que se añadan al bean debe seguir unas normas:

- Para evitar conflictos con los nombres reservados por la arquitectura EJB, ningún nombre de método puede empezar por el prefijo ejb.
- Los métodos deben declararse como públicos.
- Ningún método se puede declarar como final o estático.
- Los parámetros y la devolución deben ser válidos, de los tipos RMI-IIOP.
- La cláusula **throws** puede incluir la excepción javax.ejb.EJBException, y puede definir excepciones arbitrarias específicas de la aplicación.

### Adición y eliminación de artículos del carro

Fíjese en los métodos que mantienen la lista de artículos que mantiene el bean, los métodos addItem() y removeItem().

Pulse dos veces sobre el nodo del módulo EJB del panel de proyecto para volver al diseñador de EJB, y siga los siguientes pasos para añadir al bean un método addItem():

- 1 Pulse con el botón derecho sobre la representación del bean Cart que aparece en el diseñador de EJB y seleccione Añadir | Método.
- 2 En el inspector de métodos que aparece, escriba addItem en Nombre del método. (Si el inspector de métodos no está presente cuando desee utilizarlo para modificar algún método, pulse ese método en la representación del bean. Esto puede ser necesario si abre el panel Fuente para ver el código fuente, y luego vuelve al diseñador de EJB.)
- 3 No modifique el valor por defecto void asignado a Tipo devuelto, pero asigne el valor Item item a Parámetros de entrada. Más adelante se crea una clase Item que guarda los datos que se pasan al método.
- 4 En la lista desplegable Interfaces, especifique Remota. Si selecciona Remota, el método se declara en la interfaz remota del bean.

Para añadir al bean un método `removeItem()`:

- 1 Pulse con el botón derecho del ratón sobre la representación del bean `Cart` y seleccione Añadir | Método.
- 2 En el inspector de métodos, escriba `removeItem` en Nombre del método.
- 3 No modifique el valor por defecto `void` asignado a Tipo devuelto, pero asigne el valor `Item item` a Parámetros de entrada.
- 4 En la lista desplegable Interfaces, especifique Remota.

### **Recuperación de los artículos del bean y su coste**

Añada un método al bean `Cart` que indique los artículos que tiene el usuario en el carro:

- 1 Pulse con el botón derecho del ratón sobre la representación del bean `Cart` y seleccione Añadir | Método.
- 2 En el inspector de métodos, escriba `getContents` en el Nombre del método.
- 3 Defina el Tipo devuelto como `java.util.List` y deje en blanco el campo Parámetros de entrada.
- 4 En la lista desplegable Interfaces, especifique Remota.

Añada un método al bean `Cart` que especifique el precio total de los artículos del carro:

- 1 Pulse con el botón derecho del ratón sobre la representación del bean `Cart` y seleccione Añadir | Método.
- 2 En el inspector de métodos, escriba `getTotalPrice` en Nombre del método.
- 3 Defina Tipo devuelto como `float`, y deje vacío el campo Parámetros de entrada.
- 4 En la lista desplegable Interfaces, especifique Remota.

### **Adición de un método `purchase()`**

Añada un último método que se utilice cuando el cliente decida comprar los artículos del carro:

- 1 Pulse con el botón derecho del ratón sobre la representación del bean `Cart` y seleccione Añadir | Método.
- 2 En el inspector de métodos, escriba `purchase` en Nombre del método.
- 3 No modifique Tipo devuelto ni Parámetros de entrada.
- 4 En la lista desplegable Interfaces, especifique Remota.

## El código fuente

Pulse con el botón derecho del ratón sobre la representación del bean Cart y, a continuación, seleccione Ver código fuente del bean. El código para CartBean debe tener la siguiente apariencia:

```
package cart_session;

import javax.ejb.*;

public class CartBean implements SessionBean {
    SessionContext sessionContext;
    java.lang.String _cardHolderName;
    java.lang.String _creditCardNumber;
    java.util.Date _expirationDate;
    java.util.List _items;
    public void ejbCreate() throws CreateException {
        /**@todo: complete this method*/
    }
    public void ejbRemove() {
        /**@todo: complete this method*/
    }
    public void ejbActivate() {
        /**@todo: complete this method*/
    }
    public void ejbPassivate() {
        /**@todo: complete this method*/
    }
    public void setSessionContext(SessionContext sessionContext) {
        this.sessionContext = sessionContext;
    }
    public void addItem(Item item) {
        /**@todo: complete this method*/
    }
    public void removeItem(Item item) {
        /**@todo: complete this method*/
    }
    public java.util.List getContents() {
        /**@todo: complete this method*/
        return null;
    }
    public float getTotalPrice() {
        /**@todo: complete this method*/
    }
    public void purchase() {
        /**@todo: complete this method*/
    }
}
```

El diseñador de EJB añade comentarios al cuerpo de los métodos que aún le quedan por implementar.

En la carpeta Errores del panel de estructura aparecen dos mensajes de error. Ambos indican que no existe una clase Item en el CartBean. Pulse en

un mensaje de error del panel de estructura y la línea de código que tiene el problema aparece resaltada. El problema de la clase `Item` que falta se resolverá más adelante.

Mientras que el diseñador de EJB creaba el código que aparece en la clase `CartBean.java`, también declaraba los métodos que se han añadido a la interfaz remota del bean, `Cart.java`. Si desea ver estas declaraciones de métodos, pulse dos veces `Cart.java` en el paquete `cart_session` del panel de proyecto. Debería tener un aspecto parecido a éste:

```
package cart_session;

import javax.ejb.*;
import java.util.*;
import java.rmi.*;

public interface Cart extends javax.ejb.EJBObject {
    public void addItem(Item item) throws RemoteException;
    public void removeItem(Item item) throws RemoteException;
    public java.util.List getContents() throws RemoteException;
    public float getTotalPrice() throws RemoteException;
    public void purchase() throws RemoteException;
}
```

El diseñador de EJB también ha creado el código de la interfaz `CartHome.java`. `CartHome` contiene un solo método `create()`, al que llama cuando se necesita una nueva instancia del bean `Cart`.

```
package cart_session;

import javax.ejb.*;
import java.util.*;
import java.rmi.*;

public interface CartHome extends javax.ejb.EJBHome {
    public Cart create() throws CreateException, RemoteException;
}
```

## Inicialización de la lista de artículos

---

En este ejemplo, la lista de la compra está limitada a diez artículos. Por lo tanto, deberá inicializar la lista `_items` extendiendo la siguiente declaración de la variable `_items`:

```
java.util.List _items;
de modo que presente el siguiente aspecto:
```

```
java.util.List _items = new ArrayList(10);
```

## Adición de sentencias de importación

---

Los campos `_items` y `_expirationDate` están definidos como clases en el paquete `java.util.*`. Añada las siguientes líneas de código en la parte superior de la clase del bean que importa estas clases, inmediatamente después de la sentencia del paquete `import javax.ejb.*` que ha creado el diseñador de EJB:

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
```

## Implementación de ejbCreate()

---

Hasta aquí, el bean `Cart` realmente no hace nada. Hay que rellenar el cuerpo vacío de los métodos para proporcionarles la lógica. En primer lugar, rellene el método `ejbCreate()`. Tiene que pasar los parámetros nombre del cliente, número de su tarjeta de crédito y fecha de caducidad de esa tarjeta al método `ejbCreate()` del bean.

Para añadir los parámetros al método `ejbCreate()`:

- 1 Pulse en `ejbCreate` en la parte inferior de la representación del bean que aparece en el diseñador de EJB.
- 2 Escriba la siguiente línea en el campo Parámetros de entrada, que declara los tres parámetros:

```
String cardHolderName, String creditCardNumber, Date expirationDate
```

- 3 Vuelva al código fuente de la clase del bean y añada el código que representa la lista de los artículos de la compra y que asigna los valores de los parámetros a los campos del bean dentro del método `ejbCreate()`. Cuando haya terminado, el método `ejbCreate()` debe tener la siguiente apariencia:

```
public void ejbCreate(String cardHolderName, String creditCardNumber,
                      Date expirationDate) throws CreateException {
    _cardHolderName = cardHolderName;
    _creditCardNumber = creditCardNumber;
    _expirationDate = expirationDate;
}
```

Cada método `ejbCreate()` en la clase del bean tiene su correspondiente método `create()` en la interfaz base del bean. Como se han añadido los tres parámetros al método `ejbCreate()`, el diseñador de EJB añade los mismos parámetros al método `create()` en `CartHome.java`. Pulse dos veces sobre `CartHome.java` del paquete `cart_session`, o bien, pulse la etiqueta `CartHome.java` de la parte superior del panel de contenido para ver el código que se ha añadido:

```
public interface CartHome extends javax.ejb.EJBHome {
    public Cart create(String cardHolderName, String creditCardNumber,
        Date expirationDate)
        throws CreateException, RemoteException;
}
```

Por supuesto, puede añadir los parámetros al método `ejbCreate()` directamente en el código fuente, pero debe recordar añadirlos también en la interfaz base. Si utiliza el diseñador de EJB para añadir los parámetros, solamente tendrá que introducir la información una vez.

## Implementación de addItem() y removeItem()

---

Una vez que se haya ocupado de `ejbCreate()` y de su correspondiente método `create()` en la interfaz base del bean, vuelva al código fuente de la clase `CartBean` para implementar los métodos que haya agregado al bean. Vaya al método `addItem()` del código fuente y rellénelo para que tenga la siguiente apariencia:

```
public void addItem(Item item) {
    System.out.println("\taddItem(" + item.getTitle() + "): " + this);
    _items.add(item);
}
```

El método `addItem()` añade el artículo especificado a la lista del carro e imprime el título del artículo que se ha agregado en la consola.

Desplácese al método `removeItem()` del código fuente y añada este código para que tenga la siguiente apariencia:

```
public void removeItem(Item item) {
    System.out.println("\tremoveItem(" + item.getTitle() + "): " + this);
    if (! _items.remove(item)) {
        throw new EJBException("The item " + item.getTitle() +
            " is not in your cart.");
    }
}
```

El método `removeItem()` elimina el artículo señalado de la lista del carro, e imprime el título del artículo eliminado. Si se intenta eliminar un artículo que no está en el carro, se lanza una excepción que imprime un mensaje de error.

## Creación de una clase Item

---

Aunque el bean `Cart` ya cuenta con métodos para añadir o eliminar artículos, no se ha definido qué es exactamente un artículo. Necesita una clase `Item`.

- 1** Seleccione Archivo | Nueva clase para iniciar el Asistente para clases.
- 2** Escriba `Item` en Nombre de clase.

- 3** Seleccione sólo las opciones Pública y Generar constructor por defecto, desmarcando todas las demás.
- 4** Pulse Aceptar.

Se añade una clase `Item` al paquete `cart_session`. Modifique la clase resultante para que quede así:

```
package cart_session;

import java.io.Serializable;

public class Item implements Serializable {
    private static final long serialVersionUID = -560245896319031239L;
    private String _title;
    private float _price;
    private String _type;

    public Item(String title, float price, String type) {
        _title = title;
        _price = price;
        _type = type;
    }

    public String getTitle() {
        return _title;
    }

    public float getPrice() {
        return _price;
    }

    public String getName() {
        return _type;
    }

    public final boolean equals(Object o) {
        // dos elementos son iguales si tienen la misma clase y título
        if (!(o instanceof Item)) {
            return false;
        }
        Item i = (Item) o;
        return (getClass() == i.getClass()) &&
            (_title == null ? i._title == null : _title.equals(i._title));
    }
}
```

La clase `Item` contiene tres campos con la información relevante acerca del artículo, `_title`, `_price` y `_type`; y tres métodos accesores para recuperar información de los campos, `getTitle()`, `getPrice()` y `getType()`. También contiene un método `equals()` para determinar si la instancia de un artículo es igual a otra.

Tenga en cuenta que la clase `Item` implementa la interfaz `Serializable`, haciendo así posible escribir y leer un artículo en un flujo. Si desea más

información acerca de la serialización, consulte “Serialización” en Procedimientos iniciales con Java y también puede dirigirse a <http://java.sun.com/j2se/1.3/docs/guide/serialization/>. Y además, en la sede web de Sun encontrará más fuentes de información acerca de la serialización de objetos.

## Implementación de los métodos restantes

---

Diríjase de nuevo al código fuente de la clase del bean. Puede comprobar que el panel de estructura ya no indica la existencia de ningún error, porque el paquete tiene ahora una clase Item.

Una vez implementados los métodos para mantener la lista de artículos en una instancia Cart, añada el código necesario para implementar los demás métodos que haya definido. A continuación se muestra el método getContents() que recupera los artículos de la lista:

```
public java.util.List getContents() {
    System.out.println("\tgetContents(): " + this);
    return _items;
}
```

Este es el método getTotalPrice(), que calcula el precio total de todos los artículos del carro:

```
public float getTotalPrice() {
    System.out.println("\tgetTotalPrice(): " + this);
    float totalPrice = 0f;
    for (int i = 0, n = _items.size(); i < n; i++) {
        Item current = (Item) _items.get(i);
        totalPrice += current.getPrice();
    }
    return (long) (totalPrice * 100) / 100f;
}
```

Por último, se muestra el método purchase(), que en este ejemplo no hace más que comprobar la fecha de caducidad de la tarjeta de crédito del cliente:

```
public void purchase() throws EJBException {
    System.out.println("\tpurchase(): " + this);
    Date today = new Date();
    if(_expirationDate.before(today)) {
        throw new EJBException("Expiration date: " + _expirationDate);
    }
    System.out.println("\tPurchasing not implemented yet!");
}
```

Guarde el proyecto. Seleccione Archivo | Guardar proyecto "cart\_sessionbean.jpx".

# Utilización de los descriptores de distribución del bean

---

Mientras se crea el bean `Cart` en JBuilder, se crea un descriptor de distribución. Si desea más información sobre los descriptores de distribución, consulte el [Capítulo 12, “Distribución de enterprise beans”](#). El descriptor de distribución de este bean es muy sencillo.

Para ver el código fuente del descriptor de distribución del bean `Cart`:

- 1 Pulse la pestaña `cart_module` de la parte superior del panel de contenido, o bien, pulse dos veces sobre el nodo `cart_module` del panel de proyecto para volver al diseñador de EJB.
- 2 Pulse la pestaña Fuente del DD de EJB.
- 3 Pulse la pestaña `ejb-jar.xml` si no está ya seleccionada.

El archivo `ejb-jar.xml` es un archivo XML. Debería tener un aspecto parecido a éste:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
    2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
    <enterprise-beans>
        <session>
            <display-name>Cart</display-name>
            <ejb-name>Cart</ejb-name>
            <home>cart_session.CartHome</home>
            <remote>cart_session.Cart</remote>
            <ejb-class>cart_session.CartBean</ejb-class>
            <session-type>Stateful</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>Cart</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
    </assembly-descriptor>
</ejb-jar>
```

Si lo desea, puede modificar los campos directamente en el documento XML. También puede utilizar el Editor de descriptor de distribución con el fin de realizar cambios y adiciones a los descriptores de distribución generados.

Para modificar el descriptor de distribución `Cart` mediante el Editor de descriptor de distribución:

- 1 Pulse el icono que está a la izquierda del nodo `cart_module` del panel de proyecto para ver los contenidos del módulo.
- 2 Haga clic dos veces sobre el nodo `Cart` del módulo EJB para que se abra el Editor de descriptor de distribución.

Como podrá comprobar, en el Editor de descriptor de distribución hay numerosos paneles disponibles para el bean `Cart`. Si desea información detallada sobre el editor del descriptor de distribución, consulte el [Capítulo 13, “El editor de descriptor de distribución”](#).

## Compilación del proyecto

---

Si desea compilar su proyecto, seleccione Proyecto | Ejecutar Make del proyecto “`cart_session.jpx`”. Todos los archivos del proyecto se compilan, y se crea un archivo `cart_module.jar`. Si aparece algún error de compilación, soluciónelo y vuelva a compilar el proyecto.

Si desea más información acerca de la compilación de beans enterprise, consulte el [Capítulo 10, “Compilación de enterprise beans y creación de archivos JAR”](#).

## Ejecución del bean Cart

---

Para iniciar un contenedor local con rapidez y distribuir el archivo `cart_module.jar` a dicho contenedor que incluye el bean `Cart`:

- 1 Seleccione Herramientas | Agente de gestión de Borland Enterprise Server.
- 2 Haga clic con el botón derecho del ratón sobre el nodo `cart_module` del panel de proyecto y seleccione “Ejecutar utilizando la configuración por defecto” del menú contextual.

En el panel de mensajes de JBuilder aparecen varias líneas que documentan el proceso de inicio del contenedor EJB. También puede ver un mensaje que indica que el archivo `cart_module.jar` se ha distribuido, y que el contenedor está listo. A continuación se muestran algunas estadísticas del contenedor EJB. Si lo comprueba con cuidado, verá que no hay ninguna instancia `Cart` en la memoria. Para hacerlo, necesita un programa cliente que llame al método `create()` de la interfaz base, `CartHome`. Consulte el [Capítulo 22, “Tutorial: Creación de aplicaciones cliente de prueba”](#), en el que se explica cómo crear una aplicación cliente Java de prueba que llame a los métodos de `Cart` y que pruebe su lógica.

## Código para el bean sesión de carro de la compra

---

```
package cart_session;

import javax.ejb.*;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class CartBean implements SessionBean {
    SessionContext sessionContext;
    java.lang.String _cardHolderName;
    java.lang.String _creditCardNumber;
    java.util.Date _expirationDate;
    java.util.List _items = new ArrayList(10);

    public void ejbCreate(String cardHolderName, String creditCardNumber,
        Date expirationDate) throws CreateException {
        _cardHolderName = cardHolderName;
        _creditCardNumber = creditCardNumber;
        _expirationDate = expirationDate;
    }

    public void ejbRemove() {
        /**@todo Complete this method*/
    }
    public void ejbActivate() {
        /**@todo Complete this method*/
    }
    public void ejbPassivate() {
        /**@todo Complete this method*/
    }
    public void setSessionContext(SessionContext sessionContext) {
        this.sessionContext = sessionContext;
    }
    public void addItem(Item item) {
        System.out.println("\taddItem(" + item.getTitle() + "): " + this);
        _items.add(item);
    }
    public void removeItem(Item item) {
        System.out.println("\tremoveItem(" + item.getTitle() + "): " + this);
        if (! _items.remove(item)) {
            throw new EJBException("The item " + item.getTitle() +
                " is not in your cart.");
        }
    }
    public java.util.List getContents() {
        System.out.println("\tgetContents(): " + this);
        return _items;
    }
    public float getTotalPrice() {
        System.out.println("\tgetTotalPrice(): " + this);
        float totalPrice = 0f;
```

## Código para el bean sesión de carro de la compra

```
for (int i = 0, n = _items.size(); i < n; i++) {
    Item current = (Item) _items.get(i);
    totalPrice += current.getPrice();
}
return ((long) (totalPrice * 100))/100f;;
}
public void purchase() throws EJBException {
    System.out.println("\tpurchase(): " + this);
    Date today = new Date();
    if(_expirationDate.before(today)) {
        throw new EJBException("Expiration date: " + _expirationDate);
    }
    System.out.println("\tPurchasing not implemented yet!");
}
}
```



# Tutorial: Creación de aplicaciones cliente de prueba

La Edición WebLogic de JBuilder sólo es compatible con servidores WebLogic.

En este tutorial se crea una aplicación cliente que realiza llamadas de prueba a los métodos del bean sesión Cart que se ha creado en el Capítulo 21, "Tutorial: Desarrollo de beans sesión con el diseñador de EJB". Se pretende mostrar cómo se crea una aplicación cliente para probar un bean enterprise que ya esté creado.

Si desea ver el código creado por este tutorial, consulte ["Código para la aplicación cliente de prueba"](#) en la página 22-12.

El apartado Accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder por parte de 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 1-2.

## Apertura del proyecto cart\_session

Abra el proyecto cart\_session que se creó en el tutorial Cart:

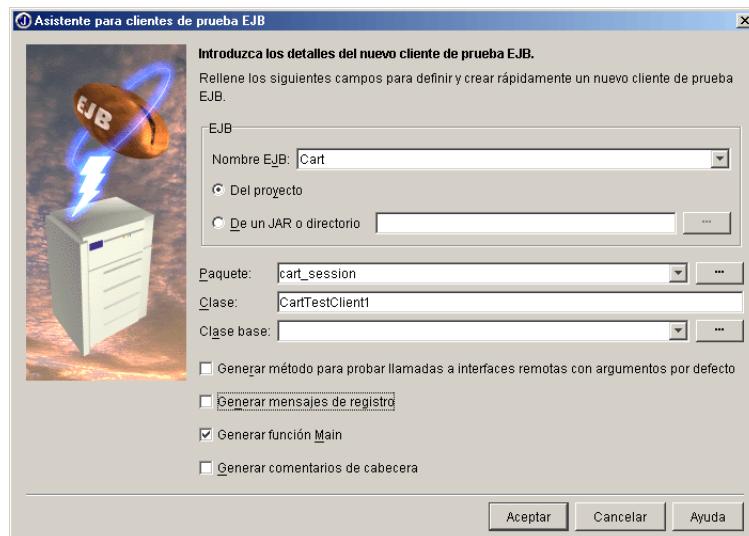
- 1 Seleccione Archivo | Abrir proyecto para presentar el cuadro de diálogo homónimo.
- 2 Desplácese al directorio cart\_session, que se encuentra en el directorio jbproject, y púlselo dos veces.
- 3 En el panel derecho del cuadro de diálogo, pulse dos veces cart\_session.jpx, o bien, seleccione cart\_session.jpx y a continuación, Aceptar.

## Utilización del Asistente para clientes de prueba EJB

El Asistente para cliente de prueba EJB puede iniciar una aplicación cliente. Para más información acerca del uso del Asistente para cliente de prueba EJB, consulte “[Creación de aplicaciones cliente de prueba](#)” en la [página 11-4](#).

Si para iniciar una aplicación desea utilizar el Asistente para cliente de prueba EJB:

- 1 Seleccione Archivo | Nuevo; se abre la galería de objetos. Pulse la pestaña Enterprise.
- 2 Pulse dos veces en el ícono Cliente de prueba EJB.



- 3 En la lista desplegable Nombre EJB, seleccione Cart. (Como Cart es el único enterprise bean de este proyecto, ya está seleccionado.)
- 4 Desactive la opción Generar mensajes de registro.
- 5 Pulse Aceptar.

El Asistente para clientes de prueba EJB genera el siguiente código:

```
package cart_session;

import javax.naming.*;
import javax.rmi.PortableRemoteObject;

public class CartTestClient1 extends Object {
    private CartHome cartHome = null;

    //Construir el cliente de prueba EJB
    public CartTestClient1() {
```

```

        initialize();
    }

    public void initialize() {
        try {
            //obtener contexto de denominación
            Context context = new InitialContext();

            //buscar nombre jndi
            Object ref = context.lookup("Cart");
            //buscar nombre jndi y convertir al tipo de la interfaz base
            cartHome = (CartHome) PortableRemoteObject.narrow(ref, CartHome.class);
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    //-----
    -
    // Métodos de utilidad
    //-----
    -

    public CartHome getHome() {
        return cartHome;
    }
    //método Main

    public static void main(String[] args) {
        CartTestClient1 client = new CartTestClient1();
        // Utilice el método getHome() del objeto cliente para llamar a la interfaz
        base
        // métodos que devuelven una referencia de interfaz remota. Después:
        // utilice esa referencia de interfaz remota para acceder a EJB.
    }
}

```

## Examinar el código generado

---

El constructor de clientes de prueba que genera el Asistente para clientes de prueba EJB lleva a cabo las siguientes tareas:

- 1 Obtiene una referencia al servidor JNDI. Lo hace obteniendo una referencia a un objeto `InitialContext`, que sirve como raíz del árbol JNDI.
- 2 Utiliza el objeto `InitialContext` obtenido para llamar al método `lookup()` y conseguir una referencia al objeto base `Cart`.
- 3 La referencia que se obtiene, `Object`, se encaja en la interfaz base del bean, y se reduce mediante el método `narrow()` de `PortableRemoteObject`. (No es necesario utilizar `PortableRemoteObject.narrow()` si el cliente es

local, lo que significa que el bean tiene una interfaz base local en lugar de una interfaz base remota.)

Las tres sentencias que realizan estas tareas aparecen en un bloque try/catch para capturar todas las excepciones lanzadas si falla cualquiera de estos pasos.

El método `main()` crea una instancia cliente de prueba con el nombre `client`. Las demás líneas de `main()` son comentarios que sugieren el modo de actuar para crear o buscar una instancia del bean e invocar a sus métodos. Los comentarios de `getHome()` se refieren al método declarado inmediatamente antes del método `main()`.

## Adición de código al cliente de prueba

---

Siga la sugerencia de los comentarios y desarrolle el método `main()` llamando al método `getHome()`, para obtener así una referencia a la interfaz base del bean `Cart`. Coloque esta línea de código inmediatamente después de la primera sentencia del método `main()`:

```
CartHome home = client.getHome();
```

Ahora puede llamar al método `create()` de la interfaz base para crear una instancia del bean. Antes de realizar esta acción, declare los valores que deseé que tenga la instancia del bean en el método `main()`:

```
String cardHolderName = "Suzy Programmer";
String creditCardNumber = "1234-5678-9012-3456";
Date expirationDate = new GregorianCalendar(2004, Calendar.JULY,
30).getTime();
```

Como `Date`, `GregorianCalendar` y `Calendar` están en el paquete `java.util`, añada esta sentencia de importación a las sentencias de importación en la parte superior del archivo:

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

## Creación de una instancia del bean Cart

---

Es muy recomendable que coloque el método `create()` dentro de un bloque try/catch. En primer lugar, declare la variable `cart` fuera de un bloque try/catch en el método `main()`, asigne a la variable `cart` el valor null y, a continuación, escriba el bloque try/catch, que intenta crear la instancia del bean y que imprime un mensaje si ésta falla:

```
Cart cart = null;

try {
    cart = home.create(cardHolderName, creditCardNumber, expirationDate);
} catch (Exception e) {
```

```
System.out.println("Could not create Cart session bean\n" + e);
}
```

## Adición y eliminación de artículos del carro

---

Una vez que se ha creado la instancia `cart`, se puede llamar a los métodos del bean. En este tutorial se pretende añadir uno o más artículos a la instancia `cart`, eliminar algún artículo, imprimir los artículos del carro y, por último, hacer el cálculo e imprimir el precio total del contenido del carro.

