

Introducción a Sistemas de Información

Por el: M. en C. RICARDO ARMANDO MACHORRO REYES.

Copyright © 2016 Ricardo Armando Machorro Reyes
Derechos reservados.
Permitido su uso para fines no comerciales.

Índice

1. Introducción al Modelo Relacional.....	1
1.1. Atributos.....	1
1.2. Tuplas.....	2
1.3. Relaciones.....	2
1.4. Tablas.....	3
1.5. Claves.....	3
1.6. Manipulación de Tablas con SQL.....	3
1.6.1. Creación de una Tabla.....	3
1.6.2. Inserción de Datos en una Tabla.....	4
1.6.3. Modificación de Datos en una tabla.....	5
1.6.4. Consulta de Datos en una Tabla.....	5
1.6.5. Eliminación de Datos en una Tabla.....	6
2. Introducción a JDBC.....	7
2.1. Biblioteca de Clases JDBC.....	7
2.2. Carga del Controlador de Conexiones.....	8
2.3. Conexión a la Base de Datos.....	9
2.4. Envío de Instrucciones SQL al Manejador.....	11
2.5. Recuperación de Datos.....	12
2.8. Programa con las clases básicas.....	13
2.8. Secuencias de Escape.....	15
2.9. Consultas con Parámetros.....	17
2.9.1. Programa con Consultas Parametrizadas.....	19
2.10. Generación Automática y Recuperación de Llaves.....	20
2.10.1. Generación y Recuperación de Llaves con MySQL y Derby.....	20
2.10.2. Generación y Recuperación de Llaves con Oracle.....	21
2.11. Procedimientos Almacenados.....	22
2.11.1. Ejemplo de Procedimientos Almacenados en MySQL.....	24
2.11.1. Invocación de Procedimientos Almacenados.....	25
3. Interfaces Sencillas.....	28
3.1. Introducción a UML.....	28
3.1.1. Diagramas de Secuencia.....	28
3.1.2. Diagramas de Clase.....	29
3.2. Inserción de Datos.....	30
3.2.1. Funcionamiento.....	30
3.2.2. Arquitectura de Tres Capas.....	32
3.2.3. Diagrama de Secuencia.....	32
3.2.4. Diagrama de Clases.....	33
3.2.5. Código de Proveedor.....	34
3.2.5. Código de CtrlInserción.....	35
3.2.6. Código de GBC.....	36
3.2.7. Diseño de la Forma.....	38
3.2.8 Código de FrmInserción.....	39
3.3. Búsqueda y Modificación.....	42
3.3.1. Funcionamiento.....	42
3.3.2. Diagrama de Clases.....	43

3.3.3. Diagrama de Secuencia.....	44
3.3.4. Código de CtrlModificación.....	45
3.3.5. Código de FrmModificación.....	47
3.4. Eliminación.....	50
3.4.1. Funcionamiento.....	50
3.4.2. Diagrama de Clases.....	51
3.4.3. Diagrama de Secuencia.....	52
3.4.4. Código de CtrlEliminación.....	53
3.4.5. Código de FrmEliminación.....	54
3.5. Consulta.....	57
3.5.1. Funcionamiento.....	57
3.5.2. Diagrama de Clases.....	58
3.5.3. Diagrama de Secuencia.....	59
3.5.4. Código de CtrlConsulta.....	59
3.5.5. Código de FrmConsulta.....	61
4. Mantenimiento de Catálogos.....	65
4.1. Funcionamiento.....	65
4.2. Manejo de la Barra de Título.....	68
4.3. Diagrama de Clases.....	68
4.4. Código de ConexionBD.....	70
4.5. Código de CreaTablas.....	71
4.6. Código de CtrlMantenimiento.....	73
4.7. Diseño de la Forma.....	76
4.8. Código de FrmMantenimiento.....	77
5. Creación de un Ejecutable.....	86
5.1. Creación del Archivo Manifest.....	86
5.1.2. Código fuente de “manifest.txt”.....	86
5.2. Creación del Archivo Ejecutable (jar).....	86
5.2.1. Código fuente de “empaca.bat”.....	87
5.3. Construcción del Instalador.....	87
5.3.1. Código fuente de “proveedores.iss”.....	88
6. Distribución en Red.....	90
6.1. Firmar Digitalmente los Archivos.....	90
6.1.1. Código de “firma.bat”.....	90
6.2. Archivo JNLP.....	91
6.2.1 Código de “proveedores_red.jnlp”.....	91
6.3. Colocar los Archivos en el Servidor Web.....	92
6.3.1. Código de “proveedores.html”.....	92

1. Introducción al Modelo Relacional.

Este capítulo explica los conceptos básicos de las **bases de datos relacionales**, que es el modelo más difundido para almacenar datos.

1.1. Atributos.

El elemento más básico que contempla el modelo relacional es el de atributo; consta de:

- **Nombre.** Texto que describe una característica; por ejemplo: “dirección”, “fecha de nacimiento”, “sexo”. Normalmente se escriben en minúsculas.
- **Dominio.** Conjunto de valores que puede tomar el atributo. A continuación se ejemplifican los dominios para atributos.

Atributo	Dominio
dirección	Texto
fecha de nacimiento	Fecha, que consta de año, mes y día.
sexo	'masculino' o 'femenino'

- **Valor.** El valor que tiene asignado en un instante dado el atributo. Debe tener las siguientes características:
 - Corresponder al dominio del atributo.
 - Puede cambiar con el tiempo.
 - Ser **atómico**; es decir que no puede tener estructura interna. Un ejemplo, de estructura interna es un valor tipo texto donde unos caracteres de representan el departamento, otros la gerencia, etc. Los nuevos modelos de normalización permiten superar esta restricción.
 - Ser **univaluado**; esto es que no contiene estructuras repetitivas, por ejemplo, no se pueden almacenar varios teléfonos en un campo; solo se almacena uno. Los nuevos modelos de normalización permiten superar esta restricción.

A continuación se ejemplifican valores para atributos.

Atributo	Valor
dirección	'Carmelitas #54'
fecha de nacimiento	Fecha(año=1980, mes='enero' día=12)
sexo	'femenino'

1.2. Tuplas.

Una tupla es un conjunto de atributos, con nombres diferentes. En la práctica, representa a un objeto y sus características. Este es un ejemplo de tupla.

Atributo	Dominio	Valor
prov_clave	Entero	1
prov_nombre	Texto	'Zoila Vaca'
prov_telefono	Texto	'01-800-MUU'
prov_email	Texto	'zolia@vaca.com.mx'
prov_direccion	Texto	'Establos #23'

1.3. Relaciones.

Una relación es un conjunto de tuplas que cumplen las siguientes reglas:

- Todas tienen el mismo número de atributos.
- Todas tienen atributos con los mismos nombres y dominios, pero pueden tener distintos valores.

Este es un ejemplo de relación:

Proveedor				
prov_clave:Entero	prov_nombre:Texto	prov_telefono: Texto	prov_email: Texto	prov_direccion: Texto
1	Zoila Vaca	01-800-MUU	zoila@vaca.com.mx	Establos #23
2	Rolando Mota	01-800-VIAJE	rolando@viajesmota.com	Misterios #12
3	Armando Pacheco	01-800-TQUILA	armando@tiquila.com	Noche de Ronda #99
4	Pancracio Wellington	01-800-PLATANOS	pancracio@platanospancracio.com	Plátanos #69

1.4. Tablas.

En la práctica los sistemas de bases de datos relacionales utilizan objetos llamados **tablas**, que normalmente se manejan como si fueran relaciones, pero actualmente permiten superar algunas restricciones del modelo relacional. En ellas, las tuplas reciben el nombre de **registros** y los atributos se siguen llamando atributos, pero también pueden llamarse **campos**.

La forma de obtener el diseño de tablas está fuera de los alcances del presente texto. Lo que se estudia a continuación es como manipularlas usando el manejador de bases de datos Derby, que viene incluido en el JDK de Java a partir de la versión 1.6.0.

1.5. Claves.

Las **claves** permiten identificar de forma única un registro. Representan la identidad de un objeto. También se les conoce como **llaves** y como **identificadores**. Hay diferentes tipos.

- **Candidatas.** Conjunto no vacío de atributos que identifican unívoca y mínimamente cada tupla.
- **Primarias.** De entre todas las claves candidatas para una relación, hay una que se escoge como oficial.
- **Alternativas.** Toda las claves candidatas que no se seleccionaron como primaria.

1.6. Manipulación de Tablas con SQL.

Para manipular tablas se utiliza un lenguaje conocido como “Lenguaje Estructurado de Consultas”, que en inglés se escribe “Structured Query Language” y es mejor conocido por sus siglas en inglés: **SQL**. A continuación se presentan las instrucciones fundamentales para manejar información.

1.6.1. Creación de una Tabla.

Se usa la instrucción CREATE TABLE. La estructura para una tabla que tiene n campos (donde n puede ser 1, 2, etc.) es algo como lo que sigue:

```
CREATE TABLE nombre_de_la_tabla (  
  nombre_del_campo_1 tipo_del_campo_1 restricciones_del_campo_1,  
  nombre_del_campo_2 tipo_del_campo_2 restricciones_del_campo_2,  
  ...  
  nombre_del_campo_n tipo_del_campo_n restricciones_del_campo_n,  
)
```

Este es un ejemplo.

```

1 CREATE TABLE Proveedor (
2   prov_clave INTEGER PRIMARY KEY,
3   prov_nombre VARCHAR(100) NOT NULL,
4   prov_telefono VARCHAR(50) NOT NULL,
5   prov_email VARCHAR(50) NOT NULL,
6   prov_direccion LONG VARCHAR NOT NULL)

```

En este caso el valor de n es 5, porque son 5 campos.

nombre_de_la_tabla es Proveedor.

nombre_del_campo_1 es prov_clave.

tipo_del_campo_1 es INTEGER.

restricciones_del_campo_1 es PRIMARY KEY.

De forma similar se interpretan las características de los otros campos.

1.6.2. Inserción de Datos en una Tabla.

Se usa la instrucción INSERT INTO. La estructura para una tabla que tiene n campos (donde n puede ser 1, 2, etc.) es algo como lo que sigue:

INSERT INTO *nombre_de_la_tabla* (*nombre_del_campo_1*, *nombre_del_campo_2*, ..., *nombre_del_campo_n*) **VALUES** (*valor_del_campo1*, *valor_del_campo2*, ..., *valor_del_campo_n*)

A continuación se presenta un ejemplo.

```

1 INSERT INTO Proveedor
2 (prov_clave, prov_nombre, prov_telefono, prov_email, prov_direccion)
3 VALUES(1, 'Zoila Vaca', '01-800-MUU', 'zoila@vaca.com.mx', 'Establos #23')

```

En este caso el valor de n es 5, porque son 5 campos.

nombre_de_la_tabla es Proveedor.

nombre_del_campo_1 es prov_clave.

valor_del_campo_1 es 1.

nombre_del_campo_2 es prov_nombre.

valor_del_campo_2 es 'Zoila Vaca'.

De forma similar se interpretan los otros campos.

1.6.3. Modificación de Datos en una tabla.

La instrucción UPDATE se utiliza para modificar el valor de campos. La estructura donde se modifica el valor de n atributos (donde n puede ser 1, 2, etc.) es algo como lo que sigue:

UPDATE *nombre_de_la_tabla* SET

nombre_del_campo_1 = *valor_del_campo1*, *nombre_del_campo_2* = *valor_del_campo2*, ...,
nombre_del_campo_n = *valor_del_campo_n*

WHERE *condición_para_seleccionar_registros*

A continuación se presenta un ejemplo.

1	UPDATE Proveedor SET
2	prov_telefono = '01-800-VACA', prov_direccion = 'Calle del Toro # 4'
3	WHERE prov_clave = 1;

En este caso el valor de n es 2, porque solo se modifican 2 campos.

nombre_de_la_tabla es Proveedor.

nombre_del_campo_1 es prov_telefono.

valor_del_campo_1 es '01-800-VACA'.

El nuevo valor de prov_telefono es '01-800-VACA'.

nombre_del_campo_2 es prov_direccion.

valor_del_campo_2 es 'Calle del Toro # 4'

El nuevo valor de prov_direccion es 'Calle del Toro # 4'.

condición_para_seleccionar_registros es prov_clave = 1.

Los cambios solo se realizan en los registros que cumplan la *condición_para_seleccionar_registros*; es decir, donde prov_clave = 1.

1.6.4. Consulta de Datos en una Tabla.

La forma más sencilla para obtener los datos almacenados en una tabla tiene la siguiente estructura:

SELECT * FROM *nombre_de_la_tabla*

WHERE *condición_para_seleccionar_registros*

A continuación se presenta un ejemplo.

1	SELECT * FROM Proveedor
2	WHERE prov_clave = 1;

En este caso:

nombre_de_la_tabla es Proveedor.

condición_para_seleccionar_registros es `prov_clave = 1`.

Se muestran todos los campos de los registros que cumplan la *condición_para_seleccionar_registros*; es decir, donde `prov_clave = 1`.

El resultado de la consulta es el siguiente:

prov_clave	prov_nombre	prov_telefono	prov_email	prov_direccion
1	Zoila Vaca	01-800-VACA	zoila@vaca.com.mx	Calle del Toro # 4

1.6.5. Eliminación de Datos en una Tabla.

Se usa la instrucción **DELETE FROM**. La estructura para eliminar los datos almacenados en una tabla es la siguiente:

DELETE FROM *nombre_de_la_tabla*

WHERE *condición_para_seleccionar_registros*

A continuación se presenta un ejemplo.

1	DELETE FROM Proveedor
2	WHERE prov_clave = 1;

En este caso:

nombre_de_la_tabla es Proveedor.

condición_para_seleccionar_registros es `prov_clave = 1`.

Se eliminan todos los registros que cumplan la *condición_para_seleccionar_registros*; es decir, donde `prov_clave = 1`.

2. Introducción a JDBC.

En este capítulo se muestran distintas formas de conectar un programa escrito en lenguaje de programación Java con una base de datos.

JDBC significa Java Data Base Connectivity. Comprende las clases utilizadas para conectar con bases de datos. Hace uso intensivo del mecanismo de excepciones, que se basa en abortar la ejecución de métodos cuando alguna instrucción falla. Las clases que describen las fallas se conoce como **excepciones**.

2.1. Biblioteca de Clases JDBC.

El primer elemento que se necesita es la librería que contiene las clases necesarias para conectar un programa con un manejador de bases de datos específico.

En el caso de Derby hay dos formas de operación:

- **Embedded.** En español se conoce como empotrado. La base de datos es manejada directamente por el programa. No necesita activar un servidor. Solamente puede ser utilizada por un programa a la vez.
- **Client.** En español se conoce como cliente. El programa se conecta a un servidor de bases de datos, que da servicio a varios programas a la vez.

A continuación se listan los nombres de las librerías para algunas bases de datos y la forma de conseguirlos.

Base de Datos	Archivo	Forma de conseguirlo
Derby (Embedded)	"derby.jar"	Incluido en el JDK 1.6.0, en la carpeta "C:\Archivos de Programa\Java\jdk1.6.0\db\lib". Para versiones posteriores se encuentra en "C:\Archivos de Programa\Sun\JavaDB\lib"
Derby (Client)	"derbyclient.jar"	Incluido en el JDK 1.6.0, en la carpeta "C:\Archivos de Programa\Java\jdk1.6.0\db\lib". Para versiones posteriores se encuentra en "C:\Archivos de Programa\Sun\JavaDB\lib"
MySQL	"mysql-connector-java-XXX-bin.jar", donde XXX es la versión. Por ejemplo, para la versión 5.0.7 el archivo es: "mysql-connector-java-5.0.7-bin.jar"	Descarga del sitio: http://mysql.com
Oracle	"ojdbc14.jar"	Descarga del sitio: http://oracle.com

Estos archivos se deben incluir CLASSPATH. Por ejemplo, en Windows puede tenerse la siguiente instrucción desde el símbolo del sistema o algún archivo bat:

```
set CLASSPATH=.;C:\Archivos de Programa\Java\jdk1.6.0\db\lib\derby.jar
```

o bien, al ejecutar algún archivo, por ejemplo la clase JDBC, se puede usar la siguiente instrucción:

```
java -cp .;C:\Archivos de Programa\Java\jdk1.6.0\db\lib\derby.jar JDBC
```

En caso de trabajar en algún ambiente de programación, normalmente se define un proyecto y en sus propiedades se puede definir la variable CLASSPATH.

2.2. Carga del Controlador de Conexiones.

Para realizar la conexión a una base de datos desde el programa, el primer paso es cargar la clase del controlador de conexiones específico de la base de datos. A continuación se lista el nombre de esta clase para distintas bases de datos.

Base de Datos	Clase
Derby (Embedded)	org.apache.derby.jdbc.EmbeddedDriver
Derby (Client)	org.apache.derby.jdbc.ClientDriver
MySQL	com.mysql.jdbc.Driver
Oracle	oracle.jdbc.OracleDriver

La carga del controlador de conexiones es realizada por la clase `Class`, que se encarga de administrar las clases de Java. En particular, cuenta con un método que se llama `forName`, que busca una clase de acuerdo a las indicaciones del `CLASSPATH` y cargarla; si no la encuentra, lanza la excepción `java.lang.ClassNotFoundException`. A continuación se ejemplifica como cargar el controlador para Derby embedded.

```
Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
```

Esta instrucción solo se ejecuta una vez en el programa y debe invocarse antes de las instrucciones de acceso a bases de datos; de lo contrario se genera una excepción de tipo `java.sql.SQLException` con el mensaje "No suitable driver".

2.3. Conexión a la Base de Datos.

El siguiente paso es establecer la conexión a la base de datos. La encargada es la clase `DriverManager`, con el método `getConnection`. Recibe tres parámetros:

- **URL.** Significa "Universal Resource Locator" (Localizador Universal de Recursos) y es una cadena que sirve para ubicar un recurso en una red. Consta de 4 partes:
 - **Protocolo.** Es el lenguaje de comunicación. Por ejemplo, `http`, `https`, `tcp`, etc. Para comunicarse desde Java a bases de datos, el protocolo es **`jdbc`**.
 - **Host.** Es la computadora donde está alojada la base de datos. Puede ser el nombre del servidor o su dirección IP. Cuando el servidor corre en la misma computadora que el programa, el nombre del servidor puede ser "127.0.0.1" o también "localhost". En algunas ocasiones incluye también el número de puerto, que es un número asignado de forma única a un programa. Por ejemplo, el servidor de Oracle generalmente utiliza el puerto 1521.
 - **Recurso.** Nombre del recurso a utilizar. Puede tratarse de un archivo, de una impresora o una base de datos. En el caso de Oracle se utiliza el `ORASID` y en otros manejadores es el nombre de la base de datos.
 - **Opciones.** Especifican formas de manejar la conexión o la base de datos. Por ejemplo, en Derby embedded la opción "create=true" permite crear la base de datos en caso de que no exista.
- **Usuario.** Nombre de usuario para realizar la conexión. Es validado por la base de datos.

- **Password.** Contraseña correspondiente a la base de datos. Es validado por la base de datos.

A continuación se muestra como construir el URL para distintos manejadores.

- **Derby (Embedded)**

Formato:

`"jdbc:derby:recurso;opciones"`

Ejemplos:

`"jdbc:derby:proveedores;create=true"`

`"jdbc:derby:pedidos"`

- **Derby (Client)**

Formato:

`"jdbc:derby://host:puerto/recurso;opciones"`

Ejemplos:

`"jdbc:derby://localhost:1527/proveedores;create=true"`

`"jdbc:derby:pedidos"`

- **MySQL**

Formato:

`"jdbc:mysql://host:puerto/recurso"`

Ejemplos:

`"jdbc:mysql://localhost/test"`

`"jdbc:mysql://localhost:3306/test"`

- **Oracle**

Formato:

`"jdbc:oracle:thin:@host:puerto:oraid"`

Ejemplo:

`"jdbc:oracle:thin:@192.168.7.221:1521:oracledb"`

El resultado de la operación `getConnection` es una instancia de la interfaz `java.sql.Connection`, que representa la conexión a la base de datos. Si no se puede realizar porque no se ha cargado el controlador de conexión, la combinación usuario/password es incorrecta, no hay red o algún otro error, se lanza una excepción del tipo `java.sql.SQLException`. A continuación se muestra un ejemplo de como establecer la conexión.

```
Connection c = DriverManager.getConnection("jdbc:derby:proveedores;create=true",
                                           "juan", "chimo1267");
```

En este ejemplo, el url es "jdbc:derby:proveedores;create=true"; el usuario es "juan" y el password es "chimo1267".

En el caso de que no se haya cargado el manejador de conexiones o el URL sea incorrecto, el mensaje de la excepción es "No suitable driver".

Al final de la sesión siempre hay que cerrar la conexión. En algunas bases de datos y configuraciones es forzoso cerrar la conexión. Se utiliza la instrucción `close`, que genera también una excepción del tipo `java.sql.Exception` cuando algo sale mal. He aquí un ejemplo.

```
c.close();
```

2.4. Envío de Instrucciones SQL al Manejador.

Hay varias formas para ejecutar una instrucción en SQL. Una de ellas es con una instancia de la interfaz `java.sql.Statement` y se usa cuando la consulta se conoce totalmente al momento de escribir el programa. El siguiente ejemplo muestra como crear un objeto de este tipo.

```
Statement s = c.createStatement();
```

En algunos se cargan registros de las bases de datos. Si estos registros se revisan una sola vez cada uno, es suficiente con la instrucción anterior; pero si han de revisarse más de una vez, o avanzar y retroceder en el juego de registros, es necesario usar la siguiente instrucción.

```
Statement s = c.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,  
                                ResultSet.CONCUR_READ_ONLY);
```

La clase `ResultSet` se representa los conjuntos de registros recuperados de la base de datos.

La constante `ResultSet.TYPE_SCROLL_INSENSITIVE` indica que los registros se pueden recorrer en diferentes direcciones (avanzar y retroceder), sin necesidad de que el conjunto de registros se mantenga sincronizado con la base de datos en todo momento.

La constante `ResultSet.CONCUR_READ_ONLY` representa que los datos se utilizarán solo para lectura. (El `ResultSet` puede usarse también para modificar datos.)

Para enviar la manejador una instrucción que modifique datos, solo hay que enviarla como parámetro del método `executeUpdate` de la clase `java.sql.Satatement`. Devuelve el número de registros

modificados. En caso de que la instrucción no se pueda ejecutar porque su formato sea incorrecto o alguna otra razón se genera una excepción del tipo `java.sql.SQLException`. He aquí un ejemplo de como enviar una instrucción **INSERT**.

```
int modificados = s.executeUpdate("INSERT INTO Proveedor "
    + "(prov_clave, prov_nombre, prov_telefono, "
    + " prov_email, prov_direccion, prov_fecha) "
    + "VALUES(1, 'Zoila Vaca', '01-800-MUU', "
    + " 'zoila@vaca.com.mx', 'Establos #23', "
    + "{d '2003-10-3'})");
```

Esta instrucción agrega un registro a la tabla Proveedor y devuelve el número de registros modificados, que en este caso es 1.

El manejo de fechas es muy diferente entre diferentes manejadores de base de datos. Para no tener que atar el programa a un manejador específico, se permite el uso de la secuencia de escape `{d '2003-10-3'}`, donde la letra `d` indica que se trata de una fecha (date en inglés). El año es 2003, el mes es octubre (mes número 10) y el día es 3. El controlador de JDBC convierte la secuencia a la sintaxis del manejador que se utiliza al momento de ejecutar la instrucción.

