**LENGUAJE DE PROGRAMACION II**

Docente: Ing. Díaz Leyva Teodoro

**Tema: JDBC: Acceso a Base de Datos** **Semana N°01**

**ODBC (**Open Database Connectivity)

Es una interface de programación de aplicaciones (API) para acceder a datos en sistemas gestores de bases de datos tanto relacionales como no relacionales, utilizando para ello SQL (Lenguaje de Consulta Estructurado).

Todas las aplicaciones que soporten ODBC reconocerán una instrucción común de Lenguaje de Consulta Estructurado (SQL).

Las aplicaciones ODBC o aplicaciones cliente envían peticiones a un servidor de bases de datos. El gestor del driver ODBC determina qué fuente de datos usar y qué driver ODBC puede comunicar con esa fuente de datos en particular. La petición se envía luego a través del driver al servidor, normalmente una aplicación de base de datos. Esta base de datos puede ser local, o en el mismo computador, o remota. Los datos solicitados se devuelven a través del gestor del driver ODBC, entonces a la aplicación del cliente. El lenguaje ODBC es una combinación de llamadas de función ODBC API y lenguaje SQL.

**JDBC**

La API (Interfaz de Programación de Aplicaciones) de JDBC provee acceso a datos desde Java. Usando esta API podemos acceder a variadas fuentes de datos, bases de datos relacionales, hojas de cálculo y archivos planos.

De una forma simple, JDBC posibilita tres cosas:

- Abrir la conexión a la base de datos
- Ejecutar consultas contra la base de datos
- Procesar los resultados

Pero JDBC abstrae al desarrollador del Sistema Gestor que tenga nuestra base de datos por lo que tendremos que cargar inicialmente el controlador que lleve a cabo dicha función, introduciendo así un nuevo paso anterior a todos los demás.

- Cargar la clase del controlador JDBC

Estos pasos están reflejado en la fig. N°1:

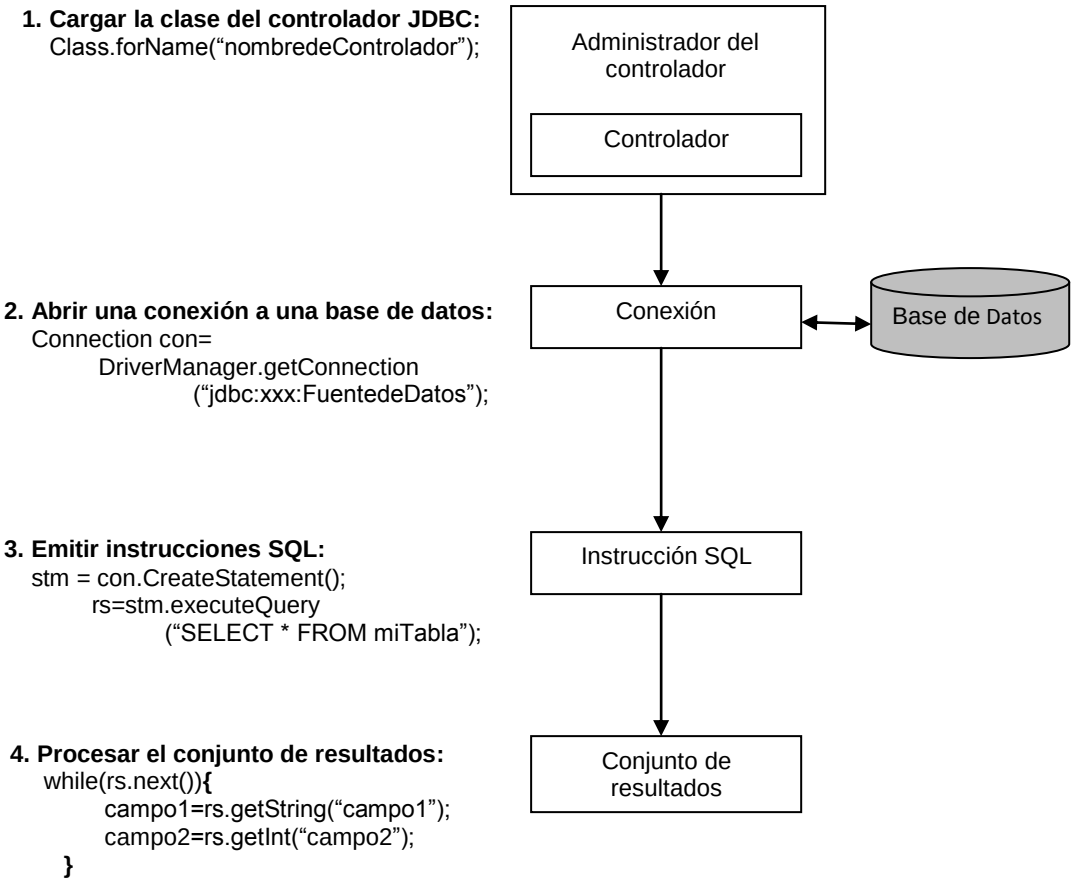



Fig. N°1. Pasos Básicos de JDBC

JDBC hace de intermediario entre nuestro programa escrito en Java y la base de datos para cumplir con estos pasos, como se muestra en la fig. N° 2

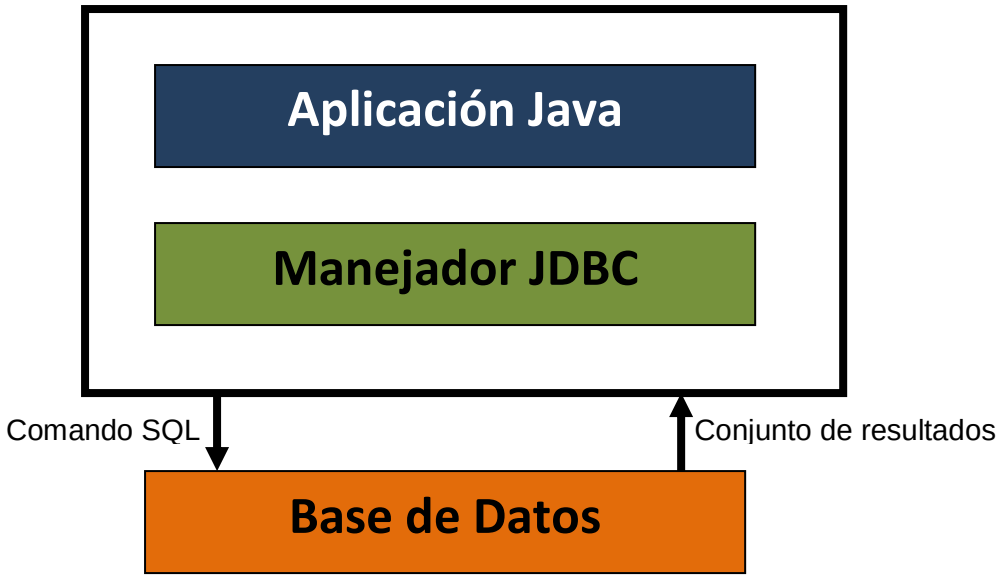



Fig. N° 2. Arquitectura de JDBC

**Clases de JDBC**

La interfaz JDBC reside en los paquetes **java.sql** y **javax.sql.** En su mayoría son interfaces, ya que la implementación específica de cada una de ellas es específica de cada empresa según su protocolo de bases de datos.

- **Connection:** Un enlace activo a una base de datos a través del cual un programa en Java puede leer y escribir datos, así como explorar la estructura de la base de datos y sus capacidades.

- **Statement:** Es la clase que proporciona los métodos para que la sentencia SQL sea ejecutada sobre la base de datos y recupere el resultado.

  Hay tres tipos de Statement y cada uno especializa al anterior:

  - **Statement:** es el encargado de ejecutar las sentencias SQL estáticas. Se crea con Connection.createStatement().