En el caso de que fallase la llamada para crear la instancia `cart`, debe colocar todas las llamadas a los métodos del bean dentro de una sentencia `if`, que comprueba si existe la instancia `cart`:

```
if (cart != null) {
}
```

Para comprobarlo, el método `main()` debe presentar el siguiente aspecto:

```
public static void main(String[] args) {
    CartTestClient1 client = new CartTestClient1();
    CartHome home = client.getHome();

    String cardHolderName = "Suzy Programmer";
    String creditCardNumber = "1234-5678-9012-3456";
    Date expirationDate = new GregorianCalendar(2004, Calendar.JULY, 30).getTime();

    Cart cart = null;

    try {
        cart = home.create(cardHolderName, creditCardNumber, expirationDate);
    } catch (Exception e) {
        System.out.println("Could not create Cart session bean\n" + e);
    }

    if (cart != null) {
    }
}
```

Coloque el código restante que haya escrito dentro de la sentencia `if`. En primer lugar debe crear una instancia `Item` para añadirla al carro:

```
Item kassemBook = new Item("J2EE Blueprints", 39.99f, "Book");
```

A continuación, escriba el bloque `try/catch` que añade el nuevo artículo al carro:

```
try {
    cart.addItem(kassemBook);
} catch (Exception e) {
    System.out.println("Could not add the book to the cart\n" + e);
}
```

Añada al carro un segundo artículo:

```
Item milesAlbum = new Item("Kind of Blue", 11.97f, "CD");

    try {
        cart.addItem(milesAlbum);
    } catch (Exception e) {
        System.out.println("Could not add the CD to the cart\n" + e);
    }
```

Llame a un método `summarize()` que debe ser creado. `summarize()` se encarga de imprimir los contenidos del carro, incluido el precio de cada artículo, y a continuación calcula el precio total de los artículos y los imprime. La llamada al método `summarize()` debe tener la siguiente apariencia:

```
try {
    summarize(cart);
} catch (Exception e) {
    System.out.println("Could not summarize the items in the cart\n" + e);
}
```

A continuación, intente eliminar un artículo del carro. Añada el siguiente código al método `main()`:

```
try {
    cart.removeItem(kassemBook);
} catch (Exception e) {
    System.out.println("Could not remove the book from the cart\n" + e);
}
```

Cree otro artículo, añádalo al carro y llame al método `summarize()` para calcular el número de artículos e volver a imprimir la cantidad total:

```
Item calvertBook = new Item("Charles Calvert's Learn JBuilder 7", 49.95f,
    "Book");

    try {
        cart.addItem(calvertBook);
    } catch (Exception e) {
        System.out.println("Could not add book to the cart\n" + e);
    }

    try {
        summarize(cart);
    } catch (Exception e) {
        System.out.println("Could not summarize the items in the cart\n" + e);
    }
```

## Final de la compra

---

Una vez creados los métodos que prueban la adición y eliminación de artículos del carro, llame al método `purchase()` para terminar la operación. Realice de nuevo la llamada dentro de un bloque `try/catch`:

```
try {
    cart.purchase();
} catch (Exception e) {
    System.out.println("Could not purchase the items:\n\t" + e);
}
```

## Eliminar la instancia del bean

---

No es totalmente necesario eliminar una instancia del bean sesión cuando se ha hecho con ella la aplicación cliente, pero es una práctica aconsejable. Si el cliente no la elimina, es probable que el servidor EJB lo haga una vez transcurrido el periodo de tiempo que se establece en el descriptor de distribución del bean. En el caso de este tutorial, añada un método `remove()` dentro de un bloque `try/catch` para terminar la sentencia `if` y el método `main()`:

```
try {
    cart.remove();
} catch (Exception e) {
    System.out.println("Could not remove the Cart bean\n" + e);
}
```

Todo el método `main()` debe tener el siguiente aspecto:

```
public static void main(String[] args) {
    CartTestClient client = new CartTestClient();
    CartHome home = client.getHome();

    String cardHolderName = "Suzy Programmer";
    String creditCardNumber = "1234-5678-9012-3456";
    Date expirationDate = new GregorianCalendar(2004, Calendar.JULY, 30).getTime();

    Cart cart = null;

    try {
        cart = home.create(cardHolderName, creditCardNumber, expirationDate);
    } catch (Exception e) {
        System.out.println("Could not create Cart session bean\n" + e);
    }

    if (cart != null) {

        Item kassemBook = new Item("J2EE Blueprints", 39.99f, "Book");

        try {
            cart.addItem(kassemBook);
        } catch (Exception e) {
            System.out.println("Could not add the book to the cart\n" + e);
        }
    }
}
```

## Adición de código al cliente de prueba

```
Item milesAlbum = new Item("Kind of Blue", 11.97f, "CD");

    try {
        cart.addItem(milesAlbum);
    } catch (Exception e) {
        System.out.println("Could not add the CD to the cart\n" + e);
    }

    try {
        summarize(cart);
    } catch (Exception e) {
        System.out.println("Could not summarize the items in the cart\n" + e);
    }

    try {
        cart.removeItem(kassemBook);
    } catch (Exception e) {
        System.out.println("Could not remove the book from the cart\n" + e);
    }

Item calvertBook = new Item("Charles Calvert's Learn JBuilder 7", 49.95f,
    "Book");

    try {
        cart.addItem(calvertBook);
    } catch (Exception e) {
        System.out.println("Could not add book to the cart\n" + e);
    }

    try {
        summarize(cart);
    } catch (Exception e) {
        System.out.println("Could not summarize the items in the cart\n" + e);
    }

    try {
        cart.purchase();
    } catch (Exception e) {
        System.out.println("Could not purchase the items:\n\t" + e);
    }

    try {
        cart.remove();
    } catch (Exception e) {
        System.out.println("Could not remove the Cart bean\n" + e);
    }
}
```

## Resumen de los artículos del carro

---

El método `main()` creado llama a un método `summarize()` que debe definir. De hecho, si mira al panel de estructura, puede comprobar que hay dos mensajes de error en la carpeta Errores, uno por cada una de las veces que llame al método `summarize()`, y que indican que `summarize()` no se encuentra en el cliente de prueba. Por lo tanto, debe comenzar la declaración del método `summarize()` fuera del método `main()`:

```
private static void summarize(Cart cart) throws Exception {
}
```

La primera línea que se añade al método imprime el título del informe que va a aparecer en la consola:

```
System.out.println("===== Cart Summary =====");
```

La segunda línea que se añade devuelve todos los artículos del carro:

```
List items = cart.getContents();
```

A continuación se añade un bucle `for` que imprime el precio, título y tipo de artículo para cada uno de los artículos del carro:

```
for (int i = 0; n = items.size(); i < n; i++) {
    Item current = (Item) items.get(i);
    System.out.println("Price: $" + current.getPrice() + "\t" +
        current.getTitle());
}
```

Por último, se añaden las líneas que calculan e imprimen el precio total de todos los artículos del carro, y se imprime la última línea del informe:

```
System.out.println("Total: $" + cart.getTotalPrice());
System.out.println("=====");
```

## Compilación de la aplicación cliente de prueba

---

Si desea compilar su aplicación cliente de prueba, pulse con el botón derecho del ratón en el nodo `CartTestClient1` del panel de proyecto y, a continuación, seleccione Ejecutar Make.

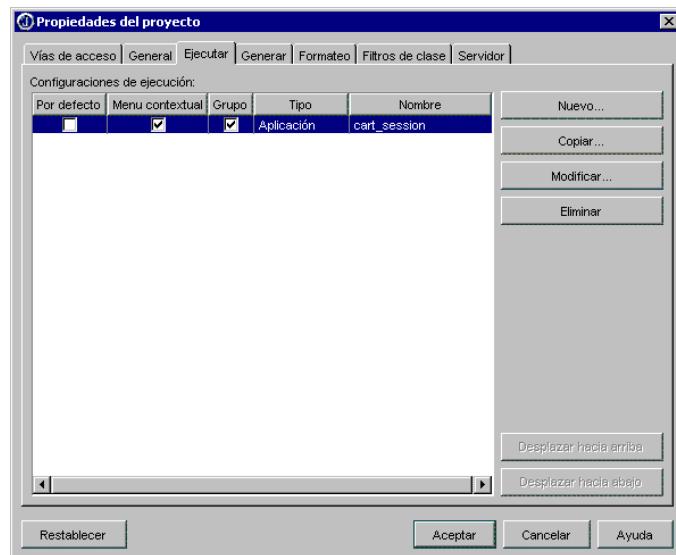
## Ejecución del cliente de prueba

Para ejecutar la aplicación cliente de prueba y probar además su lógica y la del bean sesión Cart, debe iniciar el contenedor EJB y distribuir el bean. Si desea información adicional para poder llevarlo a cabo, consulte “[Ejecución del bean Cart](#)” en la página 21-19 en el “[Tutorial: Desarrollo de beans sesión con el diseñador de EJB](#).”

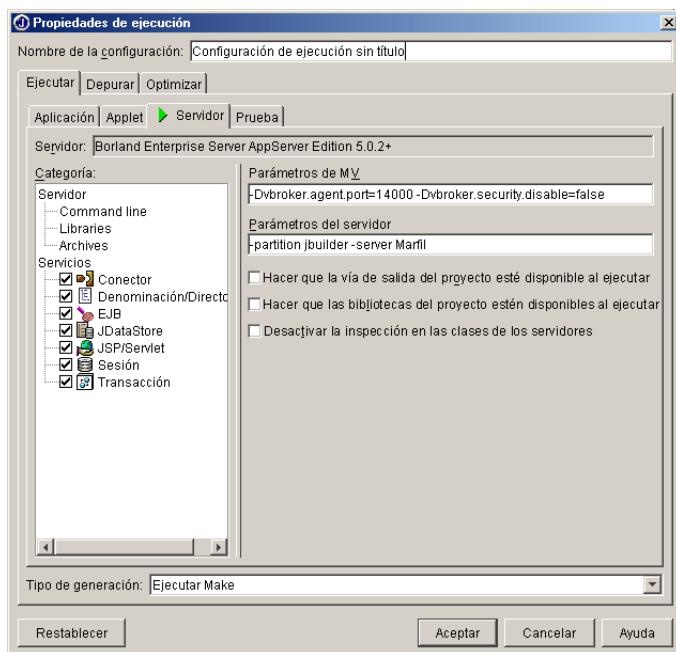
También puede crear configuraciones para la ejecución tanto del contenedor EJB como de la aplicación cliente, e iniciarlos seleccionando la configuración que desee en la flecha de lista desplegable que se encuentra junto al botón Ejecutar de la barra de herramientas.

Para crear una configuración de servidor que inicie el servidor y que distribuya el bean Cart en él:

- 1 Seleccione Ejecutar | Configuraciones y abra la pestaña Ejecutar.



- 2** Pulse el botón Nuevo y, a continuación, seleccione la pestaña Servidor si no está ya seleccionada.



- 3** En el campo Nombre de la configuración, escriba el nombre Server.
- 4** Los valores que se muestran en esta ficha son valores por defecto que se configuraron al seleccionar el servidor Borland Enterprise Server, AppServer Edition 5.0.2 -5.1.x como el servidor de aplicaciones de destino. Si necesita modificar los valores por defecto, puede hacerlo en este momento.
- 5** Elija en la lista Jars EJB el archivo JAR que contiene los beans que desea probar. Si sólo hay uno, ya estará seleccionado. Los archivos JAR enumerados se recuperan de los grupos EJB del proyecto.
- 6** Haga clic en Aceptar para cerrar todos los cuadros de diálogo.

Para crear una configuración de cliente:

- 1 Seleccione Ejecutar | Configuraciones y abra la pestaña Ejecutar.
- 2 Pulse el botón Nuevo y compruebe que se ha seleccionado la pestaña Aplicación.
- 3 En el campo Configuración, escriba Client.
- 4 Pulse el botón de puntos suspensivos junto al campo Clase principal y abra la pestaña Buscar si no está ya seleccionada. Comience escribiendo el nombre del paquete `cart_session` y aparecerá una lista de todos los

archivos de cart\_session. Seleccione CartTestClient1 y pulse Aceptar. (También puede pulsar la pestaña Examinar y desplazarse hasta la clase CartTestClient1.)

- 5 Si su servidor de aplicaciones de destino es el Borland Enterprise Server 5.0.2 -5.1.x, escriba -Dvbroker.agent.port=<port no.> en el campo Parámetros de MV, junto con el número de puerto que utiliza el agente inteligente de Visibroker. (Puede copiar la configuración del servidor que ha creado en el campo Parámetros de MV).
- 6 Haga clic sobre Aceptar dos veces.

Ahora debe iniciar el Agente de gestión de Borland Enterprise Server. Seleccione Herramientas | Agente de gestión de Borland Enterprise Server.

Ahora se puede iniciar el contenedor. Seleccione la configuración de ejecución de servidor de la lista desplegable contigua al botón Ejecutar de la barra de herramientas de JBuilder.



Se inicia el contenedor. Tenga paciencia, ya que este proceso es lento. Puede ver los progresos del proceso de inicio en la ventana de mensajes. Aquí aparecerán todos los errores que se produzcan.

A continuación, elija la configuración de ejecución cliente con el fin de ejecutar la aplicación cliente. El mensaje que aparece en el panel informa del éxito o el fracaso de la ejecución de la aplicación cliente.

## Código para la aplicación cliente de prueba

---

```
package cart_session;

import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;

public class CartTestClient1 extends Object {
    private CartHome cartHome = null;

    //Construir el cliente de prueba EJB
    public CartTestClient1() {
        initialize();
    }

    public void initialize() {
        try {
            //obtener contexto de denominación
            Context context = new InitialContext();

            //buscar nombre jndi
```

## Código para la aplicación cliente de prueba

```
Object ref = context.lookup("Cart");
//buscar nombre jndi y convertir al tipo de la interfaz base
cartHome = (CartHome) PortableRemoteObject.narrow(ref, CartHome.class);
}
catch(Exception e) {
    e.printStackTrace();
}
}

//-----
// Métodos de utilidad
//-----
public CartHome getHome() {
return cartHome;
}
//método Main

public static void main(String[] args) {
CartTestClient1 client = new CartTestClient1();
CartHome home = client.getHome();
String cardHolderName = "Suzy Programmer";
String creditCardNumber = "1234-5678-9012-3456";
Date expirationDate = new GregorianCalendar(2004, Calendar.JULY,
30).getTime();

Cart cart = null;

try {
cart = home.create(cardHolderName, creditCardNumber, expirationDate);
} catch (Exception e) {
System.out.println("Could not create Cart session bean\n" + e);
}

if (cart != null) {

Item kassemBook = new Item("J2EE Blueprints", 39.99f, "Book");
try {
cart.addItem(kassemBook);
} catch (Exception e) {
System.out.println("Could not add the book to the cart\n" + e);
}

Item milesAlbum = new Item("Kind of Blue", 11.97f, "CD");

try {
cart.addItem(milesAlbum);
} catch (Exception e) {
System.out.println("Could not add the CD to the cart\n" + e);
}

try {
summarize(cart);
} catch (Exception e) {
System.out.println("Could not summarize the items in the cart\n" + e);
}
}
```

## Código para la aplicación cliente de prueba

```
try {
    cart.removeItem(kassemBook);
} catch (Exception e) {
    System.out.println("Could not remove book from cart\n" + e);
}

Item calvertBook = new Item("Charles Calvert's Learn JBuilder 6.0",
    49.95f, "Book");

try {
    cart.addItem(calvertBook);
} catch (Exception e) {
    System.out.println("Could not add book to the cart\n" + e);
}

try {
    summarize(cart);
} catch (Exception e) {
    System.out.println("Could not summarize the items in the cart\n" + e);
}

try {
    cart.purchase();
} catch (Exception e) {
    System.out.println("Could not purchase the items: \n" + e);
}

try {
    cart.remove();
} catch (Exception e) {
    System.out.println("could not remove the Cart bean\n" + e);
}
}

private static void summarize(Cart cart) throws Exception {
    System.out.println("===== Cart Summary =====");
    List items = cart.getContents();
    for (int i = 0, n = items.size(); i < n; i++) {
        Item current = (Item) items.get(i);
        System.out.println("Price: $" + current.getPrice() + "\t" +
current.getPrice() + "\t" + current.getTitle());
    }

    System.out.println("Total: $" + cart.getTotalPrice());
    System.out.println("=====");
}
```

P a r t e

II

## Tecnologías compatibles



## Creación de productores y consumidores JMS

Es una característica de JBuilder Enterprise.

JMS (Servicio de mensajes Java), que forma parte de Java 2 Enterprise Edition (J2EE), proporciona las API necesarias para crear aplicaciones que utilizan un sistema de mensajes empresarial. Este sistema de mensajes se puede utilizar para desarrollar aplicaciones distribuidas ampliables, fiables y muy flexibles. Los sistemas de mensajes permiten la comunicación asíncrona de aplicaciones independientes.

Las aplicaciones y las clases que envían mensajes se denominan “productores”; las que reciben mensajes se denominan “consumidores”. JBuilder tiene un asistente JMS que ayuda en la creación de productores y consumidores de mensajes.

Los mensajes JMS pueden adaptarse a cualquiera de estos dos modelos:

- Publicar y suscribir

Los sistemas de mensajes de publicar y suscribir siguen un modelo basado en sucesos según el cual los productores envían o publican mensajes y los consumidores se suscriben a éstos o reciben los que les interesan. Cada mensaje publicado trata de un tema específico. Los consumidores de mensajes especifican los temas que quieren recibir.

- Punto a punto

El sistema de mensajes punto a punto requiere que el productor envíe un mensaje a un consumidor determinado. El mensaje llega a la cola de mensajes de entrada del consumidor.

Si desea información más completa acerca de JMS, consulte la documentación de JMS de Sun en <http://java.sun.com/products/jms/docs.html>. JBuilder Enterprise se suministra con SonicMQ Message Broker,

un proveedor de JMS. Para obtener más información consulte la documentación de SonicMQ.

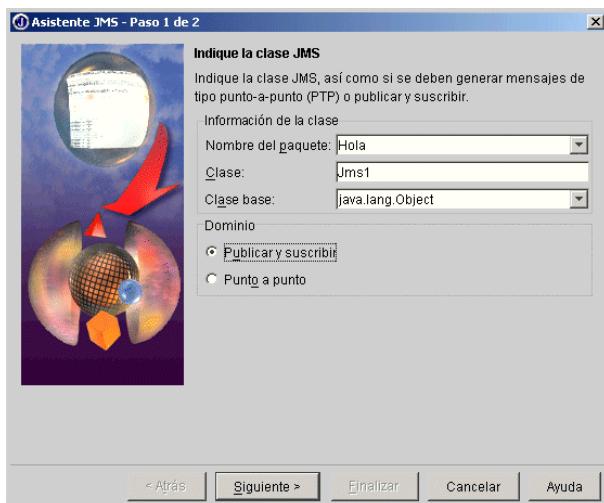
## Utilización del asistente JMS

El Asistente JMS genera una clase Java que incluye todo el código necesario para que las clases produzcan y consuman mensajes. Después basta con escribir un código muy sencillo con el fin de publicar o enviar el mensaje, si la clase es un productor. Si la clase es una consumidora de mensajes, añada el código para recibir el mensaje e implementar el método `onMessage()` para gestionar mensajes enviados o recibidos, dependiendo del tipo de modelo de mensaje que esté usando.

Para empezar a utilizar el Asistente JMS:

- 1 Elija Archivo | Nuevo, abra la ficha Enterprise y haga doble clic en el ícono JMS.

Se abre el Asistente JMS.

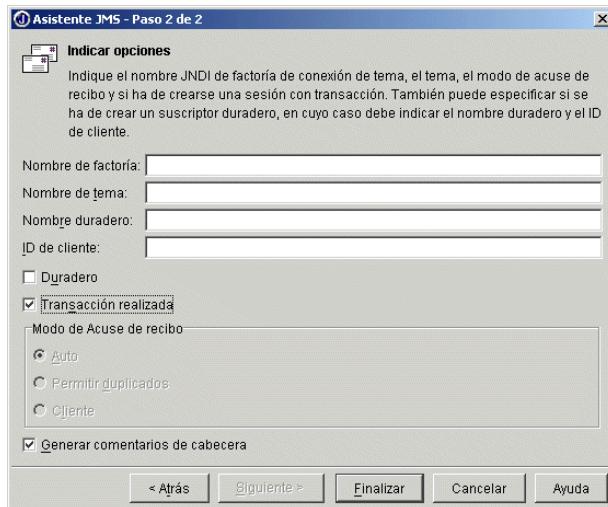


- 2 Especifique un nombre para el paquete y la clase o acepte los valores predeterminados.
- 3 Escriba en el campo Clase base la clase de la que desea que ésta sea ampliación o acepte el valor por defecto de `java.lang.Object`.
- 4 Elija Publicar y suscribir o Punto a punto, según el sistema de mensajes de la clase que va a crear.
- 5 Pulse Siguiente.

La siguiente ficha que aparece en el mensaje JMS depende del tipo de dominio seleccionado.

## Sistemas de mensajes de publicar y suscribir

Si se elige Publicar y suscribir, en el Asistente JMS aparece esta ficha:



Para terminar la clase:

**1** Indique un Nombre de factoría.

Las fábricas de conexión de temas se utilizan para configurar la conexión y la sesión del tema.

**2** Indique el nombre del tema del mensaje en el cuadro Nombre de tema.

**3** Si desea utilizar en la sesión un suscriptor duradero:

- Active la casilla de selección Duradero.
- Escriba un nombre duradero, irrepetible, para la suscripción duradera.
- Indique un identificador de cliente.

**4** Si va a ser una sesión de transacción, active la casilla de selección Transacción realizada.

Las sesiones de transacción utilizan los métodos `commit()` y `rollback()` para delimitar las transacciones locales.

**5** Si la sesión no es de transacción, elija un Modo de Acuse de recibo:

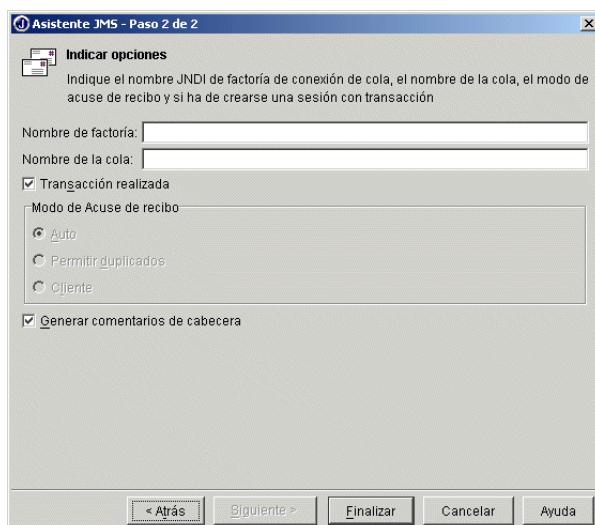
- Auto: La sesión reconoce automáticamente la recepción de un mensaje.
- Permitir duplicados: La sesión reconoce todos los mensajes y no efectúa ninguna comprobación para evitar las duplicaciones. Esto puede mejorar el tiempo de ejecución.

- Cliente: El cliente reconoce el mensaje llamando al método `acknowledge()` de éste.
- 6** Deje activada la casilla de selección Generar comentarios de cabecera si desea incluir en el código generado los comentarios que contienen el título, la descripción, etc.
- 7** Elija Finalizar.

## Sistemas de mensajes punto a punto

---

Si se elige Punto a punto, en el Asistente JMS aparece esta ficha:



Para terminar la clase:

- 1 Indique un Nombre de factoría.
- 2 Indique el nombre de la cola de mensajes en el cuadro Nombre de la cola.

- 3 Si va a ser una sesión de transacción, active la casilla de selección Transacción realizada.

Las sesiones de transacción utilizan los métodos `commit()` y `rollback()` para delimitar las transacciones locales.

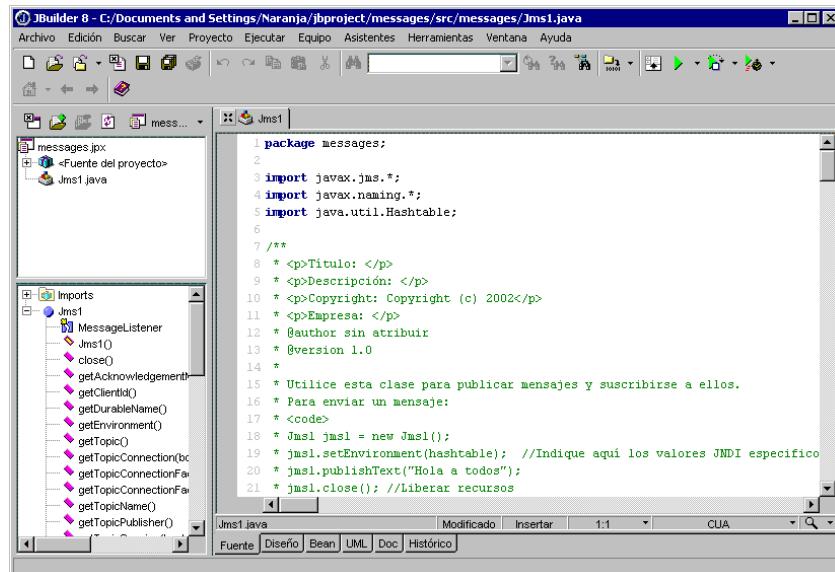
- 4 Si la sesión no es de transacción, elija un Modo de Acuse de recibo:
  - Auto: La sesión reconoce automáticamente la recepción de un mensaje.
  - Permitir duplicados: La sesión reconoce todos los mensajes y no efectúa ninguna comprobación para evitar las duplicaciones. Esto puede mejorar el tiempo de ejecución.