Cuando una instancia de `Statement` ya no se usa también debe cerrarse. Tiene un método `close` que lanza una `SQLException` cuando algo sale mal. Este es un ejemplo.

```
c.close();
```

2.5. Recuperación de Datos.

El método `executeQuery` permite recuperar datos con el uso de una instrucción **SELECT** de SQL. El valor devuelto es una instancia de la clase `java.sql.ResultSet`, que representa una colección de registros. Este es un ejemplo de como recuperar datos.

```
ResultSet rs = s.executeQuery("SELECT * FROM Proveedor");
```

La clase `java.sql.ResultSet` tiene los siguientes métodos:

- `next()`. Permite avanzar al siguiente registro. Cuando es posible avanzar, devuelve `true`. Si no existe un siguiente registro, devuelve `false`.
- `previous()`. Permite regresar al registro que le precede al actual. Cuando es posible avanzar, devuelve `true`. Si no existe un registro anterior, devuelve `false`. Requiere que al crear la instancia de `Statement` se defina del tipo `ResultSet.TYPE_SCROLL_INSENSITIVE`.

- `beforeFirst()`. Se coloca antes del primer registro. Es la posición donde se posiciona originalmente. Si posteriormente a este método se invoca `next()`, se recupera el primer registro. Requiere que al crear la instancia de `Statement` se defina del tipo `ResultSet.TYPE_SCROLL_INSENSITIVE`.
- `afterLast()`. Se coloca después del último registro. Si posteriormente a este método se invoca `previous()`, se recupera el último registro. Requiere que al crear la instancia de `Statement` se defina del tipo `ResultSet.TYPE_SCROLL_INSENSITIVE`.
- `last()`. Se coloca en el último registro. Requiere que al crear la instancia de `Statement` se defina del tipo `ResultSet.TYPE_SCROLL_INSENSITIVE`.
- `first()`. Se coloca en el primer registro. Requiere que al crear la instancia de `Statement` se defina del tipo `ResultSet.TYPE_SCROLL_INSENSITIVE`.
- `getRow()`. Devuelve el número del registro actual. El primer registro devuelve 1. invocar este método después de `last()` proporciona el número total de registros devuelto.
- `getInt("campo"), getString("campo")`. Obtiene el valor del campo indicado para el registro actual y lo convierte al tipo indicado después del prefijo `get`. Por ejemplo,

```
int a = rs.getInt("prov_clave");
```

 devuelve el valor de `prov_clave` y lo interpreta como entero. Si el valor no se puede convertir al tipo indicado, se genera una `SQLException`.
- `getDate("campo")`. Devuelve una referencia a un objeto de tipo `java.sql.Date`, que solo pone valor a día, mes y año; las horas, minutos y segundos se ponen en cero, a diferencia del tipo `java.util.Date`, del cual se deriva, que también asigna estos valores. Hay otras clases derivadas de `java.sql.Date` que se usan: `java.sql.Time` para manejar solo el tiempo (horas, minutos y segundos) y `java.sql.Timestamp` que maneja fecha y hora, con una resolución hasta nanosegundos.
- `close()`. Libera los recursos utilizados cuando se deja de usar el objeto.

2.8. Programa con las clases básicas.

El siguiente programa muestra el uso de la clase base. Se recomienda cerrar los objetos utilizados dentro de una cláusula `finally` para asegurar que siempre se ejecuten estas instrucciones, aún en caso de error.

```

1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5 import java.sql.Statement;
6 import java.text.SimpleDateFormat;
7
8 public class ClasesBase {
9     /**

```

```

10  * Ejecuta el programa. Si se presenta alguna error, se aborta la
11  * ejecución y la excepción resultante es atrapada por el ambiente de
12  * ejecución que invoca a main.
13  *
14  * @param args
15  *           Parámetros de la línea de comando.
16  * @throws ClassNotFoundException
17  *           Si no se puede cargar el controlador de conexión.
18  * @throws SQLException
19  *           Si se presenta alguna falla durante el acceso a la
20  *           base de datos.
21  */
22  public static void main(String[] args) throws ClassNotFoundException,
23      SQLException {
24      // Carga el controlador de conexiones.
25      Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
26      String url = "jdbc:derby:proveedores;create=true";
27      String usuario = "";
28      String password = "";
29      /*
30       * Normalmente las referencias a los objetos de JDBC se declaran
31       * fuera de un bloque try y se inician en null, para que el bloque
32       * finally del final pueda identificar si se pudieron crear o no.
33       */
34      Connection c = null;
35      Statement s = null;
36      ResultSet rs = null;
37      /*
38       * Las instrucciones dentro del bloque try se realizan, pero si
39       * alguna de ellas reporta un error lanzando una excepción el bloque
40       * aborta su ejecución.
41       */
42      try {
43          SimpleDateFormat fmt = new SimpleDateFormat("dd/MM/yyyy");
44          // Establece la conexión a la base de datos.
45          c = DriverManager.getConnection(url, usuario, password);
46          s = c.createStatement();
47          // Crea la tabla.
48          s.executeUpdate("CREATE TABLE Proveedor ( "
49              + "prov_clave INTEGER PRIMARY KEY, "
50              + "prov_nombre VARCHAR(100) NOT NULL, "
51              + "prov_telefono VARCHAR(50) NOT NULL, "
52              + "prov_email VARCHAR(50) NOT NULL, "
53              + "prov_direccion LONG VARCHAR NOT NULL, "
54              + "prov_fecha DATE NOT NULL)");
55          // Inserta datos en la tabla.
56          s.executeUpdate("INSERT INTO Proveedor "
57              + "(prov_clave, prov_nombre, prov_telefono, "
58              + " prov_email, prov_direccion, prov_fecha) "
59              + "VALUES(1, 'Zoila Vaca', '01-800-MUU', "
60              + " 'zoila@vaca.com.mx', 'Establos #23', "
61              + "{d '2003-10-3'})");
62          s.executeUpdate("INSERT INTO Proveedor "
63              + "(prov_clave, prov_nombre, prov_telefono, "

```

```

64         + " prov_email, prov_direccion, prov_fecha) "
65         + "VALUES(2, 'Rolando Mota', '01-800-VIAJE', "
66         + " 'rolando@viajesmota.com', 'Misterios #12', "
67         + "{d '2005-11-6'})");
68     // Recupera información de la base de datos.
69     rs = s.executeQuery("SELECT * FROM Proveedor");
70     // Revisa renglón por renglón la información devuelta.
71     while (rs.next()) {
72         // Muestra el contenido del registro.
73         System.out.println(rs.getInt("prov_clave") + " | "
74             + rs.getString("prov_nombre") + "|"
75             + rs.getString("prov_telefono") + "|"
76             + rs.getString("prov_email") + "|"
77             + rs.getString("prov_direccion") + "|"
78             + fmt.format(rs.getDate("prov_fecha")));
79     }
80     rs.close();
81     s.close();
82 } finally {
83     // Las instrucciones dentro de finally se realizan
84     // siempre; aún cuando se presente una excepción.
85     if (rs != null) { // Es true si el objeto se pudo crear.
86         try {
87             rs.close();
88         } catch (SQLException e) {
89             /*
90              * Si hay algún error al cerrar, ya no se puede hacer
91              * nada para rescatarlo. Hay que pasar a la siguiente
92              * instrucción.
93              */
94         }
95     }
96     if (s != null) {
97         try {
98             s.close();
99         } catch (SQLException e) {}
100    }
101    if (c != null) {
102        try {
103            c.close();
104        } catch (SQLException e) {}
105    }
106 }
107 }
108 }

```

2.8. Secuencias de Escape.

En algunos aspectos los manejadores de base de datos tienen una sintaxis totalmente diferente. Es por ello que JDBC introduce secuencias de escape que los controladores convierten en la sintaxis correcta para el manejador utilizado al instante de ejecutarse. Siempre se ponen entre llaves ({}). Tienen el siguiente formato: {palabra_clave parámetros}. Por ejemplo, en {d '2003-10-3'}, la palabra_clave

es `d` y el *parámetro* es `'2003-10-3'`. A continuación se listan las secuencias de escape.

- **escape. Caracteres de escape para el operador LIKE.** Normalmente el caracter `'%` corresponde a cero o más caracteres y el caracter `'_'` representa un caracter. Si se quiere eliminar este comportamiento e interpretar el caracter literalmente, debe ser precedido por una diagonal invertida (`'\'`), que se conoce como caracter de escape. Se puede especificar otro caracter de escape con la siguiente secuencia al final de la consulta:

```
{escape 'caracter_de_escape'}
```

En el siguiente ejemplo, se define una diagonal invertida como caracter de escape y selecciona a todos los proveedores cuyo correo electrónico lleve guión bajo.

```
s.executeQuery("SELECT * FROM Proveedor WHERE prov_email LIKE '%\\_%'"+
  + "{escape '\\'}");
```

- **fn. Funciones escalares.** Casi todos los manejadores de base de datos tienen definidas funciones que devuelven valores de distintos tipos. Pueden invocarse con la palabra clave `fn`, seguida de la invocación a la función deseada, incluyendo sus argumentos. El siguiente ejemplo concatena dos textos.

```
{fn concat('texto1', 'texto2')}
```

Las funciones soportadas varían según el manejador. (Nada es perfecto.)

- **d. Literales tipo fecha.** La fecha se especifica con la siguiente sintaxis:

```
{d 'año-mes-día'}
```

Donde *año* se indica con un número de cuatro dígitos, *mes* se indica con un número del 1 al 12 y *día* es el día del mes, de 1 a 31.

- **h. Literales tipo hora.** Las horas se especifican con el siguiente formato.

```
{t 'hora:minutos:segundos'}
```

hora es de 0 a 23. *minutos* y *segundos* son de 0 a 59.

- **t. Literales para fecha y hora juntas (timestamp).** Se utiliza el siguiente formato; la forma de especificar año, mes y día es la misma que para especificar solamente fechas.

```
{t 'año-mes-día hora:minutos:segundos.fracciones_de_segundo'}
```

.fracciones_de_segundo se puede omitir.

- **oj. Outer joins.** La sintaxis es la siguiente.

```
{oj outer_join}
```

La sintaxis para *outer_join* es:

tabla tipo_de_join **OUTER JOIN** *tabla_o_outer_join* **ON** *condición*

El *tipo_de_join* puede ser LEFT, RIGHT o FULL. *tabla_o_outer_join* puede ser el nombre de una tabla o bien otro *outer_join*. El siguiente ejemplo utiliza esta secuencia de escape.

```
s.executeQuery("SELECT * FROM { oj Producto LEFT OUTER JOIN "
               + "TipoDeProducto ON prod_clave = 1}");
```

Una vez más, el tipo de join que se puede usar, depende del manejador de base de datos empleado.

El uso de secuencias de escape está habilitado de forma predeterminada, pero puede habilitarse o deshabilitarse con el método de objeto `Statement.setEscapeProcessing`.

2.9. Consultas con Parámetros.

En algunas ocasiones la información a enviar a la base de datos no se conoce al momento de escribir el programa, o bien se quiere repetir varias veces la misma instrucción, pero con diferentes datos. En este caso es conveniente utilizar la clase `java.sql.PreparedStatement`, que se deriva de la clase `java.sql.Statement`.

Al momento de crear la consulta, se usan signos de interrogación (?) para definir parámetros. He aquí un ejemplo de como crearla. La variable `c` es una instancia de `java.sql.Connection`.

```
PreparedStatement ps = c.prepareStatement("INSERT INTO Proveedor "
    + "(prov_clave, prov_nombre, prov_telefono, "
    + " prov_email, prov_direccion, prov_fecha) "
    + "VALUES(?, ?, ?, ?, ?, ?)");
```

Cuando se desea realizar una consulta cuyos registros se revisan hacia adelante y hacia atrás, se puede usar una instrucción como la que sigue:

```
PreparedStatement ps = c.prepareStatement("SELECT * FROM Proveedor "
    + "WHERE prov_fecha < ?",
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
```

En este ejemplo, se crean seis parámetros al final de la consulta y están indicados por los signos de interrogación. Se identifican por su orden de aparición. Así el parámetro 1 corresponde al primer signo de interrogación, el parámetro 2 al segundo signo de interrogación y así sucesivamente.

En este momento, algunos manejadores de bases de datos compilan y optimizan la consulta, de tal forma que es más rápido crear la consulta y utilizarla varias veces, que usar varias consultas por separado.

Antes de ejecutar la instrucción se asigna valor a todos los parámetros. Dicho valor se coloca con la sintaxis adecuada en lugar del signos de interrogación. A continuación se muestra como asignar valor. La variable `fmt` apunta a una instancia de `java.util.DateFormat` y se encarga de crear un objeto de `java.util.Date` a partir de un texto.

```
ps.setInt(1, 3); // valor del signo de interrogación 1
ps.setString(2, "Armando Pacheco"); // signo 2
ps.setString(3, "01-800-TQUILA"); // signo 3
ps.setString(4, "armando@tquila.com");
ps.setString(5, "Noche de Ronda #99");
ps.setDate(6, new Date(fmt.parse("23/2/2007").getTime()));
```

En el caso de las fechas (parámetro 6), debe asignarse una instancia de `java.sql.Date`, cuyo único constructor recibe un `long` que representa la fecha. Este valor puede obtenerse con el método `getTime` de `java.util.Date`. Para obtener la fecha deseada puede utilizarse

```
java.util.Date fecha = fmt.parse("23/2/2007");
```

Una vez obtenida la fecha, el número de tipo `long` que la representa, se obtienen así

```
long número = fecha.getTime();
```

Con este número, la instancia de `java.sql.Date` se obtiene de esta forma:

```
java.sql.Date fechaSQL = new java.sql.Date(número);
```

Cuando todos los parámetros están asignados, puede invocarse `executeUpdate` o `executeQuery` según el tipo de instrucción SQL.

```
int modificados = ps.executeUpdate();
ResultSet rs = ps.executeQuery();
```

Una vez invocada, se puede cambiar el valor de los parámetros y ejecutarse una vez más. Este proceso puede repetirse las veces deseadas.

```
ps.setInt(1, 4); // Cambia los valores anteriores
ps.setString(2, "Pancracio Wellington");
ps.setString(3, "01-800-PLATANOS"); // signo 3
ps.setString(4, "pancracio@platanospancracio.com");
ps.setString(5, "Paseo de los plátanos #99");
ps.setDate(6, new Date(fmt.parse("23/2/2007").getTime()));
ps.executeUpdate();
```

Para esta clase, no se permite deshabilitar las secuencias de escape.

2.9.1. Programa con Consultas Parametrizadas.

```

1  import java.sql.Connection;
2  import java.sql.Date;
3  import java.sql.DriverManager;
4  import java.sql.PreparedStatement;
5  import java.sql.ResultSet;
6  import java.sql.SQLException;
7  import java.text.ParseException;
8  import java.text.SimpleDateFormat;
9
10 public class ConsultasParametrizadas {
11     public static void main(String[] args) throws ClassNotFoundException,
12         ParseException, SQLException {
13         Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
14         String url = "jdbc:derby:proveedores";
15         String usuario = "";
16         String password = "";
17         Connection c = null;
18         PreparedStatement ps1 = null;
19         PreparedStatement ps2 = null;
20         ResultSet rs = null;
21         try {
22             /*
23              * Crea un formato para fecha. dd - día del mes. MM - mes en dos
24              * dígitos. yyyy - año en cuatro dígitos. / - separación.
25              */
26             SimpleDateFormat fmt = new SimpleDateFormat("dd/MM/yyyy");
27             c = DriverManager.getConnection(url, usuario, password);
28             ps1 = c.prepareStatement("INSERT INTO Proveedor "
29                 + "(prov_clave, prov_nombre, prov_telefono, "
30                 + " prov_email, prov_direccion, prov_fecha) "
31                 + "VALUES(?, ?, ?, ?, ?, ?)");
32             ps1.setInt(1, 3); // valor del signo de interrogación 1
33             ps1.setString(2, "Armando Pacheco"); // signo 2
34             ps1.setString(3, "01-800-TQUILA"); // signo 3
35             ps1.setString(4, "armando@tquila.com");
36             ps1.setString(5, "Noche de Ronda #99");
37             ps1.setDate(6, new Date(fmt.parse("23/2/2007").getTime()));
38             ps1.executeUpdate(); // ejecuta la operación
39             // Ejecuta la instrucción con otros datos
40             ps1.setInt(1, 4); // Cambia los valores anteriores
41             ps1.setString(2, "Pancracio Wellington");
42             ps1.setString(3, "01-800-PLATANOS"); // signo 3
43             ps1.setString(4, "pancracio@platanospancracio.com");
44             ps1.setString(5, "Paseo de los plátanos #99");
45             ps1.setDate(6, new Date(fmt.parse("23/2/2007").getTime()));
46             ps1.executeUpdate();
47             // Realiza una búsqueda en base a la clave del proveedor.

```

```

48      // Si se encuentra, se obtiene un solo registro.
49      ps2 = c.prepareStatement("SELECT * FROM Proveedor "
50          + "WHERE prov_clave = ?");
51      ps2.setInt(1, 3);
52      rs = ps2.executeQuery();
53      if (rs.next()) {
54          System.out.println(rs.getInt("prov_clave") + " -> "
55              + rs.getString("prov_nombre"));
56      } else {
57          System.out.println("Proveedor no encontrado");
58      }
59  } finally {
60      if (rs != null) {
61          try {
62              rs.close();
63          } catch (SQLException e) {}
64      }
65      if (ps1 != null) {
66          try {
67              ps1.close();
68          } catch (SQLException e) {}
69      }
70      if (ps2 != null) {
71          try {
72              ps2.close();
73          } catch (SQLException e) {}
74      }
75      if (c != null) {
76          try {
77              c.close();
78          } catch (SQLException e) {}
79      }
80  }
81  }
82  }

```

2.10. Generación Automática y Recuperación de Llaves.

Algunas aplicaciones requieren generar automáticamente claves y detectar cual es el valor de estas. Debe hacerse al momento de insertar los datos, pues si las claves se reservaran al momento de desplegar la forma, y muchos usuarios tienen abierta la misma ventana, a cada uno de ellos se le asigna una clave diferente; si algunos no guardan sus datos, se pueden desperdiciar muchos valores de la clave. Hay dos formas de realizar esta función, dependiendo de la base de datos.

2.10.1. Generación y Recuperación de Llaves con MySQL y Derby.

Ambas bases de datos tienen la forma de declarar campos con valores generados automáticamente.

Para Derby es de la siguiente forma:

```
1 CREATE TABLE TipoDeProducto (
2   tprod_clave INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
3   tprod_nombre VARCHAR(30) UNIQUE NOT NULL
4 )
```

Para MySQL es así:

```
1 CREATE TABLE TipoDeProducto (
2   tprod_clave INTEGER PRIMARY KEY AUTO_INCREMENT,
3   tprod_nombre VARCHAR(30) UNIQUE NOT NULL
4 ) TYPE = InnoDB
5 CHARACTER SET utf8 COLLATE utf8_spanish_ci;
```

Cuando se inserta un dato, hay que indicar que se espera recuperar las llaves generadas:

```
1 // Inserta el objeto en la tabla y solicita que se capture
2 // la llave generada automáticamente.
3 ps = c.prepareStatement("INSERT INTO TipoDeProducto (tprod_nombre) VALUES (?)",
4   Statement.RETURN_GENERATED_KEYS);
```

Para recuperar la llave generada, se utiliza el método `getGeneratedKeys` de la siguiente forma (la variable `c` es la referencia a una conexión):

```
1 int nuevaClave = -1;
2 // Recupera un result set con las llaves generadas.
3 ResultSet rs = ps.getGeneratedKeys();
4 // Si solo se generó una llave, hay un solo renglón
5 if (rs.next()) {
6   // Si solo se generó una llave, está en el primer campo
7   nuevaClave = rs.getInt(1);
8 }
```

2.10.2. Generación y Recuperación de Llaves con Oracle.

En este caso el campo solo debe de ser numérico, sin una declaración especial.

```
1 CREATE TABLE TipoDeProducto (
2   tprod_clave INTEGER PRIMARY KEY,
3   tprod_nombre VARCHAR(30) UNIQUE NOT NULL
4 )
```

Hay que crear una secuencia.

```
1 CREATE SEQUENCE sec_tprod INCREMENT BY 1 START WITH 1;
```

Antes de insertar, hay que recuperar el siguiente valor de la secuencia usando la instrucción `nextval`. En el siguiente ejemplo, `s` es una referencia a un `Statement`.

```

1 ResultSet rs = s.executeQuery("SELECT sec_tprod.nextval FROM DUAL");
2 int nuevaClave = -1;
3 // Si solo se generó una llave, hay un solo renglón
4 if (rs.next()) {
5     // Si solo se generó una llave, está en el primer campo
6     nuevaClave = rs.getInt(1);
7 }

```

Lo único que resta es insertar en la tabla usando el nuevo valor

```

1 ps = c.prepareStatement("INSERT INTO TipoDeProducto(tprod_clave, tprod_nombre)"
2     + " VALUES (?, ?)");
3 ps.setInt(1, nuevaClave);

```

2.11.Procedimientos Almacenados.

Algunos manejadores de bases de datos permiten crear funciones que manipulan tablas e invocarlas desde programas. Se conocen como **procedimientos almacenados**. No todos los manejadores soportan esta característica.

Los objetos que implementan la interfaz `java.sql.CallableStatement` (derivada de `java.sql.PreparedStatement`) pueden invocar los procedimientos almacenados. Dado que la forma de invocación cambia según el manejador, se utiliza una secuencia de escape especial. Hay cuatro formas de especificarla que dependen del uso u omisión de parámetros, y de la devolución de valores. Son las siguientes:

```
{call procedimiento}
```

Para un procedimiento almacenado que no devuelve valores ni recibe parámetros.

```
{? = call procedimiento}
```

Para un procedimiento almacenado que devuelve un valor (que se maneja como el primer parámetro) y no recibe parámetros.

```
{call procedimiento(?, ?, ...)}
```

Para un procedimiento almacenado que no devuelve valores y recibe uno o más parámetros.

```
{? = call procedimiento(?, ?, ...)}
```

Para un procedimiento almacenado que devuelve un valor (que se maneja como el primer parámetro) y recibe uno o más parámetros.

En el siguiente ejemplo se crea una instancia de `CallableStatement`, se le asigna valor a sus parámetros y se invoca.

```
// Declara la invocación a suma.
CallableStatement cs = c.prepareCall("{? = call suma(?, 5) }");
// Declara el parámetro de salida que corresponde al valor de regreso.
cs.registerOutParameter(1, Types.INTEGER);
//Declara el parámetro 2, que corresponde al primer argumento de la invocación.
cs.setInt(2, 3);
// Invoca el procedimiento.
cs.executeUpdate();
// Obtiene el valor devuelto.
int x = cs.getInt(1);
// Cierra el objeto.
cs.close();
```

Cuando la invocación devuelve un `java.sql.ResultSet` y se desea recorrerlo hacia adelante y hacia atrás, también se puede crear la invocación de la siguiente forma:

```
CallableStatement cs = c.prepareCall("{call consultaProveedores}",
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
```

Hay tres tipos de parámetros, que se marcan con signos de interrogación en la secuencia de escape, de la misma forma que en una instancia de `java.sql.PreparedStatement`.