- **PreparedStatement:**se utiliza para ejecutar las sentencias SQL precompiladas. Permite que los parámetros de entrada sean establecidos de forma dinámica.

  Se crean con Connection.preparedStatement("sql string").

- **CallableStatement:** un preparedStatement que llama un procedimiento almacenado, es decir, métodos almacenados en la propia base de datos. No todos los SGBD lo admiten pero es muy útil para los que sí.

- **ResultSet**: es la clase a la que pertenece el objeto en el que se devuelve la respuesta a la petición que hicimos a la base de datos. No es más que un conjunto ordenado de filas de una tabla.

  Es lo que se devuelve cuando ejecutamos el método statement.executeQuery("sql string"). Existen métodos como next() y getXXX() para iterar por las filas y obtener los valores de los campos que queramos.

- **DatabaseMetaData:** ofrece una interfaz para operar con la estructura y capacidades de la base de datos.

  Se obtiene de connection.getMetaData().

- **ResultSetMetaData:** es el ResultSet que se devuelve al hacer un executeQuery de un objeto DatabaseMetaData.

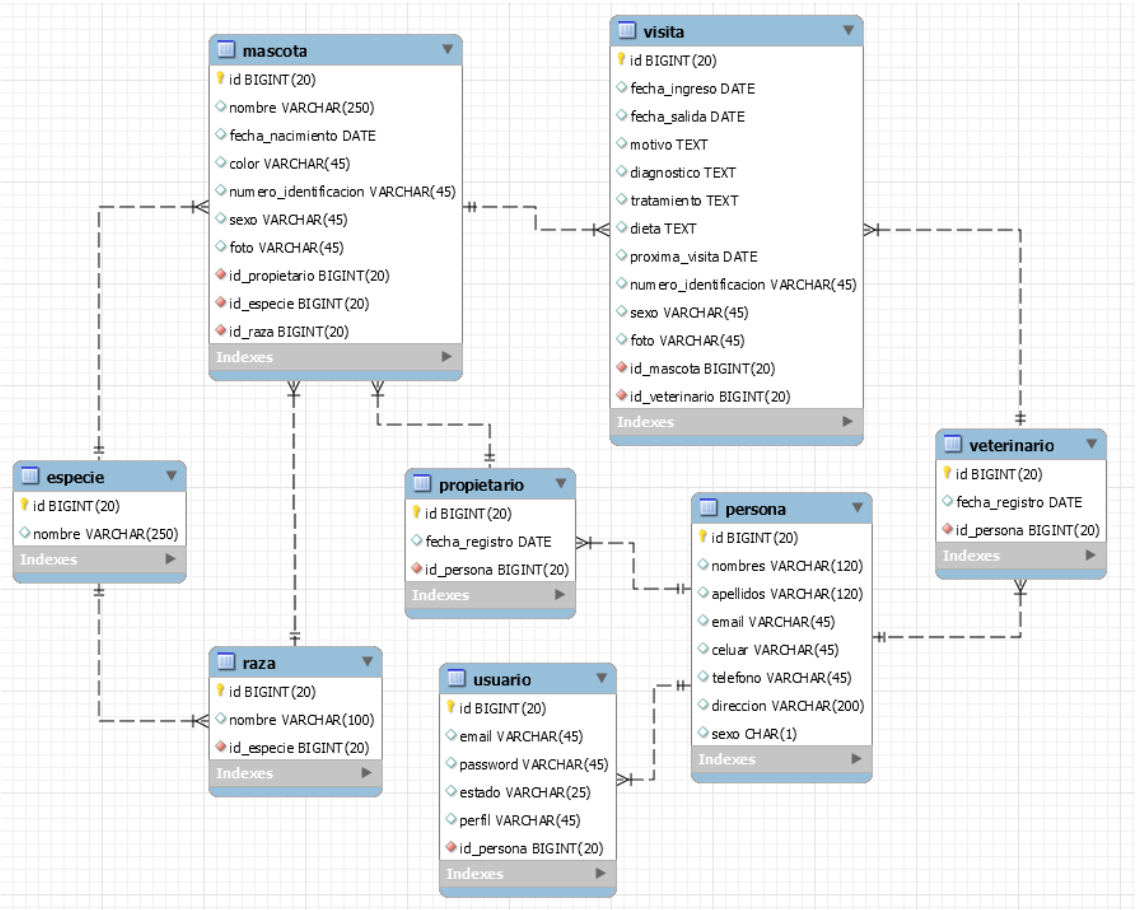
- **DriverManager:** interfaz que registra los controladores JDBC y proporciona las conexiones que permiten manejar las URL específicas de JDBC. Se consigue con el método getConnection de la propia clase**.**