- Cliente: El cliente reconoce el mensaje llamando al método `acknowledge()` de éste.
- 5** Desactive la casilla de selección Generar comentarios de cabecera si desea omitir los comentarios que contienen el título, la descripción, etc.
- 6** Elija Finalizar.

## Código adicional

---

Cuando concluye la ejecución del Asistente JMS se puede ver el código generado en el panel de código fuente:



The screenshot shows the JBUILDER 8 IDE interface. The title bar reads "JBUILDER 8 - C:/Documents and Settings/Naranja/jbproject/messages/src/messages/Jms1.java". The menu bar includes Archivo, Edición, Buscar, Ver, Proyecto, Ejecutar, Equipo, Asistentes, Herramientas, Ventana, Ayuda. The toolbar has various icons for file operations. The left pane shows a project tree with "messages.jpx" expanded, showing "Fuente del proyecto" and "Jms1.java". The right pane displays the source code for "Jms1.java". The code is as follows:

```

1 package messages;
2
3 import javax.jms.*;
4 import javax.naming.*;
5 import java.util.Hashtable;
6
7 /**
8 * <p>Titulo: </p>
9 * <p>Descripción: </p>
10 * <p>Copyright: Copyright (c) 2002</p>
11 * <p>Empresa: </p>
12 * <p>Author sin atribuir
13 * <p>Version 1.0
14 *
15 * Utilice esta clase para publicar mensajes y suscribirse a ellos.
16 * Para enviar un mensaje:
17 * <code>
18 * Jms1 jms1 = new Jms1();
19 * jms1.setEnvironment(hashtable); //Indique aqui los valores JNDI especifico
20 * jms1.publishText("Hola a todos");
21 * jms1.close(); //Librar recursos

```

The status bar at the bottom shows "Jms1.java" is modified, with tabs for Fuente, Diseño, Bean, UML, Doc, and Histórico.

El código comentado de la parte superior del archivo de clase generado demuestra la forma de enviar un mensaje de texto y la forma de recibir un mensaje. Siga este ejemplo y añada el código necesario para enviar o recibir mensajes. Si la clase es un consumidor, busque el método `onMessage()` en el código fuente y complete su implementación de forma que gestione el mensaje recibido.



# 24

## Las aplicaciones distribuidas basadas en CORBA

Es una función de JBuilder Enterprise.

Este capítulo trata sobre la creación de aplicaciones distribuidas con JBuilder, el ORB de VisiBroker (que forma parte de Borland Enterprise Server) y la arquitectura común de agente de solicitud de objetos (Common Object Request Broker Architecture - CORBA).

### Definición de CORBA

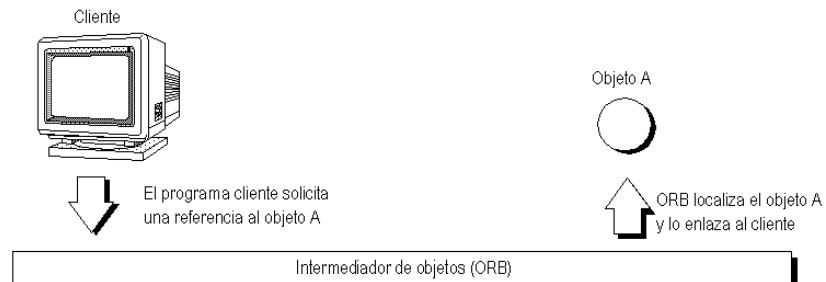
CORBA (Common Object Request Broker Architecture, arquitectura común de agente de solicitud de objetos) permite interaccionar (comunicarse entre sí) a las aplicaciones distribuidas, independientemente del lenguaje en que estén escritas y el lugar donde se encuentren.

El Object Management Group (Grupo de gestión de objetos) adoptó la especificación CORBA con el fin de resolver las complejidades y el elevado precio del desarrollo de aplicaciones de objetos distribuidos. CORBA utiliza un sistema orientado a objetos con el fin de crear componentes de software reutilizables que las aplicaciones pueden compartir. Todos los objetos encapsulan los detalles de su funcionamiento interno y presentan una interfaz bien definida, lo que reduce la complejidad de las aplicaciones. También se reduce el coste del desarrollo de aplicaciones, ya que los objetos implementados y comprobados pueden utilizarse varias veces.

El ORB (Object Request Broker, intermediador de objetos) conecta las aplicaciones clientes con los objetos que utilizan. El programa cliente no necesita saber si la implementación del objeto con la que se comunica se encuentra en el mismo ordenador o en otro lugar de la red. Basta con que conozca el nombre del objeto y entienda cómo utilizar su interfaz. El ORB

se encarga de localizar el objeto, encaminar la solicitud y devolver el resultado.

**Figura 24.1** Programa cliente actuando en un objeto



**Nota** El ORB no constituye un proceso independiente. Se trata de un conjunto de bibliotecas y recursos de red que se integra en aplicaciones de los usuarios finales y permite a las aplicaciones clientes localizar y utilizar los objetos.

## Definición del ORB de VisiBroker

El ORB de VisiBroker es un entorno completo de desarrollo y ejecución del ORB de CORBA para el desarrollo, la publicación y la gestión de aplicaciones distribuidas de Java abiertas, flexibles y compatibles. El ORB de VisiBroker forma parte de Borland Enterprise Server. Los objetos construidos con el ORB de VisiBroker son de fácil acceso desde aplicaciones web que se comunican por medio de la norma IIOP (Internet Inter-ORB Protocol, protocolo inter-ORB de Internet) del OMG para la comunicación por Internet o intranets. El ORB de VisiBroker tiene una implementación nativa de IIOP que garantiza el alto rendimiento y la compatibilidad.

## Trabajo conjunto de JBuilder y el ORB de VisiBroker

JBuilder proporciona un conjunto de asistentes y otras herramientas que simplifican el desarrollo de aplicaciones CORBA.

El tutorial del [Capítulo 25, “Tutorial: Creación de aplicaciones CORBA”](#) le ayudarán a familiarizarse con algunas de las las herramientas de desarrollo de JBuilder y el ORB de VisiBroker. Se utiliza una aplicación de ejemplo para ilustrar los distintos pasos del proceso.

Cuando se desarrollan aplicaciones distribuidas con JBuilder y el ORB de VisiBroker, deben identificarse, en primer lugar, los objetos que requiere la aplicación. El proceso suele constar de las siguientes etapas:

- 1 Escriba una especificación para cada uno de los objetos, con el Asistente para IDL, con el fin de generar el archivo IDL (Interface Definition Language, lenguaje de definición de interfaces).

IDL es el lenguaje que utilizan los implementadores para especificar las operaciones que proporcionan los objetos y la forma de llamarlas. En este ejemplo se va a definir en IDL la interfaz `Account`, con un método `balance()`, y la interfaz `AccountManager`, con un método `open()`.

- 2 Utilice el compilador IDL para generar el código stub del cliente y el código del servidor POA (Portable Object Adapter, adaptador portátil de objetos). Para más información acerca del POA, Consulte “Definición de POA” en la página 25-10.

Se utilizará el compilador `idl2java` para producir stubs de cliente (que proporcionan la interfaz de los métodos de los objetos `Account` y `AccountManager`) y clases de servidor (que proporcionan clases para la implementación de los objetos remotos).

- 3 Escriba el código de la programación cliente.

Para completar la implementación del programa cliente, inicie el ORB, establezca enlaces con los objetos `Account` y `AccountManager`, llame a sus métodos e imprima el saldo.

- 4 Escriba el código del objeto servidor.

Para completar la implementación del código del objeto servidor es necesario derivarlo de las clases `AccountPOA` y `AccountManagerPOA`, proporcionar implementaciones del método de `interface` e implementar la rutina `main` del servidor.

- 5 Compile el código del cliente y del servidor.

Para crear el programa cliente, compile su código con el stub cliente. Para crear el servidor `Account` compile el código del objeto servidor con los servidores.

- 6 Inicie el servidor.

- 7 Ejecute el programa cliente.

Si desea información más detallada sobre el uso del ORB de VisiBroker para la creación de aplicaciones distribuidas, consulte la documentación de Borland Enterprise Server, que se encuentra en el subdirectorio `doc` del directorio de instalación de Borland Enterprise Server.

Si desea más información sobre CORBA (Arquitectura común de agente de solicitud de objetos), consulte algunos enlaces prácticos:

- <http://www.borland.com/books>: Libros sobre productos de Borland y la tecnología relacionada.
- “A White Paper: Java, RMI, and CORBA” en <http://www.omg.org/news/whitepapers/wpjava.htm>

- CORBA/IIOP Specification, en Introducción a las especificaciones de OMG
- CORBA Basics, en <http://www.omg.org/gettingstarted/corbafaq.htm>

## Configuración de JBuilder para aplicaciones CORBA

---

Esta sección explica cómo configurar JBuilder con un ORB de VisiBroker u OrbixWeb con el fin de crear, ejecutar y distribuir aplicaciones CORBA.

Los tutoriales de este manual emplean el ORB de VisiBroker porque está incluido en JBuilder Enterprise, como parte de Borland Enterprise Server. Consulte la documentación de OrbixWeb si desea obtener más información acerca de las funciones especiales de ese ORB.

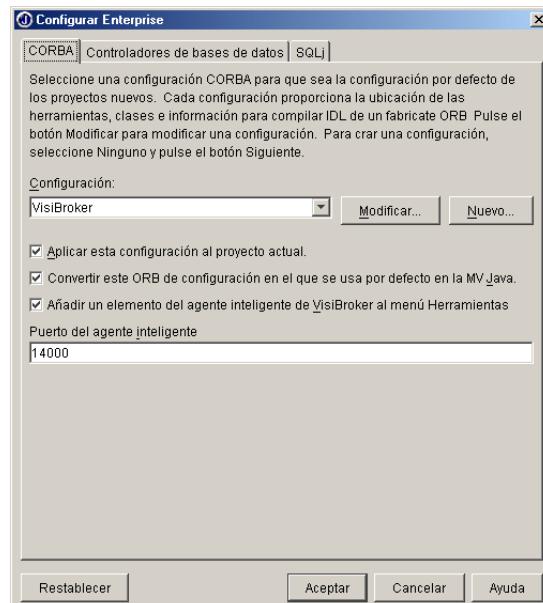
Para configurar estas piezas en su sistema:

- 1 Instale JBuilder Enterprise.
- 2 Instale un servidor de aplicaciones que contenga un ORB como, por ejemplo, el ORB de VisiBroker u OrbixWeb. Para obtener más información, consulte el [Capítulo 5, “Configuración del servidor de aplicaciones de destino”](#).

**Nota** El ORB de VisiBroker está incluido en JBuilder Enterprise, como parte de Borland Enterprise Server.

- 3 En JBuilder, al seleccionar Herramientas | Configurar Enterprise se abre el cuadro de diálogo homónimo; seleccione la pestaña CORBA. Los parámetros en este cuadro de diálogo permite a JBuilder ver los ORB.

- 4** Seleccione una configuración en la lista desplegable Configuración. Las opciones iniciales son VisiBroker y OrbixWeb. El cuadro de diálogo Configurar Enterprise tiene un aspecto semejante a éste:



- 5** Si utiliza el ORB de VisiBroker u OrbixWeb, pulse el botón Modificar y se abrirá el cuadro de diálogo Modificar configuración. (Haga clic sobre el botón Nueva para crear una configuración.)
- 6** En el cuadro Modificar configuración, pulse el botón de puntos suspensivos contiguo al campo Vía de acceso a herramientas ORB con el fin de permitir que JBuilder acceda a las herramientas de ORB. Cuando utilice el ORB de VisiBroker, diríjase al directorio que contiene el archivo `osagent.exe`. Se trata del subdirectorio `bin` del directorio de instalación de Borland Enterprise Server.
- 7** Para definir la localización de la biblioteca, pulse el botón de puntos suspensivos contiguo al campo Biblioteca para los proyectos. De esta forma se abre el cuadro de diálogo Seleccionar otra biblioteca. La biblioteca resulta necesaria para compilar los stubs y estructuras generadas y para ejecutar las aplicaciones. En VisiBroker, busque la biblioteca de Borland Enterprise Server 5.0.2 -5.1.x Client.
- a** Si desea añadir una biblioteca que no aparezca en el cuadro de diálogo Seleccionar otra biblioteca, pulse Nuevo.
  - b** Introduzca el nombre de la nueva biblioteca en el campo Nombre en el asistente.
  - c** Seleccione una ubicación para esta biblioteca: el directorio de JBuilder, el directorio del proyecto o el directorio inicial del usuario.

- d Haga clic en Añadir para desplazarse al archivo de biblioteca (JAR).
    - e Pulse Aceptar hasta volver al cuadro de diálogo Modificar configuración.
    - f Pulse de nuevo Aceptar para cerrar el cuadro de diálogo Modificar configuración.
  - 8 En el cuadro de diálogo Configurar Enterprise, deje seleccionada la opción Aplicar esta configuración al proyecto actual si el proyecto va a tener la misma configuración ORB que el proyecto por defecto.
  - 9 Deje activada la opción Convertir este ORB de configuración en el que se usa por defecto en la MV Java para definir este ORB como el predefinido de la MV de Java. Si recibe un mensaje de error, puede que no tenga suficientes derechos para modificar el archivo `orb.properties`, ubicado en el directorio `<jdk>/jre/lib/` de la instalación de JBuilder. Puede crear y/o editar manualmente este archivo, especificando la siguiente información. Este archivo de ejemplo se refiere al ORB de VisiBroker.

```
# Convertir el ORB de VisiBroker en ORB por defecto
org.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB
org.omg.CORBA.ORBSingletonClass=com.inprise.vbroker.orb.ORB
```
- 10 En el caso del ORB de VisiBroker, seleccione la opción Añadir un elemento VisiBroker Smart Agent al menú Herramientas con el fin de que dicho agente pueda ejecutarse como un servicio JBuilder durante el desarrollo. Esta opción estará desactivada si el agente no es necesario.
- 11 Seleccione un puerto en el que ejecutar el Smart Agent (agente inteligente). El puerto por defecto con el valor 14000 funciona en la mayoría de los sistemas.
- 12 Pulse Aceptar. Esta configuración se guarda en el archivo `orb.properties`.

Se ha completado la configuración del sistema para utilizar las características de JBuilder CORBA. Para ver un tutorial que utiliza estas características para crear una aplicación CORBA, consulte el [Capítulo 25, “Tutorial: Creación de aplicaciones CORBA”](#). Para más información acerca del uso de características VisiBroker con JBuilder, consulte [“Definición de interfaces en Java” en la página 24-7](#).

**Importante** Si utiliza Borland Enterprise Server, asegúrese de que el proyecto emplea el JDK 1.3. Compruebe el valor del JDK especificado en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto.

# Definición de interfaces en Java

---

En esta sección se explica la forma de utilizar los compiladores `java2iiop` y `java2idl` para generar stubs cliente y servidores desde definiciones de interfaz escritas en Java en lugar de utilizar IDL. Estos compiladores forman parte de Borland Enterprise Server, que se suministra con JBuilder.

En esta sección se tratan los temas siguientes:

- ["Utilización del compilador java2iiop" en la página 24-10.](#) Describe la creación de aplicaciones distribuidas totalmente en Java, en el entorno compatible con IIOP.
- ["Utilización del compilador java2idl" en la página 24-14.](#) Describe como convertir el código Java en interfaces IDL además de permitir generar stubs de cliente en el lenguaje de programación que se elija.

Si desea más información sobre estos compiladores consulte la documentación de Borland Enterprise Server. Esta documentación se encuentra en el subdirectorio `doc` del directorio de instalación de Borland Enterprise Server.

## Acerca de los compiladores `java2iiop` y `java2idl`

---

Borland Enterprise Server incluye los siguientes compiladores, que facilitan su manejo en un entorno Java. Entre estas funciones se encuentran:

- **`java2iiop`**

El compilador `java2iiop` permite mantenerse en un entorno cien por cien Java, si se desea. El compilador `java2iiop` toma interfaces de Java y genera stubs y estructuras básicas que cumplen las especificaciones de IIOP. Una de las ventajas del compilador `java2iiop` consiste en que las estructuras ampliables permiten pasar por valor objetos serializados de Java.

- **`java2idl`**

El compilador `java2idl` convierte el código Java en IDL, lo que permite generar stubs cliente en el lenguaje elegido. Además, como este compilador asocia las interfaces de Java a IDL, se pueden volver a implementar los objetos de Java en otro lenguaje de programación que acepte el mismo IDL.

- **Nomenclatura de web**

Nomenclatura de Web le permite asociar las URL a objetos, con lo que se puede obtener una referencia a un objeto especificando una URL.

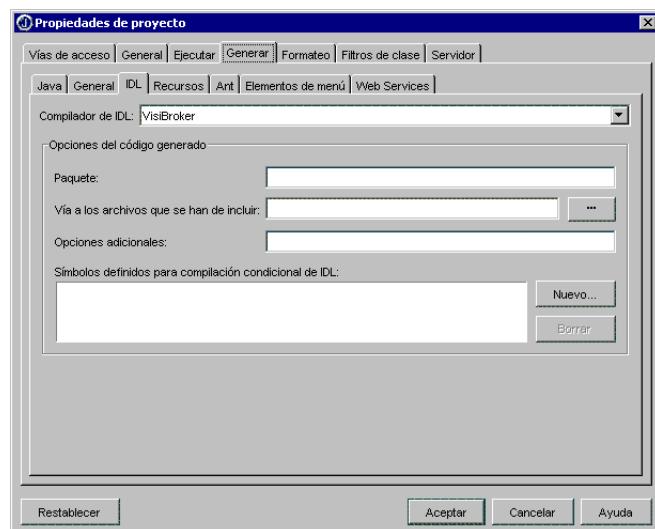
Si pretende diseñar interfaces CORBA en Java, deberá utilizar el compilador `java2iiop` para producir los stubs cliente y los servidores. Cuando se diseñan interfaces CORBA en Java no es necesario crear un archivo IDL; de hecho, si se genera no es posible modificarlo ni compilarlo. Sólo es necesario generar un archivo IDL desde una interfaz Java para:

- Rellenar el repositorio de interfaces.
- Construir aplicaciones multilenguaje.

## Acceso a los compiladores `java2iiop` y `java2idl` en JBuilder

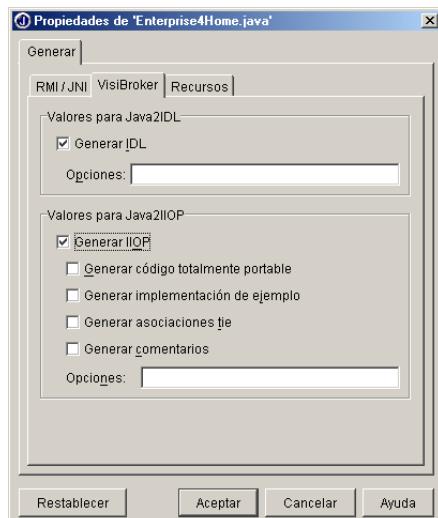
Para acceder a los compiladores `java2iiop` o `java2idl` dentro del entorno de desarrollo de JBuilder:

- 1 Asegúrese de que JBuilder está configurado correctamente, siguiendo los pasos explicados en “[Configuración de JBuilder para aplicaciones CORBA](#)” en la página 24-4.
- 2 Elija Proyecto | Propiedades de proyecto para abrir el cuadro de diálogo homónimo. Compruebe que se ha seleccionado VisiBroker en el campo Compilador IDL de la pestaña IDL de la ficha Generar.



- 3 Compruebe que la lista de bibliotecas necesarias de la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto incluye el cliente Borland Enterprise Server 5.0.2 -5.1.x Client.
- 4 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

- 5 Cuando haya creado un archivo de interfaz Java en el proyecto JBuilder, haga clic con el botón derecho del ratón en el archivo, en el panel de proyecto. Seleccione Propiedades.
- 6 Elija las opciones Generar IDL y Generar IIOP de la pestaña Visibroker, en la ficha Generar. El cuadro de diálogo tendrá un aspecto parecido a éste:



- 7 Pulse Aceptar para cerrar el cuadro de diálogo.
- 8 Haga clic con el botón derecho en el archivo de interfaz Java, en el panel de proyecto, y seleccione Ejecutar Make con el fin de compilar la definición de la interfaz y generar archivos de interfaz IIOP o IDL.

## RMI

Otra forma de llamar a métodos en objetos de Java remotos consiste en utilizar la RMI (Remote Method Invocation, llamada a métodos remotos). Borland Enterprise Server cuenta con funciones equivalentes a las de RMI, pero además, permite crear objetos de Java que se comunican con todos los demás objetos que cumplen las especificaciones de CORBA, aunque no estén escritos en Java. Si desea ver un ejemplo de definición de interfaces en Java en la que se describe una interfaz remota RMI, consulte el archivo de ejemplo `Account.java`, que se encuentra en el subdirectorio `examples/vbe/rmi-iiop/Bank` del directorio de instalación de Borland Enterprise Server.

## Utilización del compilador java2iiop

---

El compilador `java2iiop` permite definir interfaces y tipos de datos que se pueden utilizar como tales en CORBA. La ventaja es que es posible definirlos en Java, en lugar de utilizar IDL. El compilador lee el byte code de Java; no lee el código fuente, sino los archivos de clase. Así, el compilador genera los stubs y estructuras básicas que cumplen las especificaciones de IIOP necesarias para efectuar el montaje y la comunicación que requiere CORBA.

Cuando se ejecuta, el compilador `java2iiop` genera los mismos archivos que generaría si la interfaz se hubiera escrito en IDL. El compilador `java2iiop` interpreta y asocia a los tipos equivalentes de IDL los tipos de datos primitivos, como los tipos numéricos (`short`, `int`, `long`, `float` y `double`), las cadenas, los objetos de CORBA o de interfaz, los tipos Any y los códigos de tipo.

Cuando se utiliza el compilador `java2iiop` con interfaces Java, éstas se definen y se marcan como las interfaces que han de utilizarse en llamadas remotas. Se marcan haciendo que amplíen la interfaz `org.omg.CORBA.Object`. (Esta interfaz también debe definir métodos que se puedan utilizar en llamadas remotas.) Cuando se ejecuta, el compilador busca estas interfaces CORBA especiales. Cuando se encuentra una, genera todos los elementos de recodificación, lectores y escritores que permiten utilizar la interfaz para llamadas remotas. El subdirectorio `examples/vbe/rmi-iiop/Bank` del directorio de instalación de Borland Enterprise Server contiene un ejemplo de archivo de definición de interfaces Java, llamado `AccountManager.java`.

**Nota** Para los desarrolladores familiarizados con RMI (Llamadas a métodos remotos), esto equivale a ampliar la clase de la interfaz `java.rmi.Remote`.

El compilador sigue reglas distintas para las clases y las asocia a estructuras IDL o ampliables. Si desea más información sobre los tipos de datos complejos, consulte "[Correspondencia de tipos de datos complejos](#)" en la página 24-13.

### Creación de interfaces IIOP ejecutando `java2iiop`

En los archivos de ejemplo, que se encuentran en el subdirectorio `examples/vbe/rmi-iiop` del directorio de instalación de Borland Enterprise Server, se demuestra la forma de definir una interfaz en Java y compilarla mediante `java2iiop` en stubs y servidores compatibles con IIOP.

Los archivos de este ejemplo son los siguientes:

- `Bank/Account.java` - La interfaz Java que describe la interfaz remota Account RMI.
- `Bank/AccountData.java` - La clase Java que se utiliza para transmitir los datos de creación de cuenta.

- Bank/AccountManager.java - La interfaz Java que describe la interfaz CORBA AccountManager.
- Bank/AccountManagerOperations.java - Esta interfaz declara las firmas de métodos definidas en AccountManager.java.
- AccountImpl.java - El servidor que implementa la interfaz Account.
- AccountManagerImpl.java - El servidor que implementa la interfaz AccountManager.
- Client.java - La línea principal del cliente.
- Server.java - La línea principal del servidor.

La interfaz AccountManager crea una cuenta, la abre o la vuelve a abrir y presenta una lista de cuentas. La cuenta que se crea contiene operaciones que obtienen el nombre de Account y definen y obtienen el saldo de la cuenta. AccountData representa la información utilizada para crear cuentas.

Este ejemplo también demuestra el uso de RMI en lugar de IIOP para pasar clases Java nativas (las que no se asocian desde IDL) por cable, en llamadas a métodos remotos. También demuestra el uso de una interfaz remota de tipo RMI y la creación de servidores CORBA a los que acceden clientes RMI. Si desea ejecutar y ver el ejemplo, abra el archivo `rmi-iiop_java.html` del directorio `examples/vbe/rmi-iiop` de la instalación de Borland Enterprise Server.