- **IN.** De entrada. Reciben información del programa. Se declaran de la misma forma que los parámetros de `java.sql.PreparedStatement`, usando `setXX`, como en este ejemplo:

```
cs.setInt(1, 4);
```

- **OUT.** De salida. Regresan información al programa. El valor devuelto por el procedimiento almacenado es de este tipo. Se declaran con el método `registerOutParameter` de `java.sql.CallableStatement`, como se muestra:

```
cs.registerOutParameter(1, Types.INTEGER);
```

Los valores posibles de la clase `java.sql.Types` son los siguientes: BIT, TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, REAL, DOUBLE, NUMERIC, DECIMAL, CHAR, VARCHAR, LONGVARCHAR, DATE, TIME, TIMESTAMP, BINARY, VARBINARY, LONGVARBINARY, NULL, OTHER, JAVA_OBJECT, DISTINCT, STRUCT, ARRAY, BLOB, CLOB, REF, DATALINK, BOOLEAN, ROWID, NCHAR, NVARCHAR, LONGNVARCHAR, NCLOB y SQLXML.

Después de la ejecución del procedimiento, los valores devueltos se recuperan con operaciones `getXXX(número)`, como en el caso de `java.sql.ResultSet`. Así por ejemplo, el parámetro 1, declarado de salida, de tipo entero se recupera con.

```
int x = cs.getInt(1);
```

- **INOUT.** De entrada y salida. Intercambia información en los dos sentidos con el programa. Se declaran dos veces: una como IN y otra como OUT. Después de la ejecución del procedimiento, los valores devueltos se recuperan con operaciones `getXXX(número)`, como en el caso de `java.sql.ResultSet`.

Cuando el cuerpo de un procedimiento almacenado utiliza la instrucción **SELECT** de SQL se invoca con `executeQuery`. Se recomienda que los registros devueltos se procesen antes de leer cualquier parámetro de entrada.

No todos los manejadores de bases de datos soportan procedimientos almacenados.

2.11.1. Ejemplo de Procedimientos Almacenados en MySQL.

```

1 CREATE DATABASE Procedimientos;
2 USE Procedimientos;
3 /* Define la nueva cadena para separar instrucciones, que ahora es // */
4 DELIMITER //
5 /* Crea la tabla*/
6 CREATE TABLE Proveedor (prov_clave INTEGER PRIMARY KEY,
7     prov_nombre VARCHAR(100) NOT NULL,
8     prov_telefono VARCHAR(50) NOT NULL,
9     prov_email VARCHAR(50) NOT NULL,
10    prov_direccion LONG VARCHAR NOT NULL,
11    prov_fecha DATE NOT NULL)//
12 /* Inserta datos. */
13 INSERT INTO Proveedor
14     (prov_clave, prov_nombre, prov_telefono, prov_email, prov_direccion,
15     prov_fecha)
16     VALUES(1, 'Zoila Vaca', '01-800-MUU', 'zoila@vaca.com.mx',
17     'Establos #23', '2003-10-3'),
18     (2, 'Rolando Mota', '01-800-VIAJE', 'rolando@viajesmota.com',
19     'Misterios #12', '2005-11-6')//
20 /* Define un procedimiento almacenado que suma datos. */
21 CREATE FUNCTION suma (a INTEGER, b INTEGER) RETURNS INTEGER
22 RETURN a + b;
23 //
24 /* Define un procedimiento almacenado que cuenta los proveedores. */
25 CREATE PROCEDURE cuenta (OUT total INT)
26 BEGIN
27     SELECT COUNT(*) INTO total FROM Proveedor;
```

```

28 END
29 //
30 /* Define un procedimiento inserta un proveedor. */
31 CREATE PROCEDURE insertaProveedor (clave INTEGER, nombre CHAR(100),
32     telefono CHAR(50), email CHAR(50), direccion TEXT, fecha DATE)
33 BEGIN
34     INSERT INTO Proveedor
35     (prov_clave, prov_nombre, prov_telefono, prov_email, prov_direccion,
36     prov_fecha)
37     VALUES(clave, nombre, telefono, email, direccion, fecha);
38 END
39 //
40 /* Define un procedimiento almacenado que devuelve todos los proveedores. */
41 CREATE PROCEDURE consultaProveedores ()
42 BEGIN
43     SELECT * FROM Proveedor;
44 END
45 //
46 GRANT ALL PRIVILEGES ON Procedimientos.* TO 'usu'@localhost
47 IDENTIFIED BY 'ario'
48 //

```

2.11.1. Invocación de Procedimientos Almacenados.

Utiliza las definiciones del punto anterior.

```

1 import java.sql.CallableStatement;
2 import java.sql.Connection;
3 import java.sql.Date;
4 import java.sql.DriverManager;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.sql.Types;
8 import java.text.ParseException;
9 import java.text.SimpleDateFormat;
10
11 public class ProcedimientosAlmacenados {
12     public static void main(String[] args) throws ClassNotFoundException,
13         ParseException, SQLException {
14         Class.forName("com.mysql.jdbc.Driver");
15         /*
16          * En este caso la opción noAccessToProcedureBodies indica que
17          * esta conexión no tiene suficientes permisos para obtener el
18          * código del procedimiento almacenado.
19          */
20         String url = "jdbc:mysql://localhost/Procedimientos"
21             + "?noAccessToProcedureBodies=true";
22         String usuario = "usu";
23         String password = "ario";
24         Connection c = null;
25         CallableStatement cs1 = null;
26         CallableStatement cs2 = null;

```

```

27 CallableStatement cs3 = null;
28 CallableStatement cs4 = null;
29 ResultSet rs = null;
30 try {
31     /*
32      * Crea un formato para fecha. dd - día del mes. MM - mes en dos
33      * dígitos. yyyy - año en cuatro dígitos. / - separación.
34      */
35     SimpleDateFormat fmt = new SimpleDateFormat("dd/MM/yyyy");
36     c = DriverManager.getConnection(url, usuario, password);
37     // Declara la invocación a suma.
38     cs1 = c.prepareCall("{? = call suma(?, 5) }");
39     /*
40      * Declara el parámetro de salida que corresponde al valor de
41      * regreso.
42      */
43     cs1.registerOutParameter(1, Types.INTEGER);
44     /*
45      * Declara el parámetro 2, que corresponde al primer argumento de
46      * la invocación.
47      */
48     cs1.setInt(2, 3);
49     // Invoca el procedimiento.
50     cs1.executeUpdate();
51     // Obtiene el valor devuelto.
52     System.out.println("suma(3, 5) = " + cs1.getInt(1));
53     // Declara la invocación a cuenta.
54     cs2 = c.prepareCall("{call cuenta(?) }");
55     // Declara el parámetro de salida al total de registros.
56     cs2.registerOutParameter(1, Types.INTEGER);
57     cs2.executeUpdate();
58     // Obtiene el valor devuelto.
59     System.out.println("total de proveedores = " + cs2.getInt(1));
60     // Declara la invocación a insertaProveedor.
61     cs3 = c.prepareCall("{call insertaProveedor"
62         + "(?, ?, ?, ?, ?, ?) }");
63     // Declara los parámetros de entrada.
64     cs3.setInt(1, 3);
65     cs3.setString(2, "Armando Pacheco");
66     cs3.setString(3, "01-800-TQUILA");
67     cs3.setString(4, "armando@tquila.com");
68     cs3.setString(5, "Noche de Ronda #99");
69     cs3.setDate(6, new Date(fmt.parse("23/2/2007").getTime()));
70     cs3.executeUpdate();
71     // Declara la invocación a consultaProveedores.
72     cs4 = c.prepareCall("{call consultaProveedores}");
73     rs = cs4.executeQuery();
74     while (rs.next()) {
75         // Muestra el contenido del registro.
76         System.out.println(rs.getInt("prov_clave") + " | "
77             + rs.getString("prov_nombre") + "|"
78             + rs.getString("prov_telefono") + "|"
79             + rs.getString("prov_email") + "|"
80             + rs.getString("prov_direccion") + "|")

```

```
81         + fmt.format(rs.getDate("prov_fecha"));
82     }
83     } finally {
84         if (rs != null) {
85             try {
86                 rs.close();
87             } catch (SQLException e) {}
88         }
89         if (cs1 != null) {
90             try {
91                 cs1.close();
92             } catch (SQLException e) {}
93         }
94         if (cs2 != null) {
95             try {
96                 cs2.close();
97             } catch (SQLException e) {}
98         }
99         if (cs3 != null) {
100             try {
101                 cs3.close();
102             } catch (SQLException e) {}
103         }
104         if (cs4 != null) {
105             try {
106                 cs4.close();
107             } catch (SQLException e) {}
108         }
109         if (c != null) {
110             try {
111                 c.close();
112             } catch (SQLException e) {}
113         }
114     }
115 }
116 }
```

3. Interfaces Sencillas.

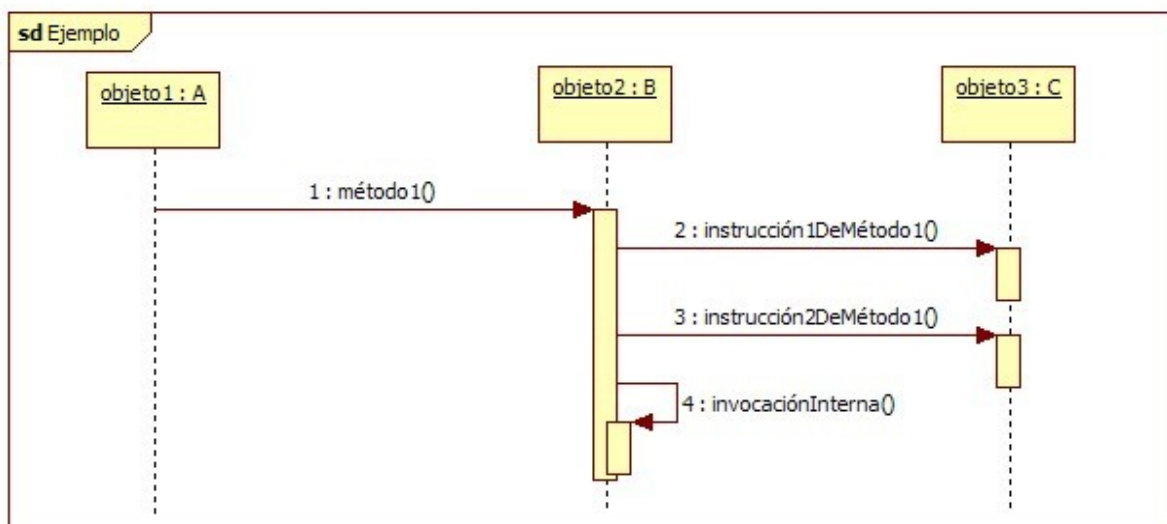
En este capítulo se muestra como realizar las operaciones fundamentales (inserción, búsqueda, modificación, borrado y consulta) desde interfaces gráficos.

3.1. Introducción a UML.

Explicar el funcionamiento de cualquier cosa es más fácil con la ayuda de dibujos. En el caso de la programación orientada a objetos, existe un estándar para dibujar clases, objetos y sus interacciones, conocido como “Lenguaje Unificado de Modelado”; mas conocido por su nombre en inglés, “Unified Modeling Language” y aún más por sus siglas en ingles: **UML**.

3.1.1. Diagramas de Secuencia.

Sirven para mostrar como es la interacción entre objetos de distintas clases. A continuación se muestra un ejemplo.



En este diagrama, A, B y C son clases. Hay tres objetos: objeto1 de la clase A, objeto2 de la clase2 y objeto3 de la clase C. Las rayas punteadas que salen de abajo de cada objeto se conocen como **líneas de tiempo** y representan a cada objeto conforme transcurre el tiempo de arriba hacia abajo. Las flechas indican instrucciones; salen del objeto que solicita su ejecución y llegan al objeto encargado de realizarla. El rectángulo que toca la flecha se conoce como **foco** y muestra el tiempo durante el cual el objeto está activo. Las flechas que salen del foco indican como se realiza la instrucción que activa el foco. En este ejemplo, la flecha 1 indica que objeto2 ejecuta el método llamado “método1”. Para ello se activa y realiza en orden las siguientes flechas:

2. El objeto2 solicita a objeto3 que ejecute instrucción1DeMétodo1().
3. El objeto2 solicita a objeto3 que ejecute instrucción2DeMétodo1().
4. El objeto2 ejecute su propio método llamado invocaciónInterna().

El código de método1 puede ser algo como lo que sigue:

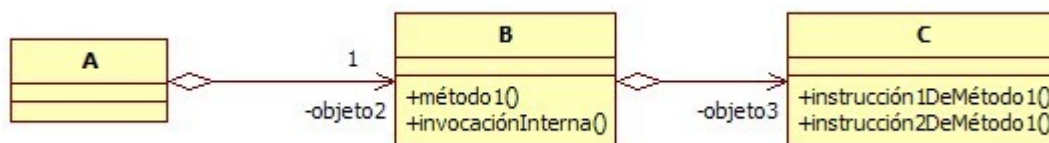
```

1 public void método1() {
2     objeto3.instrucción1DeMétodo1();
3     objeto3.instrucción2DeMétodo1();
4     invocaciónInterna();
5 }

```

3.1.2. Diagramas de Clase.

Muestran en forma resumida los atributos y operaciones de una clase. También marcan las relaciones con otras clases. He aquí un ejemplo:



Este diagrama corresponde con el diagrama de secuencia del punto anterior. Marca que tenemos tres clases.

La clase A contiene una referencia a un objeto de clase B que se llama `objeto2`; el signo menos antes del nombre indica que el acceso es privado. El rombo indica la clase que contiene el objeto y la flecha apunta al tipo de la referencia.

La clase B contiene una referencia privada a un objeto de clase C que se llama `objeto3`. También contiene dos métodos públicos: `método1` e `invocaciónInterna`.

La clase C contiene dos métodos: `instrucción1DeMétodo1` y `instrucción2DeMétodo1`.

A continuación se lista el código que corresponda a cada clase:

Clase A:

```
1 public class A {
2     private B objeto2 = new B();
3 }
```

Clase B:

```
1 public class B {
2     private C objeto3 = new C();
3     public void método1() {
4         objeto3.instrucción1DeMétodo1();
5         objeto3.instrucción2DeMétodo1();
6         invocaciónInterna();
7     }
8     public void invocaciónInterna() {
9         System.out.println("método interno");
10    }
11 }
```

Clase C:

```
1 public class C {
2     public void instrucción1DeMétodo1() {
3         System.out.println("instrucción 1");
4     }
5     public void instrucción2DeMétodo1() {
6         System.out.println("instrucción 2");
7     }
8 }
```

3.2. Inserción de Datos.

En estos ejemplos se utiliza la tabla proveedor creada en los capítulos anteriores. El comportamiento de la ventana se muestra a continuación.

3.2.1. Funcionamiento.

Al abrir la aplicación aparece una ventana como la siguiente.

Archivos de Windows

Agregar Nuevos Proveedores

Archivo

Clave:

Nombre:

Teléfono:

Correo Electrónico:

Fecha Inicial:

Dirección:

El menú “Archivo” contiene las siguientes opciones:

- **Agregar.** Para agregar los datos capturados como un nuevo registro en la base de datos.
- **Salir.** Para salir de la aplicación.

Para usarla se sigue el siguiente procedimiento:

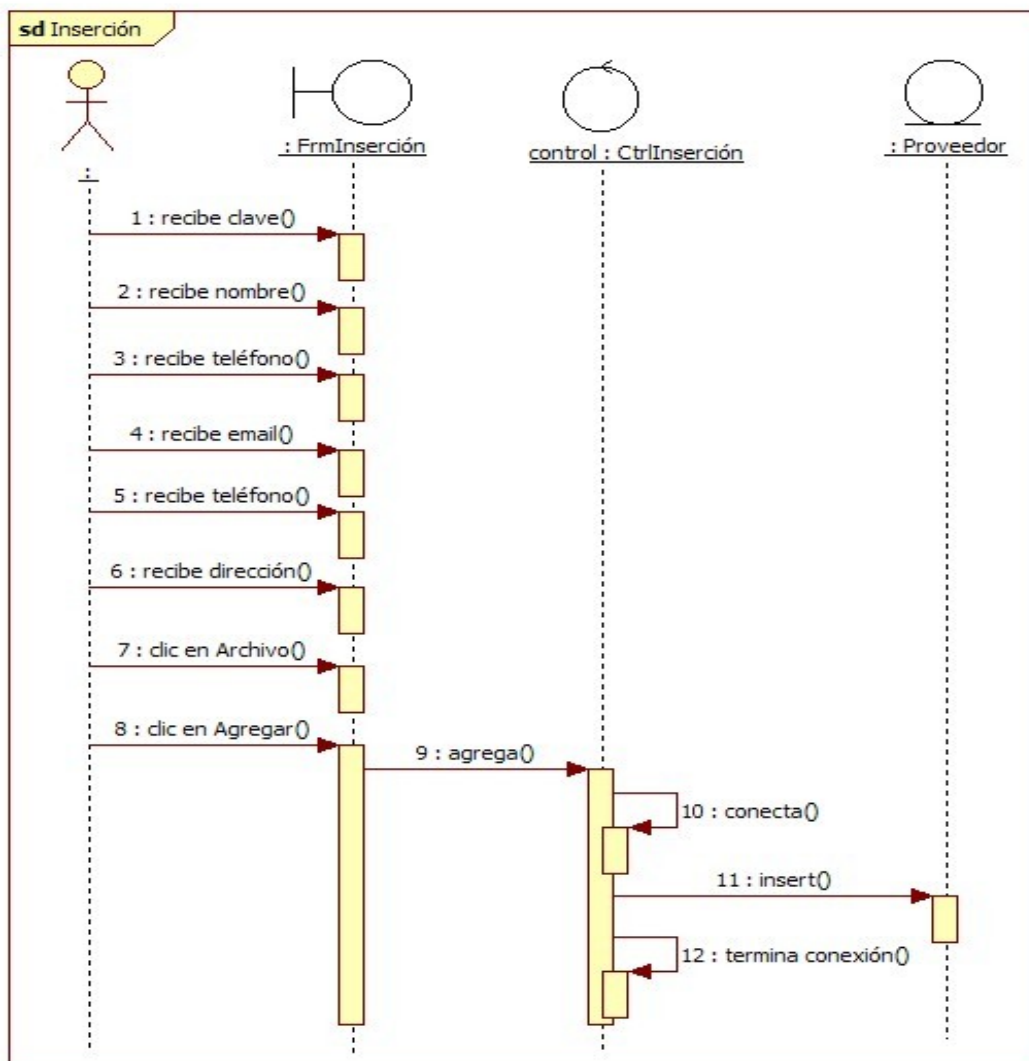
1. Se introduce el valor para los campos:
 - clave
 - nombre
 - teléfono
 - correo electrónico
 - fecha inicial (es la fecha de la primera compra realizada)
 - dirección
2. Se selecciona el menú “Archivo”.
3. Se selecciona la opción “Agregar”.
4. La aplicación intenta almacenar los datos creando un nuevo registro. En caso de éxito se muestra un cuadro de mensaje que despliega el mensaje “Proveedor Agregado”. En caso de que no se pueda realizar exitosamente la operación, muestra un cuadro de error que despliega la razón del fallo.

3.2.2. Arquitectura de Tres Capas.

En el diseño de las clases se utiliza un esquema conocido como tres capas, donde se utilizan tres clases para realizar cada función del programa. Son las siguientes:

- **boundary.** Se encargan de interactuar con los usuarios y de la lógica de la aplicación. Utilizan el prefijo Frm.
- **control.** Se encargan de manejar el acceso a la base de datos y asegurar las reglas del negocio. Utilizan el prefijo Ctrl.
- **entity.** Se encargan de almacenar los datos.

3.2.3. Diagrama de Secuencia.



El diagrama muestra los pasos a realizar, pero algunos de ellos son solo descriptivos; muestran lo que se debe realizar, por ejemplo en los pasos 1 a 6, aunque no necesariamente se genere un método con los nombres indicados.

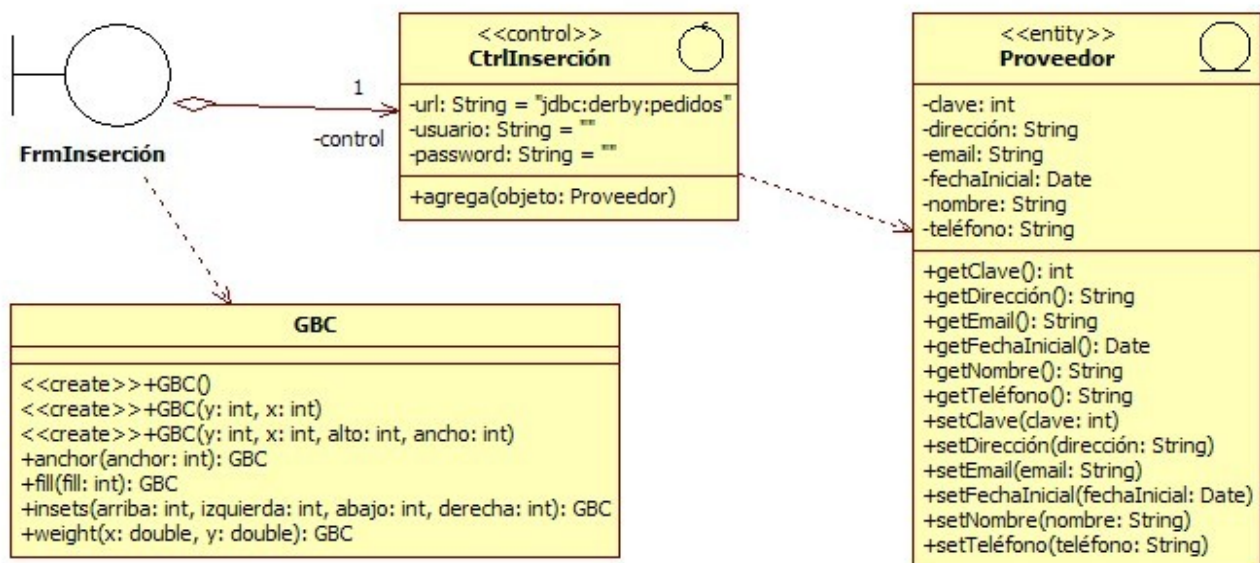
El muñeco de palo se conoce como actor y representa alguien externo al programa. Puede ser otra clase, una persona u otro sistema.

La arquitectura de tres capas está presente aquí. La clase entity es Proveedor, la clase control es CtrlInserción y la clase boundary es FrmInserción. Los íconos mostrados son los típicos para estos tipos de clases.

En este dibujo, la clase entity representa a la tabla Proveedor. El paso 11 representa la operación insert sobre la tabla.

3.2.4. Diagrama de Clases.

Se utiliza la clase GBC para simplificar la construcción de interfaces utilizando el GridBagLayout.



Además de la tabla para la clase proveedor, se crea una clase que contiene los mismos campos, más operaciones set y get para modificar y consultar, respectivamente, estos valores.