- **SQLException:** La clase base de las excepciones de JDBC

## PASOS PARA OPERAR CON JDBC SOBRE UNA BASE DE DATOS

Para desarrollar los pasos con mayor detalle, necesitamos primero una base de datos, la cual iniciaremos creando la base de datos “clinicamas” y la tabla “persona” en el Servidor de base de datos MySQL, se enfoca en gestionar una clínica de mascotas. Básicamente se registrará las visitas de los animales que previamente deben ser registrados junto con los propietarios.

Esquema de la base de datos “clinicamas”:



--Creación de la base de datos clinicamas

```
create  database  clinicamas;
```

--seleccionar  la base de datos

```
use clinicamas;
```

--Creación de tablas de la base de datos

```
create table persona(
 id bigint  primary key auto_increment not null,
 nombres varchar(120),
 apellidos varchar(120),
 email  varchar(45),
 celuar varchar(45),
 telefono varchar(45),
 direccion varchar(200),
 sexo char(1)
)ENGINE= InnoDB;
```

***Aclaración:***

**PRIMARY KEY.** Se llama **clave primaria** a un campo o a una combinación de campos que identifica de forma única a cada fila de una tabla. De esta manera no puede haber dos filas en una tabla que tengan la misma clave primaria.

**InnoDB**: Tablas de transacción segura con bloqueo de fila y claves foráneas.

--Insertar datos o registros en la tabla persona

```
insert into persona(nombres ,apellidos ,email  ,celuar ,telefono ,direccion ,sexo )
values('Antonio','Silva Zevallos','ant@hotmail.com','955788523','611622','Jr. Los
Nogales N°225 San Isidro','M');

insert into persona(nombres ,apellidos ,email  ,celuar ,telefono ,direccion ,sexo )
values('Alberto','Cerron Castañeda','cerr@hotmail.com','955118523','611624','Jr. Los
Alisos N°111 Surco','M');

insert into persona(nombres ,apellidos ,email  ,celuar ,telefono ,direccion ,sexo )
values('Silvia','Morales Alva','sil@hotmail.com','933118523','699624','Jr. Los Pinos
N°618 Los Olivos','F');
```

Tenemos la base de datos **clinicamas** con una tabla persona, a la cual vamos a conectarnos con los pasos descritos anteriormente:

1. **Registrar la clase del controlador JDBC**

   Driver para acceder a MySQL:

```
Class.forName("com.mysql.jdbc.Driver");
```

***Aclaración***

Para conectarse a una base de datos a través de JDBC desde una aplicación Java, lo primero que se necesita es cargar el driver que se encargará de convertir la información que se envía a través de la aplicación a un formato que lo entienda la base de datos. Esta parte del código sería la única que dependería del tipo de driver y del tipo de base de datos.

La siguiente es una lista de tres formas diferentes de registrar un controlador JDBC:

- Class.forName(String driverName).newInstace()
- DriverManager.registerDriver(Driver driverName)
- jdbc.driver property

Class.forName(String driverName).newInstace() es la forma más común para registrar un controlador JDBC.