Antes de generar las interfaces IIOP en JBuilder, compruebe que el compilador `java2iiop` está seleccionado en el cuadro de diálogo Propiedades. Para ello:

- 1** Haga clic con el botón derecho del ratón en el archivo de interfaz Java, AccountManager, en el panel de proyecto. Elija Propiedades en el menú contextual.
- 2** Seleccione la opción Generar IIOP de la pestaña VisiBroker, en la ficha Generar, y pulse Aceptar.
- 3** Haga clic con el botón derecho en el archivo de interfaz de Java y elija Ejecutar Make en el menú contextual.

El compilador `java2iiop` genera todos los archivos auxiliares habituales, como las clases Helper y Holder. Si desea más información sobre los archivos generados, consulte la documentación de Borland Enterprise Server.

**Nota** Los compiladores `java2iiop` e `idl2java` generan los mismos archivos.

En la documentación de Borland Enterprise Server se facilita más información, como código de muestra y aplicaciones de ejemplo, para completar el proceso de desarrollo con el ORB de VisiBroker. En general, después de generar stubs y estructuras básicas, es necesario crear las clases servidor y cliente. A continuación, siga estos pasos:

1 Utilice la clase de estructura para crear una implementación del objeto de servidor; si desea ver un ejemplo, consulte el archivo `Server.java` del directorio `examples/vbe/rmi-iiop` de la instalación de Borland Enterprise Server.

2 Compile la clase de servidor, haciendo clic con el botón derecho del ratón en el archivo del servidor, que aparece en el panel de proyecto, y seleccione Ejecutar Make.

**Nota** Si utiliza `Server.java` puede ser necesario cambiar el nombre de `Makefile` y `Makefile.java`, que se encuentran en el directorio `examples/vbe/rmi-oop`. Pueden interferir con el compilador de JBuilder.

3 Escriba el código del cliente. El archivo `Client.java` del subdirectorio `examples/vbe/rmi-iiop` del directorio de instalación de Borland Enterprise Server contiene un ejemplo.

4 Compile el código del cliente haciendo clic con el botón derecho en el archivo del cliente, que aparece en el panel de proyecto, y seleccione Ejecutar Make.

**Nota** Si utiliza `Client.java` puede ser necesario cambiar el nombre de `Makefile` y `Makefile.java`, que se encuentran en el directorio `examples/vbe/rmi-iiop`. Pueden interferir con el compilador de JBuilder.

5 Inicie el agente inteligente del ORB de VisiBroker seleccionándolo en el menú Herramientas. Para que esta opción se encuentre disponible es necesario haber configurado JBuilder de la forma descrita en “[Configuración de JBuilder para aplicaciones CORBA](#)” en la [página 24-4](#).

6 Inicie los objetos de servidor haciendo clic con el botón derecho en el archivo del servidor, en el panel de proyecto, y elija Ejecutar.

7 Inicie el cliente haciendo clic en su archivo con el botón derecho, en el panel de proyecto, y elija Ejecutar.

**Importante** En los ejemplos que se muestran en este capítulo se da por supuesto que se está ejecutando un Agente inteligente del ORB de VisiBroker. Cerciórese de que el agente inteligente se encuentra en ejecución, seleccionando Herramientas | VisiBroker Smart Agent. Si desea más información sobre el Smart Agent (agente inteligente) consulte la documentación de Borland Enterprise Server.

## Correspondencia de tipos de datos primitivos a IDL

Los stubs cliente que genera `java2iiop` gestionan el montaje de los parámetros de los tipos de datos primitivos de Java que representan solicitudes de operación, de modo que se puedan transmitir al servidor de objetos. Cuando se recodifica un tipo de datos primitivo de Java para su transmisión, hay que convertirlo a un formato compatible con IIOP. Si

desea más información, consulte la documentación de Borland Enterprise Server.

## Correspondencia de tipos de datos complejos

Este apartado muestra cómo utilizar el compilador `java2iiop` para manejar tipos de datos complejos, como las interfaces, las matrices, las clases de Java y las estructuras ampliables. Si desea más información, consulte la documentación de Borland Enterprise Server.

### Interfaces

Las interfaces Java se representan en IDL como interfaces CORBA. Deben ser herederas de la interfaz `org.omg.CORBA.Object`. Se pasan por referencia cuando se pasan objetos que implementan estas interfaces.

**Nota** El compilador `java2iiop` no acepta métodos sobrecargados en interfaces CORBA.

Las interfaces que no son ampliaciones de `org.omg.CORBA.Object` se asocian a una estructura ampliable.

### Matrices

Las matrices son otro tipo de datos complejo que se puede definir en las clases. Si tiene interfaces o definiciones que utilicen matrices, éstas se asocian a secuencias ilimitadas de CORBA.

## Correspondencia de clases de Java

En las clases Java se pueden definir tipos de datos no estándar. Algunos de ellos se parecen a las estructuras de IDL (también llamados *structs*). Si se define una clase de Java de forma que cumpla unos requisitos determinados, el compilador `java2iiop` la asocia a una estructura de IDL. Las clases de Java se pueden asociar a estructuras de IDL si cumplen todos los requisitos siguientes:

- La clase es final.
- La clase es pública.
- La clase no utiliza herencia de implementación.
- Los miembros de datos de la clase son públicos.

El compilador `java2iiop` comprueba si las clases cumplen todos estos requisitos. Si es así, las asocia a estructuras de IDL; si no cumplen todos los requisitos, las asocia a estructuras ampliables.

### Estructuras ampliables

Las clases de Java que no cumplen los requisitos mencionados anteriormente se asocian a estructuras ampliables. Las estructuras ampliables son extensiones de estructuras CORBA compatibles con

versiones posteriores. Cuando se utilizan estructuras ampliables, los objetos pasan por valor.

El paso por valor es la capacidad de pasar el estado del objeto a otro programa Java. Si hay una definición de clase del objeto de Java en el servidor, el programa Java puede llamar a métodos del objeto clonado que tengan el mismo estado que el objeto original.

El uso de estructuras ampliables es una ampliación de Borland Enterprise Server del IDL OMG. Proporciona una palabra clave adicional: `extensible`. Si desea trabajar exclusivamente en CORBA o si desea llevar el código a otros ORB, debe usar estructuras de IDL y no estructuras ampliables. Las estructuras ampliables permiten utilizar clases que se pueden definir en Java pero no en IDL, a causa de las limitaciones de CORBA.

El ORB de VisiBroker utiliza la serialización de Java para pasar las clases como estructuras ampliables. La serialización de Java comprime el estado de los objetos de Java en un flujo en serie de octetos que se pueden pasar en la transmisión como parte de la solicitud.

**Nota** A causa de la serialización de Java, todos los tipos de datos que se pasan deben ser serializables, esto es, deben implementar `java.io.Serializable`.

Las estructuras ampliables permiten pasar correctamente los datos que hacen uso de punteros. Por ejemplo, es habitual definir listas vinculadas en el código de la aplicación, pero no es posible definir una lista vinculada en IDL estándar. Este problema se resuelve definiendo la lista por medio de estructuras ampliables. Cuando se pasan estructuras ampliables se mantienen los punteros.

## Utilización del compilador `java2idl`

---

IDL es el lenguaje en el que se basan las aplicaciones CORBA. Las interfaces IDL definen las fronteras de los distintos componentes. IDL puede utilizarse para definir los atributos de un componente, las excepciones que lanza y los métodos de los que dispone su interfaz.

Es posible generar interfaces IDL a partir de interfaces Java definidas en un archivo `.java` mediante el compilador `java2idl`. La utilización de interfaces de Java para describir objetos de CORBA tiene ciertas limitaciones. Se pueden utilizar todos los tipos primitivos de Java. No obstante, sólo se pueden utilizar objetos de Java para definir la interfaz si el objeto implementa `java.io.Serializable`.

A continuación se muestra la forma de generar interfaces IDL a partir de una definición de interfaz Java.

- 1 Haga clic con el botón derecho en un archivo de interfaz Java, en el panel de proyecto. Elija Propiedades en el menú contextual.

- 2** Active la opción Generar IDL de la pestaña VisiBroker, en la ficha Generar, y pulse Aceptar.
- 3** Haga clic con el botón derecho en el archivo de interfaz de Java y elija Ejecutar Make en el menú contextual.

El compilador `java2idl` genera todos los archivos auxiliares habituales, como las clases Helper y Holder. Haga clic en el signo "más" a la izquierda del archivo de interfaz de Java, en el panel de proyecto, con el fin de abrir los archivos generados. Si desea más información sobre los archivos generados, consulte la documentación de Borland Enterprise Server.



# 25

## Tutorial: Creación de aplicaciones CORBA

Este tutorial es una  
característica de JBuilder  
Enterprise

En este tutorial vamos a construir un ejemplo de aplicación cliente que puede consultar el saldo de una cuenta bancaria. En este tutorial se describen las etapas que hay que seguir para crear los archivos básicos necesarios para ejecutar y distribuir una aplicación distribuida sencilla basada en CORBA.

Con esta explicación aprenderá a:

- Definir interfaces de objetos en IDL.
- Generar stubs de cliente y código de servidor.
- Implementar el programa cliente: inicializar ORB, establecer un enlace con el objeto `AccountManager`, obtener el saldo, imprimir el balance y gestionar las excepciones.
- Implementar el objeto servidor, inicializar el ORB, crear el POA (Portable Object Adapter, adaptador portátil de objetos), crear el objeto servidor `Account Manager`, activar el objeto servidor, activar el gestor de POA y el POA, y prepararse para recibir solicitudes.
- Realizar una compilación completa del ejemplo.
- Ejecutar el ejemplo con un cliente Java: iniciar Agente inteligente del ORB de Visibroker, iniciar el servidor `AccountManager` y ejecutar el programa cliente.

Para obtener más información sobre CORBA, consulte el [Capítulo 24, “Las aplicaciones distribuidas basadas en CORBA”](#).

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 [1-2](#).

## Paso 1: Configuración del proyecto

---

En este tutorial se supone que está familiarizado con las herramientas de diseño de JBuilder. Si es la primera vez que utiliza JBuilder, en “El entorno de JBuilder” de *Introducción a JBuilder* encontrará una descripción introductoria.

- 1 Configure JBuilder y VisiBroker de forma que se detecten mutuamente, como se describe en “[Configuración de JBuilder para aplicaciones CORBA](#)” en la página [24-4](#)
- 2 Seleccione Archivo | Proyecto nuevo para crear un proyecto nuevo.
- 3 Cambie el nombre a BankTutorial.
- 4 Cambie el nombre del directorio a BankTutorial. Asegúrese de que la opción Generar archivo de notas del proyecto se encuentra seleccionada.
- 5 Haga clic en Siguiente para avanzar al Paso 2 del asistente.
- 6 Como JDK, utilice el JDK 1.3. Si desea informarse sobre la configuración del JDK, consulte el tema “Configuración del JDK” en el capítulo “Creación y gestión de proyectos” de *Creación de aplicaciones con JBuilder*.
- 7 Asegúrese de que Borland Enterprise Server 5.0.2 -5.1.x Client aparece en la lista de la pestaña Bibliotecas necesarias.
- 8 Pulse Finalizar en el segundo paso.
- 9 Compruebe que el ORB de VisiBroker se ha definido como compilador IDL por defecto. Para ello:
  - a Seleccione Proyecto | Propiedades de proyecto para abrir el cuadro de diálogo Propiedades de proyecto.
  - b Seleccione la pestaña Generar. Pulse la pestaña IDL de la ficha Generar.
  - c Compruebe que el ORB de VisiBroker está seleccionado en la lista desplegable Compilador IDL.
  - d Pulse Aceptar para cerrar el cuadro de diálogo.

El proyecto BankTutorial aparece en el panel de proyecto. A continuación se definirán las interfaces de objetos, escribiendo la interfaz de Account en IDL.

## Paso 2: Definición de las interfaces de los objetos CORBA en IDL

El primer paso para la creación de una aplicación CORBA consiste en indicar todos los objetos y sus interfaces, utilizando el IDL (Lenguaje de definición de interfaces) de OMG. IDL tiene una sintaxis parecida a la de C++, y se puede utilizar para definir módulos, interfaces, estructuras de datos, etc. Se puede establecer correspondencias entre el IDL y diversos lenguajes de programación. La asignación de IDL para Java se resume en la documentación de Borland Enterprise Server.

Aquí se muestra cómo crear el archivo `BankTutorial.idl` para este ejemplo. La interfaz `Account` dispone de un método para obtener el saldo actual. La interfaz `AccountManager` crea una cuenta para el usuario si no existe aún.

Para crear el archivo IDL y definir las interfaces de objeto:

- 1 Elija Archivo | Nuevo y seleccione IDL de ejemplo de la pestaña CORBA de la galería de objetos.
- 2 Escriba `BankTutorial.idl` en el campo Archivo. Pulse Aceptar. Se crea un archivo IDL de ejemplo, que se añade al proyecto.
- 3 Borre el código generado e introduzca el siguiente código IDL.

```
module Bank {
    interface Account {
        float balance();
    };
    interface AccountManager {
        Account open(in string name);
    };
}
```

- 4 Elija Archivo | Guardar todo. El archivo IDL se debe guardar antes de la compilación.

En la tercera ficha se utiliza el compilador IDL de Borland Enterprise Server para generar las rutinas de stub y el código de servidor de la especificación IDL.

## Paso 3: Generación de stubs de cliente y servidores

---

En esta ficha se utiliza el compilador IDL de Borland Enterprise Server, también llamado `idl2java`, para generar las rutinas de stub y el código de servidor de la especificación IDL. El programa cliente utiliza las rutinas de stub para llamar a operaciones en un objeto. El código de servidor, junto con el código escrito, crea un servidor que implementa el objeto. Cuando se completa, el código del programa cliente y el objeto servidor se utiliza como entrada en el compilador Java, que produce las clases ejecutables cliente y servidor.

Para generar los stubs de cliente y los servidores:

- 1 Haga clic con el botón derecho en el archivo `BankTutorial.idl`, en el panel de proyecto.
- 2 Seleccione Ejecutar Make para llamar al compilador `idl2java`.

Si el archivo requiere algún tratamiento especial, puede especificar las propiedades IDL haciendo clic con el botón derecho sobre el archivo IDL en el panel de proyecto y seleccionando Propiedades. Abre la pestaña IDL de la ficha Generar del cuadro de diálogo Propiedades de proyecto, donde se pueden establecer las opciones para compilar interfaces remotas definidas en IDL.

### Archivos generados

---

Ya que Java sólo admite una interfaz pública o una clase por archivo, la compilación del archivo IDL genera varios archivos .java. Estos archivos se guardan en un subdirectorio llamado `Generated Source`, que es el nombre de módulo especificado en el archivo IDL y es el paquete al que pertenecen los archivos generados. Si desea ver los archivos generados, pulse el icono de ampliación contiguo a `BankTutorial.idl`.

Se generan los siguientes archivos:

- `_AccountManagerStub.java` - Código stub para el objeto `AccountManager` del cliente.
- `_AccountStub.java` - Código stub para el objeto `Account` del cliente.
- `Account.java` - La declaración de interfaz de `Account`.
- `AccountHelper.java` - Declara la clase `AccountHelper`, que define métodos de utilidades.
- `AccountHolder.java` - Declara la clase `AccountHolder`, que proporciona un contenedor intermedio para el paso de objetos `Account`.
- `AccountManager.java` - La declaración de interfaz de `AccountManager`.
- `AccountManagerHelper.java` - Declara la clase `AccountManagerHelper`, que define métodos de utilidades.

- AccountManagerHolder.java - Declara la clase AccountManagerHolder, que proporciona un contenedor intermedio para el paso de objetos AccountManager.
- AccountManagerOperation.java - Esta interfaz declara las firmas de método definidas en la interfaz AccountManager del archivo Bank.idl.
- AccountManagerPOA.java - Código de servidor POA (código base de implementación) para la implementación del objeto AccountManager del servidor.
- AccountManagerPOATie.java - Clase utilizada para implementar el objeto AccountManager en el servidor por medio del mecanismo de enlace. Si desea más información sobre el mecanismo tie consulte la documentación de Borland Enterprise Server.
- AccountOperations.java - Esta interfaz declara las firmas de método definidas en la interfaz Account del archivo Bank.idl.
- AccountPOA.java - Código de servidor POA (código base de implementación) para la implementación del objeto Account del servidor.
- AccountPOATie.java - Clase utilizada para implementar el objeto Account en el servidor por medio del mecanismo de enlace. Si desea más información sobre el mecanismo tie consulte la documentación de Borland Enterprise Server.

Si desea más información sobre los archivos generados, consulte la documentación de Borland Enterprise Server.

En el siguiente paso se implementa el programa cliente.

## Paso 4: Implementación del cliente

---

El cliente de la aplicación servidor CORBA puede ser otra aplicación que se ejecute en el equipo local, como se explica en este tutorial. También puede ser un cliente HTML que se ejecute en un navegador. Se puede crear un applet, una página JavaServer (JSP) o un servlet. Asimismo es posible crear una página JSP o un servlet que utilice InternetBeans. Si desea obtener más información acerca del desarrollo de aplicaciones web con JBuilder, consulte la *Guía del Desarrollador de aplicaciones Web*.

Muchas de las clases que se utilizan para implementar el cliente de bank se encuentran en el paquete Bank, que genera el compilador idl2java. Los pasos siguientes sirven para crear un cliente Java que inicializa el ORB, se asocia al objeto AccountManager, obtiene el saldo y lo imprime, y gestiona las excepciones.

Para crear el cliente:

- 1** Cree una aplicación, que se utilizará para crear la interfaz de usuario.  
Para ello:
  - a** Seleccione Archivo | Nuevo y elija Aplicación en la pestaña General de la galería de objetos, para iniciar el Asistente para aplicaciones.
  - b** Acepte todas las opciones por defecto tanto en el paso 1 como en el paso 2. Pulse en Siguiente para pasar al siguiente paso.
  - c** En el paso 3, asegúrese de que la opción Crear una configuración de ejecución está activada.
  - d** Pulse Finalizar para crear el archivo de aplicación.
- 2** Cree la interfaz AccountManager.
  - a** Elija Asistentes | Usar interfaz CORBA. No elija esta opción en el primer paso del asistente.
  - b** Pulse Siguiente para pasar al Paso 2.
  - c** Elija BankTutorial.idl como archivo IDL.
  - d** No modifique el nombre de la clase, que debe ser AccountManagerClientImpl1.
  - e** Asegúrese de que banktutorial.Bank.AccountManager está seleccionado en la lista Interfaz.
  - f** Pulse Siguiente para pasar al Paso 3.
  - g** Acepte el nombre de campo por defecto, myAccountManager. Pulse el botón Finalizar. Se crea el archivo AccountManagerClientImpl1.java y se muestra en el panel de proyecto.
- 3** Repita los mismos pasos para crear la interfaz Account:
  - a** Elija Asistentes | Usar interfaz CORBA. No elija esta opción en el primer paso del asistente.
  - b** Pulse Siguiente para pasar al Paso 2.
  - c** Elija BankTutorial.idl como archivo IDL.
  - d** Cambie el nombre de la clase a AccountClientImpl1.
  - e** Elija banktutorial.Bank.Account en la lista Interfaz.
  - f** Pulse Siguiente para pasar al Paso 3.
  - g** Acepte el nombre de campo por defecto, myAccount. Pulse el botón Finalizar. Se crea el archivo AccountClientImpl1.java y se muestra en el panel de proyecto.
- 4** Elija Archivo | Guardar todo.

## Realización de una asociación con el objeto AccountManager

---

El paso siguiente consiste en conectar la interfaz de fábrica (AccountManager) al Intermediador de objetos (Object Request Broker, ORB). Para ello:

- 1 Haga doble clic en `Marco1.java`, en el panel de proyecto.
- 2 Haga clic en la pestaña Diseño para abrir el archivo en el diseñador de interfaces.
- 3 Seleccione la pestaña CORBA en la paleta de componentes.
- 4 Seleccione un objeto `OrbConnect`.
- 5 Haga clic en el árbol de componentes para añadir un objeto `OrbConnect` al marco.
- 6 Seleccione el objeto `myAccountManager` en el árbol de componentes.
- 7 En el Inspector, elija la propiedad `ORBConnect`. Seleccione `orbConnect1` en la lista desplegable.
- 8 Seleccione el objeto `OrbConnect1` en el árbol de componentes.
- 9 En el Inspector, asigne a su propiedad `initialize` el valor `true`.
- 10 Elija Archivo | Guardar todo.

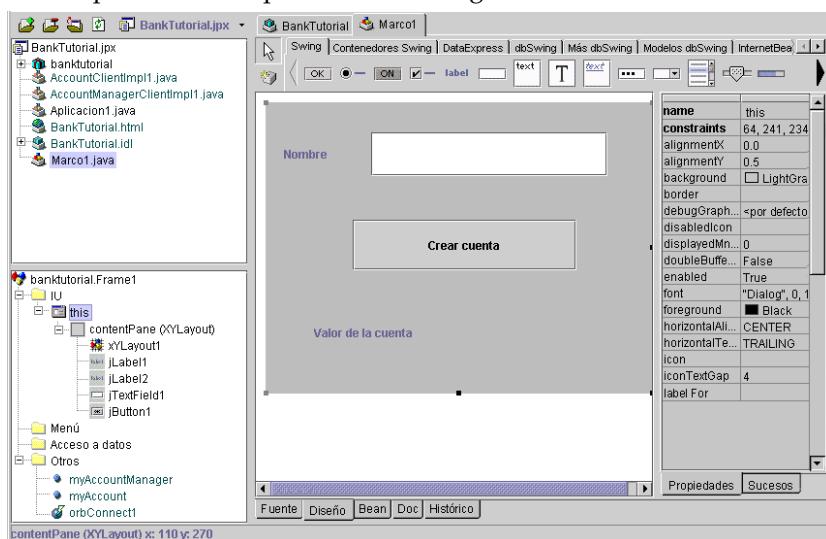
## Asociación de la clase englobadora durante la ejecución

---

Para asociar la clase englobadora durante la ejecución:

- 1 Seleccione `contentPane` en el árbol de componentes.
- 2 Asigne a su propiedad `layout` el valor `XYLayout`.
- 3 Seleccione un control `JButton` de la pestaña Swing de la paleta de componentes. Haga clic en el centro del marco para añadir el control `JButton` a la superficie de diseño. Este botón se utiliza para crear una cuenta nueva y asociarla a la referencia `AccountManager`.
- 4 Desde el Inspector, seleccione la propiedad `text` de `JButton`. Escriba `Crear cuenta`. Es posible que sea necesario cambiar el tamaño del control.
- 5 Seleccione un control `JLabel` de la pestaña Swing de la paleta de componentes. Colóquelo en la esquina superior izquierda de la superficie de diseño.
- 6 En el Inspector, cambie la propiedad `text` a `Nombre`. Es posible que sea necesario cambiar el tamaño del control de modo que quepa el texto.

- 7 Seleccione un control JTextField de la pestaña Swing. Colóquelo a la derecha del control JLabel que se acaba de añadir. Este control se utiliza para introducir el nombre de la cuenta nueva.
- 8 En el Inspector, borre jTextField1 de la propiedad text y déjela en blanco.
- 9 Seleccione otro control JLabel de la pestaña Swing. Coloque este control debajo del objeto JButton añadido en el paso 3. Esta etiqueta muestra el saldo de la cuenta. Cambie el tamaño de este control de forma que tenga casi la anchura del marco.
- 10 En el Inspector, asigne a su propiedad text el valor Valor de la cuenta. Si lo desea, cambie las propiedades de alineación horizontal para que el texto quede centrado en el control. El diseñador de interfaces de usuario presenta un aspecto similar al siguiente:



#### Importante

Si se fuera a desarrollar este programa en un entorno de producción no se utilizaría XYLayout, ya que está destinado únicamente al diseño. En su lugar se utilizaría otra configuración, como GridBagConstraints, que ajusta la presentación al sistema operativo. Para cambiar la configuración de diseño, seleccione el objeto contentPane en el Inspector y cambie la propiedad layout.

- 11 Seleccione el control JButton. Haga clic en la pestaña Sucesos del Inspector. Seleccione el suceso actionPerformed y haga doble clic en el cuadro de valor para crear el suceso. El foco se desplaza al editor.
  - 12 Escriba el código siguiente en el suceso jButton1ActionPerformed:
- ```
myAccount.setCorbaInterface(myAccountManager.open(jTextField1.getText()));
jLabel2.setText("The account balance is " + myAccount.balance());
```
- 13 Elija Archivo | Guardar todo.

Las clases clientes, `Application1.java` y `Frame1.java`, hacen lo siguiente:

- 1** Inicializan el ORB especificado en las propiedades de `OrbConnect`.
- 2** Se asocia a un objeto `AccountManager`.

Para poder llamar al método `balance()`, el programa cliente debe utilizar el método `bind()` para establecer una conexión con el servidor que implemente el objeto `AccountManager`. El compilador `idl2java` genera automáticamente la implementación del método `bind()`. El método `bind()` solicita el ORB para localizar el servidor y establecer la conexión. Si se localiza el servidor y la conexión se establece, se crea un objeto proxy que representa el objeto `AccountManagerPOA` del servidor. Se devuelve al programa cliente una referencia al objeto `AccountManager`.

Ahora la clase cliente debe llamar al método `open()` del objeto `AccountManager` con el fin de obtener una referencia del objeto `Account` del nombre de cliente especificado.

- 3** Obtiene el saldo de la cuenta por medio de la referencia de objeto devuelta por `bind()`.

Después de que el programa cliente establezca una conexión con un objeto `Account` se puede utilizar el método `balance()` para obtener el saldo. En realidad, el método `balance()` del cliente es un stub, generado por el compilador `idl2java`, que recopila todos los datos necesarios para la solicitud y los envía al objeto servidor.