3.2.5. Código de Proveedor.

```
1 import java.util.Date;
2
3 /**
4  * Representa los datos almacenados de cada Proveedor.
5  */
6 public class Proveedor {
7     private int clave;
8     private String dirección;
9     private String email;
10    private Date fechaInicial;
11    private String nombre;
12    private String teléfono;
13    public int getClave() {
14        return clave;
15    }
16    public String getDirección() {
17        return dirección;
18    }
19    public String getEmail() {
20        return email;
21    }
22    public Date getFechaInicial() {
23        return fechaInicial;
24    }
25    public String getNombre() {
26        return nombre;
27    }
28    public String getTeléfono() {
29        return teléfono;
30    }
31    public void setClave(int clave) {
32        this.clave = clave;
33    }
34    public void setDirección(String dirección) {
35        this.dirección = dirección;
36    }
37    public void setEmail(String email) {
38        this.email = email;
39    }
40    public void setFechaInicial(Date fechaInicial) {
41        this.fechaInicial = fechaInicial;
42    }
43    public void setNombre(String nombre) {
44        this.nombre = nombre;
45    }
46    public void setTeléfono(String teléfono) {
47        this.teléfono = teléfono;
48    }
49 }
```

3.2.5. Código de CtrlInserción.

```

1 import java.sql.Connection;
2 import java.sql.Date;
3 import java.sql.DriverManager;
4 import java.sql.PreparedStatement;
5 import java.sql.SQLException;
6
7 public class CtrlInserción {
8     private final String url = "jdbc:derby:proveedores";
9     private final String usuario = "";
10    private final String password = "";
11    /**
12     * Agrega la información a la que apunta objeto.
13     *
14     * @param objeto
15     *      Referencia al objeto que contiene los datos
16     * @throws SQLException
17     *      Si algo falla.
18     */
19    public void agrega(Proveedor objeto) throws SQLException {
20        Connection c = null;
21        PreparedStatement ps = null;
22        try {
23            c = DriverManager.getConnection(url, usuario, password);
24            ps = c.prepareStatement("INSERT INTO Proveedor "
25                + "(prov_clave, prov_nombre, prov_telefono, "
26                + " prov_email, prov_direccion, prov_fecha) "
27                + "VALUES(?, ?, ?, ?, ?, ?)");
28            ps.setInt(1, objeto.getClave());
29            ps.setString(2, objeto.getNombre());
30            ps.setString(3, objeto.getTeléfono());
31            ps.setString(4, objeto.getEmail());
32            ps.setString(5, objeto.getDirección());
33            ps.setDate(6, new Date(objeto.getFechaInicial().getTime()));
34            ps.executeUpdate();
35        } finally {
36            if (ps != null) {
37                try {
38                    ps.close();
39                } catch (SQLException e) {}
40            }
41            if (c != null) {
42                try {
43                    c.close();
44                } catch (SQLException e) {}
45            }
46        }
47    }
48 }

```

3.2.6. Código de GBC.

Esta clase se utiliza para facilitar el uso del GridBagLayout.

```

1  /* Copyright 2016 Ricardo Armando Machorro Reyes
2  *
3  * Licensed under the Apache License, Version 2.0 (the "License");
4  * you may not use this file except in compliance with the License.
5  * You may obtain a copy of the License at
6  *
7  *     http://www.apache.org/licenses/LICENSE-2.0
8  *
9  * Unless required by applicable law or agreed to in writing, software
10 * distributed under the License is distributed on an "AS IS" BASIS,
11 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 * See the License for the specific language governing permissions and
13 * limitations under the License.
14 */
15 import java.awt.GridBagConstraints;
16
17 /**
18  * Esta clase está hecha para simplificar el uso de GridBag Constraints.
19  */
20 public class GBC extends GridBagConstraints {
21     /**
22      * Crea un objeto de GBC alineado a la esquina superior izquierda de la
23      * celda.
24      */
25     public GBC() {
26         anchor = BASELINE_LEADING;
27     }
28     /**
29      * Crea un objeto de GBC de una celda de alto y ancho. El componente que
30      * contiene está alineado a la esquina superior izquierda de la celda.
31      *
32      * @param y
33      *         Renglón de la celda.
34      * @param x
35      *         Columna de la celda.
36      */
37     public GBC(int y, int x) {
38         gridy = y;
39         gridx = x;
40         anchor = BASELINE_LEADING;
41     }
42     /**
43      * Crea una instancia de GBC. El componente que contiene está alineado a
44      * la esquina superior izquierda de la celda.
45      *
46      * @param y
47      *         Renglón de la celda.
48      * @param x
49      *         Columna de la celda.
50      * @param alto

```



```

51      *           Número de renglones que abarca el componente.
52      * @param ancho
53      *           Número de columnas que abarca el componente.
54      */
55      public GBC(int y, int x, int alto, int ancho) {
56          gridy = y;
57          gridx = x;
58          gridheight = alto;
59          gridwidth = ancho;
60          anchor = BASELINE_LEADING;
61      }
62      /**
63       * Asigna la alineación del componente
64       *
65       * @param anchor
66       *           Alineación del componente.
67       *
68       * @return El mismo objeto de GBC que recibió el mensaje.
69       */
70      public GBC anchor(int anchor) {
71          this.anchor = anchor;
72          return this;
73      }
74      /**
75       * Asigna la forma de llenar la celda cuando el componente que contiene
76       * es más pequeño que esta:
77       *
78       * @param fill
79       *           Forma de llenar la celda.
80       *
81       * @return El mismo objeto de GBC que recibió el mensaje.
82       */
83      public GBC fill(int fill) {
84          this.fill = fill;
85          return this;
86      }
87      /**
88       * Asigna el ancho del borde de la celda.
89       *
90       * @param arriba
91       *           Número de pixeles del borde superior.
92       * @param izquierda
93       *           Número de pixeles del borde izquierdo.
94       * @param abajo
95       *           Número de pixeles del borde inferior.
96       * @param derecha
97       *           Número de pixeles del borde derecho.
98       * @return El mismo objeto de GBC que recibió el mensaje.
99       */
100     public GBC insets(int arriba, int izquierda, int abajo, int derecha) {
101         insets.top = arriba;
102         insets.left = izquierda;
103         insets.bottom = abajo;
104         insets.right = derecha;

```

```

105         return this;
106     }
107     /**
108     * Asigna la forma de crecer de la celda.
109     *
110     * @param x
111     *          Forma de crecer horizontalmente.
112     * @param y
113     *          Forma de crecer verticalmente.
114     * @return El mismo objeto de GBC que recibió el mensaje.
115     */
116     public GBC weight(double x, double y) {
117         weightx = x;
118         weighty = y;
119         return this;
120     }
121 }

```

3.2.7. Diseño de la Forma.

	0	1
0	Clave:	<input type="text"/>
1	Nombre:	<input type="text"/>
2	Teléfono:	<input type="text"/>
3	Correo Electrónico:	<input type="text"/>
4	Fecha Inicial:	<input type="text"/>
5	Dirección:	<input type="text"/>

El campo nombre crece horizontalmente al cambiar el tamaño de la ventana y el campo dirección crece a lo largo y ancho.

3.2.8 Código de FrmInserción.

```

1  import java.awt.GridBagLayout;
2  import java.awt.event.ActionEvent;
3  import java.awt.event.ActionListener;
4  import java.sql.SQLException;
5  import java.text.NumberFormat;
6  import java.text.ParseException;
7  import java.text.SimpleDateFormat;
8  import java.util.Date;
9  import javax.swing.JFormattedTextField;
10 import javax.swing.JFrame;
11 import javax.swing.JLabel;
12 import javax.swing.JMenu;
13 import javax.swing.JMenuBar;
14 import javax.swing.JMenuItem;
15 import javax.swing.JOptionPane;
16 import javax.swing.JPanel;
17 import javax.swing.JScrollPane;
18 import javax.swing.JTextArea;
19 import javax.swing.JTextField;
20 import javax.swing.SwingUtilities;
21
22 public class FrmInserción extends JFrame implements ActionListener {
23     public static void main(String[] args) {
24         try {
25             Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
26             SwingUtilities.invokeLater(new Runnable() {
27                 public void run() {
28                     new FrmInserción().setVisible(true);
29                 }
30             });
31         } catch (Exception e) {
32             e.printStackTrace();
33         }
34     }
35     // Crea el objeto que se conecta a la base de datos
36     private final CtrlInserción control = new CtrlInserción();
37     private final JLabel lblClave = new JLabel("Clave:");
38     private final NumberFormat fmt = NumberFormat.getInstance();
39     private final JFormattedTextField txtClave = new JFormattedTextField(
40         fmt);
41     {
42         txtClave.setColumns(11);
43         txtClave.setHorizontalAlignment(JTextField.TRAILING);
44     }
45     private final JLabel lblNombre = new JLabel("Nombre:");
46     private final JTextField txtNombre = new JTextField(50);
47     private final JLabel lblTeléfono = new JLabel("Teléfono:");
48     private final JTextField txtTeléfono = new JTextField(50);
49     private final JLabel lblEmail = new JLabel("Correo Electrónico:");
50     private final JTextField txtEmail = new JTextField(50);
51     private final JLabel lblFecha = new JLabel("Fecha Inicial:");
52     private final SimpleDateFormat fmtF = new SimpleDateFormat("dd/MM/yyyy");

```

```

53     private final JFormattedTextField txtFecha = new JFormattedTextField(
54         fmtF);
55     {
56         txtFecha.setColumns(10);
57     }
58     private final JLabel lblDirección = new JLabel("Dirección:");
59     private final JTextArea txtDirección = new JTextArea(3, 50);
60     private final JMenuItem itmAgregar = new JMenuItem("Agregar");
61     {
62         itmAgregar.addActionListener(this);
63     }
64     private final JMenuItem itmSalir = new JMenuItem("Salir");
65     {
66         itmSalir.addActionListener(this);
67     }
68     private final JMenu menuArchivo = new JMenu("Archivo");
69     {
70         menuArchivo.add(itmAgregar);
71         menuArchivo.add(itmSalir);
72     }
73     private final JMenuBar menuBar = new JMenuBar();
74     {
75         menuBar.add(menuArchivo);
76     }
77     private final JPanel panel = new JPanel(new GridBagLayout());
78     {
79         panel.add(lblClave, new GBC(0, 0).insets(12, 12, 12, 12));
80         panel.add(txtClave, new GBC(0, 1).insets(12, 0, 12, 12));
81         panel.add(lblNombre, new GBC(1, 0).insets(0, 12, 12, 12));
82         panel.add(txtNombre, new GBC(1, 1).insets(0, 0, 12, 12).weight(1.0,
83             0.0).fill(GBC.HORIZONTAL));
84         panel.add(lblTeléfono, new GBC(2, 0).insets(0, 12, 12, 12));
85         panel.add(txtTeléfono, new GBC(2, 1).insets(0, 0, 12, 12).weight(
86             1.0, 0.0));
87         panel.add(lblEmail, new GBC(3, 0).insets(0, 12, 12, 12));
88         panel.add(txtEmail, new GBC(3, 1).insets(0, 0, 12, 12).weight(1.0,
89             0.0));
90         panel.add(lblFecha, new GBC(4, 0).insets(0, 12, 12, 12));
91         panel.add(txtFecha, new GBC(4, 1).insets(0, 0, 12, 12));
92         panel.add(lblDirección, new GBC(5, 0).insets(0, 12, 12, 12));
93         panel.add(new JScrollPane(txtDirección), new GBC(5, 1).insets(0, 0,
94             12, 12).fill(GBC.BOTH).weight(1.0, 1.0));
95     }
96     public FrmInserción() {
97         setTitle("Agregar Nuevos Proveedores");
98         setJMenuBar(menuBar);
99         setContentPane(new JScrollPane(panel));
100        setDefaultCloseOperation(EXIT_ON_CLOSE);
101        pack();
102    }
103    public void actionPerformed(ActionEvent evt) {
104        Object origen = evt.getSource();
105        if (origen == itmAgregar) {
106            agrega();

```

```

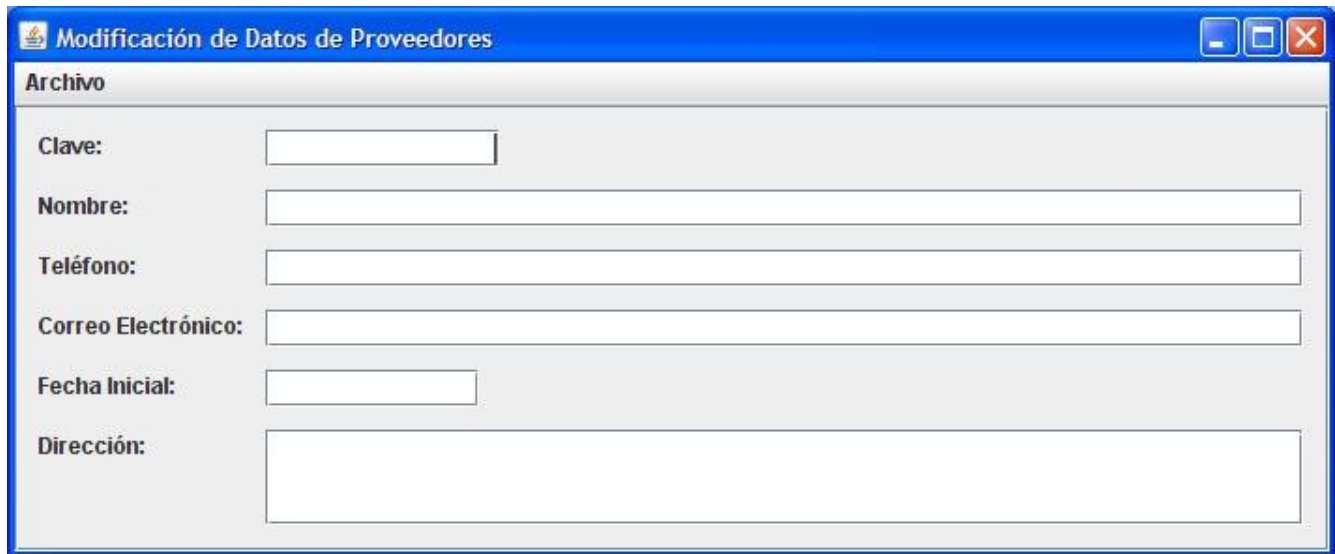
107     } else if (origen == itmSalir) {
108         System.exit(0);
109     }
110 }
111 private void agrega() {
112     try {
113         // Obtiene la información a enviar a la base de datos
114         String nombre = txtNombre.getText().trim();
115         String teléfono = txtTeléfono.getText().trim();
116         String email = txtEmail.getText().trim();
117         String dirección = txtDirección.getText().trim();
118         // Valida la clave
119         try {
120             txtClave.commitEdit();
121         } catch (ParseException e) {
122             throw new SQLException("Clave Incorrecta");
123         }
124         // Valida la fecha
125         try {
126             txtFecha.commitEdit();
127         } catch (ParseException e) {
128             throw new SQLException("Fecha Incorrecta");
129         }
130         Number clave = (Number) txtClave.getValue();
131         Date fecha = (Date) txtFecha.getValue();
132         // Crea el objeto que contiene la información a guardar
133         Proveedor pro = new Proveedor();
134         // Asigna la información a guardar
135         pro.setClave(clave.intValue());
136         pro.setNombre(nombre);
137         pro.setTeléfono(teléfono);
138         pro.setEmail(email);
139         pro.setDirección(dirección);
140         pro.setFechaInicial(fecha);
141         // Guarda los datos
142         control.agrega(pro);
143         JOptionPane.showMessageDialog(this, "Proveedor Agregado");
144     } catch (SQLException e) {
145         e.printStackTrace();
146         error(e.getMessage());
147     }
148 }
149 private void error(String mensaje) {
150     JOptionPane.showMessageDialog(this, mensaje, "Error",
151         JOptionPane.ERROR_MESSAGE);
152 }
153 }

```

3.3. *Búsqueda y Modificación.*

3.3.1. Funcionamiento.

Al abrir la aplicación aparece una ventana como la siguiente. La forma tiene el mismo diseño que en la inserción.



The screenshot shows a Windows-style application window titled "Modificación de Datos de Proveedores". It features a menu bar with the option "Archivo". Below the menu bar, there are six text input fields, each preceded by a label: "Clave:", "Nombre:", "Teléfono:", "Correo Electrónico:", "Fecha Inicial:", and "Dirección:". The fields are arranged vertically and are currently empty.

El menú “Archivo” contiene las siguientes opciones:

- **Buscar.** Para recuperar la información de la base de datos que se desea modificar.
- **Guardar.** Para guardar las modificaciones realizadas.
- **Salir.** Para salir de la aplicación.

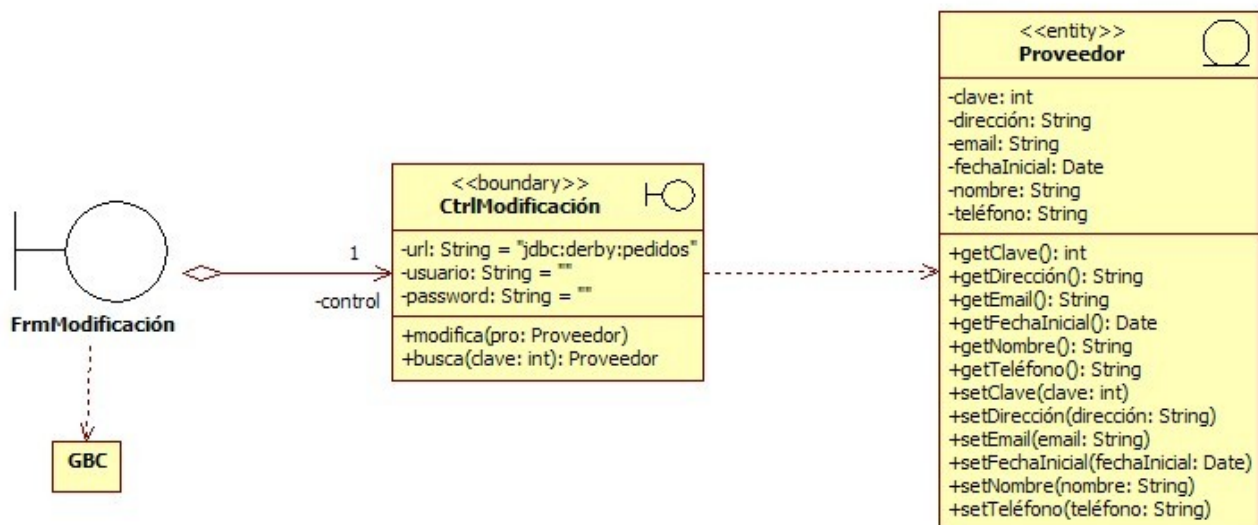
Para usarla se sigue el siguiente procedimiento:

1. Se introduce la clave del proveedor a modificar.
2. Se selecciona el menú “Archivo”.
3. Se selecciona la opción “Buscar”.
4. La aplicación busca en la base de datos el proveedor correspondiente a la clave introducida. En caso de éxito muestra su información en los campos de texto correspondientes. En caso de que no se pueda realizar exitosamente la operación, muestra un cuadro de error que despliega la razón del fallo.
5. Se introduce el valor para los campos (la clave no se modifica porque es la llave primaria):
 - nombre
 - teléfono

- correo electrónico
 - fecha inicial (es la fecha de la primera compra realizada)
 - dirección
6. Se selecciona el menú “Archivo”.
 7. Se selecciona la opción “Guardar”.
 8. La aplicación intenta almacenar los cambios en los datos. En caso de éxito se muestra un cuadro de mensaje que despliega el mensaje “Proveedor Modificado”. En caso de que no se pueda realizar exitosamente la operación, muestra un cuadro de error que despliega la razón del fallo.

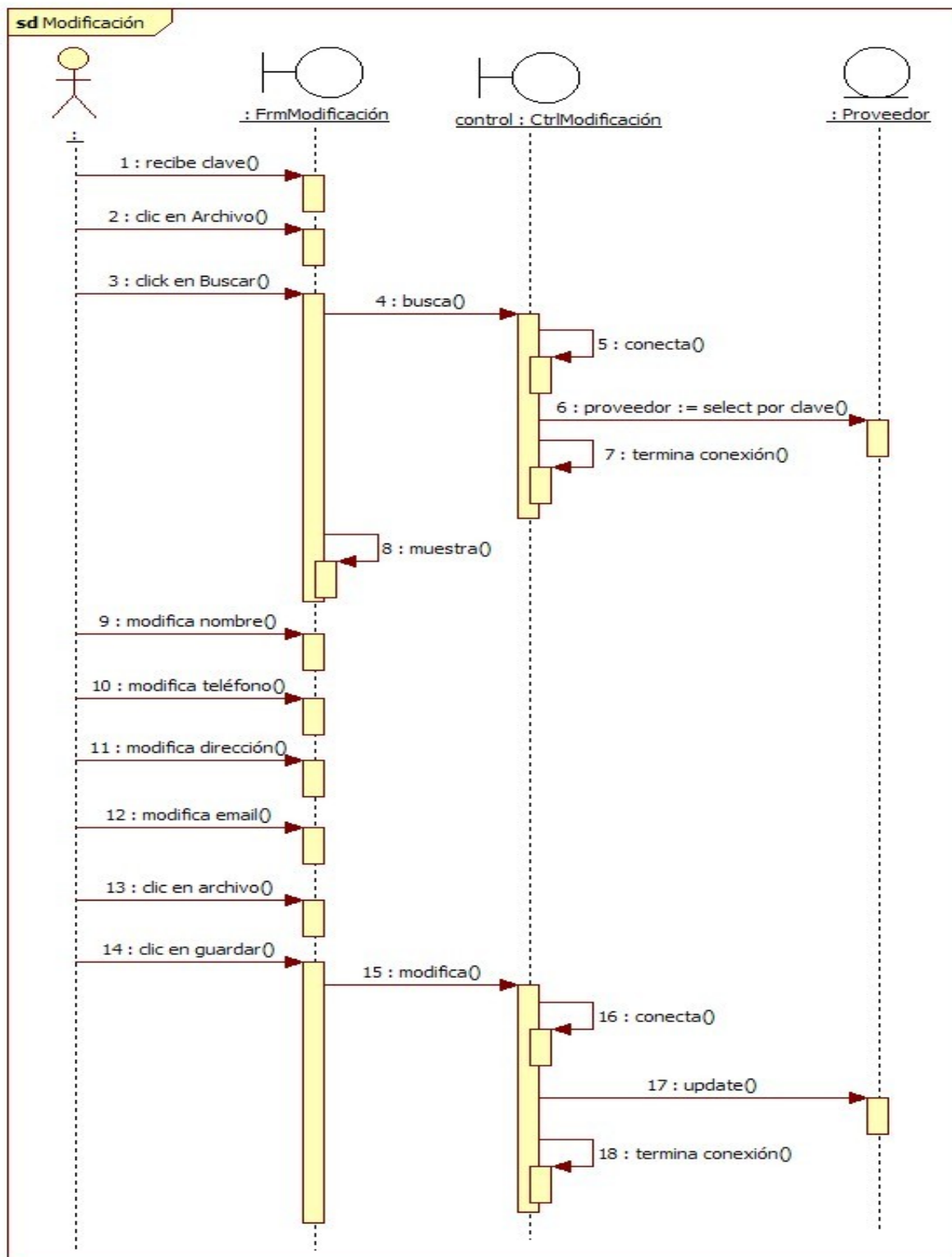
3.3.2. Diagrama de Clases.

Como la clase GBC ya se conoce, se omiten sus detalles.



La estructura es parecida a la inserción, pero aquí se contemplan los métodos para buscar y modificar.