DriverManager provee el método DriverManager.registerDriver (Driver driverName) para registrar controladores JDBC. El siguiente fragmento de código le muestra que tan sencillo es este método, note que es necesario crear una nueva instancia del tipo Driver y pasarlo como parámetro: **DriverManager.registerDriver(new com.mysql.jdbc.Driver());**

2. **Abrir la conexión a la base de datos** clinicamas

```
String usuario="root";
String password="123";
String url="jdbc:mysql://localhost/clinicamas";
Connection cn = DriverManager.getConnection(url, usuario, password);
```

***Aclaración***

Un objeto **Connection** representa una conexión a una base de datos. Una sesión con una conexión incluye las sentencias SQL que son ejecutadas y los resultados que son devueltos a través de dicha conexión. Una misma aplicación puede tener una o más conexiones con una sola base de datos o puede tener conexiones con varias bases de datos diferentes.

La forma estándar de establecer una conexión con una base de datos es llamando al método **DriverManager.getConnection**. Este método toma como parámetros una cadena de caracteres que contiene una URL, un usuario y una clave. La clase **DriverManage** trata de localizar el driver que pueda conectar con la base de datos representada por esa URL.

Una URL de JDBC facilita una forma de identificar una base de datos de forma que el driver apropiado la reconozca y establezca una conexión con ella. La sintaxis estándar para URLs de JDBC es la siguiente:

<protocolo>:<subprotocolo>:<subnombre>

- **Protocolo** es siempre **jdbc.**
- El **subprotocolo** se refiere al controlador que se va a usar y ha suministrado la empresa distribuidora.
- **Subnombre** identifica la base de datos a la que conectarse, pudiendo contener parámetros de conexión.

Ejemplo de JDBC-ODBC

```
String url=jdbc:odbc:myBD;
```

En este ejemplo el valor de <subprotocolo> es odbc y queremos conectarnos con un ODBC DSN llamado myBD, que es el valor para <subname>. Recuerde que cuando usa ODBC es responsable de configurar correctamente el DSN en la computadora del cliente antes de usar su aplicación.

Ejemplo de MySQL:

```
String url = "jdbc:mysql://localhost/BD";
```

Ejemplo de Oracle:

```
String url="jdbc:oracle:thin:@localhost:1521:xe";
```

El valor del <subprotocolo> es **oracle:thin**. La palabra Oracle es un estándar en los controladores de Oracle, por su parte, **thin** se refiere al mecanismo de conexión específico que Oracle utiliza. Finalmente, @localhost:1521:xe, dice al controlador de Oracle el host, port, y la instancia de la base de datos con la que se realizara la conexión.

### 3. Ejecutar consultas contra la base de datos clinicamas

```
Statement stm = cn.createStatement();

String sql=SELECT * FROM persona;

stm. executeQuery(sql);
```

***Aclaración***

Un objeto **Statement** es el que envía las sentencias SQL al controlador de la base de datos. Simplemente se creará un objeto **Statement** y se ejecuta, utilizando el método SQL apropiado con la sentencia SQL que se desea enviar. Para una sentencia **SELECT**, el método a ejecutar es **executeQuery**. Para sentencias que crean o modifican tablas, el método a utilizar es **executeUpdate**.

Se toma un objeto de una conexión activa para crear un objeto **Statement**. En el ejemplo, utilizamos el objeto **Connection** (**cn)** para crear el objeto **Statement**(**stm)**.

En este momento **stm** existe, pero no tiene ninguna sentencia SQL que pasarle al controlador de la base de datos. Necesitamos suministrarle el método que utilizaremos para ejecutar **stm**. En el ejemplo, suministramos **executeQuery** con la sentencia SQL: String sql=SELECT *FROM persona;

**4. Procesar los resultados**

El **ResultSet** recibe los datos que devuelve **executeQuery**(sql);

```
ResultSet rs= stm. executeQuery(sql);
```

***Aclaración***
Como se ha visto **executeQuery** de las diferentes interfaces devuelven bien el número de líneas modificadas si realizan una actualización, un **ResultSet** en caso de ser una consulta, así que veremos cómo tratar esta clase especial.