- 4** Muestra el saldo.

En el siguiente paso se implementa el servidor.

## Paso 5: Implementación del servidor

---

En este paso se muestra la forma de implementar el servidor. Inicializar el ORB, crear el POA (Portable Object Adapter - adaptador portátil de objetos), el objeto servidor `Account Manager`, activar el gestor de POA y el POA, y prepararse para recibir solicitudes. Muchos de los archivos que se utilizan para implementar el servidor de Bank se encuentran en el paquete `BankTutorial`, que se genera cuando se compila el archivo IDL.

Para crear este servidor:

- 1** Seleccione Archivo | Nuevo y, a continuación, Aplicación servidor CORBA en la ficha CORBA de la galería de objetos. Pulse Aceptar para mostrar el Asistente de aplicación de servidor CORBA.
- 2** Asegúrese de que `BankTutorial.idl` está seleccionado en el campo Archivo IDL.
- 3** Elija Generar aplicación visible con monitor para crear un monitor de servidor.

- 4 Haga clic en Siguiente para avanzar al paso siguiente del asistente.
- 5 Asegúrese de que la opción Crear una configuración de ejecución está activada.
- 6 Pulse Finalizar para crear el servidor y cerrar el asistente.

Se generan el paquete `banktutorial.Bank.server` y el archivo `BankServerApp.java`. El archivo `BankAppGenFileList.html`, también se añade al proyecto, y contiene la lista de los archivos generados.

El programa servidor, llamado `BankServerApp.java`, hace lo siguiente:

- Inicializa el ORB (Gestor de solicitud de objetos).
- Crea un adaptador portátil de objetos (POA) con las características necesarias.
- Crea el objeto servidor Account Manager.
- Activa el objeto servidor.
- Activa el administrador de POA (y el POA).
- Espera solicitudes entrantes.

Si desea ver una descripción de los archivos generados, consulte la documentación de Borland Enterprise Server.

## Definición de POA

---

El POA (Portable Object Adaptor, adaptador portátil de objetos) intercepta la solicitud del cliente e identifica el objeto que la cumple. A continuación se llama al objeto, y la respuesta se devuelve al cliente. El POA sustituye al BOA (Basic Object Adaptor, adaptador básico de objetos) y se desarrolló para ofrecer portabilidad en el servidor.

El POA sirve para:

- Permitir a los programadores crear implementaciones de objetos intercambiables entre los distintos productos de ORB.
- Proporcionar apoyo a los objetos con identidad persistente.
- Proporcionar apoyo para la activación transparente de objetos.
- Permitir a un servidor aceptar simultáneamente varias identidades de objetos.
- Permitir la coexistencia de varias instancias del POA en un servidor.
- Proporcionar apoyo para los objetos transitorios, reduciendo al mínimo la programación y el proceso.
- Proporcionar apoyo para la activación implícita de servidores con identificadores de objetos asignados por el POA.

- Permitir que las implementaciones de objetos sean las responsables del comportamiento de éstos.
- Evitar la necesidad de que el ORB mantenga un estado persistente para describir los objetos y sus identidades, si su estado se ha almacenado, si se han utilizado previamente determinados valores de identidad, si los objetos han dejado de existir, etc.
- Proporcionar un mecanismo ampliable para la asociación de información sobre los criterios de los objetos implementados en el POA.
- Permitir que los programadores construyan implementaciones de objetos herederas de clases de esqueleto, generadas por compiladores OMG IDL, o implementaciones DSI.

Si desea más información sobre el POA consulte la documentación de Borland Enterprise Server.

## Paso 6: Cómo obtener una implementación para la interfaz CORBA

---

Ahora, la aplicación contiene una interfaz de usuario que va a crear objetos Account con un saldo de cero. Para dar más interés a este ejemplo, podríamos añadir una base de datos de clientes, números de cuenta y saldos, y consultarlas para anotar en el debe o haber de la cuenta los depósitos o retiradas de fondos. Sin embargo, en este ejemplo sencillo únicamente añadiremos una implementación que genera el saldo asociado a un determinado nombre a partir del número de caracteres de ese nombre.

Para implementar la interfaz de CORBA:

- 1 Amplíe el objeto banktutorial.Bank.server del panel de proyecto para que queden visibles todos los archivos.
- 2 Haga doble clic en el archivo AccountImpl.java para abrirlo en el panel de contenido.
- 3 Busque el método balance() (puede pulsar *Ctrl+F*). Sustituya return 0 por

```
return _name.length()*100;
```
- 4 Elija Archivo | Guardar todo.

En el siguiente paso se compila la aplicación por medio del comando Proyecto | Ejecutar Make.

## Paso 7: Compilación de la aplicación

---

Para compilar el proyecto, elija Proyecto | Ejecutar Make del proyecto "BankTutorial.jpx".

En el panel de mensajes se mostrarán los errores que se detecten. Corrija los errores de sintaxis antes de pasar al siguiente paso.

### Antes de la compilación

Si se ha configurado el Agente inteligente de forma que se ejecute en un puerto distinto del 14000, es necesario introducir el siguiente parámetro de en el campo Parámetros de MV de la pestaña Aplicaciones de la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto):

```
-Dvbroker.agent.port=port number
```

## Paso 8: Ejecución de la aplicación Java

---

Para ejecutar la aplicación del tutorial:

- 1 Inicie el agente inteligente del ORB de VisiBroker.
- 2 Ponga en marcha la implementación del servidor.
- 3 Ejecute la aplicación cliente.
- 4 Pruebe y publique la aplicación.

Estos pasos se tratan más detalladamente en los apartados siguientes.

### Inicio del agente inteligente del ORB de VisiBroker.

---

El Agente inteligente del ORB de Visibroker, u OsAgent, proporciona un servicio de localización de objetos. El cliente encuentra el servidor por medio de OsAgent, que también permite al servidor anunciar sus servicios; así es como se encuentran mutuamente. Normalmente, es conveniente que el agente inteligente del ORB de Visibroker se ejecute por lo menos en un servidor de la red local. El agente inteligente del ORB de VisiBroker se describe en la documentación de Borland Enterprise Server.

Para iniciar el agente inteligente del ORB de Visibroker, elija Herramientas | Agente inteligente de VisiBroker. El agente inteligente del ORB de Visibroker se activa y desactiva desde el menú Herramientas.

Si tiene Windows NT y desea iniciar el agente inteligente como servicio de NT, debe registrar los servicios ORB como servicios NT durante la instalación. Si desea instrucciones, consulte la guía *Instalación de Borland Enterprise Server*. Si se han registrado los servicios, se puede iniciar el

agente inteligente del ORB de Visibroker como servicio NT desde el panel de control de servicios, sin necesidad de utilizar el menú Herramientas.

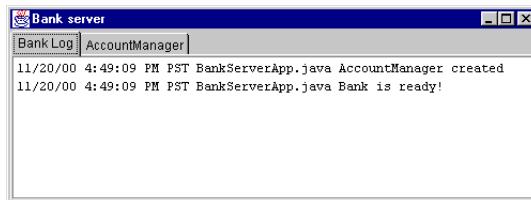
Si lo prefiere, puede ejecutar el agente inteligente del ORB deVisiBroker desde la línea de comandos, ejecutando `osagent.exe` desde el directorio `bin` de la instalación de Borland Enterprise Server.

## Inicio del servidor

---

Para iniciar la implementación del servidor, haga clic con el botón derecho del ratón en el archivo `BankServerApp.java`, en el panel de proyecto.

Seleccione Ejecutar. En la lista desplegable, seleccione Utilizar “Aplicación servidor”. Se abre la ventana de servidor Bank. Tiene el siguiente aspecto:



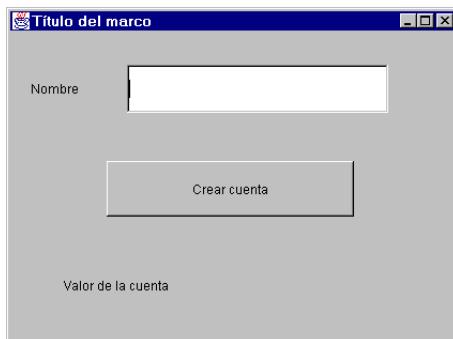
La ejecución de `BankServerApp.java` llama a la máquina virtual y ofrece otras funciones especiales. Si desea ver una lista de las opciones de línea de comandos y más información consulte la documentación de Borland Enterprise Server.

## Ejecución del cliente

---

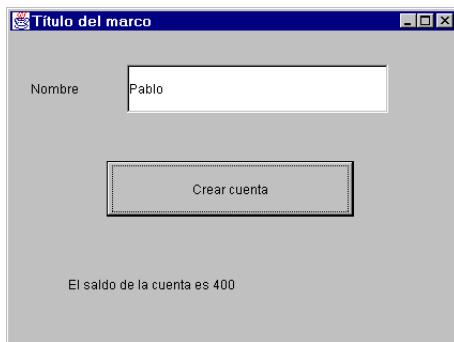
Para ejecutar el cliente Java, haga clic con el botón derecho del ratón en `Aplicación1.java`, en el panel de proyecto. Seleccione Ejecutar. En el menú desplegable, seleccione Utilizar “Aplicación1”.

La aplicación en ejecución presentará este aspecto:



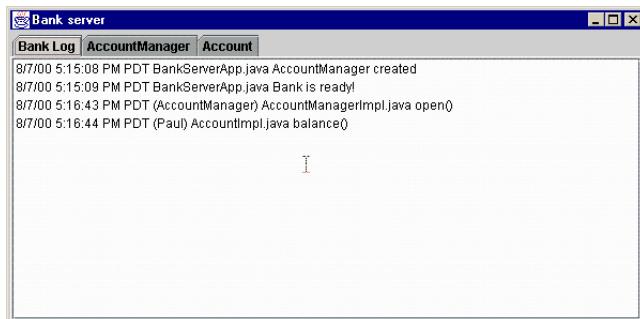
## Paso 8: Ejecución de la aplicación Java

Escriba un nombre cualquiera en el campo de texto y pulse el botón Crear cuenta. El saldo generado aparece en la etiqueta, como se muestra a continuación:



El saldo de la cuenta se calcula sumando el número de caracteres del nombre y multiplicándolo por 100.

El monitor de servidor imprime un mensaje cada vez que crea un objeto Account. En el lado del servidor aparecerá lo siguiente:



La ficha AccountManager muestra el número de objetos AccountManager que se han creado. La interfaz crea una cuenta para el usuario si no existe aún. En este ejemplo sólo se ha creado un objeto de interfaz AccountManager. La ficha Account muestra el número de objetos Account que se han creado. Siempre que se llama al método balance() se crea un objeto Account.

## Distribución de la aplicación

El ORB de VisiBroker se utiliza también en la fase de distribución. Esta fase se alcanza cuando un desarrollador ha creado programas cliente o aplicaciones servidor, las ha probado y las tiene preparadas para trabajar con ellas. En esta etapa, el administrador del sistema puede publicar los programas cliente en los sistemas de los usuarios finales o en las aplicaciones servidor de los ordenadores utilizados como servidores.

Para la distribución, el ORB de VisiBroker acepta programas cliente en la interfaz del cliente. Es necesario instalar el ORB en todos los ordenadores que ejecutan el programa cliente. Los clientes que utilizan el ORB en el mismo anfitrión, comparten el ORB. El ORB de VisiBroker acepta también varias aplicaciones de nivel intermedio. Es necesario instalar el ORB en todos los aparatos que ejecutan la aplicación servidor. Las aplicaciones u objetos de servidor que utilizan el ORB en el mismo equipo servidor, comparten el ORB. Los clientes pueden ser GUI, applets y programas clientes. Las aplicaciones de servidor contienen la lógica empresarial del nivel intermedio.

JBuilder incluye una sola LICENCIA DE DESARROLLO para Borland Enterprise Server. Esta licencia se debe utilizar en combinación con JBuilder. Si intervienen más programadores en el proyecto, cada uno de ellos deberá tener su propia licencia de desarrollo con Borland Enterprise Server. Los programas distribuidos deben tener una licencia de distribución para cada ordenador servidor.

## Otras aplicaciones de ejemplos

---

Borland Enterprise Server incluye aplicaciones CORBA de ejemplo, en el subdirectorio `examples` del directorio de instalación de Borland Enterprise Server. En los ejemplos se incorporan las funciones descritas en la documentación de Borland Enterprise Server.



# A

## Sugerencias acerca de los servidores Borland

La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

Este documento pretende reunir la información sobre el uso de servidores Borland con JBuilder en un solo sitio. Mucha de esta misma información se encuentra dispersa por otros capítulos de la *Guía del desarrollador de Enterprise JavaBeans*. Deberá estudiar esos capítulos para aprender a utilizar JBuilder con el fin de crear y distribuir los enterprise beans.

### Configuración y uso de Borland Enterprise Server 5.0.2 -5.1.x

En este apartado se trata la configuración de Borland Enterprise Server 5.0.2 – 5.1.x para utilizarlo con JBuilder.

#### Particiones, servicios de partición y API J2EE

Las particiones le proporcionan contenedores y servicios para alojar sus aplicaciones. Todos los contenedores de una partición pueden ser activados o desactivados. Un servidor puede tener muchas particiones, y puesto que cada una es un proceso distinto, las funciones de una aplicación se pueden distribuir entre múltiples procesos (particiones).

Adicionalmente, una partición puede ser clonada (copiada) en el mismo servidor o en otro servidor enterprise localizado en la misma red de área local. La clonación de una partición puede ahorrar cantidades importantes de tiempo en el proceso de distribución, debido a que no sólo copia la totalidad de los módulos distribuidos, sino también los parámetros de configuración de todos los servicios asociados, que pueden requerir un tiempo y esfuerzos considerables para completarse.

Los componentes de la aplicación se alojan o bien en el contenedor web o en el contenedor EJB, o simplemente en la propia partición (en el caso de los conectores). Puede distribuir archivos EAR de aplicación o recopilatorios más pequeños a las particiones. Las particiones son “inteligentes” y colocarán los componentes de su aplicación en los contenedores adecuados. Puesto que puede crear tantas particiones como desee, también puede tener tantas instancias de contenedor como desee. Borland proporciona hasta dos contenedores distintos por cada instancia de partición:

- Contenedor Web (Tomcat 4.0): para alojar JSP y servlets, y
- Contenedor EJB (Borland): para alojar componentes EJB

Cada partición proporciona además Servicios de partición y API J2EE como Servicios de partición. Estos servicios y API son:

- Servicio de denominación: en Borland Enterprise Server, el Servicio de denominación es gestionado por el ORB VisiBroker.
- JDataStore: la base de datos de Borland 100% Java.
- JDBC: para conectarse con bases de datos y modelar sus datos.
- Java Mail: un servicio de e-mail Java.
- JTA: la API transaccional de Java.
- JAXP: la API de Java para analizar XML.
- JNDI: la interfaz de Denominación y Directorio de Java.
- RMI-IIOP: llamada a métodos remotos (RMI) realizada mediante protocolo Internet inter-ORB (IIOP).

El agente de administración gestiona todas las particiones. El puerto por defecto para administración es 42424. Por defecto, el servidor se instala con dos particiones: “standard” y “webservices”.

## Archivos de propiedades del servidor

---

Los archivos de configuración por defecto se guardan en `RAÍZ_DEL_PPSERVER_HOME\var\servers\defaults\adm\properties`. Las particiones creadas en JBuilder utilizan la configuración de los archivos de propiedades por defecto. Para asegurarse de que las particiones creadas en JBuilder utilizan las configuraciones del servidor necesarias, modifique los archivos de propiedades en el directorio de valores por defecto.

- Los archivos de propiedades para configurar su servidor se almacenan en `RAÍZ_DEL_APPSERVER\var\servers\NOMBRE_SERVIDOR\adm\properties`.

- Los archivos de propiedades de partición se almacenan en RAÍZ\_DEL\_APPSERVER\var\servers\NOMBRE\_SERVIDOR\adm\properties\partition\NOMBRE\_PARTICIÓN.

A continuación se relacionan los archivos de propiedades utilizados con más frecuencia:

- RAÍZ\_DEL\_APPSERVER\var\servers\NOMBRE\_SERVIDOR\adm\properties\management\_vbroker.properties: **propiedades del Agente de administración**
- RAÍZ\_DEL\_APPSERVER\var\servers\NOMBRE\_SERVIDOR\adm\properties\server.config: **configuración del servidor**
- RAÍZ\_DEL\_APPSERVER\var\servers\NOMBRE\_SERVIDOR\adm\properties\partition\NOMBRE\_PARTICIÓN\partition.properties: **propiedades de partición**
- RAÍZ\_DEL\_APPSERVER\var\servers\NOMBRE\_SERVIDOR\adm\properties\partition\NOMBRE\_PARTICIÓN\tomcat\conf\server.xml: **propiedades del servicio Tomcat**
- RAÍZ\_DEL\_APPSERVER\var\servers\NOMBRE\_SERVIDOR\adm\properties\partition\NOMBRE\_PARTICIÓN\services\jdatastore.properties: **propiedades del servicio JDataStore**

## Cambio del puerto de administración

---

- 1 En RAÍZ\_DEL\_APPSERVER\bin\iastool.config, cambie vmprop default.mgmtport.
- 2 En RAÍZ\_DEL\_APPSERVER\var\servers\NOMBRE\_SERVIDOR\adm\properties\management\_vbroker.properties, cambie vbroker.agent.port.
- 3 En RAÍZ\_DEL\_APPSERVER\var\servers\NOMBRE\_SERVIDOR\adm\properties\partition\NOMBRE\_PARTICIÓN\management\_vbroker.properties, cambie vbroker.agent.port.

**Nota** Esto debe hacerse con todas las particiones que vayan a utilizarse en JBuilder.

## Configuración de la seguridad del agente de administración

---

Ponga vbroker.security.disable a false en RAÍZ\_DEL\_APPSERVER\var\servers\ SERVER\_NAME\adm\properties\management\_vbroker.properties.

Por defecto, este parámetro no está incluido en el archivo de propiedades y debe ser añadido a mano.

## Configuración de la seguridad de la partición

---

Ponga `vbroker.security.disable` a `false` en `RAÍZ_APPSERVER\var\servers\SERVER_NAME\adm\properties\partition\PARTITION_NAME\vbroker.properties`.

## Configuración de la política del cargador de clases para la partición

---

Cada partición carga recopilatorios dependiendo de la configuración de `container.classloader.policy` en el archivo de propiedades `partition.properties` en `RAÍZ_APPSERVER\var\servers\SERVER_NAME\adm\properties\partition\NOMBRE_PARTICIÓN`. Por defecto, está configurado como `per_module`. Con este valor, la partición carga cada recopilatorio utilizando un nuevo cargador de clases. En este caso no se pueden compartir recursos entre los recopilatorios. Si su aplicación tiene recopilatorios individuales con dependencias, todos los recopilatorios dependientes deben empaquetarse en un EAR para asegurarse de que los recursos comunes estén disponibles. Si no desea distribuir sus aplicaciones en un EAR, puede cambiar este valor a 'container' lo que asegurará que la partición cargará todos los recopilatorios utilizando un solo cargador de clases.

## Configuración de la distribución de WAR expandidos para la partición

---

Ponga `container.auto.expand.wars=true` en `RAÍZ_APPSERVER\var\servers\NOMBRE_SERVIDOR\adm\properties\partition\NOMBRE_PARTICIÓN\partition.properties` para activar la distribución de WAR expandidos.

## Asuntos internacionales

---

Borland Enterprise Server 5.0.2 – 5.1.x no es compatible con los caracteres japoneses del nombre del JNDI.

# Borland Enterprise Server como servidor web de aplicaciones con JBuilder

## Configuración del servidor

Borland Enterprise Server 5.0.2– 5.1.x tiene que instalarse y configurarse utilizando Herramientas | Configurar Servidores. El servidor, los nombres de particiones y el puerto del agente de administración pueden configurarse en la ficha Propiedades personalizadas. El puerto de administración para un servidor local se leerá y configurará en JBuilder cuando configure el directorio raíz del servidor. El puerto de administración se utiliza en JBuilder para detectar el servidor durante el inicio y la distribución. Puede cambiar el puerto de administración haciendo clic en el botón Configuración avanzada de la ficha Propiedades personalizadas. Esto sólo debería utilizarse si se está distribuyendo a un servidor remoto con un puerto de administración diferente al puerto por defecto. Cuando cambie el número de puerto asegúrese de que el número que ha introducido es el correcto según está configurado en el servidor. El servidor no se iniciará sin el puerto de administración correcto. Toda la comunicación cliente-servidor se realiza mediante el puerto del agente inteligente. Este es el puerto que utiliza el agente inteligente CORBA para la comunicación. Puede configurarse en la pestaña CORBA del cuadro de diálogo Configurar Enterprise accesible desde el menú Herramientas.

JBuilder, por defecto, utiliza una partición llamada 'jbuilder' como destino de distribución para Borland Enterprise Server 5.0.2/5.1.x. Si esta partición no existe, se creará una. Esta partición es una copia de la partición 'standard' y compartirá el mismo número de puerto para los servicios Tomcat y JDatastore. JBuilder le permite iniciar múltiples particiones utilizando configuraciones de ejecución. Para hacer esto, cree múltiples configuraciones de ejecución de servidor como se explica a continuación:

- 1 Seleccione Ejecutar | Configuraciones.
- 2 Haga clic en Nuevo.
- 3 Pulse la pestaña Servidor.
- 4 Modifique los parámetros de Servidor y cambie el nombre de la partición por el que desee utilizar.
- 5 Repita los pasos anteriores para crear una segunda configuración de ejecución para ejecutar otra partición.

**Nota** Asegúrese de que ha configurado números de puerto distintos para los servicios Tomcat y JDataStore del servidor antes de iniciar la partición.

Asegúrese también de que tiene activado el servicio de denominación sólo para una de las particiones, con el fin de evitar conflictos con el servicio de denominación. Para hacer esto, modifique la configuración de ejecución de la partición y desactive el servicio de denominación.

## Agente de gestión

---

El agente de administración debe iniciarse antes de que se lleve a cabo cualquier acción relacionada con el servidor (inicio del servidor, distribución, etc.). Puede iniciarse desde Herramientas | Agente de administración de Borland Enterprise Server.

## Inicio del servidor

---

Inicie el agente de administración utilizando Herramientas | Agente de administración de Borland Enterprise Server. Esto iniciará un proceso de servidor (**ias**), el agente de administración y el agente CORBA. Después de que el agente de administración se haya iniciado, inicie la partición haciendo clic con el botón derecho en el módulo EJB/Web de la vista de proyecto y seleccione Ejecutar usando valores por defecto. También puede crear una configuración de ejecución Servidor y ejecutar el proyecto utilizándola. La ejecución de la partición de JBuilder abarca las siguientes cosas:

- 1** Crea la partición, si no está presente en el servidor. El nombre de partición utilizado en este paso se deriva del de la configuración de ejecución utilizada para iniciar el servidor. Si está iniciando el servidor utilizando la configuración por defecto, se utilizará el nombre de la partición según esté configurado en las propiedades del servidor de aplicaciones.
- 2** Distribuye los recursos definidos en `jndi-definitions.xml` (si existe) a la raíz del directorio de la partición p.e. `RAÍZ_DEL_APPSERVER\var\servers\NOMBRE_SERVIDOR\partitions\NOMBRE_PARTICIÓN\`.

Este archivo se creará si su proyecto contiene un módulo EJB 2.0 con recursos de fuentes de datos/mensajes definidos.

**Nota**

Esta acción puede desactivarse desmarcando Distribuir `jndi-definitions.xml` en las propiedades del servicio de Distribución en la ficha Servidor del cuadro de diálogo Propiedades de proyecto.

- 3** Copia las bibliotecas seleccionadas al directorio `lib` de la partición. Todos los recopilatorios de este directorio se añadirán a la vía de acceso a clases de la partición cuando ésta se inicie.

El directorio del servidor es RAÍZ\_DEL\_APPSERVER\var\servers\NOMBRE\_SERVIDOR\partitions\NOMBRE\_PARTICIÓN\lib. Para poder seleccionar las bibliotecas que desea que se copien a la partición, haga clic en la sección Bibliotecas de la configuración de ejecución del servidor.

- 4 Elimina todos los recopilatorios distribuidos a la partición, si esta opción está seleccionada. Esto se puede configurar en la sección Recopilatorios en la configuración de ejecución del servidor. Seleccione la opción "Eliminar recopilatorios ya distribuidos al servidor", si desea comenzar con una partición limpia.
- 5 Distribuye recopilatorios si se han seleccionado. Por defecto, están seleccionados todos los recopilatorios distribuibles del proyecto. Para cambiar esto, haga clic en la sección Recopilatorios de la configuración de ejecución del servidor y desmarque los recopilatorios que no desea distribuir.
- 6 Inicie la partición. Cuando el inicio de la partición se haya completado, verá el mensaje 'La partición <NOMBRE\_PARTICIÓN> ha sido iniciada'. Los recopilatorios incluidos en el inicio se cargarán y estarán accesibles.

**Nota** Por defecto, se inician todos los servicios asociados con una partición. Esto podría producir que la partición tarde un mayor tiempo en iniciarse. Para reducir el tiempo de inicio, desmarque los servicios que no sean necesarios en la configuración de ejecución Servidor.