3.3.3. Diagrama de Secuencia.



3.3.4. Código de CtrlModificación.

```

1  import java.sql.Connection;
2  import java.sql.Date;
3  import java.sql.DriverManager;
4  import java.sql.PreparedStatement;
5  import java.sql.ResultSet;
6  import java.sql.SQLException;
7
8  public class CtrlModificación {
9      private final String url = "jdbc:derby:proveedores";
10     private final String usuario = "";
11     private final String password = "";
12     /**
13      * Devuelve el proveedor que corresponde a la clave recibida.
14      *
15      * @param clave
16      *         Clave de proveedor a buscar.
17      * @return El proveedor que corresponde a la clave.
18      * @throws SQLException
19      *         Si esa clave no está registrada o hay algún fallo.
20      */
21     public Proveedor busca(int clave) throws SQLException {
22         Connection c = null;
23         PreparedStatement ps = null;
24         ResultSet rs = null;
25         try {
26             // Establece la conexión.
27             c = DriverManager.getConnection(url, usuario, password);
28             // busca el objeto.
29             ps = c.prepareStatement("SELECT * FROM Proveedor "
30                                   + "WHERE prov_clave = ?");
31             ps.setInt(1, clave);
32             rs = ps.executeQuery();
33             // Verifica si se encuentra el objeto.
34             if (rs.next()) {
35                 /*
36                  * Como si se vanza al primer renglón del resultado, si hay
37                  * proveedor para esa clave.
38                  */
39                 Proveedor pro = new Proveedor();
40                 pro.setClave(clave);
41                 pro.setNombre(rs.getString("prov_nombre"));
42                 pro.setTeléfono(rs.getString("prov_telefono"));
43                 pro.setEmail(rs.getString("prov_email"));
44                 pro.setDirección(rs.getString("prov_direccion"));
45                 pro.setFechaInicial(rs.getDate("prov_fecha"));
46                 return pro;
47             } else {
48                 /*
49                  * Como no se vanza al primer renglón del resultado, no hay

```

```

50         * proveedor para esa clave. Lanza la excepción.
51         */
52         throw new SQLException("Proveedor no encontrado");
53     }
54     } finally {
55         // Termina la conexión.
56         if (rs != null) {
57             try {
58                 rs.close();
59             } catch (SQLException e) {}
60         }
61         if (ps != null) {
62             try {
63                 ps.close();
64             } catch (SQLException e) {}
65         }
66         if (c != null) {
67             try {
68                 c.close();
69             } catch (SQLException e) {}
70         }
71     }
72 }
73 /**
74  * Guarda en la base de datos el propiedades del producto recibido. Los
75  * valores se guardan en el registro cuyo campo "clave" coincida con el
76  * valor de la propiedad "clave" del objeto.
77  *
78  * @param objeto
79  *         El producto que se desea guardar.
80  * @throws SQLException
81  *         Si se presenta algún fallo.
82  */
83 public void modifica(Proveedor pro) throws SQLException {
84     Connection c = null;
85     PreparedStatement ps = null;
86     try {
87         // Establece la conexión.
88         c = DriverManager.getConnection(url, usuario, password);
89         // Busca el registro y realiza los cambios.
90         ps = c.prepareStatement("UPDATE Proveedor "
91             + "SET prov_nombre = ?, prov_telefono = ?, "
92             + "prov_email = ?, prov_direccion = ?, "
93             + "prov_fecha = ? WHERE prov_clave = ?");
94         ps.setString(1, pro.getNombre());
95         ps.setString(2, pro.getTeléfono());
96         ps.setString(3, pro.getEmail());
97         ps.setString(4, pro.getDirección());
98         ps.setDate(5, new Date(pro.getFechaInicial().getTime()));
99         ps.setInt(6, pro.getClave());
100        ps.executeUpdate();
101    } finally {
102        // Termina la conexión.
103        if (ps != null) {

```

```

104         try {
105             ps.close();
106         } catch (SQLException e) {}
107     }
108     if (c != null) {
109         try {
110             c.close();
111         } catch (SQLException e) {}
112     }
113 }
114 }
115 }

```

3.3.5. Código de FrmModificación.

```

1  import java.awt.GridBagLayout;
2  import java.awt.event.ActionEvent;
3  import java.awt.event.ActionListener;
4  import java.sql.SQLException;
5  import java.text.NumberFormat;
6  import java.text.ParseException;
7  import java.text.SimpleDateFormat;
8  import java.util.Date;
9  import javax.swing.JFormattedTextField;
10 import javax.swing.JFrame;
11 import javax.swing.JLabel;
12 import javax.swing.JMenu;
13 import javax.swing.JMenuBar;
14 import javax.swing.JMenuItem;
15 import javax.swing.JOptionPane;
16 import javax.swing.JPanel;
17 import javax.swing.JScrollPane;
18 import javax.swing.JTextArea;
19 import javax.swing.JTextField;
20 import javax.swing.SwingUtilities;
21
22 public class FrmModificación extends JFrame implements ActionListener {
23     public static void main(String[] args) {
24         try {
25             Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
26             SwingUtilities.invokeLater(new Runnable() {
27                 public void run() {
28                     new FrmModificación().setVisible(true);
29                 }
30             });
31         } catch (Exception e) {
32             e.printStackTrace();
33         }
34     }
35     private final CtrlModificación control = new CtrlModificación();
36     private final JLabel lblClave = new JLabel("Clave:");
37     private final NumberFormat fmt = NumberFormat.getInstance();

```

```

38     private final JFormattedTextField txtClave = new JFormattedTextField(
39         fmt);
40     {
41         txtClave.setColumns(11);
42         txtClave.setHorizontalAlignment(JTextField.TRAILING);
43     }
44     private final JLabel lblNombre = new JLabel("Nombre:");
45     private final JTextField txtNombre = new JTextField(50);
46     private final JLabel lblTeléfono = new JLabel("Teléfono:");
47     private final JTextField txtTeléfono = new JTextField(50);
48     private final JLabel lblEmail = new JLabel("Correo Electrónico:");
49     private final JTextField txtEmail = new JTextField(50);
50     private final JLabel lblFecha = new JLabel("Fecha Inicial:");
51     private final SimpleDateFormat fmtF = new SimpleDateFormat("dd/MM/yyyy");
52     private final JFormattedTextField txtFecha = new JFormattedTextField(
53         fmtF);
54     {
55         txtFecha.setColumns(10);
56     }
57     private final JLabel lblDirección = new JLabel("Dirección:");
58     private final JTextArea txtDirección = new JTextArea(3, 50);
59     private final JMenuItem itmBuscar = new JMenuItem("Buscar");
60     {
61         itmBuscar.addActionListener(this);
62     }
63     private final JMenuItem itmGuardar = new JMenuItem("Guardar");
64     {
65         itmGuardar.addActionListener(this);
66     }
67     private final JMenuItem itmSalir = new JMenuItem("Salir");
68     {
69         itmSalir.addActionListener(this);
70     }
71     private final JMenu menuArchivo = new JMenu("Archivo");
72     {
73         menuArchivo.add(itmBuscar);
74         menuArchivo.add(itmGuardar);
75         menuArchivo.add(itmSalir);
76     }
77     private final JMenuBar menuBar = new JMenuBar();
78     {
79         menuBar.add(menuArchivo);
80     }
81     private final JPanel panel = new JPanel(new GridBagLayout());
82     {
83         panel.add(lblClave, new GBC(0, 0).insets(12, 12, 12, 12));
84         panel.add(txtClave, new GBC(0, 1).insets(12, 0, 12, 12));
85         panel.add(lblNombre, new GBC(1, 0).insets(0, 12, 12, 12));
86         panel.add(txtNombre, new GBC(1, 1).insets(0, 0, 12, 12).weight(1.0,
87             0.0).fill(GBC.HORIZONTAL));
88         panel.add(lblTeléfono, new GBC(2, 0).insets(0, 12, 12, 12));
89         panel.add(txtTeléfono, new GBC(2, 1).insets(0, 0, 12, 12).weight(
90             1.0, 0.0));
91         panel.add(lblEmail, new GBC(3, 0).insets(0, 12, 12, 12));

```

```

92     panel.add(txtEmail, new GBC(3, 1).insets(0, 0, 12, 12).weight(1.0,
93         0.0));
94     panel.add(lblFecha, new GBC(4, 0).insets(0, 12, 12, 12));
95     panel.add(txtFecha, new GBC(4, 1).insets(0, 0, 12, 12));
96     panel.add(lblDirección, new GBC(5, 0).insets(0, 12, 12, 12));
97     panel.add(new JScrollPane(txtDirección), new GBC(5, 1).insets(0, 0,
98         12, 12).fill(GBC.BOTH).weight(1.0, 1.0));
99 }
100 public FrmModificación() {
101     setTitle("Modificación de Datos de Proveedores");
102     setJMenuBar(menuBar);
103     setContentPane(new JScrollPane(panel));
104     setDefaultCloseOperation(EXIT_ON_CLOSE);
105     pack();
106 }
107 public void actionPerformed(ActionEvent evt) {
108     Object origen = evt.getSource();
109     if (origen == itmBuscar) {
110         busca();
111     } else if (origen == itmGuardar) {
112         guarda();
113     } else if (origen == itmSalir) {
114         System.exit(0);
115     }
116 }
117 private void busca() {
118     try {
119         txtClave.commitEdit();
120         Number clave = (Number) txtClave.getValue();
121         Proveedor pro = control.busca(clave.intValue());
122         txtNombre.setText(pro.getNombre());
123         txtTeléfono.setText(pro.getTeléfono());
124         txtEmail.setText(pro.getEmail());
125         txtDirección.setText(pro.getDirección());
126         txtFecha.setValue(pro.getFechaInicial());
127     } catch (ParseException e) { // generado por el commit edit
128         error("Clave Incorrecta");
129         e.printStackTrace();
130     } catch (SQLException e) {
131         e.printStackTrace();
132         error(e.getMessage());
133     }
134 }
135 private void error(String mensaje) {
136     JOptionPane.showMessageDialog(this, mensaje, "Error",
137         JOptionPane.ERROR_MESSAGE);
138 }
139 private void guarda() {
140     try {
141         String nombre = txtNombre.getText().trim();
142         String teléfono = txtTeléfono.getText().trim();
143         String email = txtEmail.getText().trim();
144         String dirección = txtDirección.getText().trim();
145         try {

```

```

146         txtClave.commitEdit();
147     } catch (ParseException e) {
148         throw new SQLException("Clave Incorrecta");
149     }
150     try {
151         txtFecha.commitEdit();
152     } catch (ParseException e) {
153         throw new SQLException("Fecha Incorrecta");
154     }
155     Number clave = (Number) txtClave.getValue();
156     Date fecha = (Date) txtFecha.getValue();
157     Proveedor pro = new Proveedor();
158     pro.setClave(clave.intValue());
159     pro.setNombre(nombre);
160     pro.setTeléfono(teléfono);
161     pro.setEmail(email);
162     pro.setDirección(dirección);
163     pro.setFechaInicial(fecha);
164     control.modifica(pro);
165     JOptionPane.showMessageDialog(this, "Proveedor Modificado");
166 } catch (SQLException e) {
167     e.printStackTrace();
168     error(e.getMessage());
169 }
170 }
171 }

```

3.4. Eliminación.

3.4.1. Funcionamiento.

Al abrir la aplicación aparece una ventana como la siguiente. La forma tiene el mismo diseño que en la inserción.

Es importante visualizar la información antes de borrarla para asegurar que realmente es lo que se desea eliminar. Aún más; después de seleccionar la eliminación, se pide confirmar el borrado.

El único dato que se puede modificar en esta forma es la clave.

El menú “Archivo” contiene las siguientes opciones:

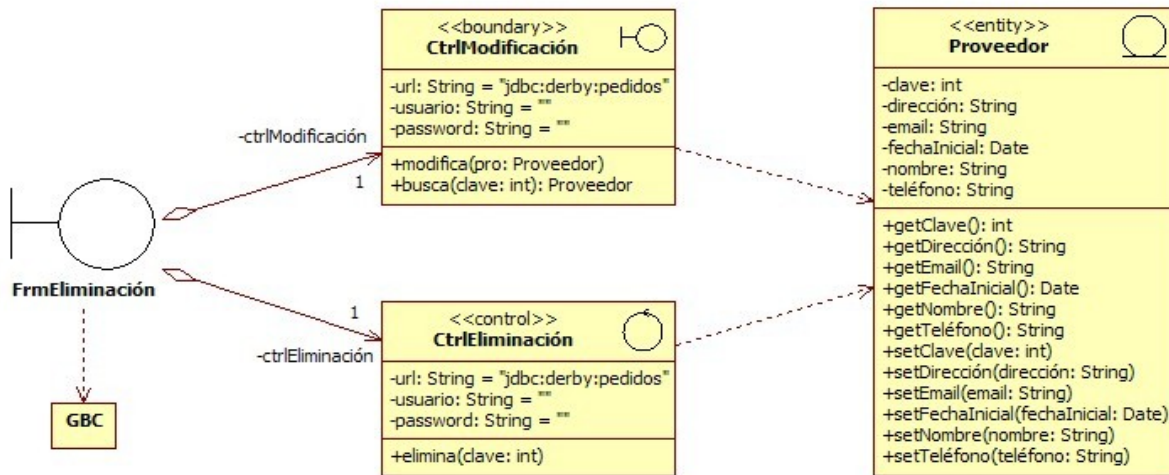
- **Buscar.** Para recuperar la información de la base de datos que se desea eliminar.
- **Eliminar.** Para eliminar el proveedor mostrado.
- **Salir.** Para salir de la aplicación.

Para usarla se sigue el siguiente procedimiento:

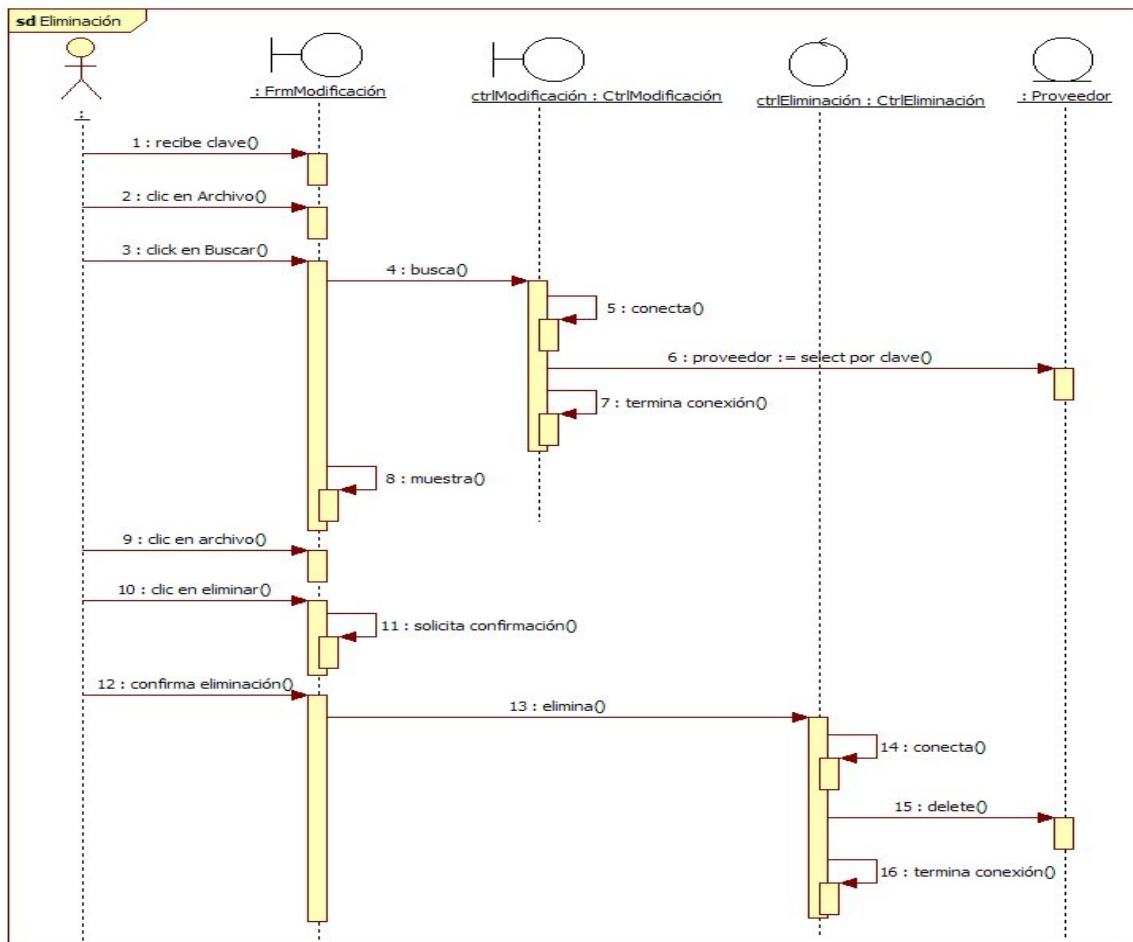
1. Se introduce la clave del proveedor a modificar.
2. Se selecciona el menú “Archivo”.
3. Se selecciona la opción “Buscar”.
4. La aplicación busca en la base de datos el proveedor correspondiente a la clave introducida. En caso de éxito muestra su información en los campos de texto correspondientes. En caso de que no se pueda realizar exitosamente la operación, muestra un cuadro de error que despliega la razón del fallo.
5. Se selecciona el menú “Archivo”.
6. Se selecciona la opción “Eliminar”.
7. Se presenta un cuadro de confirmación para ratificar la decisión de borrar el registro. Las opciones que presenta son “Sí”, “No” o “Cancelar”.
8. Cuando el usuario contesta “No” o “Cancelar” se detiene la operación. Cuando contesta “Sí”, avanza al paso 9.
9. La aplicación intenta eliminar el registro seleccionado. En caso de éxito se muestra un cuadro de mensaje que despliega el mensaje “Proveedor Eliminado”. En caso de que no se pueda realizar exitosamente la operación, muestra un cuadro de error que despliega la razón del fallo.

3.4.2. Diagrama de Clases.

Como la clase `CtrlModificación` ya contiene una función de búsqueda, no es necesario volver a programarla. Simplemente se utiliza una instancia de esta clase.



3.4.3. Diagrama de Secuencia.



3.4.4. Código de CtrlEliminación.

```

1  import java.sql.Connection;
2  import java.sql.DriverManager;
3  import java.sql.PreparedStatement;
4  import java.sql.SQLException;
5
6  public class CtrlEliminación {
7      private final String url = "jdbc:derby:proveedores";
8      private final String usuario = "";
9      private final String password = "";
10     /**
11      * Elimina de la base de datos el producto que corresponde a la clave
12      * recibida.
13      *
14      * @param clave
15      *         Clave del producto a eliminar.
16      * @throws SQLException
17      *         Si no existe un objeto con esa clave o se presenta
18      *         algún fallo.
19      */
20     public void elimina(int clave) throws SQLException {
21         Connection c = null;
22         PreparedStatement ps = null;
23         try {
24             // Establece la conexión.
25             c = DriverManager.getConnection(url, usuario, password);
26             /*
27              * Busca el proveedor y lo elimina. Como resultado, la variable
28              * "eliminados" contiene el número de registros eliminados.
29              */
30             ps = c.prepareStatement("DELETE FROM Proveedor "
31                                     + "WHERE prov_clave = ?");
32             ps.setInt(1, clave);
33             int eliminados = ps.executeUpdate();
34             if (eliminados == 0) {
35                 throw new SQLException("Proveedor no Encontrado");
36             }
37         } finally {
38             // Termina la conexión.
39             if (ps != null) {
40                 try {
41                     ps.close();
42                 } catch (SQLException e) {}
43             }
44             if (c != null) {
45                 try {
46                     c.close();
47                 } catch (SQLException e) {}
48             }
49         }
50     }
51 }

```

3.4.5. Código de FrmEliminación.

```

1  import java.awt.GridBagLayout;
2  import java.awt.event.ActionEvent;
3  import java.awt.event.ActionListener;
4  import java.sql.SQLException;
5  import java.text.NumberFormat;
6  import java.text.ParseException;
7  import java.text.SimpleDateFormat;
8  import javax.swing.JFormattedTextField;
9  import javax.swing.JFrame;
10 import javax.swing.JLabel;
11 import javax.swing.JMenu;
12 import javax.swing.JMenuBar;
13 import javax.swing.JMenuItem;
14 import javax.swing.JOptionPane;
15 import javax.swing.JPanel;
16 import javax.swing.JScrollPane;
17 import javax.swing.JTextArea;
18 import javax.swing.JTextField;
19 import javax.swing.SwingUtilities;
20
21 public class FrmEliminación extends JFrame implements ActionListener {
22     public static void main(String[] args) {
23         try {
24             Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
25             SwingUtilities.invokeLater(new Runnable() {
26                 public void run() {
27                     new FrmEliminación().setVisible(true);
28                 }
29             });
30         } catch (Exception e) {
31             e.printStackTrace();
32         }
33     }
34     private final CtrlModificación ctrlModificación = new CtrlModificación();
35     private final CtrlEliminación ctrlEliminación = new CtrlEliminación();
36     private final JLabel lblClave = new JLabel("Clave:");
37     private final NumberFormat fmt = NumberFormat.getInstance();
38     private final JFormattedTextField txtClave = new JFormattedTextField(
39         fmt);
40     {
41         txtClave.setColumns(11);
42         txtClave.setHorizontalAlignment(JTextField.TRAILING);
43     }
44     private final JLabel lblNombre = new JLabel("Nombre:");
45     private final JTextField txtNombre = new JTextField(50);
46     {
47         txtNombre.setEditable(false);
48     }
49     private final JLabel lblTeléfono = new JLabel("Teléfono:");
50     private final JTextField txtTeléfono = new JTextField(50);
51     {
52         txtTeléfono.setEditable(false);

```

```

53     }
54     private final JLabel lblEmail = new JLabel("Correo Electrónico:");
55     private final JTextField txtEmail = new JTextField(50);
56     {
57         txtEmail.setEditable(false);
58     }
59     private final JLabel lblFecha = new JLabel("Fecha Inicial:");
60     private final SimpleDateFormat fmtF = new SimpleDateFormat("dd/MM/yyyy");
61     private final JFormattedTextField txtFecha = new JFormattedTextField(
62         fmtF);
63     {
64         txtFecha.setColumns(10);
65         txtFecha.setEditable(false);
66     }
67     private final JLabel lblDirección = new JLabel("Dirección:");
68     private final JTextArea txtDirección = new JTextArea(3, 50);
69     {
70         txtDirección.setEditable(false);
71     }
72     private final JMenuItem itmBuscar = new JMenuItem("Buscar");
73     {
74         itmBuscar.addActionListener(this);
75     }
76     private final JMenuItem itmEliminar = new JMenuItem("Eliminar");
77     {
78         itmEliminar.addActionListener(this);
79     }
80     private final JMenuItem itmSalir = new JMenuItem("Salir");
81     {
82         itmSalir.addActionListener(this);
83     }
84     private final JMenu menuArchivo = new JMenu("Archivo");
85     {
86         menuArchivo.add(itmBuscar);
87         menuArchivo.add(itmEliminar);
88         menuArchivo.add(itmSalir);
89     }
90     private final JMenuBar menuBar = new JMenuBar();
91     {
92         menuBar.add(menuArchivo);
93     }
94     private final JPanel panel = new JPanel(new GridBagLayout());
95     {
96         panel.add(lblClave, new GBC(0, 0).insets(12, 12, 12, 12));
97         panel.add(txtClave, new GBC(0, 1).insets(12, 0, 12, 12));
98         panel.add(lblNombre, new GBC(1, 0).insets(0, 12, 12, 12));
99         panel.add(txtNombre, new GBC(1, 1).insets(0, 0, 12, 12).weight(1.0,
100             0.0).fill(GBC.HORIZONTAL));
101         panel.add(lblTeléfono, new GBC(2, 0).insets(0, 12, 12, 12));
102         panel.add(txtTeléfono, new GBC(2, 1).insets(0, 0, 12, 12).weight(
103             1.0, 0.0));
104         panel.add(lblEmail, new GBC(3, 0).insets(0, 12, 12, 12));
105         panel.add(txtEmail, new GBC(3, 1).insets(0, 0, 12, 12).weight(1.0,
106             0.0));