El **conjunto de resultados** es una lista ordenada de filas de tabla en JDBC. Para extraer los resultados de este conjunto de datos se puede ayudar uno de los siguientes métodos:

- Para desplazarse por el conjunto de resultados sólo se dispone de ***next().***
- Para recuperar los valores de las columnas deseadas se dispone de:

**ResultSet.getXXX**(numero_columna) o **ResultSet.getXXX**(nombre_columna), donde **XXX** es un tipo de dato JDBC.

Se utiliza bucle while para leer el conjunto de registros del ResultSet:

```
while(rs.next()) {
        System.out.println("Código:"+ rs.getString("id"));
        System.out.println("Nombre:"+ rs.getString("nombres"));

    }
```

En este caso se imprime el código y nombre de las personas, si en caso que el ResultSet tuviera un sólo un registro se utilizaría **if** para la lectura.

### 5. Cerrar la conexión a la base de datos

```
cn.close();
```

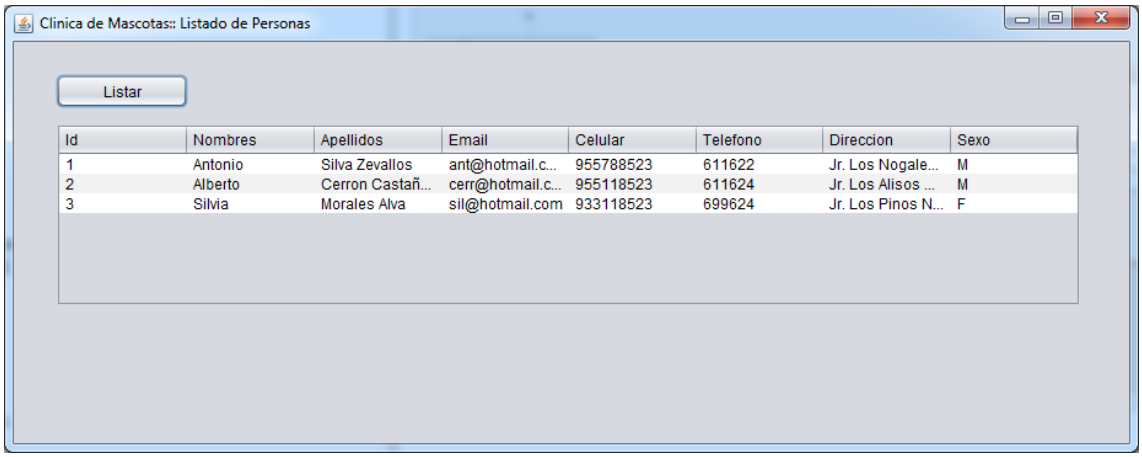
***Aclaración***

Cuando ya no se desee mantener un diálogo con la base de datos se optará por cerrar la conexión.

Este es el paso más sencillo de todos, basta con llamar al método ***Connection.close().***

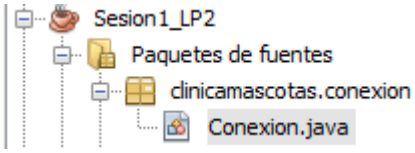
Ejemplo

Listar las personas en una tabla:



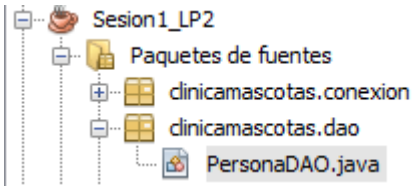

Desarrollo:

- Crear proyecto con el nombre Sesion1_LP2, en seguida crea un paquete clinicamascotas.conexion y la clase Conexión:




```
package clinicamascotas.conexion;

import java.sql.Connection;
import java.sql.DriverManager;

public class Conexion {
    private static final String url = "jdbc:mysql://localhost/clinicamas";
    private static final String user = "root";
    private static final String pass = "123";
    private static Connection cn;
    public static Connection abrir() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            cn = DriverManager.getConnection(url, user, pass);
        } catch (Exception ex) {
            return null;
        }
        return cn;
    }
}
```