## Distribución

---

Cuando se utiliza JBuilder hay dos opciones de distribución disponibles:

- Distribuir al servidor utilizando el asistente de distribución: Herramientas | Distribución Enterprise. Ésta es una herramienta de Borland Enterprise Server y requiere que esté iniciado el agente de administración. El asistente detectará todas las particiones del servidor, incluso las que no han sido iniciadas. Seleccione la partición adecuada en la lista desplegable y haga clic en 'Finalizar' para distribuir el recopilatorio. Si la partición ya ha sido iniciada, reiníciela para acceder al recopilatorio distribuido. Este asistente sólo puede utilizarse para la acción 'Distribuir'. El asistente puede utilizarse para distribuir a servidores remotos. Para configurar las propiedades del servidor remoto, utilice Herramientas | Configurar servidores e indique el servidor, los nombres de las particiones y el número de puerto de administración del servidor remoto.
- Distribuir al servidor utilizando la opción del menú contextual de todos los nodos distribuibles. JBuilder reconoce a los módulos EJB, WAR y EAR como nodos distribuibles. Esta acción necesita que el agente de administración esté iniciado. La partición será reiniciada después de todas las acciones de distribución (distintas a 'list'). Esta opción puede

utilizarse para distribuir a un servidor remoto. El servidor, los nombres de partición y los números de puerto de administración pueden indicarse utilizando la ficha de propiedades de distribución de un módulo EJB. Se puede acceder a ella haciendo clic con el botón derecho en el módulo en la vista de proyecto y seleccionando 'Propiedades'.

## Ejecución del cliente de prueba

---

Puede crear un cliente de prueba utilizando el Asistente para Clientes de prueba EJB de la pestaña Enterprise de la Galería de objetos. Antes de ejecutar el cliente de prueba, cree una configuración de ejecución de aplicación y ponga el parámetro de MV - Dvroker.agent.port=<PUERTO\_AGENTE\_INTELIGENTE>. La comunicación cliente-servidor no funcionará sin este parámetro.

## Soluciones a problemas de aplicaciones web

---

La instalación por defecto del servidor de aplicaciones incluye ROOT.war, que contiene el contexto por defecto. Si distribuye un EAR que contiene un contexto por defecto, debe eliminar ROOT.war (o cambiar su extensión) para que no se cargue y cause un conflicto. Si se distribuyen WAR (con la política del cargador de clases del contenedor), el WAR resultante se copia automáticamente a la partición como ROOT.war, en cuyo caso no hay conflicto. Tenga en cuenta que es el parámetro <context-root>!ROOT!</context-root> en el archivo web-borland.xml file el que define al contexto como raíz, no el nombre del archivo.

Si está trabajando en un contexto con nombre pero tiene el contexto raíz en un recopilatorio ya distribuido, el navegador web interno de JBuilder intentará cargarse una vez que el recopilatorio haya sido cargado por el contenedor, porque el contexto raíz puede, potencialmente, coincidir con cualquier URL. Probablemente recibirá un mensaje de error 'Documento no hallado'. Para evitar esto, elimine ROOT.war si no lo está utilizando o sobre escribiendo.

Cuando crea una configuración de ejecución de servidor para iniciar el servidor de aplicaciones en JBuilder, la configuración creada no aparece en el menú contextual del jar EJB. Durante el inicio la distribución no está disponible para este servidor de aplicaciones. Utilice *F9* para iniciar la aplicación.

## La consola de Borland Enterprise Server con JBuilder

La consola de Borland Enterprise Server detectará un servidor (agente de administración) iniciado en JBuilder, pero no detectará particiones iniciadas en JBuilder. Para utilizar la consola, apague el agente de administración y la partición en JBuilder e inicie el servidor fuera de JBuilder. A continuación puede iniciar la partición utilizando la consola.

## Configuración y uso de Borland AppServer 4.5.1

JBuilder 8 admite Borland AppServer 4.5 como servidor de aplicaciones para el desarrollo de proyectos. Tan sólo son necesarios algunos cambios manuales para un funcionamiento correcto:

- 1 En general, el desarrollo con BAS 4.5 será más limpio y fluido si no configura Borland Enterprise Server 5.0.2/5.1.x.
- 2 Si tiene configurado Borland Enterprise Server 5.0.2/5.1.x , necesita hacer lo siguiente para asegurarse de que las clases de BAS 4.5 se utilizan donde es adecuado:
  - a Por defecto, el ORB de Borland Enterprise Server 5.0.2/5.1.x estará definido como el que tiene que utilizar JBuilder. Seleccione Herramientas | Configurar Enterprise | CORBA | Modificar para modificar Visibroker, y, a continuación, modifique la vía de acceso de las herramientas ORB de modo que sea el directorio `home\bin` de Borland AppServer4.5 y la biblioteca para que sea la biblioteca de cliente de Borland AppServer 4.5. Asegúrese de eliminar todas las clases y archivos generados, y genere de nuevo su proyecto si anteriormente el AppServer configurado para él era Borland Enterprise Server 5.0.2/5.1.x.

O
  - b Cree una configuración de ORB para el ORB Borland AppServer 4.5. Seleccione Herramientas | Configurar Enterprise | CORBA | "Nueva" configuración, y configure la vía de acceso a las herramientas ORB para que sea el directorio `home\bin` de Borland AppServer4.5 y la biblioteca para que sea la biblioteca de cliente de Borland AppServer 4.5. Configure el comando del compilador IDL como '`idl2java`' y la opción del comando para el directorio de salida como '`-root_dir`'. En la pestaña CORBA marque "Aplicar esta configuración al proyecto actual", desmarque todas las demás, y pulse Aceptar. Ahora, cuando cambie a un proyecto que utilice Borland Enterprise Server 5.0.2/5.1.x, seleccione la configuración por defecto de Visibroker y en la pestaña CORBA marque "Aplicar esta configuración al proyecto actual", desmarque todas las demás y pulse Aceptar.

**Nota** Asegúrese de que su proyecto sólo contiene referencias a bibliotecas de Borland AppServer 4.5 (seleccione Proyecto | Propiedades | pestaña Vías de acceso | Bibliotecas necesarias); deben eliminarse todas las referencias a bibliotecas Borland Enterprise Server 5.0.2/5.1.x, o tendrá conflictos de clases cuando genere o distribuya.

- 3** Los asistentes de Distribución y de Creación de clientes jar (Herramientas | Distribución de EJB) no funcionarán para distribuir a Borland AppServer 4.5, por lo que estos menús están deshabilitados. Puede distribuir a Borland AppServer 4.5 utilizando los menús de distribución accesibles con el botón derecho en los módulos EJB y en los grupos EAR o en la consola de Borland AppServer 4.5.

Solución:

- a** Cambie las siguientes bibliotecas de forma que apunten a los jar instalados con Borland AppServer 4.5.1:
  - Servidor de JDataStore
  - Data Express
- b** Elimine el directorio de trabajo AppServer en el directorio de proyecto.

**Advertencia**

No intente utilizar JDataStore 5 con Borland AppServer 4.5.1. Esta configuración no se ha comprobado.

# B

## Sugerencias acerca de WebLogic Server

Este documento pretende reunir la información sobre el uso de servidores WebLogic con JBuilder en un solo sitio. Esta misma información se encuentra dispersa por otros capítulos de la *Guía del desarrollador de Enterprise JavaBeans*. Deberá estudiar esos capítulos para aprender a utilizar JBuilder con el fin de crear y distribuir los enterprise beans.

JBuilder es compatible con WebLogic Server 5.1, 6.x y 7.x.

### Configuración de JBuilder

Cuando se configura JBuilder para utilizar el servidor WebLogic, se crean automáticamente las bibliotecas necesarias. Si desea ver una lista de las bibliotecas de WebLogic, consulte “[Las bibliotecas generadas](#)” en la página 5-6.

### WebLogic 7.x

WebLogic 7.x se instala en una estructura de directorios diferente a la de versiones anteriores, pero todavía utiliza un directorio RAÍZ\_DE\_BEAS. Si instaló WebLogic 7.x en un directorio RAÍZ\_DE\_BEAS, el directorio de instalación debería tener este aspecto: [drive]:/bea/weblogic700/server.

Bajo este directorio, el directorio raíz real del servidor es server.

- El directorio bin es server/bin.
- El directorio lib es server/lib.

Si se utiliza el Kit de servicios web para Java y se producen errores de tipo `java.lang.VerifyError` cuando se distribuye la aplicación Axis, la vía de acceso a clases para el inicio del servidor debe modificarse como sigue:

- 1 Seleccione Herramientas | Configurar servidores.
- 2 Seleccione WebLogic Application Server 7.x a la izquierda.
- 3 En la pestaña Clases de la derecha, añada `JB_HOME/lib/jaxrpc.jar` y mueva esta entrada al principio de la lista.

## Configuración de WebLogic 7.x en JBuilder 8

Seleccione WebLogic Application Server 7.x en el panel a la izquierda y especifique el directorio raíz. Si desea más información pulse el botón Ayuda del cuadro de diálogo.

Pulse la pestaña Personalizar y el botón Ayuda para llenar los campos de la ficha.

Cuando crea una configuración de ejecución de servidor para iniciar el servidor de aplicaciones en JBuilder, la configuración creada no aparece en el menú contextual del jar EJB. Durante el inicio la distribución no está disponible para este servidor de aplicaciones. Utilice Ejecutar | Ejecutar proyecto (*F9*) para iniciar la aplicación.

## WebLogic 6.x

---

JBuilder admite EJB 1.1/EJB 2.0 para WebLogic Server 6.x.

JBuilder necesita WebLogic 6.0 con Service Pack 2 o WebLogic 6.1 con Service Pack 3 para su correcto funcionamiento.

Para configurar JBuilder para que utilice WebLogic 6.x, seleccione Herramientas | Configurar servidores y en el panel, a la izquierda, seleccione WebLogic Application Server 6.x. Utilice las fichas General y Personalizar para llenar los campos necesarios. Para obtener ayuda sobre cómo llenar estos campos, pulse el botón Ayuda del cuadro de diálogo cuando cualquiera de las dos fichas esté seleccionada.

Para asegurarse de que el servidor se inicia de modo apropiado, compruebe que los siguientes parámetros de MV están configurados correctamente:

- `-Dweblogic.Domain` donde *Domain* apunta al nombre de dominio de WebLogic 6.x (el nombre por defecto es "mydomain")
- `bea.home` le dirige al directorio correcto para el directorio inicial BEA (el directorio por defecto es el directorio superior del directorio inicial de WebLogic 6.x)

Para utilizar DataExpress para EJB con WebLogic, necesita añadir `weblogic.jar` y un archivo `jndi.properties` a la vía de acceso a clases

de JBuilder. De otra forma, las búsquedas JNDI del Diseñador de interfaces de usuario no funcionarán.

## WebLogic 5.1

---

JBuilder admite por defecto Service Pack 11.

Si ha instalado WebLogic Server 5.1, éste habrá creado una biblioteca cliente WebLogic 5.1 para su proyecto. Si esta biblioteca forma parte de la vía de acceso a clases de su proyecto, puede que aparezcan extraños errores del compilador al generar su EJB. El archivo `weblogicaux.jar` redefine algunas clases Java.

## Utilización de código ya creado

---

Si tiene descriptores de distribución para enterprise beans EJB 2.0 WebLogic creados previamente, puede crear un módulo EJB que los contenga. Existen dos posibilidades:

- Utilizar el Asistente para módulo EJB a partir de descriptores, que crea un módulo EJB que contiene los descriptores de distribución.
- Utilizar el asistente para Proyecto para código existente, que crea un proyecto de JBuilder con nuevos módulos EJB que contienen los descriptores de distribución.

Consulte “[Creación de módulos EJB a partir de descriptores de distribución](#)” en la página 6-4 para obtener más información.

## Creación de beans entidad WebLogic en JBuilder

---

Si desea crear un bean entidad EJB 1.1 con WebLogic 5.1, el Modelador de Bean entidad EJB 1.x de JBuilder incluye un campo Nombre de agrupación, donde se especifica el nombre de la agrupación para los beans de WebLogic con persistencia gestionada por contenedor.

El Diseñador de EJB de JBuilder permite utilizar características propias de WebLogic. El inspector de beans entidad incluye opciones como la concurrencia optimista y los grupos de campos. Existe una versión para WebLogic del Editor de mapa de tablas. Para crear relaciones entre beans entidad, se dispone también de una versión especial para WebLogic del inspector de relaciones, así como de un Editor de relaciones SGBDR. Para obtener más información acerca de estas herramientas, consulte “[Cómo añadir referencias de tabla de WebLogic](#)” en la página 7-13 y “[Especificación de una relación WebLogic 6.x o 7.x](#)” en la página 7-20.

## El editor de descriptor de distribución

---

El Editor de descriptor de distribución presenta varios paneles específicos de WebLogic con el fin de modificar descriptores de distribución de WebLogic. Si el servidor de aplicaciones seleccionado para el proyecto actual es WebLogic Server 6.0 o posterior, cuando el módulo EJB es el nodo actual, el Editor de descriptor de distribución muestra una ficha adicional: la ficha Propiedades de WebLogic 6.x o 7.x.

Cuando se selecciona un enterprise bean en el panel de proyecto, con WebLogic 6.x o 7.x como servidor de aplicaciones de destino, pueden aparecer otros paneles específicos de WebLogic. Estos paneles son: un panel General de WebLogic, un panel Propiedades de WebLogic, un panel Caché de WebLogic, un panel Aislamiento de transacciones de WebLogic y un panel Métodos idempotentes. Por último, también existe un panel Competencias de seguridad de WebLogic para WebLogic 5.1, 6.x o 7.x.

## Compilación

---

Si desea configurar opciones EJBC para compilar los beans WebLogic, pulse con el botón secundario del ratón sobre el nodo del módulo EJB, del panel de proyecto, seleccione Propiedades, seleccione la ficha Generar y, a continuación, la ficha WebLogic. Esta ficha permite especificar opciones EJBC.

## Distribuir

---

La opción de menú Herramientas | Distribución Enterprise abre el cuadro de diálogo Configuración de distribución de WebLogic. Configure las opciones de distribución con este cuadro de diálogo. JBuilder pasa los valores especificados a la herramienta de distribución de WebLogic.

Si el servidor de aplicaciones de destino es WebLogic 7.x, al pulsar con el botón secundario del ratón sobre el archivo JAR o el módulo EJB del panel de proyecto, no aparecen elementos de menú contextual para ejecutar o depurar. Durante el inicio, la distribución para WebLogic 7.x no está disponible.

# Depuración remota de Páginas JavaServer

---

Para depurar las JSP remotamente, abra el proyecto que contiene la aplicación web que desea depurar y sigua estos pasos:

- 1** Abra el nodo de la aplicación web en el panel de proyecto y abra a su vez el nodo de los descriptores de distribución que se encuentra debajo del nodo de la aplicación web.
- 2** Haga doble clic en `weblogic.xml` y añada los siguientes elementos descriptores:

```
<jsp-descriptor>
  <jsp-param>
    <param-name>keepgenerated</param-name>
    <param-value>true</param-value>
  </jsp-param>
  <jsp-param>
    <param-name>debug</param-name>
    <param-value>true</param-value>
  </jsp-param>
  <jsp-param>
    <param-name>precompile</param-name>
    <param-value>true</param-value>
  </jsp-param>
  <jsp-param>
    <param-name>compileFlags</param-name>
    <param-value>-g</param-value>
  </jsp-param>
</jsp-descriptor>
```

- 3** Vuelva a generar el proyecto y distribuya la aplicación web (como un WAR).
- 4** Lance el servidor Weblobic con los parámetros de depuración. Para hacer esto, modifique `startWebLogic.sh` en el directorio de dominio y añada los siguientes parámetros a la variable OPCIONES\_JAVA:

```
-Xdebug -Xnoagent -Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=n
```

Estos pasos pueden realizarse en un ordenador local provisto de Weblogic instalado y configurado en JBuilder.

En el ordenador local, sigua estos pasos:

- 1** Transfiera el proyecto al ordenador locan en el que deseé depurar la JSP y abra el proyecto en JBuilder.
- 2** Defina el servidor destino como Tomcat 4.0 o 3.3.
- 3** Bajo la vía de acceso a archivos fuente del proyecto (normalmente `<RAÍZ_DEL_PROYECTO>/src`), cree un directorio denominado `jsp_servlet`. Si necesita utilizar una vía de acceso a archivos fuente distinta de la vía de

acceso por defecto, puede realizar esta configuración en la ficha General del cuadro de diálogo Propiedades de proyecto.

- 4 Copie las JSP que desee depurar en el directorio que acaba de especificar.
- 5 Abra las JSP y defina los puntos de interrupción donde sea necesario.
- 6 Seleccione Ejecutar | Configuraciones.
- 7 Pulse Nuevo, después haga clic en la pestaña Aplicación y a continuación en la pestaña Depurar.
- 8 Marque la opción Activar depuración remota.
- 9 Introduzca el nombre del host del servidor remoto en el campo Nombre del anfitrión y asigne el valor Ninguno a Tipo de generación. Acepte todos los demás valores por defecto.
- 10 Vinculese con el servidor remoto pulsando en el icono Depurar proyecto de la barra de herramientas.
- 11 Cargue su JSP en un navegador web. El depurador se detiene en los puntos de interrupción de la JSP.

## Asuntos internacionales

---

Los beans entidad locales desarrollador para WebLogic Server 7.0 no pueden tener caracteres japoneses en sus nombres.



# Sugerencias acerca de WebSphere Application Server

La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

Este documento pretende reunir la información sobre el uso de servidores WebSphere con JBuilder en un solo sitio. Esta misma información se encuentra dispersa por otros capítulos de la *Guía del desarrollador de Enterprise JavaBeans*. Deberá estudiar esos capítulos para aprender a utilizar JBuilder con el fin de crear y distribuir los enterprise beans.

JBuilder es compatible con WebSphere 3.5, WebSphere 4.0 Single Server y WebSphere 4.0 Advanced Edition.

## Instalación de WebSphere y configuración de JBuilder

En el caso de WebSphere 3.5, debe instalar el WebSphere FixPack2 con el objeto de conseguir la versión más actualizada del JDK de IBM.

Cuando se configura JBuilder para utilizar un servidor WebSphere, se crean automáticamente las bibliotecas necesarias. Si desea ver una lista de las bibliotecas de WebSphere, consulte “[Las bibliotecas generadas](#)” en la página 5-6.

## WebSphere Server 3.5

JBuilder es compatible con WebSphere 3.5, que se puede utilizar para crear componentes EJB 1.0.

Debe utilizar para sus proyectos con WebSphere el JDK de IBM (JDK 1.2.2) que se distribuye con WebSphere. Al instalar el servidor de aplicaciones WebSphere, ese JDK se define automáticamente y se configura como el

JDK para el proyecto actual. El JDK para el proyecto también se configura en el JDK de IBM al cambiar los servidores de aplicaciones. Como este JDK no admite JDI, no se puede utilizar para depurar. WebSphere 3.5 FixPack 3 cuenta ahora con una corrección para el JDK que incluye soporte para JDI. El fix pack se puede descargar conectándose a la página de asistencia de IBM <http://www.ibm.com/support/us/>.

Una vez haya instalado el fix pack, copie los siguientes archivos de `JDK_HOME\jre\bin` a `JDK_HOME\bin`: `dt_socket.dll`, `dt_shmem.dll` y `jdwp.dll`. No podrá depurar si estos archivos no están en el `JDK_HOME\bin`. El depurador no se detiene en los puntos de interrupción del servidor. Esto es un fallo del JDK de IBM.

El Modelador de beans entidad EJB 1.x no admite persistencia gestionada por contenedores para los beans de WebSphere 3.5 ya que WebSphere utiliza un sistema de correspondencia propia para persistencia gestionada por contendor.

WebSphere 3.5 no admite la distribución en el inicio. Primero debe iniciar el servidor de aplicaciones y, a continuación, es necesario distribuir sus archivos jar. Debido a esta restricción, están desactivadas todas las opciones que tienen que ver con la selección de archivos jar de EJB para la distribución en el inicio del servidor de aplicaciones.

Para que la distribución funcione adecuadamente en sistemas Unix, antes de que JBuilder se inicie, la variable de sistema `PATH` debe modificarse para incluir (al principio de la vía de acceso) el directorio `<RAÍZ_DE_IBM_JDK>\bin`.

## Generación de descriptores de distribución de WebSphere

---

Cuando se utiliza el Asistente para Enterprise JavaBean 1.x o el Modelador de entidades EJB 1.x con el fin de crear beans entidad para WebSphere 4.0 Advanced Edition, los asistentes no generan los descriptores de distribución `Map.mapxmi` y `Schema.dbxmi`. Los genera más adelante `EjbDeploy`, durante el proceso de generación del bean. Después se puede modificar la asignación y volver a ejecutar el Make para conservar esta asignación en el JAR.

Si ha utilizado el Modelador de entidades para crear los beans entidad y los nombres de los campos del bean no se corresponden directamente con las columnas de la base de datos, puede hacer que JBuilder cree descriptores de distribución CMP de WebSphere que redefinan el comportamiento por defecto de `EjbDeploy`:

- 1** Si WebSphere 4.0 Advanced Edition es el servidor seleccionado, haga clic con el botón derecho del ratón sobre el nodo del módulo EJB que aparece en el panel de proyecto.
- 2** Seleccione Propiedades en el menú contextual.

- 3** Abra la pestaña WebSphere AE 4.0.
- 4** Active la casilla de selección Generar descriptores CMP.
- 5** Realice los cambios.

La lista Archivos de correspondencia de datos contiene los archivos de correspondencia utilizados para asignar tipos primitivos de Java a tipos de base de datos. Puede modificar, añadir y eliminar archivos de correspondencia. La lista Sustitución de tipo de base de datos en descriptores generados muestra la sustitución que JBuilder realiza si los tipos primitivos de Java no pueden asignarse directamente a un tipo en la base de datos. Puede modificar esta lista. Por ejemplo, podría sustituir otro tipo Java por un tipo original concreto en una base de datos. También puede añadir sustituciones adicionales a la lista.

- 6** Pulse Aceptar.

Cuando se compila el bean, se generan dos descriptores de distribución (`Map.mapxmi` y `Schema.dbxmi`) para los beans entidad con persistencia gestionada por el contenedor, sólo con la edición Advanced de WebSphere 4.0. Si utiliza WebSphere 4.0 y cambia el servidor de destino a la otra versión, es necesario volver a compilar el proyecto con el fin de garantizar que JBuilder genere los descriptores de distribución correctos.

Si el nombre de esquema utilizado para la base de datos es diferente del nombre de usuario, modifique manualmente el archivo `Schema.dbxmi` de la manera siguiente:

- 1** Vuelva a generar el proyecto con el fin de generar los descriptores CMP.
- 2** Haga doble clic en el nodo del módulo EJB en el panel de proyecto para abrir el visualizador de módulos EJB
- 3** Haga clic en el visor de código fuente de `Schema.dbxmi`
- 4** Añada la siguiente línea antes de la última del código fuente:

```
<RDBSchema:RDBSchema xmi:id="RDBSchema_1" name=<nombre del esquema>
database=<id-de la base de datos> tables=<id-de la tabla>/>
```

- 5** Vuelva generar el proyecto con el fin de incorporar el cambio.

Los beans para WebSphere 3.5 cumplen la especificación EJB 1.0. Estos componentes no utilizan descriptores de distribución XML. En su lugar, el archivo JAR generado contiene un archivo `.ser` para cada bean.

## Importación de descriptores de WebSphere 4.0 Advanced Edition a JBuilder

Para importar descriptores propietarios IBM a JBuilder:

- 1 Cree un proyecto.
- 2 Seleccione Proyecto | Propiedades de proyecto. En la ficha Servidor defina como servidor de aplicaciones de destino WebSphere 4.0 Advanced Edition.
- 3 Si no ha creado un módulo en JBuilder para sus EJB, hágalo.
- 4 Haga clic con el botón derecho sobre módulo EJB y seleccione Propiedades.
- 5 Utilice la ficha EJB para añadir descriptores propios al módulo.
- 6 Pulse el botón derecho sobre el archivo de proyecto y seleccione Añadir Archivos/Paquetes. Importe el código fuente del bean al proyecto.
- 7 Haga doble clic en el nodo del módulo EJB en el panel de proyecto para abrir el visualizador de módulos EJB
- 8 En JBuilder, los descriptores propietarios se regeneran en base a la marca cronológica del descriptor estándar (`ejb-jar.xml`). Pulse el archivo `ejb-jar.xml` de la pestaña Fuente del DD para cambiar la marca cronológica.
- 9 Vuelva a generar el proyecto.

## Distribución de enterprise beans en servidores WebSphere

Si utiliza WebSphere 4.0 Advanced Edition, deberá crear un grupo EAR que contenga los beans. Después, deberá distribuir el grupo EAR. Si desea obtener más información sobre la creación de grupos de EAR, consulte ["Creación de archivos EAR" en la página 12-6](#).

La opción de menú Herramientas | Distribución Enterprise abre el cuadro de diálogo Configuración de distribución de WebSphere. Configure las opciones de distribución con este cuadro de diálogo. JBuilder pasa los valores especificados a la herramienta de distribución de WebSphere. La herramienta de distribución de WebSphere 4.0 difiere según la versión del servidor utilizada. La herramienta de distribución de WebSphere Single Server es SEAppInstaller. La herramienta de distribución de Advanced Edition es SEAppInstaller. Por ello, el aspecto del cuadro de diálogo Configuración de distribución varía entre las dos ediciones de WebSphere Server 4.0.

Si su servidor de aplicaciones de destino es WebSphere 3.5 o WebSphere 4.0 Advanced Edition y desea que se genere un archivo XML como entrada en la utilidad WebSphere XMLConfig, marque la casilla Generar XML. Si esta opción no está marcada, el archivo no se creará. Si ha introducido modificaciones en el archivo XML que se ha creado (de nombre `deploy_<selectednode>.xml` y que aparece debajo del nodo del módulo EJB o del grupo EAR), no marque esta opción para asegurarse de que no pierde los cambios. Si se pulsa el módulo EJB con el botón derecho del ratón y elige Opciones de distribución de `<nombre del archivo jar>.jar` aparecen los comandos de distribución, con los que se puede generar un archivo XML llamado `deploy.xml`. Este archivo aparece bajo el nodo del proyecto.

