Práctica 7

JDBC

Programación Orientada a Objetos

Humberto Alejandro Ortega Alcocer

Opción: 1 “Insertar Perros en una Base de Datos”

14 de enero de 2021

Grupo: 2CM1

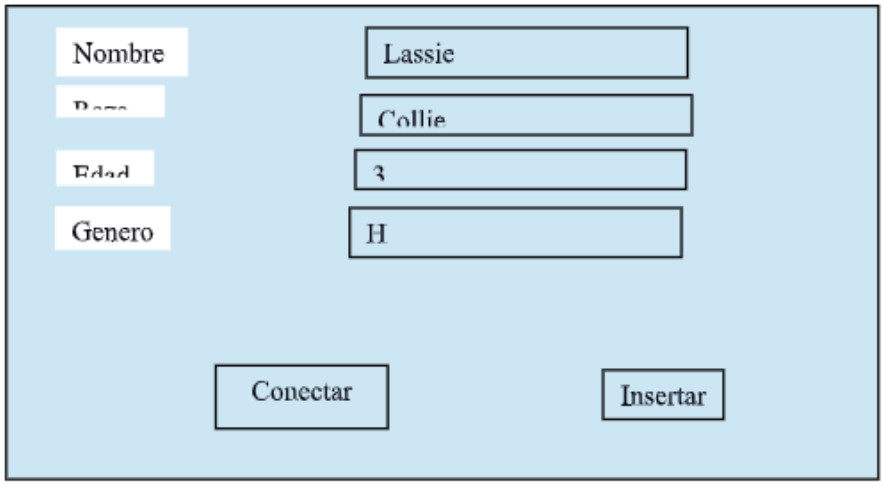
Introducción

Las bases de datos son un elemento clave en el desarrollo de sistemas de información que sean capaces de realizar operaciones con datos de forma eficiente y funcional. En el caso de Java, el JDBC es el complemento que nos permite conectarnos con distintos gestores de bases de datos como MySQL, PostgreSQL, Microsoft SQL y, al ser agnóstico a cualquiera de los gestores, nos permite tener garantía de que nuestro código podrá ser adaptado de forma fácil a cualquiera de los gestores que deseemos utilizar.

Para esta práctica, se realizó un formulario sencillo de captura de información y su posterior almacenamiento en una base de datos. La información a registrar corresponde a un perro, por lo que son pocos los campos a implementar y conectar con la base de datos.

El planteamiento de la práctica, en palabras del profesor Roberto Tecla, es:

1.-**Insertar Perros** en una base de datos
Hacer un programa que utilice la API **JDBC** y que tenga una GUI que permita insertar "perros" en una base de datos que tiene una tabla Perro con el siguiente esquema Perro(nombre, raza, edad, genero). La GUI consistirá en 4 etiquetas, 4 campos de entrada y 2 botones. Un botón para conectar a la base de datos y otro para insertar los datos de un perro.



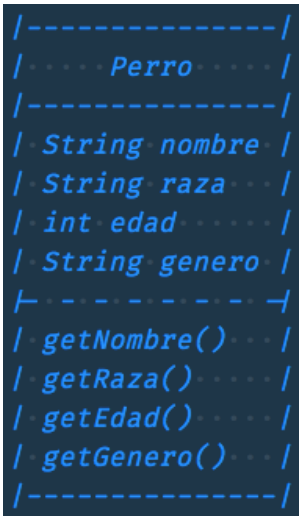
Desarrollo

En esta práctica, debemos comenzar por entender cómo se opera en un esquema de operación con una base de datos externa con la cual interactuará nuestro programa. Para esto, primero nos aseguraremos de tener el *Driver* adecuado para nuestro gestor de bases de datos instalado en nuestro entorno de desarrollo. Si se trabaja con un IDE, se puede especificar, mediante el sistema de compilación definido por el mismo, la dependencia hacia un *Driver* particular pero, en mi caso, al no usar un IDE dedicado, he tenido que descargar el *Driver* para PostgreSQL (que es el gestor de base de datos que utilizo) y ligarlo mediante un *export* antes de realizar la compilación.