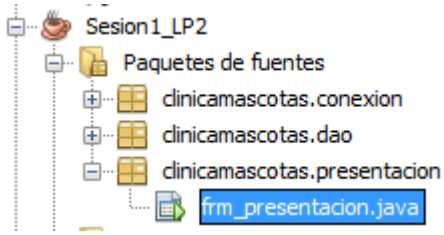
- Crear paquete clinicamascotas.dao y la clase PersonaDAO:

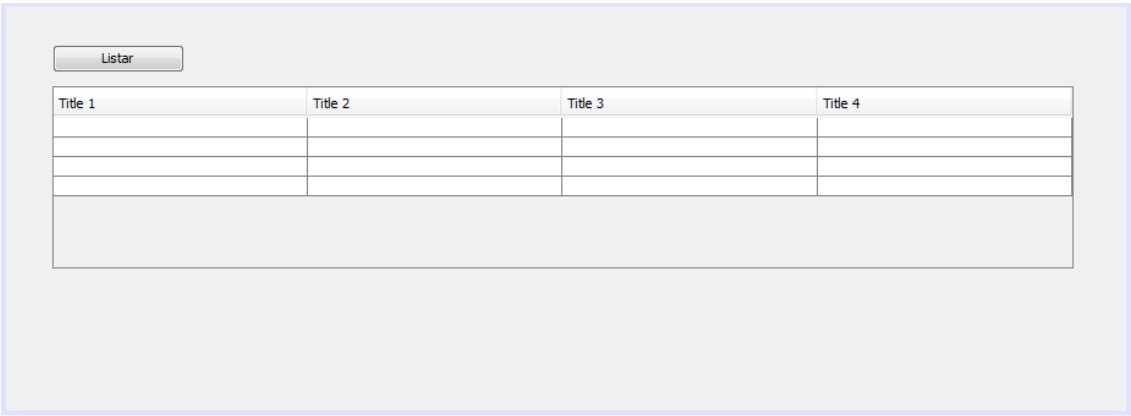



```
package clinicamascotas.dao;

import clinicamascotas.conexion.Conexion;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

public class PersonaDAO {

    //declaración de variables
    static Connection cn;
    static Statement stm;
    static ResultSet rs;

    public static void listar(JTable t) {

        //encabezado de la tabla
        String[] encabezadoTabla = {"Id", "Nombres", "Apellidos", "Email", "Celular",
"Telefono", "Direccion", "Sexo"};

        //modelo de tabla
        DefaultTableModel mod_tabla = new DefaultTableModel(null, encabezadoTabla);

       //asignar modelo al jTable Personas
        t.setModel(mod_tabla);

        //abrir conexion a la base de datos
        cn = Conexion.abrir();

        try {
            //objeto Statement es el que envía las sentencias SQL al controlador
            //de de base de datos
            stm = cn.createStatement();

            //ejecutar objeto stm con la intruccion sql
            rs = stm.executeQuery("select*from persona");
```

```
            //leer el conjunto de registros  y asignar al modelo de tabla
            while (rs.next()) {
                mod_tabla.addRow(new Object[]{rs.getInt(1), rs.getString(2), rs.getString(3),
rs.getString(4),rs.getString(5), rs.getString(6), rs.getString(7), rs.getString(8)});
            }

            //cerrar objetos
            stm.close();
            cn.close();

        } catch (SQLException ex) {

        }

    }
}
```

- Crear paquete **clinicamascotas.presentacion** y un formulario con un botón y una tabla:



- En el constructor de del formulario asigne título  al formulario:

```
public frm_presentacion() {
      initComponents();

      setTitle("Clinica de Mascotas:: Listado de Personas");

   }
```

- En el boton btn_listar llame al método listar de la clase PersonaDAO

```
private void btn_listarActionPerformed(java.awt.event.ActionEvent evt) {

          PersonaDAO.listar(tabla_personas);
   }
```

_____**EJECUTAR**