Configure las opciones de distribución para servidores WebSphere mediante el cuadro de diálogo Propiedades. Consulte “[Definición de las opciones de distribución mediante el cuadro de diálogo Propiedades](#)” en la página 12-10 para obtener más información.

## El editor de descriptor de distribución

---

El Editor de descriptores de distribución presenta un panel Propiedades de WebSphere 4.0. Consulte “[Panel Propiedades específicas del servidor](#)” en la página 13-27 para obtener información sobre la utilización del panel.

En el panel WebSphere 4.0 Properties (Propiedades del WebSphere 4.0), los buscadores del bean aparecen al final de la lista de propiedades. Es posible elegir el tipo de consulta para la que se desea utilizar el buscador: SQL SELECT, SQL WHERE, EJB query language (Lenguaje de consultas EJB) o SQL Method (Método SQL). Seleccione el tipo de buscador deseado. El valor por defecto es una cláusula WHERE (SQL WHERE). El tipo de buscador seleccionado se añade al descriptor de distribución propio de WebSphere `ibm-ejb-jar-ext.xmi`. A continuación, se puede dirigir al panel Finders (Buscadores) y definir la búsqueda utilizando la consulta del tipo seleccionado en el panel WebSphere 4.0 Properties (Propiedades de WebSphere 4.0) como valor de la cláusula Where, aunque el tipo de consulta no sea una cláusula WHERE.

## Ejecución y distribución

---

Después de seleccionar Ejecutar o Depurar en el menú contextual, debe seleccionar un comando del menú Distribuir con el objeto de distribuir los beans. Consulte “[Distribución en tiempo real en un servidor de aplicaciones](#)” en la página 12-11.

## Activar la depuración de WebSphere 4.0 Advanced Edition

WebSphere Server 4.0 Advanced Edition 4.0 inicia otra máquina virtual de Java para depurar, por lo que debe seguir estas instrucciones con el objeto de depurar los enterprise beans, JSP o servlets:

- 1 Inicie el servidor WebSphere Server 4.0. Observe qué número de puerto utiliza el servidor.
- 2 Inicie aplicación de consola WebSphere Server 4.0 (que encontrará en <raíz de ws4\_ae>bin\adminclient.bat si lo está ejecutando desde Windows).
- 3 Cuando aparezca la aplicación de consola, amplíe el árbol de la izquierda hasta que pueda ver el nodo Servidores de aplicaciones. Seleccione el nodo.
- 4 En el panel que aparece a la derecha de la consola, abra la pestaña Configuración de la MVJ. Pulse el botón Configuración avanzada. (Quizás necesite desplazar el panel para ver el botón.)
- 5 Ien el cuadro de diálogo que se abre, introduzca las siguientes opciones de MV:

```
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdwp:transport=dt_socket,server=y,address=5000,suspend=n
```
- 6 Para iniciar el servidor dentro de JBuilder en modo depuración y enlazarlo a él, selección Ejecutar | Configuraciones y pulse el botón Nueva en el cuadro de diálogo que aparece. En el campo Configuración, especifique un nombre para el nuevo servidor de configuración de ejecución para poder depurar en el servidor de aplicaciones.
- 7 En la ficha Ejecutar, seleccione Servidor en la lista desplegable Tipo.
- 8 En el campo Dirección de transporte de depuración, introduzca el número que había anotado anteriormente durante el proceso de inicio del servidor.
- 9 Haga dos veces clic en Aceptar para cerrar los cuadros de diálogo.
- 10 Seleccione la configuración de ejecución que acaba de crear en la lista desplegable del ícono Depurar, en el menú de herramientas de JBuilder. El servidor se inicia el modo depuración y el proceso de depuración se enlaza a él. Ahora puede depurar el enterprise bean o la aplicación web y podrá parar en el punto de interrupción definido en el bean JSP o servlet.

# Activación de la depuración remota

---

Los pasos a seguir para activar la depuración remota de un bean que se ejecuta en un servidor WebSphere varían según la versión de WebSphere utilizada.

## WebSphere Server 3.5

---

Si va a utilizar WebSphere Server 3.5, siga estos pasos con el fin de activar la depuración remota en Windows:

- 1** Copie `dt_shem.dll` y `dt_socket.dll` de `RAIZ_WEBSPHERE/jdk/jre/bin` a `RAIZ_WEBSPHERE/jdk/bin`.
- 2** Modifique el script `adminserver` en `RAIZ_WEBSPHERE/bin/debug` y añada los siguientes parámetros de depuración remota en la línea de comandos Java:

```
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdwp:transport=dt_socket,server=y,address=5000,suspend=n
```

- 3** En JBuilder, seleccione Proyecto | Propiedades de proyecto y abra la pestaña Depurar.
- 4** Marque las opciones Activar depuración remota y Acoplarse, y pulse Aceptar.

Cuando esté preparado para depurar, acópelo al proceso remoto seleccionando Ejecutar | Depurar proyecto.

## WebSphere Single Server 4.0

---

Si está utilizando WebSphere Single Server 4.0, siga estas indicaciones para activar la depuración remota:

- 1** Inicie el servidor con la opción `-script`:
- ```
RAIZ_WEBSPHERE/bin/startserver -script
```
- Este comando escribe un script denominado `launch` en el directorio `RAIZ_WEBSPHERE/bin`.
- 2** Modifique este script y añada los siguientes parámetros de depuración remota a la línea de comandos Java:
- ```
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdwp:transport=dt_socket,server=y,address=5000,suspend=n
```
- 3** En JBuilder, seleccione Proyecto | Propiedades de proyecto. Haga clic en la pestaña Depurar.
  - 4** Marque las opciones Activar depuración remota y Acoplarse, y pulse Aceptar.

Cuando esté preparado para depurar, acópelo al proceso remoto seleccionando Ejecutar | Depurar proyecto.

## WebSphere Server Advanced Edition 4.0

---

WebSphere Server 4.0 Advanced Edition inicia otra máquina virtual de Java para depurar, por lo que debe seguir estas instrucciones con el objeto de depurar los enterprise beans:

- Nota** No podrá depurar JSP en una sesión de depuración remota.
- 1** Inicie el servidor WebSphere Server 4.0.
  - 2** Inicie aplicación de consola WebSphere Server 4.0 (que encontrará en <raíz\_de\_ws4\_ae>bin\adminclient.bat si lo está ejecutando desde Windows).
  - 3** Cuando aparezca la aplicación de consola, amplíe el árbol de la izquierda hasta que pueda ver el nodo Servidores de aplicaciones. Seleccione el nodo.
  - 4** En el panel que aparece a la derecha de la consola, abra la pestaña Configuración de la MVJ. Pulse el botón Configuración avanzada. (Quizás necesite desplazar el panel para ver el botón.)
  - 5** Ien el cuadro de diálogo que se abre, introduzca las siguientes opciones de MV:

```
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdwp:transport=dt_socket,server=y,address=5000,suspend=n
```
  - 6** Si inicia el servidor fuera de JBuilder, seleccione Ejecutar | Configuraciones y pulse el botón Siguiente en el cuadro de diálogo que aparece. En el campo Configuración, especifique un nombre para el nuevo servidor de configuración de ejecución para poder depurar en el servidor de aplicaciones.
  - 7** Abra la pestaña Depurar y realice los cambios en el panel Depurar:
    - a** Active la casilla de selección Activar depuración remota.
    - b** Seleccione la opción Acoplarse.
    - c** En el campo Número de puerto, introduzca el número que había anotado anteriormente durante el proceso de inicio del servidor.
    - d** Haga clic en Aceptar para cerrar todos los cuadros de diálogo.
  - 8** Seleccione la configuración de ejecución que acaba de crear en la lista desplegable del ícono Depurar, en el menú de herramientas de JBuilder.
- De esta forma, la acoplará al proceso del servidor de aplicaciones. Ahora ya puede distribuir los enterprise beans y detenerse en los

puntos de interrupción establecidos en los beans cuando se ejecuta un cliente que llama a esos beans.

Para iniciar el servidor en modo depuración dentro de JBuilder, siga estos pasos:

- Seleccione Ejecutar | Configuraciones.
- Seleccione el nodo Comando e introduzca el número de puerto para depuración remota en el campo Dirección de transporte de depuración.

Ahora, pulsando el botón de depuración de la barra de herramientas se lanza el servidor en modo depuración y se adjunta a él, todo en una única sesión.



# D

## Sugerencias acerca de iPlanet Application Server

La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

Este documento pretende reunir la información sobre el uso de iPlanet Application Server 6.x+ con JBuilder en un solo sitio. Esta misma información se encuentra dispersa por otros capítulos de la *Guía del desarrollador de Enterprise JavaBeans*. Deberá estudiar esos capítulos para aprender a utilizar JBuilder con el fin de crear y distribuir los enterprise beans.

### iPlanet 6.0 Service Pack 3

Con iPlanet 6.0, utilice Service Pack 3. JBuilder solamente se ha probado utilizando iPlanet Service Pack 3.

### Ejecución o depuración en iPlanet

Antes de comenzar a ejecutar o depurar una aplicación, inicie el servidor iPlanet fuera de JBuilder..

### Iniciar y detener iPlanet desde dentro de JBuilder

Si elige ejecutar o depurar un componente de servidor con iPlanet como servidor de aplicaciones de destino, el proceso de inicio de iPlanet es más complicado. Aparecen más cuadros de mensajes. JBuilder debe detener los procesos KJS de iPlanet que se ejecuten fuera de JBuilder, distribuir el recopilatorio seleccionado, e iniciar un solo proceso KJS dentro de

JBuilder. En cada uno de estos pasos aparece un mensaje. Al igual que sucede con otros servidores, puede ver los progresos del proceso de inicio en la ventana de mensajes. No es posible utilizar JBuilder hasta que no aparece el mensaje que indica que el motor está preparado. Una vez que aparece este mensaje, puede retroceder en el panel de mensajes con el fin de comprobar que no se han producido errores.

Si desea detener el servidor iPlanet, pulse el botón rojo de parada de la parte inferior derecha del panel de mensajes. En ese momento, los procesos KJS normales se reiniciarán fuera de JBuilder.

Si se produce un error en iPlanet, debe utilizar la herramienta de administración de iPlanet para apagar y reiniciar el servidor. Puede acceder a ella desde el menú Herramientas de JBuilder, si ha seleccionado la opción Añadir herramientas iPlanet al menú Herramientas al configurar iPlanet para JBuilder con Herramientas | Configurar servidores.

Al detener el servidor iPlanet, el archivo JAR que se está ejecutando no se distribuye. Para volver a distribuirlo, reinicie el servidor y distribuya el archivo JAR.

## Ejecución o depuración de un servlet o JavaServer Page

---

Si se está ejecutando o depurando un servlet o una página JavaServer Page (JSP) y no aparece en el panel Ver el contenido esperado, y, en su lugar, se presentan errores, pruebe con los siguientes pasos:

- 1 Detenga el servidor web antes de iniciar iPlanet dentro de JBuilder.
- 2 Cuando aparezca el mensaje informándole de que el motor ya está preparado, reinicie el servidor.

## Preparación para la depuración remota de aplicaciones iPlanet

---

Para preparar la depuración remota de aplicaciones iPlanet:

- 1 Seleccione Proyecto | Propiedades de proyecto.
- 2 Haga clic en la pestaña Depurar.
- 3 Marque la opción Activar depuración remota
- 4 Seleccione la opción Acoplarse. No debe elegir la opción Iniciar.
- 5 Pulse Aceptar.

# Configuraciones de distribución

---

En el cuadro de diálogo Configuración de distribución de iPlanet 6.x+ (Herramientas | Distribución Enterprise), la opción Mostrar distribuciones solamente funciona si se ha seleccionado la opción Servidor local como la opción Destino, aunque sea posible elegir la opción Mostrar distribuciones en la lista desplegable Acción.

## Creación de clientes para iPlanet

---

Para comenzar a crear una aplicación cliente con JBuilder:

- 1 En la ficha Propiedades de iPlanet 6.x+ del Editor de descriptor de distribución, asigne el valor true a la propiedad `iiop` de su bean enterprise.
- 2 Abra el código fuente del bean enterprise en el editor, y utilice el asistente para clientes de prueba de la galería de objetos para crear el archivo fuente cliente.
- 3 Dentro del código fuente cliente, modifique el código “búsqueda” para que utilice la convención de nomenclatura que usa iPlanet:

```
Object ref = initial.lookup("<convección iPlanet>/<Nombre del bean>")
```

Consulte la documentación de iPlanet si desea más detalles acerca de la convención de nomenclatura de iPlanet.

Ahora puede modificar y ampliar el código fuente de la aplicación cliente para que implemente la lógica que necesita esa aplicación. Cuando termine, debe crear configuraciones de ejecución para el servidor y la aplicación cliente tal y como se detalla en “[Uso de la aplicación cliente de prueba con el enterprise bean](#)” en la página 11-9.

## Inicio de la vista web

---

Para asegurarse de que la vista web se inicia, cierre manualmente el servidor web antes de utilizar Ejecutar web en JBuilder. Cuando aparezca el mensaje ENGINE IS READY inicie el servidor web.



# E

## Sugerencias acerca de Sybase Enterprise Application Server

La Edición WebLogic de JBuilder no es compatible con este servidor de aplicaciones.

Este documento pretende reunir la información sobre el uso de Sybase Enterprise Application Server con JBuilder en un solo sitio. Esta misma información se encuentra dispersa por otros capítulos de la *Guía del desarrollador de Enterprise JavaBeans*. Deberá estudiar esos capítulos para aprender a utilizar JBuilder con el fin de crear y distribuir los enterprise beans.

JBuilder es compatible con Sybase Enterprise Application Server 4.1.

### El editor de descriptor de distribución

Al hacer doble clic en los nodos del módulo EJB, EAR o WAR, del panel de proyecto, con Sybase como servidor de aplicaciones de destino, aparece una pestaña Editor DD de EA Server en el panel de contenido. Pulse sobre esta pestaña y se abrirá el Editor DD de EA Server.

Utilice el panel Editor DD de EA Server con el fin de especificar las propiedades que se almacenarán en el descriptor de distribución específico de Sybase (archivo `sybase-easerver-config.xml`). No obstante, puede seguir utilizando el panel Editor DD si desea especificar opciones comunes a todos los servidores de aplicaciones.

Para más información acerca de la utilización del panel Editor DD de EA Server, consulte “[Panel Editor DD de EA Server](#)” en la página 13-28.

## Configuración de JBuilder

---

Cuando se configura JBuilder para utilizar el servidor Sybase Enterprise Application Server, se crean automáticamente las bibliotecas necesarias. Si desea ver una lista de las bibliotecas de Sybase, consulte “[Las bibliotecas generadas](#)” en la página 5-6.

# Índice

## A

- 
- activar
    - beans entidad 16-4
    - beans sesión 15-7
  - administrador del sistema
    - competencias EJB 3-4
  - afterBegin() 15-10
  - afterCompletion() 15-10
  - agente
    - de gestión
      - de Borland Enterprise Server 5-7
      - iniciar 5-7, A-6
      - seguridad A-3
    - inteligente 5-7
  - añadir
    - métodos empresariales a EJB 8-9
    - propiedades a EJB 8-9
  - API
    - de JNDI 19-2
    - de transacciones de Java (JTA) 20-5
  - aplicaciones
    - cliente
      - de prueba
        - tutorial 22-1
      - de prueba EJB
        - ejecutar 11-12
        - tutorial 22-1
    - CORBA
      - configurar JBuilder para 24-4
      - ejemplo 25-15
    - de varios niveles
      - para sistemas distribuidos 3-1
    - del cliente EJB de ejemplo
      - SortClient 19-1
    - distribuidas
      - CORBA 24-1
      - EJB 3-1
      - ejemplos 24-10
      - ORB de VisiBroker 24-1, 24-2
    - J2EE
      - compatible con JBuilder 2-5
      - más información acerca de 2-9
      - ventajas 2-1
    - archivos
      - de propiedades del servidor
        - Borland A-2
    - EAR
      - crear 2-9, 12-6
    - IDL 24-7
  - generar 24-14
  - JAR 2-9
  - WAR 2-9
  - arquitectura
    - Enterprise JavaBeans 3-4
    - multinivel
      - ventajas 2-4
  - asignación de columna
    - multitabla de WebLogic 7-13
  - asistentes
    - Configuración de Cactus 11-17
    - Diseñador de EJB 2.0 6-7
    - distribución de Enterprise 12-7
    - Enterprise JavaBean 8-5
    - generador de Bean EJB 1.x 8-13
    - generador de interfaces EJB 1.x 8-15
    - JMS 23-2
    - Modelador de Bean entidad EJB 1.x 9-1
  - Módulo
    - EJB 8-2
    - EJB a partir de descriptores 8-4
  - para
    - cliente de prueba EJB 11-1
      - JUnit (test) 11-13
      - test Cactus 11-22
      - tipos de cliente de prueba 11-3
    - EAR 2-9, 12-6
    - EJB
      - desactivado 5-9
      - Enterprise JavaBean 8-5
      - generador de Bean EJB 1.x 8-13
      - generador de interfaces EJB 1.x 8-15
      - Módulo EJB 6-2, 8-2
      - Módulo EJB a partir de descriptores 6-4, 8-4
      - Proyecto para código existente 6-4
      - Usar cliente de prueba 11-7
    - EJB Enterprise 12-8
    - el generador de Bean EJB 1.x 8-13
    - el Generador de interfaces de EJB 1.x 8-15
    - Enterprise JavaBean 1.x 8-5
    - grupos EJB vacíos 6-2, 8-2
    - JMS 23-2
    - la configuración de Cactus 11-17
    - la distribución a Borland Enterprise Server 12-7
    - modeladores de bean entidad EJB 1.x 9-1
    - módulos EJB a partir de descriptores 6-4, 6-5, 8-4
    - usar clientes de prueba EJB 11-7

Proyecto para código existente 6-4, 6-6  
Usar cliente de prueba 11-7  
atributos  
  transacción 20-3

## B

---

bean  
  con estado  
    ciclo de vida 15-7  
  entidad 3-7  
    almacenar datos 14-1  
    claves principales 16-3, 19-4  
    compartir interfaces base y remota 16-11  
    crear BMP 7-25  
    crear WebLogic B-3  
    create() 18-5, 19-3  
    definición 16-1  
    ejemplo 16-10, 16-12, 16-14  
    eliminar 16-1  
    escribir 16-3  
    estado de activación 16-9  
    estado de espera 16-9  
    etapa inexistente 16-9  
    etapas 16-8  
    fuentes de datos 13-37  
    hacer referencia 19-3  
    interfaz base 16-11  
    interfaz base local 18-4  
    interfaz remota 16-12  
    local por defecto en el diseñador de EJB 6-21  
    métodos 16-5  
    métodos de búsqueda 13-48, 16-7, 18-6, 19-3  
    persistencia 16-1  
    suministrar datos 14-1  
  entidad (2.0)  
    añadir métodos base 7-24  
    crear 7-1  
    crear relaciones 7-16  
    ejbSelect() 7-23  
    métodos del buscador WebSphere 4.0 13-50  
  entidad (CMP 2.0)  
    crear a partir de una fuente de datos  
      importada 7-2  
    editar propiedades 7-7  
    eliminar relaciones 7-22  
  entidad de ejemplo  
    SavingsAccount 16-14  
  gestionado por mensajes 3-8, 17-1  
    ciclo de vida 17-2  
    crear 6-19  
    crear con JBuilder 17-5  
    descriptores de distribución 17-6  
    escribir 17-3

sesión 3-7, 15-1  
  búfer sin estado 15-6  
  ciclo de vida 15-6  
  con estado 15-1, 19-2  
  crear 2.0 6-9  
  crear con JBuilder 15-4  
  create() 18-3  
  escribir 15-2  
  hacer referencia 19-2  
  interfaz base local 18-3  
  interfaz SessionSynchronization 15-8  
  métodos de creación 19-2  
  remoto por defecto en el diseñador de EJB 6-21  
  remove() 19-6  
  sin estado 15-2, 19-2  
  tipos 15-1  
  tutorial 21-1  
  sin estado  
    ciclo de vida 15-6  
BeansExpress  
  exponer métodos EJB 8-11  
beforeCompletion() 15-10  
bibliotecas  
  Borland AppServer 4,5 5-6  
  Borland Enterprise Server 5.0.2 - 5.1.x 5-6  
  servidor de aplicaciones 5-6  
  WebLogic Server 5.1 5-6  
  WebLogic Server 6.x+ 5-6  
  WebSphere 4.0 Single Server 5-6  
  WebSphere Server Advanced Edition 4.0 5-6  
Borland  
  AppServer 4,5  
    instalación y configuración A-9  
  asistencia  
    a desarrolladores 1-4  
    técnica 1-4  
  contacto 1-4  
  e-mail 1-6  
  Enterprise Server 5.0.2 - 5.1.x  
    distribuir A-7  
    iniciar A-6  
    instalación y configuración A-5  
  grupos de noticias 1-5  
  informar sobre errores 1-6  
  recursos en línea 1-5  
  servidor de aplicaciones de destino 5-1  
  World Wide Web 1-5  
borrar  
  enterprise beans 18-9  
  instancias de EJB 18-2  
  vistas EJB 6-18  
botones  
  Modificar relación RDBMS 7-20

- buscar  
    6-18  
enterprise beans con clave principal 18-9  
objetos de entidad  
    clave principal 18-9
- ## C
- 
- cachés entidad  
    WebLogic 13-31
- Cactus  
    configuración para probar EJB 11-17  
    configurar 11-17  
    ejecutar pruebas 11-24
- campos  
    añadir y eliminar EJB 6-12
- cancelar transacciones 20-8
- clases  
    asignar a IDL 24-13  
    de  
        bean entidad  
            escribir 16-3  
            requisitos 16-3  
        bean sesión  
            requisitos 15-2  
        clave principal  
            JBuilder (englobamientos) 7-7  
            requisitos de beans entidad 16-3  
    de clave principal 16-3, 19-4  
    de clave principal englobada 7-7
- clave principal  
    localizar objetos de entidad 18-6  
    obtener 18-9
- clientes  
    crear con JBuilder 19-10  
    crear iPlanet 6.x+ D-3  
    de prueba  
        crear EJB 11-1  
        declarar instancia de 11-7  
        ejecutar EJB 11-9  
        tipos 11-3  
        usar EJB 11-7  
    eliminar instancias de beans 19-6  
    enterprise bean 19-1  
    gestión de las transacciones 19-8  
    localizar interfaz base 19-2  
    obtener interfaz remota 19-2
- comando  
    Añadir | ejbCreate 6-14  
    Borrar vista 6-18  
    Buscar EJB 6-18  
    Cambiar el nombre a la vista 6-17  
    modificar  
        propiedades CMP de columna 7-4
- propiedades de columna 7-4  
propiedades de tabla 7-4  
referencia de tablas 7-17  
Ordenar los EJB 6-19  
Regenerar interfaces 6-15  
Vistas | Borrar vista 6-18  
Vistas | Cambiar el nombre a la vista 6-17  
Vistas | Copiar selección 6-18  
Vistas | Eliminar selección 6-18  
Vistas | Mover selección 6-17
- Common Object Request Broker Architecture 24-1
- comparar  
    EJB 18-9
- compartir interfaces base y remota 16-11
- competencias  
    de distribución EJB 3-4  
    de operación EJB 3-4  
    de seguridad 12-5  
        crear 13-42  
        crear WebLogic 13-42  
        EJB 13-15, 13-42  
    distribuir 3-4  
    EJB 3-2  
        de aplicación 3-3  
        de infraestructura 3-3  
    EJB de infraestructura 3-3  
    operación 3-4
- compiladores  
    java2iiop 24-7, 24-8, 24-10, 24-14
- componentes  
    del cliente EJB 14-2  
    del servidor EJB 14-2  
    EJB 14-1
- comprobar enterprise beans 11-9
- configuraciones  
    de distribución (cuadros de diálogo) 12-1, 12-9  
    WebLogic B-4
- de ejecución  
    cliente y servidor 11-9  
    Servidor 11-9
- consola de Borland Enterprise Server A-9
- consultas en EJB QL 7-22
- consumidores  
    de mensajes 17-1  
    JMS 23-1
- contenedor EJB  
    aceptación de transacciones 20-2  
    activar beans sesión 15-7  
    ciclo de vida de beans con estado 15-7  
    ciclo de vida de beans entidad 16-8  
    ciclo de vida de beans sin estado 15-6  
    crea EJBObject 18-9  
    definición 3-5  
    desactivar beans sesión 15-6