```

```

107     panel.add(lblFecha, new GBC(4, 0).insets(0, 12, 12, 12));
108     panel.add(txtFecha, new GBC(4, 1).insets(0, 0, 12, 12));
109     panel.add(lblDirección, new GBC(5, 0).insets(0, 12, 12, 12));
110     panel.add(new JScrollPane(txtDirección), new GBC(5, 1).insets(0, 0,
111         12, 12).fill(GBC.BOTH).weight(1.0, 1.0));
112 }
113 public FrmEliminación() {
114     setTitle("Eliminación de Datos de Proveedores");
115     setJMenuBar(menuBar);
116     setContentPane(new JScrollPane(panel));
117     setDefaultCloseOperation(EXIT_ON_CLOSE);
118     pack();
119 }
120 public void actionPerformed(ActionEvent evt) {
121     Object origen = evt.getSource();
122     if (origen == itmBuscar) {
123         busca();
124     } else if (origen == itmEliminar) {
125         elimina();
126     } else if (origen == itmSalir) {
127         System.exit(0);
128     }
129 }
130 private void busca() {
131     try {
132         txtClave.commitEdit();
133         Number clave = (Number) txtClave.getValue();
134         Proveedor pro = ctrlModificación.busca(clave.intValue());
135         txtNombre.setText(pro.getNombre());
136         txtTeléfono.setText(pro.getTeléfono());
137         txtEmail.setText(pro.getEmail());
138         txtDirección.setText(pro.getDirección());
139         txtFecha.setValue(pro.getFechaInicial());
140     } catch (ParseException e) { // generado por el commit edit
141         e.printStackTrace();
142         error("Clave Incorrecta");
143     } catch (SQLException e) {
144         e.printStackTrace();
145         error(e.getMessage());
146     }
147 }
148 private void elimina() {
149     try {
150         String nombre = txtNombre.getText().trim();
151         txtClave.commitEdit();
152         Number clave = (Number) txtClave.getValue();
153         int respuesta = JOptionPane.showConfirmDialog(this,
154             "¿Realmente desea eliminar al proveedor " + clave
155             + " - " + nombre + "?");
156         if (respuesta == JOptionPane.YES_OPTION) {
157             ctrlEliminación.elimina(clave.intValue());
158             JOptionPane.showMessageDialog(this, "Proveedor Eliminado");
159             txtClave.setValue(null);
160             txtNombre.setText("");

```

```

161         txtTeléfono.setText("");
162         txtEmail.setText("");
163         txtDirección.setText("");
164         txtFecha.setValue(null);
165     }
166     } catch (ParseException e) { // generado por el commit edit
167         e.printStackTrace();
168         error("Clave Incorrecta");
169     } catch (SQLException e) {
170         e.printStackTrace();
171         error(e.getMessage());
172     }
173 }
174 private void error(String mensaje) {
175     JOptionPane.showMessageDialog(this, mensaje, "Error",
176                                 JOptionPane.ERROR_MESSAGE);
177 }
178 }

```

3.5. Consulta.

3.5.1. Funcionamiento.

Al abrir la aplicación, se muestra un listado de los proveedores registrados.



Clave	Nombre	Teléfono	Correo Electr...	Dirección	Fecha de Inic...
1	Zoila Vaca	01-800-MUU	zoila@vaca.c...	Establos #23	3/10/2003
2	Rolando Mota	01-800-VIAJE	rolando@via...	Misterios #12	6/11/2005
3	Armando Pa...	01-800-TQUI...	armando@t...	Noche de R...	23/02/2007
4	Pancraccio W...	01-800-PLA...	pancracio@...	Paseo de lo...	23/02/2007

El menú “Archivo” contiene la siguiente opción:

- **Salir.** Para salir de la aplicación.

El menú “Ver” contiene la siguiente opción:

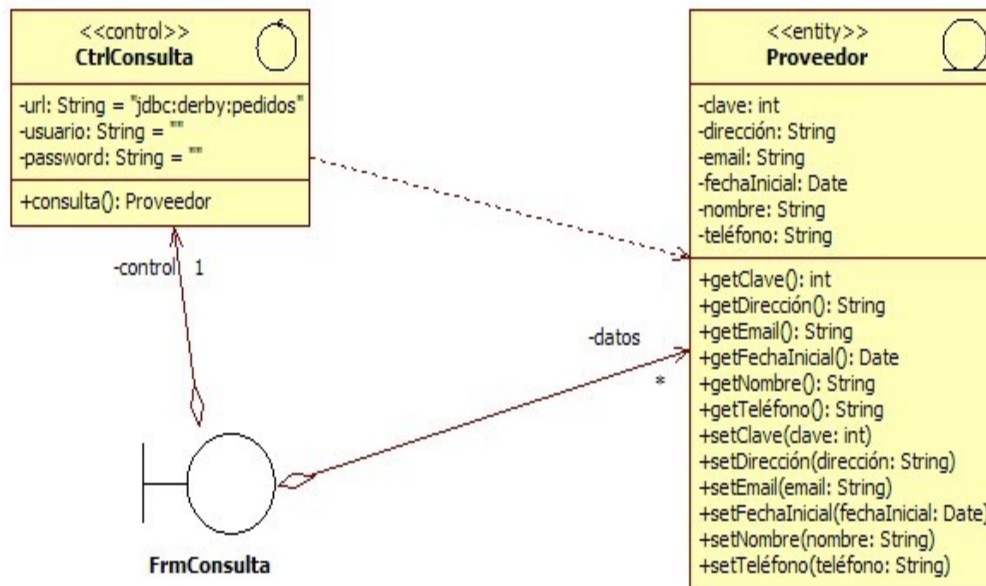
- **Actualizar.** Para actualizar el listado con la información más reciente, almacenada en la base de datos. Esta opción es útil para ambientes donde otros usuarios pueden acceder simultáneamente a la aplicación y modificar datos.

Para usarla se sigue el siguiente procedimiento:

1. Se introduce la clave del proveedor a modificar.
2. Se selecciona el menú “Ver”.
3. Se selecciona la opción “Actualizar”.
4. La aplicación busca en la base de datos los proveedores registrados y los muestra. En caso de que no se pueda realizar exitosamente la operación, muestra un cuadro de error que despliega la razón del fallo.

3.5.2. Diagrama de Clases.

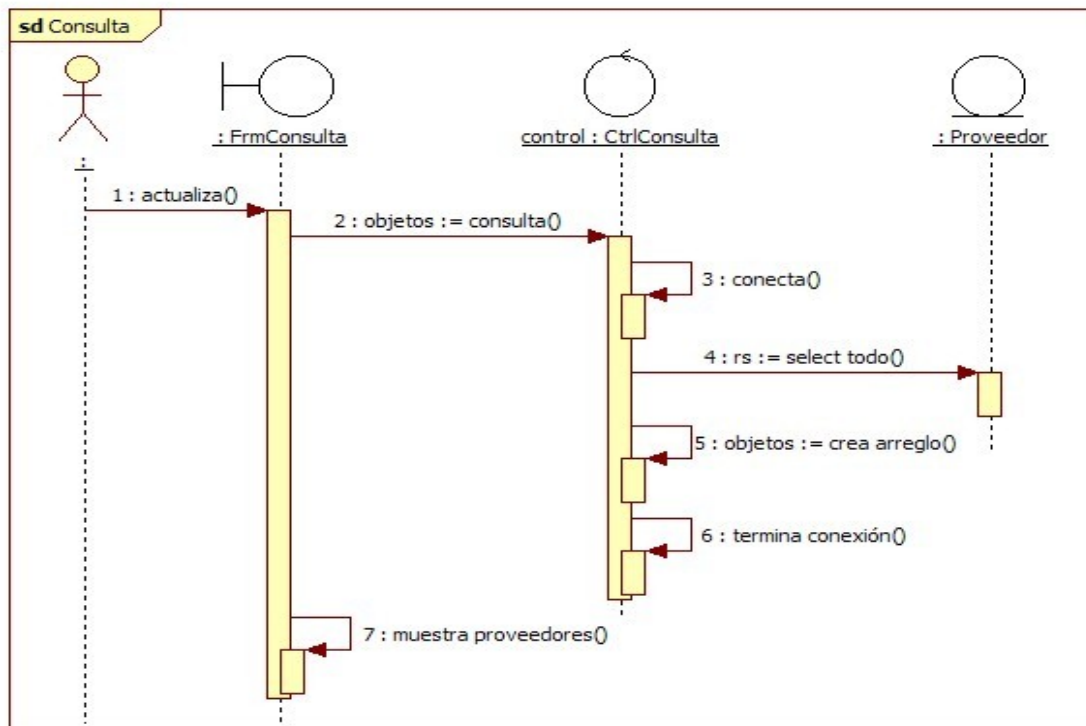
Los datos mostrados en a tabla, se almacenan en un arreglo de proveedores.



El arreglo se muestra en la flecha que dice datos. El asterisco (*) indica que almacena muchos proveedores.

3.5.3. Diagrama de Secuencia.

Dado que la actualización puede generarse automáticamente u por solicitud del usuario, el siguiente diagrama no muestra la forma de lanzar la operación.



3.5.4. Código de CtrlConsulta.

```

1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5 import java.sql.Statement;
6
7 public class CtrlConsulta {
8     private final String url = "jdbc:derby:proveedores";
9     private final String usuario = "";
10    private final String password = "";
11    /**
12     * Obtiene todos los proveedores.
13     *
14     * @return Un arreglo con todos los proveedores registrados.
15     * @throws SQLException
16     *         si algo falla.
17     */
18    public Proveedor[] consulta() throws SQLException {

```

```

19 Connection c = null;
20 Statement s = null;
21 ResultSet rs = null;
22 try {
23     c = DriverManager.getConnection(url, usuario, password);
24     /*
25      * Crea el objeto que envía consultas. TYPE_SCROLL_INSENSITIVE -
26      * Los ResultSet creados se pueden recorrer hacia adelante y
27      * hacia atrás. CONCUR_READ_ONLY - Los ResultSet creados son de
28      * solo lectura.
29      */
30     s = c.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
31         ResultSet.CONCUR_READ_ONLY);
32     // Solicita todos los proveedores registrados.
33     rs = s.executeQuery("SELECT prov_clave, prov_nombre, "
34         + "prov_telefono, prov_email, prov_direccion "
35         + "FROM Proveedor");
36     // Obtiene el número total de registros.
37     // Primero se coloca en la última posición de la respuesta.
38     rs.last();
39     // El número de renglón es el número de elementos.
40     int total = rs.getRow();
41     // Crea un arreglo que reserva espacio para colocar todos los
42     // proveedores.
43     Proveedor[] objetos = new Proveedor[total];
44     // Se coloca antes del primer renglón para leer la respuesta.
45     rs.beforeFirst();
46     // Revisa todos los renglones de la respuesta.
47     for (int i = 0; rs.next() && i < objetos.length; i++) {
48         Proveedor pro = new Proveedor();
49         pro.setClave(rs.getInt("prov_clave"));
50         pro.setNombre(rs.getString("prov_nombre"));
51         pro.setTeléfono(rs.getString("prov_telefono"));
52         pro.setEmail(rs.getString("prov_email"));
53         pro.setDirección(rs.getString("prov_direccion"));
54         objetos[i] = pro;
55     }
56     // Devuelve el arreglo.
57     return objetos;
58 } finally {
59     if (rs != null) {
60         try {
61             rs.close();
62         } catch (SQLException e) {}
63     }
64     if (s != null) {
65         try {
66             s.close();
67         } catch (SQLException e) {}
68     }
69     if (c != null) {
70         try {
71             c.close();
72         } catch (SQLException e) {}

```



```

73         }
74     }
75 }
76 }

```

3.5.5. Código de FrmConsulta.

```

1  import java.awt.Dimension;
2  import java.awt.event.ActionEvent;
3  import java.awt.event.ActionListener;
4  import java.sql.SQLException;
5  import java.util.Date;
6  import javax.swing.JFrame;
7  import javax.swing.JMenu;
8  import javax.swing.JMenuBar;
9  import javax.swing.JMenuItem;
10 import javax.swing.JOptionPane;
11 import javax.swing.JScrollPane;
12 import javax.swing.JTable;
13 import javax.swing.ListSelectionModel;
14 import javax.swing.SwingUtilities;
15 import javax.swing.table.AbstractTableModel;
16
17 public class FrmConsulta extends JFrame implements ActionListener {
18     public static void main(String[] args) {
19         try {
20             Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
21             SwingUtilities.invokeLater(new Runnable() {
22                 public void run() {
23                     FrmConsulta ventana = new FrmConsulta();
24                     ventana.setVisible(true);
25                     ventana.actualiza();
26                 }
27             });
28         } catch (Exception e) {
29             e.printStackTrace();
30         }
31     }
32     private final CtrlConsulta control = new CtrlConsulta();
33     /**
34      * Proveedores a desplegar. Originalmente es un arreglo vacío.
35      */
36     private Proveedor[] datos = new Proveedor[0];
37     /**
38      * Especifica el contenido de la tabla que muestra los proveedores.
39      */
40     private final AbstractTableModel modelo = new AbstractTableModel() {
41         /**
42          * Devuelve la clase que describe el tipo de datos de cada columna.
43          */
44         @Override
45         public Class<?> getColumnClass(int columna) {

```

```

46         switch (columna) {
47             case 0:
48                 return Integer.class;
49             case 5:
50                 return Date.class;
51             default:
52                 return String.class;
53         }
54     }
55     /**
56      * Devuelve el número de columnas.
57      */
58     @Override
59     public int getColumnCount() {
60         return 6;
61     }
62     /**
63      * Devuelve el encabezado de cada columna.
64      */
65     @Override
66     public String getColumnName(int columna) {
67         switch (columna) {
68             case 0:
69                 return "Clave";
70             case 1:
71                 return "Nombre";
72             case 2:
73                 return "Teléfono";
74             case 3:
75                 return "Correo Electrónico";
76             case 4:
77                 return "Dirección";
78             default:
79                 return "Fecha de Inicio";
80         }
81     }
82     /**
83      * Devuelve el número de renglones. Lo toma de la longitud del
84      * arreglo que contiene el catálogo.
85      */
86     @Override
87     public int getRowCount() {
88         return datos.length;
89     }
90     /**
91      * Devuelve el valor de cada celda. Lo toma del arreglo que contiene
92      * el catálogo.
93      */
94     @Override
95     public Object getValueAt(int renglón, int columna) {
96         switch (columna) {
97             case 0:
98                 return datos[renglón].getClave();
99             case 1:

```

```

100         return datos[renglón].getNombre();
101     case 2:
102         return datos[renglón].getTeléfono();
103     case 3:
104         return datos[renglón].getEmail();
105     case 4:
106         return datos[renglón].getDirección();
107     default:
108         return datos[renglón].getFechaInicial();
109     }
110 }
111 /**
112  * Indica que las celdas de esta tabla no se pueden modificar por el
113  * usuario.
114  */
115 @Override
116 public boolean isCellEditable(int renglón, int columna) {
117     return false;
118 }
119 // Como no se puede modificar, no es necesario definir setValueAt
120 };
121 private final JTable tabla = new JTable(modelo);
122 {
123     tabla.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
124     tabla.setShowGrid(false);
125 }
126 private final JScrollPane scroll = new JScrollPane(tabla);
127 {
128     scroll.setPreferredSize(new Dimension(500, 100));
129 }
130 private final JMenuItem itmActualizar = new JMenuItem("Actualizar");
131 {
132     itmActualizar.addActionListener(this);
133 }
134 private final JMenuItem itmSalir = new JMenuItem("Salir");
135 {
136     itmSalir.addActionListener(this);
137 }
138 private final JMenu menuArchivo = new JMenu("Archivo");
139 {
140     menuArchivo.add(itmSalir);
141 }
142 private final JMenu menuVer = new JMenu("Ver");
143 {
144     menuVer.add(itmActualizar);
145 }
146 private final JMenuBar menuBar = new JMenuBar();
147 {
148     menuBar.add(menuArchivo);
149     menuBar.add(menuVer);
150 }
151 public FrmConsulta() {
152     setTitle("Datos de Proveedores");
153     setJMenuBar(menuBar);

```

```
154         setContentPane(scroll);
155         setDefaultCloseOperation(EXIT_ON_CLOSE);
156         pack();
157     }
158     public void actionPerformed(ActionEvent evt) {
159         Object origen = evt.getSource();
160         if (origen == itmActualizar) {
161             actualiza();
162         } else if (origen == itmSalir) {
163             System.exit(0);
164         }
165     }
166     public void actualiza() {
167         try {
168             datos = control.consulta();
169             modelo.fireTableDataChanged();
170         } catch (SQLException e) {
171             e.printStackTrace();
172             error(e.getMessage());
173         }
174     }
175     private void error(String mensaje) {
176         JOptionPane.showMessageDialog(this, mensaje, "Error",
177                                     JOptionPane.ERROR_MESSAGE);
178     }
179 }
```

4. Mantenimiento de Catálogos.

Por catálogo se entiende, todo el conjunto de registros almacenados en una tabla.

Este tipo de forma, integra las funciones de todas las ventanas del capítulo anterior. Consta de dos partes:

- Una tabla donde se muestra el contenido del catálogo.
- Una forma de propiedades, donde se muestra el valor de los valores que se pueden ver para el objeto seleccionado en la tabla; permite modificar algunos de los datos mostrados y también puede usarse para dar de alta un nuevo registro.

Cuando la forma de propiedades es muy complicada, normalmente se despliega aparte, en un cuadro de diálogo que se activa al hacer doble clic sobre el proveedor deseado, o mediante la barra de menú, o con un menú contextual.

En este caso no se muestran diagramas de secuencia puesto que son muy parecidos a los casos anteriores.

4.1. Funcionamiento.

La aplicación se puede instalar de dos formas:

- Con un archivo instalador. Crea un grupo en la barra de inicio con el nombre “Proveedores”.
- Con el java Web Start desde un sitio web. Crea un grupo en la barra de inicio con el nombre “Proveedores Red”.

En los dos casos, se crea un acceso directo llamado proveedores, con el siguiente ícono.

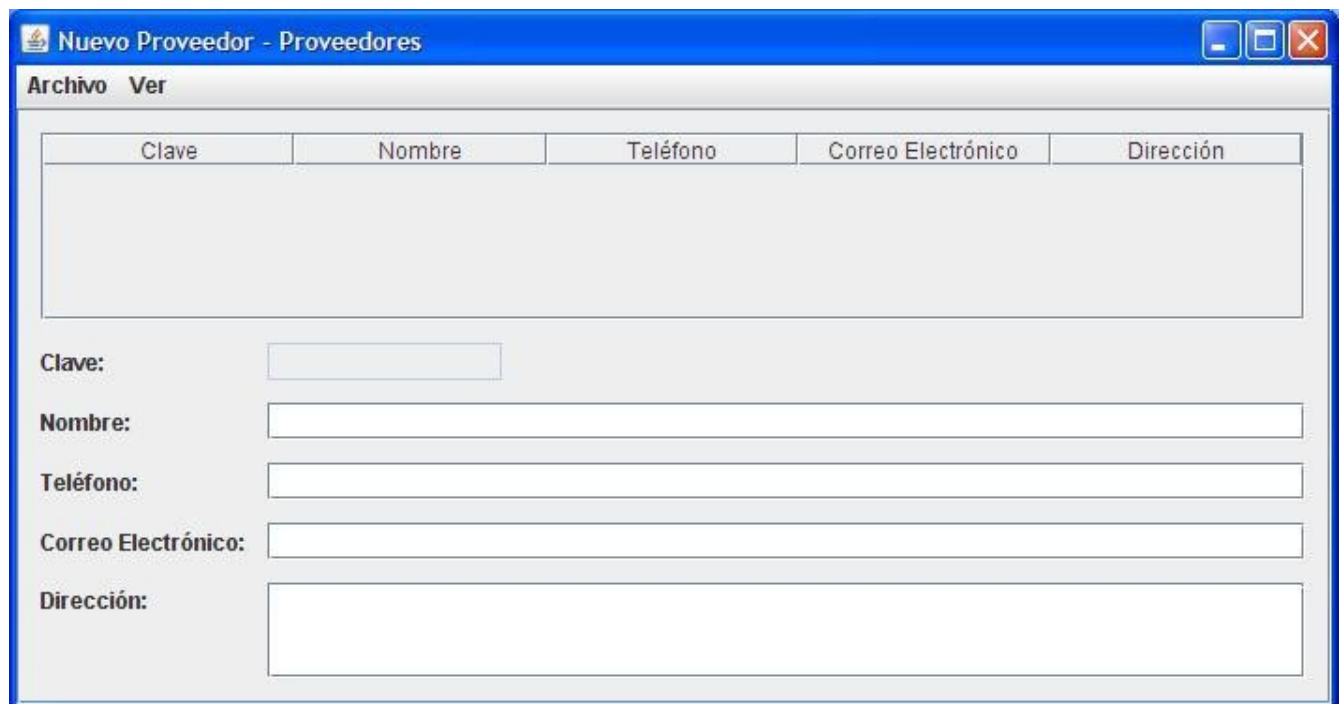


Al lanzarse, verifica que la base de datos ya esté creada. En caso contrario, presenta un cuadro de opciones como el siguiente:



Cuando el usuario desea seleccionar “Crear Base de Datos”, se intenta realizar esta opción. Si no tiene éxito, presenta un cuadro de error describiendo la falla.

Cuando la base de datos está creada, se despliega una ventana como la siguiente:



Clave	Nombre	Teléfono	Correo Electrónico	Dirección

Clave:

Nombre:

Teléfono:

Correo Electrónico:

Dirección:

El menú “Archivo” tiene las siguientes opciones:

- **Nuevo Proveedor.** Para crear un nuevo proveedor.
- **Guardar.** Para guardar las modificaciones realizadas o el nuevo proveedor creado.
- **Eliminar.** Para eliminar el proveedor seleccionado en la tabla.

El menú “Ver” tiene la siguiente opción

- **Actualizar.** Para sincronizar la forma con el contenido actual de la base de datos.

La forma se utiliza de la siguiente manera:

- **Creación un nuevo proveedor.**

1. Si la barra de título no indica “Nuevo Proveedor”, el usuario selecciona el menú “Archivo” y luego, la opción “Nuevo Proveedor”. La barra de título se ajusta para indicar esta condición.
2. El usuario introduce los datos para los siguientes campos:
 - nombre
 - teléfono
 - correo electrónico
 - dirección
3. El usuario selecciona el menú “Archivo” y luego, la opción “Guardar”.
4. La aplicación intenta agregar el nuevo registro. Si tiene éxito, despliega un cuadro con el mensaje: “Proveedor Agregado”. En caso contrario presenta un cuadro de error con la descripción de la falla.
5. La tabla se sincroniza con la base de datos y la forma de propiedades se pone en blanco. La barra de título indica “Nuevo Proveedor – Proveedores”.