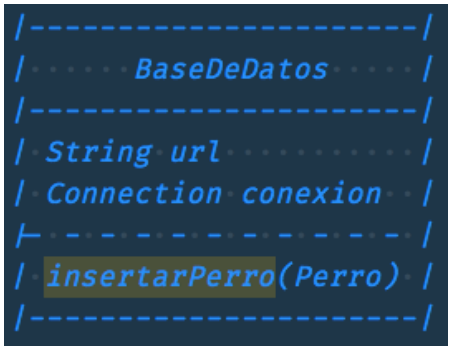
Para simplificar este proceso de exportar la ruta para que el compilador de Java encuentre la ruta al *Driver*, escribí un *script* sencillo (en bash) que permite realizar la tarea de compilación y ejecución de forma apropiada. Los contenidos del script son:

```
# Este script realiza la compilación y ejecución de la práctica 7.

# Para usar el JDBC de Postgres.
export CLASSPATH=postgresql-42.2.18.jar:.

# Para compilar y ejecutar.
javac *.java \
  && java GUIPerro

# Humberto Alcocer.
```

Esto asumiendo que colocamos el archivo `.jar` dentro del mismo directorio dónde se encuentran el resto de los archivos `.java` que contienen nuestro programa. Teniendo nuestras dependencias ligadas apropiadamente, proseguimos con el desarrollo de la práctica determinando la clase Perro, la cual representa, como su nombre menciona, un Perro y nos permitirá realizar operaciones de forma que se encuentre encapsulado su comportamiento interno. El diagrama UML de la clase Perro es:



Como se puede observar, no merece la pena ahondar en la implementación de esta clase ya que únicamente representa un Perro. La clase que revisaremos es *BaseDeDatos* ya que en ella se realiza la conexión con JDBC. El diagrama UML de la clase es:



La implementación de esta clase es muy simple, y para entenderla debemos revisar sus dos funciones: el constructor, dónde ocurre la tarea de conexión con nuestra base de datos en sí, y el método de inserción, dónde se prepara la consulta con la información necesaria para insertarla a la base de datos. La implementación de la clase es:

```java
public class BaseDeDatos {
  private String url = "jdbc:postgresql:poo";
  private Connection conexion;

  // Constructor de la clase.
  public BaseDeDatos() {
    // Realizamos el intento de conexión dentro de un bloque try/catch.
    try {
      // Class.forName("org.postgresql.Driver");
      conexion = DriverManager.getConnection(this.url);
    } catch (Exception e) {
      System.out.println("Ha ocurrido un problema conectando con la base de datos:");
      e.printStackTrace();
    }
  }

  // Método para realizar la inserción de un nuevo Perro en la base de datos.
  public boolean insertarPerro(Perro nuevoPerro) {
    String consulta = "INSERT INTO perro VALUES (?,?,?,?)";

    // Ejecutamos la consulta dentro de un bloque try/catch.
    try {
      CallableStatement statement = conexion.prepareCall(consulta);

      // Ajustamos los parámetros de la consulta.
      statement.setString(1, nuevoPerro.getNombre());
      statement.setString(2, nuevoPerro.getRaza());
      statement.setInt(3, nuevoPerro.getEdad());
      statement.setString(4, nuevoPerro.getGenero());

      // Realizamos la consulta en la base de datos.
      statement.executeUpdate();

      // Si no se arrojó una excepción, asumimos que la operación fue exitosa.
      return true;
    } catch (Exception e) {
      // Mostramos información del error.
      System.out.println("Ocurrió un error al tratar de insertar el nuevo Perro:");
      e.printStackTrace();

      // La ejecución no fue exitosa.
      return false;
    }
  }
}
```

Como se puede observar, la conexión es muy simple. En mi gestor local de base de datos (PostgreSQL) he decidido quitar el requisito de un usuario y contraseña para fines de simpleza en esta práctica por lo que la cadena de conexión realmente es trivial. La base de datos se llama “poo” y la conexión se realiza dentro del constructor de la clase.

El método de inserción también es muy simple, primero preparamos la consulta a realizar, colocamos los parámetros para la misma, y ejecutamos la consulta. Si se arroja una excepción, sea cual sea el motivo, se regresará un valor falso y si la operación es exitosa, se regresa un valor verdadero.