- implementa la interfaz base 18-1
  - persistencia gestionada por contenedor 16-2
  - proveedor 3-3
  - contexto
    - de denominación
      - obtener 19-2
    - de denominación inicial
      - obtener 19-2
  - controladores
    - añadir controladores de bases de datos al proyecto 5-12
    - base de datos 5-11
    - JDBC 5-11
      - añadir al proyecto 5-12
  - convenciones de la documentación 1-2, 1-4
  - convertir Java a IDL 24-14
  - Copia automática de descriptores al guardar 10-1
  - copiar EJB entre vistas 6-18
  - CORBA 24-1
  - correspondencia de tipos primitivos 24-12
  - crear beans sesión EJB 2.0 6-1
  - create()
    - excepciones 18-3, 18-5
  - cuadros
    - de diálogo
      - Configuración de distribución de iPlanet 6.x+ D-3
      - Configurar servidores 5-1
      - Importar EJB 6-15
      - Propiedades CMP 7-4
      - Propiedades de generación 10-5
      - Propiedades de la fuente de datos 7-4
      - Propiedades de proyecto
        - Ficha Servidor 5-9
      - Proveedor de esquemas de bases de datos 7-2
  - D
    - DataExpress para componentes EJB 14-1
    - definiciones de interfaz Java 24-7
    - depuración
      - activar WebSphere 4.0 Advanced Edition 11-25, C-6
      - aplicaciones iPlanet de forma remota 11-30, D-2 remota
        - iPlanet 11-30, D-2
        - para WebSphere C-7
        - para WebSphere Server 3.5 C-7
        - para WebSphere Server Advanced Edition 4.0 C-8
        - para WebSphere Single Server 4.0 C-7
        - WebSphere 3.5 11-26
      - desactivar 15-6, 16-4
    - referencias
      - de entorno de recurso 13-25
      - de recursos 13-14
      - EJB 13-12
      - locales del EJB 13-23
      - transacciones de contenedor 13-32
      - ver código fuente de EJB 2.0 6-23
      - visualizar 13-2
      - WebLogic 13-26
- instancias de beans sesión 15-6
- desarrollador de beans
  - competencias EJB 3-3
  - tareas 3-6
- descriptores de distribución
  - aislamiento de transacciones de WebLogic 13-35
  - archivos
    - XML 10-5
  - caché de WebLogic 13-31
  - cambio de la información del bean 13-5
  - competencias de seguridad 13-15, 13-42
  - comprobar 13-51
  - copiar automáticamente 10-1
  - crear 12-2, 13-1
  - ejemplo de bean entidad 16-20, 16-21
  - fuentes de datos 13-37
  - identidad de seguridad EJB 13-21
  - importación en Sybase 13-31
  - importar
    - IBM C-4
    - WebSphere 4.0 C-4
- información
  - contenida 12-3
  - de ensamblaje de la aplicación 12-5
  - de estructura 12-4
- insertar
  - en grupos EAR 10-1
  - en módulos EJB 10-1
- métodos
  - de búsqueda 13-48
  - de transacción 13-32
  - del buscador WebSphere 4.0 13-50
  - idempotentes de WebLogic 13-36
- modificar código fuente directamente 13-5
- niveles de aislamiento de transacciones 13-39
- permisos para los métodos 13-44
- persistencia EJB 1.1 13-46
- propiedades 13-16
  - de beans gestionados por mensajes 13-9
  - de fuente de datos 13-39
  - de Sybase 13-28
  - del entorno 13-11
  - específicas del servidor 13-27
- propósito 12-3
- referencias
  - de entorno de recurso 13-25
  - de recursos 13-14
  - EJB 13-12
  - locales del EJB 13-23
  - transacciones de contenedor 13-32
  - ver código fuente de EJB 2.0 6-23
  - visualizar 13-2
  - WebLogic 13-26

WebSphere C-2  
deshacer distribución de un JAR de EJB 12-11  
desplazar EJB entre vistas 6-17  
destino  
    JMS 17-2, 17-6  
    JMS de cola 17-2  
    JMS de tema 17-2  
Diseñador  
    de Beans  
        Ficha Métodos 8-11  
        Ficha Propiedades 8-9  
    de EJB  
        configurar opciones del IDE 6-24  
        convenciones de nomenclatura 6-21  
        copiar EJB entre vistas 6-18  
        desplazar EJB entre vistas 6-17  
        eliminar EJB 6-20  
        importar EJB 6-15  
        importar fuente de datos 7-2  
        mostrar 6-7  
        ordenar beans 6-19  
        organizar beans con vistas 6-17  
        vista por defecto 6-17  
        volver a 6-23  
distribuir  
    Archivos JAR de EJB 12-8  
    en servidores WebSphere C-4  
    enterprise beans 12-1, 12-7  
    enterprise beans en iPlanet 12-1, 12-9  
    enterprise beans en WebLogic 12-1, 12-9  
    enterprise beans en WebSphere 12-1, 12-9  
    JAR de EJB al contenedor en ejecución 12-11  
duración  
    de la suscripción 17-6

## E

Editor  
    DD de EA Server (panel) 13-28  
    de descriptor de distribución 13-1  
        competencias de seguridad 13-42  
        comprobar descriptores 13-51  
        mostrar 6-23, 13-2  
        niveles de aislamiento de  
            transacciones 13-39  
    panel  
        Aislamiento de transacciones de  
            WebLogic 6.x o 7.x 13-35  
        Caché de WebLogic 6.x o 7.x 13-31  
        CMP 1.1 13-46  
        Data Source 13-37  
        de beans gestionados por mensajes 13-9  
        EJB Local References 13-23  
        EJB References 13-12

Environment 13-11  
específico de servidor 13-27  
Finders 13-48  
General 13-6  
General de WebLogic 6.x o 7.x 13-26  
Métodos idempotentes de WebLogic 6.x o  
7.x 13-36  
Properties 13-16  
Properties específicas del servidor 13-27  
Resource Env Refs 13-25  
Resource References 13-14  
Security Identity 13-21  
    Security Role References 13-15  
permisos para los métodos 13-44  
propiedades de fuente de datos 13-39  
transacciones de contenedor 13-32  
de referencias de tablas 7-10  
de relaciones WebLogic RDBMS 7-20  
ejbActivate() 15-3, 16-4  
EJBC (opciones) B-4  
ejbCreate() 6-14, 16-5  
    ejemplo 16-6  
    llamada por el contenedor 16-6  
    requisitos 15-3, 16-5  
ejbFindByPrimaryKey() 16-7  
ejbHome()  
    añadir a beans entidad 7-24  
ejb-jar.xml 10-5  
ejbLoad() 16-5  
ejbPassivate() 15-3, 16-4  
ejbPostCreate() 16-6  
ejbRemove() 15-3, 16-4  
ejbSelect()  
    añadir a beans entidad 7-23  
ejbStore() 16-5  
ejemplo de rmi-iop 24-10  
eliminar  
    EJB de las vistas 6-18  
    EJB entre vistas 6-18  
    instancias de beans entidad 16-1, 19-4  
    instancias de EJB 18-2  
    instancias de enterprise beans 18-9  
    JAR de EJB distribuido 12-11  
    vistas EJB 6-18  
ensamblador de aplicaciones  
    competencias EJB 3-3  
enterprise beans 3-1  
    acceso local y remoto 3-8  
    añadir campos 6-12  
    añadir métodos 6-13  
    borrar campos 6-13  
    cómo funcionan 3-6  
    comparación 18-9  
    compilar 10-5

comprobar 11-9  
configuración de Cactus para pruebas 11-17  
copiar entre vistas 6-18  
crear  
    beans sesión EJB 2.0 6-1  
    con asistentes 8-5  
desarrollar con JBuilder 4-1  
desplazar entre vistas 6-17  
distribuir 12-1  
    en servidor iPlanet 12-9  
    en servidores WebLogic 12-9  
    en servidores WebSphere 12-9, C-4  
    en tiempo real 12-11  
ejecutar 11-9  
eliminar  
    de las vistas 6-18  
    del diseñador de EJB 6-20  
    instancias 18-9  
    métodos 6-14  
entity 3-7  
errores en el diseñador de EJB 6-22  
generar desde interfaz remota 8-13  
gestionar recursos 15-3  
importar 6-15  
métodos  
    de comprobación remota 11-4  
    empresariales 16-8  
modificar  
    2.0 6-11  
    atributos 6-11  
obtener información 19-9  
opciones de distribución 12-10  
organizar en el diseñador de EJB 6-17, 6-19  
propiedades de generación 10-1, 10-4  
referencias a tablas 7-10  
regenerar interfaces 6-15  
tests  
    con Cactus 11-16, 11-22  
    JUnit 11-12  
tipos 3-7  
transacciones 20-3  
Visualización del código fuente 6-11  
enterprise beans (CMP 2.0)  
    generar clases de bean 7-6  
Enterprise JavaBeans  
    desarrollo 3-1  
    finalidad 3-1  
errores sintácticos  
    en el diseñador de EJB 6-22  
esqueletos compatibles con IIOP 24-7  
    generar 24-7  
esquema  
    crear a partir de EJB 7-24  
    de exportación

a LDD de SQL 7-24  
exportar a LDD de SQL 7-24  
importar a proyecto de EJB 7-2  
modificar  
    fuente de datos importada 7-4  
    propiedades de fuente de datos 7-4

estado  
    de espera  
        vuelta de los beans entidad 16-10

preparado  
    para métodos  
        definición 15-7  
        en transacción 15-8  
    para transacción 15-10

estructuras ampliables 24-13

excepciones  
    CreateException 18-3, 18-5  
    FinderException 18-6  
    nivel de aplicación 20-7  
    RemoteException 18-3, 18-5, 18-6, 18-8  
    transacción 20-7  
        de aplicaciones 20-8  
        de sistema 20-7

---

**F**

fábricas de conexión de temas 23-3  
findByPrimaryKey() 18-6, 19-4  
formato  
    binario  
        módulos EJB 6-2, 8-1  
    XML  
        módulos EJB 6-2, 8-1

fuentes  
    Convenciones empleadas en la documentación  
        de JBuilder 1-2  
    de datos  
        EJB 13-37  
        exportar desde el diseñador de EJB 7-24  
        importar al diseñador de EJB 7-2  
        modificar esquema importado 7-4  
        niveles de aislamiento de  
            transacciones 13-39  
        propiedades 13-39

---

**G**

generar  
    enterprise beans 10-5  
    interfaces EJB 6-15  
gestión de puerto  
    Borland A-3  
getEJBHome()  
    EJBObject 18-9  
getEJBMetaData() 18-2

EJBHome 18-2  
getHandle()  
  EJBObject 18-9  
getHomeHandle()  
  EJBHome 18-2  
getMetaData()  
  enterprise bean 19-9  
getPrimaryKey()  
  EJBObject 18-9  
getRollbackOnly() 20-8  
grupos de campos  
  WebLogic 6.x o 7.x 7-13  
grupos de noticias  
  public 1-6  
grupos EAR  
  copiar descriptores de distribución 10-1  
  insertar descriptores de distribución 10-1

|  
identificadores  
  EJB 18-2  
  obtener enterprise bean 18-9  
importar  
  descriptores de distribución EJB 6-6  
  enterprise beans 6-15  
InitialContext 20-5  
inspectores  
  bean sesión 6-9  
  campo de bean entidad 7-16  
  campo de bean sesión 6-12  
  EJB 6-11, 7-7  
  método de bean entidad 7-16  
  método de bean sesión 6-13  
  relaciones 7-17  
Interfaces  
  UserTransaction 19-8  
interfaces  
  base  
    amplía EJBHome 18-2  
    beans entidad 18-4  
    beans sesión 18-3  
    crear 18-1, 18-3  
    crear para bean enterprise existente 8-15  
    definición 18-1  
    denominación 18-1  
    ejemplo de bean entidad 18-7  
    ejemplo de bean sesión 18-3  
    local 18-2  
      beans entidad 6-21, 18-4  
      beans sesión 18-3  
      crear 18-3  
      métodos de búsqueda 18-6  
    localizada por el cliente 19-1, 19-2  
  métodos de búsqueda 18-6, 19-2  
  métodos de creación 19-2, 19-4  
  métodos de eliminación 19-4  
  remota  
    beans sesión 6-21  
    requisitos de beans entidad 18-5  
    requisitos de beans sesión 18-3  
base y remota  
  compartir beans entidad 16-11  
  clientes compatibles con IIOP 24-7  
CORBA 24-7  
EJBLocalObject  
  ampliada por la interfaz local 18-8  
EJBMetaData 19-9  
EJBObject 18-9  
  ampliada por la interfaz remota 18-8  
EnterpriseBean  
  ampliada por SessionBean 15-2  
  ampliado por EntityBean 16-4  
EntityBean  
  implementaciones de beans entidad 16-3  
  implementar 16-4  
  métodos 16-4  
IDL  
  creación desde interfaces Java 24-8  
  desde interfaces Java 24-14  
IIOP  
  creación desde interfaces Java 24-8  
  definición 24-10  
  generar 24-10  
inicio 18-1  
Java 24-13  
  convertir en IDL 24-7, 24-14  
local  
  crear 18-8  
  EJBLocalObject 18-8  
  referencia a bean sesión 19-2  
  referencias  
    a bean entidad 19-3  
LocalHome 18-2  
MessageDrivenBean 17-4  
MessageListener 17-4  
remota 18-8  
  amplía EJBObject 18-9  
  crear 18-8  
  crear para bean existente 8-15  
  definición 18-1  
  ejemplo de bean sesión 18-8  
  generar EJB 8-13  
  métodos  
    empresariales 8-11  
    referencia 19-2  
  referencia a bean entidad 19-3  
  referencia a bean sesión 19-2

- referencia obtenida por el cliente 19-1  
requisitos 18-8  
remota y base  
compartir beans entidad 16-11  
SessionBean  
ampliar 15-2  
métodos 15-2  
SessionContext 15-10  
SessionSynchronization 6-9  
beans sesión con estado 15-8  
métodos 15-10  
UserTransaction 20-4, 20-5  
invalidateFinderCollectionAtCommit 7-7  
iPlanet 6.0  
Service Pack 3 D-1  
iPlanet 6.x+  
configuraciones de distribución D-3  
crear clientes D-3  
proceso de inicio D-1  
sugerencias D-1  
isIdentical()  
EJBObject 18-9
- J**
- 
- JAR de EJB distribuidos  
listar 12-11  
JNDI  
servicio de denominación 20-5  
servicios de denominación 19-2  
jndi-definitions.xml file 7-4  
JUnit  
ejecutar el test de EJB 11-15
- L**
- 
- listar JAR distribuidos 12-11  
localizar beans  
vistas 6-18  
lookup() 19-2
- M**
- 
- Mandatory (atributo de transacción) 13-32  
mapeo  
clases Java a IDL 24-13  
tipos de datos complejos a IDL 24-13  
tipos de datos primitivos a IDL 24-12  
matrices  
asignar a secuencias ilimitadas de  
CORBA 24-13  
mensajes  
JMS 17-1, 23-1  
modelo "punto a punto" 23-1  
modelo de publicación y suscripción 23-1
- punto a punto 17-2  
metadatos  
definición 19-9  
métodos  
añadir EJB 6-13  
base 7-24  
base empresariales 7-24  
comprobación de enterprise beans 11-4, 11-12, 11-16  
de búsqueda 13-48, 16-5, 18-6, 19-2, 19-3  
añadir a beans entidad (2.0) 7-22  
beans entidad 1.1 16-3  
crear 16-7  
por defecto 19-4  
prefijo 18-6  
requisitos 16-7  
WebSphere 4.0 13-50  
de creación  
bean entidad 16-5, 19-3  
bean sesión 19-2  
ejbCreate()  
crear en el diseñador de EJB 6-14  
pasar parámetros 6-14  
empresariales 16-8  
añadir a enterprise bean 8-9  
beans entidad 16-5  
escribir 15-3  
exponer EJB 8-11  
interfaz local 18-8  
interfaz remota 8-11, 18-8  
llamados por el cliente 19-1, 19-5  
idempotente 13-36  
modelos  
de mensajes  
suscribir y publicar 17-2  
de mensajes "publicación y suscripción" 23-3  
de mensajes "punto a punto" 23-1, 23-4  
modificar el descriptor de distribución 13-1  
modos de reconocimiento  
beans gestionados por mensajes 17-6  
mensajes JMS 23-3  
módulos EJB 8-2  
a partir de descriptores de distribución 6-4, 8-4  
copia automática de descriptores 10-1  
copiar descriptores de distribución 10-1  
crear 8-2  
definición 6-2, 8-1, 8-2  
extensiones del archivo 6-2, 8-1  
formatos 6-2, 8-1  
importar beans 6-15  
insertar descriptores de distribución 10-1  
propiedades de generación 10-1, 10-4  
tipos 6-2, 8-1

multiplicidad  
muchos a muchos 7-20  
uno a uno 7-20

## N

Never (atributo de transacción) 13-32  
nivel  
de aislamiento 13-39  
WebLogic 6.x 13-35  
de aislamiento de transacciones  
WebLogic 6.x o 7.x 13-35  
del cliente 2-4  
tecnologías 2-6  
intermedio 2-4  
tecnologías 2-6  
nombres de archivos EJB  
Diseñador de EJB 6-21  
Nomenclatura de Web 24-7  
normas de aislamiento de transacciones  
WebLogic 6.x o 7.x 13-35  
NotSupported (atributo de transacción) 13-32

## O

onMessage() 17-2, 17-4  
escribir 17-4  
opciones  
Activar servidor 5-1  
Crear siempre JAR al generar el proyecto 10-1  
del IDE  
configurar el diseñador de EJB 6-24  
Eliminar archivos stub al cambiar de  
AppServer 10-1  
Regenerar interfaces siempre 6-15  
ORB (Intermediador de objetos-Object Request  
Broker)  
configurar 24-4  
por defecto 24-4  
propiedades 24-4  
ORB de VisiBroker 24-2  
aplicaciones de ejemplo 25-15  
compilador IDL 24-7, 24-14  
compilador IIOP 24-7, 24-10  
con JBuilder 24-2  
configurar para JBuilder 24-4  
ejemplo de rmi-iiop 24-10  
y RMI 24-9  
OrbixWeb  
configurar para JBuilder 24-4  
ordenar beans 6-19

## P

panel

Aislamiento de transacciones de WebLogic 6.x o  
7.x 13-35  
Caché de WebLogic 6.x o 7.x 13-31  
de beans gestionados por mensajes  
propiedades 13-9  
de fuente del DD 13-2  
EJB Local References 13-23  
General de WebLogic 6.x o 7.x 13-26  
Métodos idempotentes de WebLogic 6.x o  
7.x 13-36  
Properties  
específicas del servidor EJB 13-27  
Resource Env Refs 13-25  
Security Identity 13-21  
Security Roles  
Editor de descriptor de distribución 13-42  
partición  
Borland A-1  
distribución de WAR expandidos A-4  
política del cargador de clases A-4  
seguridad A-4  
pauta de diseño  
Session Bean wrap Entity Bean 14-1  
permisos para los métodos 12-5, 13-44  
persistencia 3-7  
beans entidad 1.1 13-46  
desventajas  
de la gestión por bean 16-2  
de la gestión por contenedor 16-2  
EJB 1.1 gestionado por contenedor 13-46  
gestionad  
por bean 16-20  
gestionada  
por bean 16-1, 16-2  
desventajas 16-2  
métodos de búsqueda 16-3  
por contenedor 13-46, 16-1, 16-2, 16-21  
clase de clave no principal 16-6  
limitaciones 16-2  
modificar propiedades de Borland 7-4  
ventajas 16-2  
por contenedor y gestionada por bean 16-2  
gestionada por bean 16-2  
gestionado por contenedor 16-1  
ventajas de la gestión por contenedor 16-2  
plataformas  
convenciones 1-4  
política del cargador de clases  
para partición A-4  
proceso de inicio  
iPlanet 6.x+ D-1  
productores  
de mensajes 17-1  
JMS 23-1

- propiedades  
    CacheCreate 7-7  
    checkExistenceBeforeCreate 13-16  
    CMP 7-7  
        modificar Borland 7-4  
    configurar  
        componentes Sybase 13-30  
        paquetes de Sybase 13-28  
de EJB específicas del servidor  
    editar 13-5, 13-27  
de fuente de datos  
    modificar esquema 7-4  
de generación  
    bean 10-4  
    Copia automática de descriptores al guardar 10-1  
    grupo EAR 10-1  
    modificación 10-1  
    módulo EJB 10-4  
    módulos EJB 10-1  
del entorno  
    EJB 13-11  
descriptor de distribución 13-16  
EJB 13-16  
específicas de Sybase 13-28  
findByPrimaryKeyLoadState 13-16  
findPrimaryKeyBehavior 13-16  
getPrimaryKeyAfterInsertSql 13-16  
getPrimaryKeyBeforeInsertSql 13-16  
ignoreColumnsOnInsert 13-16  
jdbcAccesserFactory 13-16  
maxBeanInTransaction 13-16  
maxBeansInCache 7-7, 13-16  
maxBeansInPool 7-7, 13-16  
optimisticConcurrencyBehavior 13-16  
persistencia gestionada por contenedor 7-7  
primaryKeyGenerator 13-16  
transactionCommitMode 7-7, 13-16  
proveedor  
    de beans  
        competencias EJB 3-3  
        tareas 3-6  
    de contenedores  
        competencias EJB 3-3  
    de servidor  
        EJB 3-3  
        JMS 17-1  
        Sonic MQ Message Broker 17-8  
proyecto  
    configuración para Cactus 11-17  
    crear a partir de los EJB 6-4  
    por defecto  
        añadir controladores de bases de datos 5-12  
puerto  
    Agente inteligente de VisiBroker 5-7  
    cambiar la gestión de Borland A-3
- 
- R**
- redistribuir  
    JAR de EJB 12-11  
referencias  
    a enterprise beans  
        con identificadores 19-6  
        métodos de búsqueda 19-4  
        métodos de creación 19-3  
de competencias de seguridad 12-6  
de entorno de recurso  
    EJB 13-25  
de recursos  
    EJB 13-14  
de tablas  
    editar 7-17  
    EJB 13-12  
regenerar automáticamente interfaces EJB 6-15  
relaciones  
    bidireccionales 7-17  
    de muchos a muchos 7-17  
    EJB 2.0 7-16  
    EJB 2.0 WebLogic 6.x o 7.x 7-20  
    entre EJB 7-17  
    unidireccionales 7-17  
remove()  
    EJBHome 18-2  
    EJBObject 18-9  
Required (atributo de transacción) 13-32  
RequiresNew (atributo de transacción) 13-32  
RMI  
    y ORB de VisiBroker 24-9  
rollback() 20-8
- 
- S**
- seguridad  
    definición para agente de gestión A-3  
selector de mensajes 17-2, 17-6  
services packs  
    añadir 5-5  
Servicio de mensajes Java 23-1  
servidor  
    activar 5-1  
    admitido 5-1  
    Borland  
        sugerencias A-1  
de aplicaciones  
    activar 5-1  
    configurar destino 5-1  
    destino  
        Borland 5-1

iPlanet 5-1  
WebLogic 5-1  
WebSphere 5-1  
EJB 3-5  
    definición 3-5  
ejecutar EJB 11-9  
seleccionar 5-9  
WebLogic  
    distribuir en 12-9  
    instalación y configuración B-1  
    sugerencias B-1  
WebSphere  
    descriptores de distribución C-2  
    descriptores de distribución propios C-4  
    distribuir en C-4  
    sugerencias C-1  
sesiones de transacción 23-3  
setEntityContext() 16-4  
setRollbackOnly() 15-10  
setSessionContext() 15-3  
sistemas de mensajes  
    publicación y suscripción 23-1  
    punto a punto 23-1  
Sonic MQ Message Broker 17-8  
stubs  
    compatibles con IIOP 24-7  
    generar 24-7  
    de cliente  
        generar 24-7  
        generar cliente 10-1  
Supports (atributo de transacción) 13-32  
suscriptor duradero 23-3  
Sybase  
    importar descriptores de distribución 13-31  
    sugerencias E-1

## T

test de módulos  
    configuración de Cactus para EJB 11-17  
    ejecutar  
        tests Cactus 11-24  
        tests JUnit para EJB 11-15  
    probar un EJB  
        mediante Cactus 11-16, 11-22  
        mediante JUnit 11-12  
tipos de datos  
    correspondencia de primitivos 24-12  
    datos complejos, correspondencia 24-13  
transacciones  
    atributos 13-32, 20-3  
    característica  
        aislamiento 20-1  
        atomicidad 20-1

    conurrencia 20-1  
    persistencia 20-1  
consistencia 20-1  
demarcación 20-3, 20-5  
gestionadas  
    por bean 20-3  
    por contenedor 13-32, 20-3, 20-4  
        añadir 13-32  
        por el cliente 19-8  
global 20-4  
limitaciones 19-8, 20-5  
local 20-4  
niveles de aislamiento 13-39  
normas 13-32  
políticas de aislamiento de WebLogic 6.x o  
    7.x 13-35  
rolling back. 20-8  
y enterprise beans 20-3  
transacciones de contenedor 13-32  
transacciones gestionadas por bean 20-5  
tutoriales  
    crear aplicaciones cliente de prueba 22-1  
    diseñar beans sesión con el diseñador de  
        EJB 21-1  
    java2iiop 24-10  
    tutorial de CORBA 25-1

## U

---

unsetEntityContext() 16-4  
Usenet, grupos de noticias 1-6

## V

---

VisiBroker  
    Agente inteligente 5-7  
    poner a disposición de JBuilder  
        ORB 5-7

vistas  
    buscar EJB 6-18  
    cambiar el nombre al EJB por defecto 6-17  
    copiar EJB entre 6-18  
    crear 6-17  
    desplazar EJB entre 6-17  
    EJB por defecto 6-18  
    eliminar EJB 6-18  
    ordenar beans 6-19

## W

---

WebLogic  
    configuración de opciones EJBC B-4  
    descriptores de distribución 10-5  
    paneles del Editor de descriptor de  
        distribución B-4

servidor de aplicaciones de destino 5-1  
utilizar código ya creado B-3

WebLogic 5.1  
instalación y configuración B-3

WebLogic 6.x  
instalación y configuración B-2

WebLogic 6.x o 7.x  
asignación multitable 7-13  
grupos de campos 7-13

WebLogic 7.x  
instalación y configuración B-1  
weblogic-ejb-jar.xml 10-5

WebSphere  
servidor de aplicaciones de destino 5-1

WebSphere Server 3.5  
sugerencias C-1