- **Modificación de datos de un proveedor.**

1. El usuario selecciona en la tabla el proveedor que se desea modificar.
2. El programa despliega en la forma de propiedades los datos del proveedor seleccionado. La barra de título se ajusta para indicar la clave y el nombre del proveedor seleccionado, seguido del texto “ - Proveedores”.
3. El usuario puede modificar los datos para los siguientes campos:
 - nombre

- teléfono
 - correo electrónico
 - dirección
4. El usuario selecciona el menú “Archivo” y luego, la opción “Guardar”.
 5. La aplicación intenta modificar el registro. Si tiene éxito, despliega un cuadro con el mensaje: “Proveedor Modificado”. En caso contrario presenta un cuadro de error con la descripción de la falla.
 6. La tabla se sincroniza con la base de datos y la forma de propiedades se pone en blanco. La barra de título indica “Nuevo Proveedor – Proveedores”.
- **Eliminación de un proveedor.**
 1. El usuario selecciona en la tabla el proveedor que se desea eliminar.
 2. El programa despliega en la forma de propiedades los datos del proveedor seleccionado. La barra de título se ajusta para indicar la clave y el nombre del proveedor seleccionado, seguido del texto “ - Proveedores”.
 3. El usuario selecciona el menú “Archivo” y luego, la opción “Eliminar”.
 4. La aplicación intenta eliminar el registro. Si tiene éxito, despliega un cuadro con el mensaje: “Proveedor Eliminado”. En caso contrario presenta un cuadro de error con la descripción de la falla.
 5. La tabla se sincroniza con la base de datos y la forma de propiedades se pone en blanco. La barra de título indica “Nuevo Proveedor – Proveedores”.
 - **Actualizar la tabla de proveedores.**
 1. El usuario selecciona el menú “Ver” y luego, la opción “Actualizar”.
 2. La tabla se sincroniza con la base de datos y la forma de propiedades se pone en blanco. La barra de título indica “Nuevo Proveedor – Proveedores”. En caso de error el programa presenta un cuadro de error con la descripción de la falla.

4.2. Manejo de la Barra de Título.

Normalmente la barra de título muestra el nombre del objeto que es está modificando, seguido de un separador y el nombre de la aplicación. También se indica la condición de crear un nuevo objeto, seguida de un separador y el nombre de la aplicación.

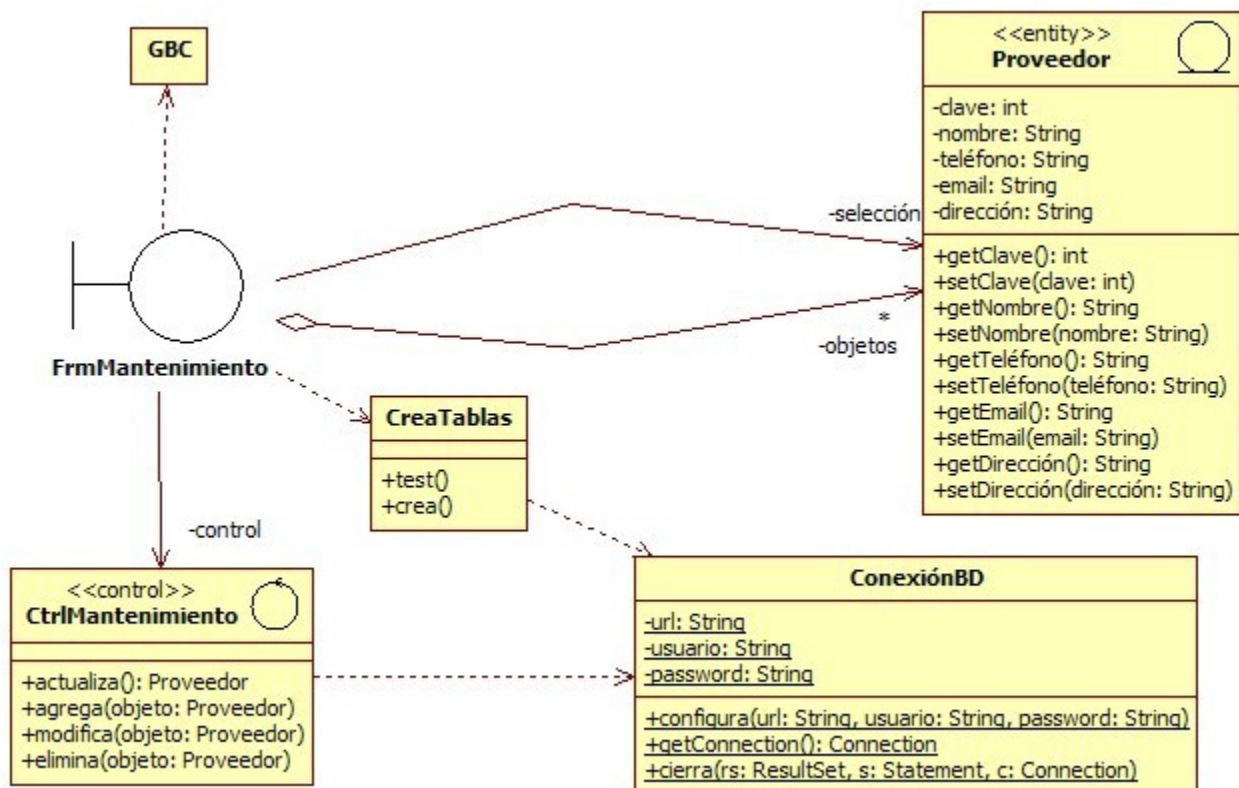
4.3. Diagrama de Clases.

Para simplificar el manejo de distintas funciones de JDBC se introduce la clase `ConexiónBD`, que permite realizar las siguientes funciones:

- Utilizar el mismo URL, usuario y contraseña en toda la aplicación.
- Establecer conexiones
- Terminar conexiones.
- Recuperar las llaves generadas automáticamente.

La clase `CreaTablas` se encarga de verificar si la base de datos existe y de crearla.

Le flecha sin rombo marcada como selección, marca una referencia que puede apuntar a cualquier proveedor; en particular hace referencia al que está seleccionado en la tabla, y que se almacena dentro del arreglo llamado `objetos`.



4.4. Código de ConexiónBD

```

1  /* Copyright 2016 Ricardo Armando Machorro Reyes
2  *
3  * Licensed under the Apache License, Version 2.0 (the "License");
4  * you may not use this file except in compliance with the License.
5  * You may obtain a copy of the License at
6  *
7  *     http://www.apache.org/licenses/LICENSE-2.0
8  *
9  * Unless required by applicable law or agreed to in writing, software
10 * distributed under the License is distributed on an "AS IS" BASIS,
11 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 * See the License for the specific language governing permissions and
13 * limitations under the License.
14 */
15 import java.sql.Connection;
16 import java.sql.DriverManager;
17 import java.sql.ResultSet;
18 import java.sql.SQLException;
19 import java.sql.Statement;
20
21 /**
22  * Clase que facilita el trabajo con JDBC.
23  */
24 public class ConexiónBD {
25     private static String url;
26     private static String usuario;
27     private static String password;
28     /**
29      * Asigna los valores de url, usuario y password para toda la aplicación.
30      *
31      * @param url
32      *         URL para establecer conexiones.
33      * @param usuario
34      *         Nombre de usuario para establecer conexiones.
35      * @param password
36      *         Contraseña de usuario para establecer conexiones.
37      */
38     public static void configura(String url, String usuario,
39                                 String password) {
40         ConexiónBD.url = url;
41         ConexiónBD.usuario = usuario;
42         ConexiónBD.password = password;
43     }
44     /**
45      * Intenta establecer una conexión con la base de datos.
46      *
47      * @return La conexión establecida.
48      * @throws SQLException
49      *         Si algo falló y no pudo establecerse la conexión.
50      */
51     public static Connection getConnection() throws SQLException {
52         if (url == null) {

```

```

53         throw new SQLException("Te falta configurar url");
54     }
55     Connection c = DriverManager.getConnection(url, usuario, password);
56     return c;
57 }
58 /**
59  * Cierra los elementos de una conexión, siempre y cuando no valgan null.
60  *
61  * @param rs
62  *         ResultSet a cerrar siempre y cuando no valga null.
63  * @param s
64  *         Statement a cerrar siempre y cuando no valga null.
65  * @param c
66  *         Connection a cerrar siempre y cuando no valga null.
67  */
68 public static void cierra(ResultSet rs, Statement s, Connection c) {
69     if (rs != null) {
70         try {
71             rs.close();
72         } catch (SQLException e) {}
73     }
74     if (s != null) {
75         try {
76             s.close();
77         } catch (SQLException e) {}
78     }
79     if (c != null) {
80         try {
81             c.close();
82         } catch (SQLException e) {}
83     }
84 }
85 }

```

4.5. Código de CreaTablas

```

1  import java.sql.Connection;
2  import java.sql.Statement;
3
4  /**
5   * Permite verificar la existencia de la base de datos y crearla en caso de
6   * que sea necesario. También define el url, usuario y password necesarios
7   * para usarla. Está pensada principalmente para poder usar el ejemplo en una
8   * sola computadora. Para un ambiente multiusuario, esta clase no es
9   * necesaria.
10  *
11  */
12 public class CreaTablas {
13     /**
14      * Verifica si la la base de datos ya está creada.
15      *
16      * @throws SQLException

```

```

17      *           Si la base de datos no está creada o falla algo más.
18      */
19      public void test() throws Exception {
20          // Configura los parámetros de conexión.
21          ConexiónBD.configura("jdbc:derby:proveedores2", "", "");
22          Connection c = null;
23          Statement s = null;
24          try {
25              // Establece la conexión.
26              c = ConexiónBD.getConnection();
27              // Crea el objeto que realiza la consulta
28              s = c.createStatement();
29              // Solicita el número de registros de la tabla Proveedor.
30              s.executeQuery("SELECT count(*) FROM Proveedor");
31          } finally {
32              // Termina la conexión.
33              ConexiónBD.cierra(null, s, c);
34          }
35      }
36      /**
37       * Crea la base de datos.
38       *
39       * @throws SQLException
40       *           Si falla algo.
41       */
42      public void crea() throws Exception {
43          // Configura los parámetros de conexión.
44          ConexiónBD.configura("jdbc:derby:proveedores2;create=true", "", "");
45          Connection c = null;
46          Statement s = null;
47          try {
48              // Establece la conexión.
49              c = ConexiónBD.getConnection();
50              // Crea el objeto que realiza la consulta
51              s = c.createStatement();
52              // Crea la tabla.
53              s.executeUpdate("CREATE TABLE Proveedor ( "
54                  + "prov_clave INTEGER PRIMARY KEY "
55                  + "GENERATED ALWAYS AS IDENTITY, "
56                  + "prov_nombre VARCHAR(100) NOT NULL, "
57                  + "prov_telefono VARCHAR(50) NOT NULL, "
58                  + "prov_email VARCHAR(50) NOT NULL, "
59                  + "prov_direccion LONG VARCHAR NOT NULL)");
60              // Configura los parámetros de conexión.
61              ConexiónBD.configura("jdbc:derby:proveedores2", "", "");
62          } finally {
63              // Termina la conexión.
64              ConexiónBD.cierra(null, s, c);
65          }
66      }
67  }

```

4.6. Código de CtrlMantenimiento.

```

1  import java.sql.Connection;
2  import java.sql.PreparedStatement;
3  import java.sql.ResultSet;
4  import java.sql.SQLException;
5  import java.sql.Statement;
6
7  /**
8   * Realiza el acceso a la base de datos para objetos de la clase Proveedor.
9   */
10 public class CtrlMantenimiento {
11     /**
12      * Obtiene todos los proveedores.
13      *
14      * @return Un arreglo con todos los proveedores registrados.
15      * @throws SQLException
16      *         Si se presenta algún fallo.
17      */
18     public Proveedor[] actualiza() throws SQLException {
19         Connection c = null;
20         Statement s = null;
21         ResultSet rs = null;
22         try {
23             c = ConexiónBD.getConnection();
24             // Crea el objeto que envía consultas.
25             // TYPE_SCROLL_INSENSITIVE - Los ResultSet creados se
26             // pueden recorrer hacia adelante y hacia atrás.
27             // CONCUR_READ_ONLY - Los ResultSet creados son de solo lectura.
28             s = c.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
29                                   ResultSet.CONCUR_READ_ONLY);
30             // Solicita todos los proveedores registrados.
31             rs = s.executeQuery("SELECT prov_clave, prov_nombre, "
32                               + "prov_telefono, prov_email, prov_direccion "
33                               + "FROM Proveedor");
34             // Obtiene el número total de registros.
35             // Primero se coloca en la última posición de la respuesta.
36             rs.last();
37             // El número de renglón es el número de elementos.
38             int total = rs.getRow();
39             // Crea un arreglo que reserva espacio para colocar todos los
40             // proveedores.
41             Proveedor[] objetos = new Proveedor[total];
42             // Se coloca antes del primer renglón para leer la respuesta.
43             rs.beforeFirst();
44             // Revisa todos los renglones de la respuesta.
45             for (int i = 0; rs.next() && i < objetos.length; i++) {
46                 Proveedor pro = new Proveedor();
47                 pro.setClave(rs.getInt("prov_clave"));
48                 pro.setNombre(rs.getString("prov_nombre"));
49                 pro.setTeléfono(rs.getString("prov_telefono"));
50                 pro.setEmail(rs.getString("prov_email"));
51                 pro.setDirección(rs.getString("prov_direccion"));
52                 objetos[i] = pro;

```

```

53         }
54         // Devuelve el arreglo.
55         return objetos;
56     } finally {
57         // Termina la conexión.
58         ConexiónBD.cierra(rs, s, c);
59     }
60 }
61 /**
62  * Agrega un proveedor a la base de datos y le asigna automáticamente la
63  * clave
64  *
65  * @param objeto
66  *         El proveedor que se desea agregar.
67  * @return El proveedor recibido, pero con la llave primaria asignada.
68  * @throws SQLException
69  *         Si se presenta algún fallo.
70  */
71 public void agrega(Proveedor objeto) throws SQLException {
72     Connection c = null;
73     PreparedStatement ps = null;
74     try {
75         // Establece la conexión.
76         c = ConexiónBD.getConnection();
77         // Inserta el objeto en la tabla y solicita que se capture
78         // la llave generada automáticamente.
79         ps = c.prepareStatement("INSERT INTO Proveedor "
80                                + "(prov_nombre, prov_telefono, "
81                                + "prov_email, prov_direccion) "
82                                + "VALUES (?, ?, ?, ?)");
83         ps.setString(1, objeto.getNombre());
84         ps.setString(2, objeto.getTeléfono());
85         ps.setString(3, objeto.getEmail());
86         ps.setString(4, objeto.getDirección());
87         ps.executeUpdate();
88     } finally {
89         // Termina la conexión.
90         ConexiónBD.cierra(null, ps, c);
91     }
92 }
93 /**
94  * Guarda en la base de datos las propiedades del proveedor recibido. Los
95  * valores se guardan en el registro cuyo campo "clave" coincida con el
96  * valor de la propiedad "clave" del objeto.
97  *
98  * @param objeto
99  *         El proveedor que se desea guardar.
100  * @throws SQLException
101  *         Si se presenta algún fallo.
102  */
103 public void modifica(Proveedor objeto) throws SQLException {
104     Connection c = null;
105     PreparedStatement ps = null;
106     try {

```

```

107         // Establece la conexión.
108         c = ConexiónBD.getConnection();
109         // Busca el registro y realiza los cambios.
110         ps = c.prepareStatement("UPDATE Proveedor "
111             + "SET prov_nombre = ?, prov_telefono = ?, "
112             + "prov_email = ?, prov_direccion = ? "
113             + "WHERE prov_clave = ?");
114         ps.setString(1, objeto.getNombre());
115         ps.setString(2, objeto.getTeléfono());
116         ps.setString(3, objeto.getEmail());
117         ps.setString(4, objeto.getDirección());
118         ps.setInt(5, objeto.getClave());
119         ps.executeUpdate();
120     } finally {
121         // Termina la conexión.
122         ConexiónBD.cierra(null, ps, c);
123     }
124 }
125 /**
126  * Elimina de la base de datos el proveedor recibido. Toma como criterio
127  * de eliminación el valor de la propiedad "clave".
128  *
129  * @param objeto
130  *         El objeto a eliminar.
131  * @throws SQLException
132  *         Si o se presenta algún fallo.
133  */
134 public void elimina(Proveedor objeto) throws SQLException {
135     Connection c = null;
136     PreparedStatement ps = null;
137     try {
138         // Establece la conexión.
139         c = ConexiónBD.getConnection();
140         // Busca el registro y lo elimina.
141         ps = c.prepareStatement("DELETE FROM Proveedor "
142             + "WHERE prov_clave = ?");
143         ps.setInt(1, objeto.getClave());
144         ps.executeUpdate();
145     } finally {
146         // Termina la conexión.
147         ConexiónBD.cierra(null, ps, c);
148     }
149 }
150 }

```

4.7. Diseño de la Forma.

	0	1										
0	<table border="1"> <thead> <tr> <th>Clave</th> <th>Nombre</th> <th>Teléfono</th> <th>Correo Electrónico</th> <th>Dirección</th> </tr> </thead> <tbody> <tr> <td colspan="5"></td> </tr> </tbody> </table>	Clave	Nombre	Teléfono	Correo Electrónico	Dirección						
Clave	Nombre	Teléfono	Correo Electrónico	Dirección								
1	Clave:	<input type="text"/>										
2	Nombre:	<input type="text"/>										
3	Teléfono:	<input type="text"/>										
4	Correo Electrónico:	<input type="text"/>										
5	Dirección:	<input type="text"/>										

4.8. Código de FrmMantenimiento.

```

1  import java.awt.Dimension;
2  import java.awt.GridBagLayout;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.sql.SQLException;
6  import javax.swing.JFrame;
7  import javax.swing.JLabel;
8  import javax.swing.JMenu;
9  import javax.swing.JMenuBar;
10 import javax.swing.JMenuItem;
11 import javax.swing.JOptionPane;
12 import javax.swing.JPanel;
13 import javax.swing.JScrollPane;
14 import javax.swing.JTable;
15 import javax.swing.JTextArea;
16 import javax.swing.JTextField;
17 import javax.swing.ListSelectionModel;
18 import javax.swing.SwingUtilities;
19 import javax.swing.event.ListSelectionEvent;
20 import javax.swing.event.ListSelectionListener;
21 import javax.swing.table.AbstractTableModel;
22
23 /**
24  * Administra el catálogo de proveedores. Permite realizar altas, bajas,
25  * cambios y consultas.
26  */
27 public class FrmMantenimiento extends JFrame implements ActionListener,
28     ListSelectionListener {
29     public static void main(String[] args) {
30         try {
31             // Carga el controlador que permite conectar la base de datos.
32             Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
33             SwingUtilities.invokeLater(new Runnable() {
34                 public void run() {
35                     try {
36                         // Prueba si ya está creada la base de datos.
37                         // Esta prueba solo se recomienda para sistemas
38                         // que corren en una sola computadora.
39                         CreaTablas ctrl = new CreaTablas();
40                         ctrl.test();
41                     } catch (Exception e) {
42                         // Se generó un error. Esto puede deberse a que
43                         // la base de datos no está creada.
44                         creaTablas(e.getMessage());
45                     }
46                     FrmMantenimiento ventana = new FrmMantenimiento();
47                     ventana.setVisible(true);
48                 }
49             });
50         } catch (Exception e) {
51             e.printStackTrace();
52         }
53     }
54 }

```

```

53     }
54     /**
55     * Muestra un mensaje de error y solicita autorización para crear la base
56     * de datos con todo y tablas. En caso afirmativo, intenta crear toda la
57     * estructura de la base de datos. En caso de error, muestra un mensaje,
58     * reportando el problema y termina.
59     *
60     * @param mensaje
61     *         Mensaje de error a mostrar.
62     */
63     private static void creaTablas(String mensaje) {
64         Object[] opciones = { "Crear Base de Datos", "Terminar" };
65         int seleccion = JOptionPane.showOptionDialog(null, mensaje, "Error",
66             JOptionPane.YES_NO_OPTION, JOptionPane.ERROR_MESSAGE, null,
67             opciones, null);
68         if (seleccion == JOptionPane.YES_OPTION) {
69             try {
70                 CreaTablas ctrl = new CreaTablas();
71                 ctrl.crea();
72             } catch (Exception ex) {
73                 // Posiblemente no pude crear la base de datos.
74                 JOptionPane.showMessageDialog(null, ex.getMessage(),
75                     "Error", JOptionPane.ERROR_MESSAGE);
76                 System.exit(1);
77             }
78         } else {
79             System.exit(1);
80         }
81     }
82     /**
83     * Objeto que realiza la conexión a la base de datos.
84     */
85     private final CtrlMantenimiento control = new CtrlMantenimiento();
86     /**
87     * Catálogo de proveedores.
88     */
89     private Proveedor[] objetos = new Proveedor[0];
90     /**
91     * Referencia al objeto seleccionado en la tabla que muestra los
92     * proveedores registrados.
93     */
94     private Proveedor selección = null;
95     /**
96     * Especifica el contenido de la tabla que muestra los proveedores.
97     */
98     private final AbstractTableModel modelo = new AbstractTableModel() {
99         /**
100         * Devuelve la clase de describe el tipo de datos de cada columna.
101         */
102         public Class getColumnClass(int columna) {
103             switch (columna) {
104                 case 0:
105                     return Integer.class;
106                 default:

```

```

107         return String.class;
108     }
109 }
110 /**
111  * Devuelve el número de columnas.
112  */
113 public int getColumnCount() {
114     return 5;
115 }
116 /**
117  * Devuelve el encabezado de cada columna.
118  */
119 public String getColumnName(int columna) {
120     switch (columna) {
121         case 0:
122             return "Clave";
123         case 1:
124             return "Nombre";
125         case 2:
126             return "Teléfono";
127         case 3:
128             return "Correo Electrónico";
129         default:
130             return "Dirección";
131     }
132 }
133 /**
134  * Devuelve el número de renglones. Lo toma de la longitud del
135  * arreglo que contiene el catálogo.
136  */
137 public int getRowCount() {
138     return objetos.length;
139 }
140 /**
141  * Devuelve el valor de cada celda. Lo toma del arreglo que contiene
142  * el catálogo.
143  */
144 public Object getValueAt(int renglón, int columna) {
145     switch (columna) {
146         case 0:
147             return new Integer(objetos[renglón].getClave());
148         case 1:
149             return objetos[renglón].getNombre();
150         case 2:
151             return objetos[renglón].getTeléfono();
152         case 3:
153             return objetos[renglón].getEmail();
154         default:
155             return objetos[renglón].getDirección();
156     }
157 }
158 /**
159  * Indica que las celdas de esta tabla no se pueden modificar por el
160  * usuario.

```

```

161         */
162         public boolean isCellEditable(int renglón, int columna) {
163             return false;
164         }
165     };
166     /**
167      * Tabla donde se muestra el catálogo de proveedores.
168      */
169     private final JTable tblObjetos = new JTable(modelo);
170     {
171         // No muestra cuadrícula.
172         tblObjetos.setShowGrid(false);
173         // Solo se puede seleccionar un renglón a la vez.
174         tblObjetos.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
175         tblObjetos.getSelectionModel().addListSelectionListener(this);
176     }
177     private final JScrollPane scrObjetos = new JScrollPane(tblObjetos);
178     {
179         scrObjetos.setPreferredSize(new Dimension(200, 100));
180     }
181     private final JLabel lblClave = new JLabel("Clave:");
182     private final JTextField txtClave = new JTextField(11);
183     {
184         txtClave.setColumns(11);
185         txtClave.setEditable(false);
186         txtClave.setHorizontalAlignment(JTextField.TRAILING);
187     }
188     private final JLabel lblNombre = new JLabel("Nombre:");
189     private final JTextField txtNombre = new JTextField(50);
190     private final JLabel lblTeléfono = new JLabel("Teléfono:");
191     private final JTextField txtTeléfono = new JTextField(50);
192     private final JLabel lblEmail = new JLabel("Correo Electrónico:");
193     private final JTextField txtEmail = new JTextField(50);
194     private final JLabel lblDirección = new JLabel("Dirección:");
195     private final JTextArea txtDirección = new JTextArea(3, 50);
196     private final JMenuItem itmNuevo = new JMenuItem("Nuevo Proveedor");
197     {
198         itmNuevo.addActionListener(this);
199     }
200     private final JMenuItem itmGuardar = new JMenuItem("Guardar");
201     {
202         itmGuardar.addActionListener(this);
203     }
204     private final JMenuItem itmEliminar = new JMenuItem("Eliminar");
205     {
206         itmEliminar.addActionListener(this);
207     }
208     private final JMenuItem itmActualizar = new JMenuItem("Actualizar");
209     {
210         itmActualizar.addActionListener(this);
211     }
212     private final JMenuItem itmSalir = new JMenuItem("Salir");
213     {
214         itmSalir.addActionListener(this);

```

```

215     }
216     private final JMenu menuArchivo = new JMenu("Archivo");
217     {
218         menuArchivo.add(itmNuevo);
219         menuArchivo.add(itmGuardar);
220         menuArchivo.add(itmEliminar);
221         menuArchivo.add(itmSalir);
222     }
223     private final JMenu menuVer = new JMenu("Ver");
224     {
225         menuVer.add(itmActualizar);
226     }
227     private final JMenuBar menuBar = new JMenuBar();
228     {
229         menuBar.add(menuArchivo);
230         menuBar.add(menuVer);
231     }
232     private final JPanel panel = new JPanel(new GridBagLayout());
233     {
234         panel.add(scrlObjetos, new GBC(0, 0, 1, 2).insets(12, 12, 12, 12)
235             .fill(GBC.BOTH).weight(1.0, 1.0));
236         panel.add(lblClave, new GBC(1, 0).insets(0, 12, 12, 12));
237         panel.add(txtClave, new GBC(1, 1).insets(0, 0, 12, 12));
238         panel.add(lblNombre, new GBC(2, 0).insets(0, 12, 12, 12));
239         panel.add(txtNombre, new GBC(2, 1).insets(0, 0, 12, 12).weight(1.0,
240             0.0).fill(GBC.HORIZONTAL));
241         panel.add(lblTeléfono, new GBC(3, 0).insets(0, 12, 12, 12));
242         panel.add(txtTeléfono, new GBC(3, 1).insets(0, 0, 12, 12).weight(
243             1.0, 0.0));
244         panel.add(lblEmail, new GBC(4, 0).insets(0, 12, 12, 12));
245         panel.add(txtEmail, new GBC(4, 1).insets(0, 0, 12, 12).weight(1.0,
246             0.0));
247         panel.add(lblDirección, new GBC(5, 0).insets(0, 12, 12, 12));
248         panel.add(new JScrollPane(txtDirección), new GBC(5, 1).insets(0, 0,
249             12, 12).fill(GBC.BOTH).weight(1.0, 0.0));
250     }
251     public FrmMantenimiento() {
252         setTitle("Proveedores");
253         setJMenuBar(menuBar);
254         setContentPane(new JScrollPane(panel));
255         setDefaultCloseOperation(EXIT_ON_CLOSE);
256         pack();
257         actualiza();
258     }
259     /**
260     * Se invoca al seleccionar algún elemento de menú.
261     */
262     public void actionPerformed(ActionEvent evt) {
263         // Obtiene el item que genera el evento.
264         Object origen = evt.getSource();
265         // Detecta el objeto que genera el evento.
266         if (origen == itmNuevo) {
267             nuevo();
268         } else if (origen == itmGuardar) {

```

```

269         guarda();
270     } else if (origen == itmEliminar) {
271         elimina();
272     } else if (origen == itmActualizar) {
273         actualiza();
274     } else if (origen == itmSalir) {
275         System.exit(0);
276     }
277 }
278 /**
279  * Detecta el cambio en el objeto seleccionado en la tabla.
280  */
281 public void valueChanged(ListSelectionEvent evt) {
282     // Descarta el evento cuando no es un evento final de cambio
283     // de selección.
284     if (evt.getValueIsAdjusting()) {
285         return;
286     }
287     // Obtiene el renglón seleccionado.
288     int renglón = tblObjetos.getSelectedRow();
289     if (renglón >= 0) {
290         // Obtiene el objeto del renglón seleccionado.
291         selección = objetos[renglón];
292         // Muestra el proveedor seleccionado.
293         txtClave.setText(String.valueOf(selección.getClave()));
294         txtNombre.setText(selección.getNombre());
295         txtTeléfono.setText(selección.getTeléfono());
296         txtEmail.setText(selección.getEmail());
297         txtDirección.setText(selección.getDirección());
298         /*
299          * Indica al usuario que la forma está preparada para editar el
300          * proveedor seleccionado y muestra tanto su clave como su
301          * nombre.
302          */
303         setTitle(selección.getClave() + ":" + selección.getNombre()
304             + " - Proveedores");
305     }
306 }
307 /**
308  * Actualiza el la información mostrada, consultando la base de datos.
309  */
310 private void actualiza() {
311     try {
312         objetos = control.actualiza();
313         modelo.fireTableDataChanged();
314         nuevo();
315     } catch (SQLException e) {
316         e.printStackTrace();
317         error(e.getMessage());
318     }
319 }
320 private void elimina() {
321     // Si no hay selección, no borra nada.
322     if (selección == null) {

```

```

323         return;
324     }
325     // Confirma que realmente ha de borrar el objeto.
326     int respuesta = JOptionPane.showConfirmDialog(this,
327         "¿Realmente desea eliminar al proveedor "
328         + selección.getClave() + ":" + selección.getNombre()
329         + "?", "Eliminar", JOptionPane.YES_NO_OPTION);
330     if (respuesta == JOptionPane.YES_OPTION) {
331         try {
332             // Elimina el objeto de la base de datos.
333             control.elimina(selección);
334             // Actualiza el catálogo.
335             objetos = control.actualiza();
336             // Notifica a la tabla que los datos han cambiado.
337             modelo.fireTableDataChanged();
338             actualiza();
339             JOptionPane.showMessageDialog(this, "Proveedor Eliminado");
340         } catch (SQLException e) {
341             e.printStackTrace();
342             error(e.getMessage());
343         }
344     }
345 }
346 /**
347  * Muestra mensajes de error.
348  *
349  * @param mensaje
350  *         El mensaje a mostrar.
351  */
352 private void error(String mensaje) {
353     JOptionPane.showMessageDialog(this, mensaje, "Error",
354         JOptionPane.ERROR_MESSAGE);
355 }
356 /**
357  * Obtiene los datos capturados. Si se trata de un objeto nuevo no
358  * recupera la clave, pues esta será asignada al momento de registrar el
359  * objeto en la base de datos.
360  *
361  * @return Un objeto que contiene los datos capturados.
362  * @throws Exception
363  *         Si algún valor capturado no es correcto.
364  */
365 private Proveedor getObjeto() throws Exception {
366     String nombre = txtNombre.getText().trim();
367     String teléfono = txtTeléfono.getText().trim();
368     String email = txtEmail.getText().trim();
369     String dirección = txtDirección.getText().trim();
370     Proveedor objeto = new Proveedor();
371     objeto.setNombre(nombre);
372     objeto.setTeléfono(teléfono);
373     objeto.setEmail(email);
374     objeto.setDirección(dirección);
375     if (selección != null) {
376         // Está modificando un objeto. Hay que recuperar la clave.

```

```

377         int clave = -1;
378         try {
379             clave = Integer.parseInt(txtClave.getText().trim());
380         } catch (NumberFormatException e) {
381             throw new Exception("Clave Incorrecta");
382         }
383         objeto.setClave(clave);
384     }
385     return objeto;
386 }
387 /**
388  * Guarda en la base de datos los datos mostrados en la forma de
389  * propiedades. En el caso de registrar un nuevo objeto, recupera la
390  * clave asignada por la base de datos.
391  */
392 public void guarda() {
393     try {
394         // Obtiene los datos capturados por el usuario.
395         Proveedor objeto = getObjeto();
396         // Define el tipo de operación a realizar.
397         if (selección == null) {
398             // No hay un objeto seleccionado, por lo tanto se está
399             // creando un nuevo objeto.
400             // Agrega el objeto en la base de datos y lo recupera
401             // con la clave asignada por la base de datos.
402             control.agrega(objeto);
403             // Actualiza el catálogo.
404             actualiza();
405             mensaje("Proveedor Agregado");
406         } else {
407             // Hay un objeto seleccionado, por lo tanto se está
408             // modificando la información de un objeto.
409             control.modifica(objeto);
410             // Actualiza el catálogo.
411             actualiza();
412             mensaje("Proveedor Modificado");
413         }
414     } catch (Exception e) {
415         e.printStackTrace();
416         error(e.getLocalizedMessage());
417     }
418 }
419 /**
420  * Muestra un mensaje informativo.
421  *
422  * @param mensaje
423  *         El mensaje a mostrar.
424  */
425 private void mensaje(String mensaje) {
426     JOptionPane.showMessageDialog(this, mensaje);
427 }
428 /**
429  * Permite capturar un nuevo objeto.
430  */

```



```
431     private void nuevo() {  
432         selección = null;  
433         tblObjetos.clearSelection();  
434         txtClave.setText("");  
435         txtNombre.setText("");  
436         txtTeléfono.setText("");  
437         txtEmail.setText("");  
438         txtDirección.setText("");  
439         /*  
440          * Indica al usuario que la forma está preparada para crear un nuevo  
441          * proveedor.  
442          */  
443         setTitle("Nuevo Proveedor - Proveedores");  
444     }  
445 }
```

5. Creación de un Ejecutable.

Los siguientes pasos permiten crear un programa instalador.

5.1. Creación del Archivo Manifest.

Crea un archivo de definiciones (en inglés se le conoce como manifest), donde se indica cual es la clase principal y los archivos jar de los que depende.

5.1.2. Código fuente de “manifest.txt”.

```
1 Main-Class: FrmMantenimiento  
2 Class-Path: derby.jar derbyLocale_es.jar  
3
```

La última línea está en blanco y que después del signo “:” sigue un espacio en blanco.

5.2. Creación del Archivo Ejecutable (jar).

Crea un archivo de instrucciones que realice las siguientes operaciones:

1. Compilar las del programa clases dejar los archivos compilados en una carpeta aparte.
2. Empacar las clases para generar el archivo ejecutable (de extensión “.jar”).
3. Ejecutar el programa para probar que funciona correctamente.

A continuación se muestra un ejemplo:

5.2.1. Código fuente de “empaca.bat”.

```

1 rem edita este archivo para que la ruta de los programas
2 rem javac y jar coincida con tu computadora
3 rem crea el directorio clases
4 mkdir clases
5
6 rem compila con los siguientes parámetros:
7 rem javac es el compilador
8 rem -sourcepath src src es el directorio donde están los
9 rem archivos fuente
10 rem -d clases clases es el directorio donde se dejan
11 rem las clases compiladas
12 rem src\*.java archivos a compilar
13 "C:\Archivos de programa\Java\jdk1.6.0_03\bin\javac" -sourcepath src -d
   clases src\*.java
14
15 rem crea un archivo de distribución
16 rem jar es la insrtucción para empacar
17 rem c crear un nuevo archivo
18 rem f proveedores.jar el archivo generado se llama proveedores.jar
19 rem m manifest.txt usa el archivo manifest.txt como archivo de
20 rem definiciones
21 rem -C clases temporalmente se cambia a la carpeta clases
22 rem . incluye todo el directorio actual (clases)
23 "C:\Archivos de programa\Java\jdk1.6.0_03\bin\jar" cvfm proveedores.jar
   manifest.txt -C clases .
24
25 rem elimina la carpeta de clases
26 rmdir /s /q clases
27
28 rem ejecuta el programa
29 "C:\Archivos de programa\Java\jdk1.6.0_03\bin\java" -jar proveedores.jar

```

Las líneas 13 y 23 son demasiado largas y se están desplegadas en dos renglones, pero deben ponerse en un solo renglón del archivo real.

Como resultado tienes el archivo proveedores.jar, que se puede ejecutar haciendo doble clic en su ícono, o tecleando la instrucción:

```
java -jar proveedores.jar
```

Donde quiera que copies “proveedores.jar”, debes colocar también los archivos “derby.jar” y “derbyLocale_es.jar”.

5.3. Construcción del Instalador.

Las instrucciones siguientes son para el programa Inno Setup, que se encarga de generar programas de instalación. Se necesita un archivo de definiciones como el siguiente:

5.3.1. Código fuente de “proveedores.iss”.

```

1  [Setup]
2  ; Nombre de la aplicación
3  AppName=Proveedores
4  ; Nombre de la aplicación con versión
5  AppVerName=Proveedores 0.1
6  ; Nombre del que publica
7  AppPublisher=Ramptors Network
8  ; Dirección del sitio de la aplicación
9  AppPublisherURL=http://ramptors.net/gil/java/proveedores
10 ; Carpeta base de la aplicación al instalar {pf} es el directorio de
11 ; aplicaciones
12 DefaultDirName={pf}\Proveedores
13 ; Nombre del grupo en el menú de inicio
14 DefaultGroupName=Proveedores
15 ; Nombre del archivo instalador
16 OutputBaseFilename=setup-proveedores-0.1
17 ; Algoritmo de compresión
18 Compression=lzma
19 ; Tipo de compresión
20 SolidCompression=true
21 ;directorio donde se deja el instalador
22 OutputDir=instalador
23 ; Idiomas del instalador
24 [Languages]
25 Name: es; MessagesFile: SpanishStd-5-5.1.11.isl
26 ; Archivos a incluir en el instalador
27 [Files]
28 ; Ícono
29 Source: proveedores.ico; DestDir: {app}; Flags: ignoreversion
30 ; Archivo jar
31 Source: proveedores.jar; DestDir: {app}; Flags: ignoreversion
32 ; Archivo jdbc
33 Source: derby.jar; DestDir: {app}; Flags: ignoreversion
34 ; Archivo jdbc
35 Source: derbyLocale_es.jar; DestDir: {app}; Flags: ignoreversion
36 ; Runtime de java para no preocuparnos si está instalado o no
37 Source: C:\Archivos de programa\Java\jre1.6.0_03\*; DestDir: {app}\jre;
   Flags: ignoreversion recursesubdirs createallsubdirs
38 [Icons]
39 ; Accesos en la barra de inicio
40 Name: {group}\Proveedores; Filename: {app}\jre\bin\javaw.exe; Parameters: "-
   jar proveedores.jar"; WorkingDir: {app}; IconFilename: {app}\proveedores.ico

```

Las líneas 37 y 40 se muestran es varios renglones, pero cada una debe ocupar una sola línea del archivo real.

A partir de este archivo, el programa Inno Setup crea el archivo “setup-proveedores-0.1.exe” dentro de la carpeta “instalador”. El programa permite instalar el programa en cualquier computadora sin necesidad de tener instalado Java. También crea un acceso directo en el menú de archivo. Otra forma de escribir la línea 40 del archivo es:

```
Name: {group}\Proveedores; Filename: {app}\jre\bin\javaw.exe; Parameters: "-cp  
proveedores.jar;derby.jar;derbyLocale_es.jar FrmMantenimiento"; WorkingDir: {app};  
IconFilename: {app}\proveedores.ico
```

Esta forma de invocación permite tener varias clases principales en el mismo archivo jar. La opción -cp de los parámetros indica el CLASSPATH, que es el conjunto de archivos y directorios donde se buscan las clases utilizadas por un programa.

6. Distribución en Red.

Cuando una aplicación es usada por muchos usuarios y es necesario que estos se actualicen a la última versión, la distribución con instaladores es poco recomendable. En vez de esto, se usa la distribución en red con Java Web Start. Con esta técnica, que funciona vía web, los usuarios utilizan siempre la misma versión. Los usuarios se conectan a una página web desde la cual automáticamente se comprueba la versión y de ser necesario, se descarga el nuevo código.

La primera vez que se ejecuta, se crean íconos para lanzar la ejecución en el menú de inicio y en el escritorio.

Para poder realizar la distribución en red, es necesario contar con el archivo jar de la aplicación (por ejemplo, “proveedores.jar”) y las bibliotecas auxiliares (“derby.jar” y “derbyLocale_es.jar”). Posteriormente hay que seguir los pasos listados en este capítulo.

6.1. Firmar Digitalmente los Archivos.

Cuando un programa distribuido vía red necesita hacer uso de los recursos de una computadora, es necesario que se firme digitalmente para garantizar su autenticidad. El primer paso es crear un certificado digital y con este, se procede a firmar los archivos.

6.1.1. Código de “firma.bat”.

```
1 rem edita este archivo para que la ruta de los programas
2 rem keytool y jarsigner coincida con tu computadora
3
4 rem crea un certificado digital
5 rem -genkeypair genera un par de llaves (pública y privada)
6 rem -dname      X.500 Distinguished Name se usa para identificar
7 rem             entidades, y tiene las siguientes partes
8 rem             CN = Nombre común
```

```

 9 rem          OU = Unidad dentro de la organización
10 rem          O = Nombre de la organización
11 rem          L = Nombre de la localidad
12 rem          S = Nombre del estado
13 rem          C = Pais
14 rem -alias      Identificador para el par de llaves
15 rem -keypass     Contraseña para proteger las llaves
16 rem -keystore    Archivo donde se almacenan las llaves
17 rem -storepass   Contraseña del keystore
18 rem -validity    Número de días que el certificado se considera
19 rem              válido
20 "C:\Archivos de programa\Java\jdk1.6.0_03\bin\keytool" -genkeypair -dname
   "CN=Pedidos, OU=TIC, O=UTN, C=mx" -alias utn -keypass utn123 -keystore
   demostore -storepass stol23 -validity 180
21
22 rem firma el archivo proveedores.jar usando
23 rem -signedjar    Nombre del archivo firmado
24 rem se está usando el par de llaves bajo el alias utn
25 "C:\Archivos de programa\Java\jdk1.6.0_03\bin\jarsigner" -keystore demostore
   -storepass stol23 -keypass utn123 -signedjar sproveedores.jar proveedores.jar
   utn
26
27 rem firma el archivo derby.jar
28 "C:\Archivos de programa\Java\jdk1.6.0_03\bin\jarsigner" -keystore demostore
   -storepass stol23 -keypass utn123 -signedjar sderby.jar derby.jar utn
29
30 rem firma el archivo derbyLocale_es.jar
31 "C:\Archivos de programa\Java\jdk1.6.0_03\bin\jarsigner" -keystore demostore
   -storepass stol23 -keypass utn123 -signedjar sderbyLocale_es.jar
   derbyLocale_es.jar utn

```

Las líneas 20, 25 , 28 y 31 deben ocupar un solo renglón.

6.2. Archivo JNLP

Este archivo describe la aplicación, los archivos que al conforman y el sitio desde el cual se descarga.

6.2.1 Código de “proveedores_red.jnlp”.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <jnlp spec="1.0+" codebase="http://192.168.7.35/proveedores"
3   href="proveedores_red.jnlp">
4   <information>
5     <!-- Titulo de la Aplicación -->
6     <title>Mantenimiento de Proveedores</title>
7     <!-- Proveedor de la Aplicación -->
8     <vendor>Ramptors Network</vendor>
9     <description>Ejemplo de Mantenimiento de Catálogos</description>
10    <!-- Permite que se ejecute una copia local de la
11      aplicación cuando la red no responde -->
12    <offline-allowed />
13    <!-- Ícono de la aplicación -->

```

```

14     <icon href="proveedor32.gif" />
15     <!-- Accesos directos -->
16     <shortcut online="true">
17         <!-- Crea un acceso directo en el escritorio -->
18         <desktop />
19         <!-- Grupo del menú de inicio -->
20         <menu submenu="Proveedores Red" />
21     </shortcut>
22 </information>
23 <!-- Restricciones de seguridad -->
24 <security>
25     <!-- Esta aplicación requiere el máximo de privilegios; para
26         ello es necesario que todos los jar utilizados estén firmados
27         con la misma llave
28     -->
29     <all-permissions />
30 </security>
31 <!-- Archivos que conforman la aplicación -->
32 <resources>
33     <!-- Versión de Java necesaria para ejecutar la aplicación -->
34     <j2se version="1.6+" />
35     <!-- Archivos que conforman la aplicación -->
36     <jar href="sproveedores.jar" />
37     <jar href="sderby.jar" />
38     <jar href="sderbyLocale_es.jar" />
39 </resources>
40 <!-- Clase principal de la aplicación -->
41 <application-desc main-class="FrmMantenimiento" />
42 </jnlp>

```

6.3. Colocar los Archivos en el Servidor Web.

En el servidor se colocan el archivo de extensión jnlp y los archivos a los que hace referencia, que en este caso son: “proveedor32.gif”, “sproveedores.jar”, “sderby.jar” y “sderbyLocale_es.jar”. También se necesita un archivo de extensión html que se usa para lanzar la aplicación por primera vez.

6.3.1. Código de “proveedores.html”.

```

1 <!-- versión de HTML empleada. -->
2 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
3     "http://www.w3.org/TR/html4/strict.dtd">
4 <HTML>
5 <HEAD>
6     <!-- Juego de caracteres utilizado para codificar el documento. -->
7     <META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8     <TITLE>Mantenimiento de Proveedores</TITLE>
9 </HEAD>
10 <BODY>
11     <!-- Control ActiveX para Internet Explorer que lanza archivos JNLP -->
12     <OBJECT
13         codebase="http://java.sun.com/update/1.6.0/jinstall-1_6-windows-

```



```
14 i586.cab#Version=6,0,0,0"  
15 classid="clsid:5852F5ED-8BF4-11D4-A245-0080C6F74284" height=0 width=0>  
16 <PARAM name="app"  
17 value="http://192.168.7.35/proveedores/proveedores.jnlp">  
18 <PARAM name="back" value="true">  
19 <!-- Esta sección es para navegadores que no pueden crear el control -->  
20 <A href="proveedores.jnlp">Mantenimiento de Proveedores</A> Esta  
21 aplicaci&oacute;n requiere tener instalado Java 6, que se puede  
22 descargar de <A href="http://java.com"> http://java.com </A>  
23 </OBJECT>  
24 </BODY>  
25 </HTML>
```

La línea 13 se muestra en dos renglones, pero debe ocupar un solo renglón y sin espacios en blanco.