En cuanto a la interfaz gráfica se refiere, la clase que implementa tanto dicha interfaz como la lógica del programa es GUIPerro. El diagrama UML de la clase es:

Realmente en esta clase la parte gráfica se puede omitir ya que es muy simple su implementación. Para la parte lógica, la función “actionPerformed” es llamada cuando el usuario hace click en el botón “conectar” o en “enviar”. La idea es que todos los campos se encuentren bloqueados, salvo el botón “conectar”, hasta que

el usuario utilice el botón “conectar” para realizar la conexión con la base de datos. Una vez realizada la conexión con la base de datos el programa entonces bloquea el botón “conectar” y desbloquea el resto de los elementos.

Cuando el usuario hace click en el botón “enviar” se captura el texto contenido en cada uno de los campos y se llama a la función de inserción dentro de la base de datos. La implementación de la función “actionPerformed” es:

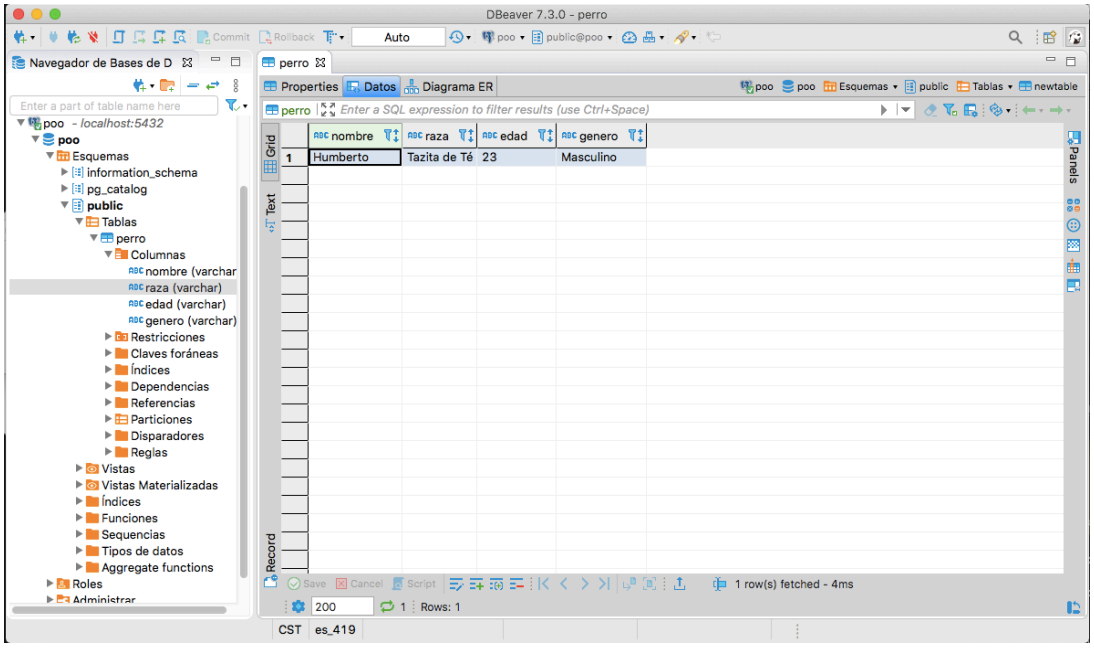
```
public void actionPerformed(ActionEvent evento) {
  // Obtenemos el origen del evento.
  JButton botonOrigen = (JButton) evento.getSource();

  // Verificamos el botón que fue presionado.
  if (botonOrigen == this.botonConectar) {
    // Realizamos la conexión.
    this.baseDeDatos = new BaseDeDatos();

    // Cambiamos el estado del GUI.
    this.botonConectar.setEnabled(false);
    this.botonEnviar.setEnabled(true);
    this.campoNombre.setEnabled(true);
    this.campoEdad.setEnabled(true);
    this.campoRaza.setEnabled(true);
    this.campoGenero.setEnabled(true);

    // Mostramos un mensaje de confirmación.
    JOptionPane.showMessageDialog(null, "¡Conexión con la Base de Datos realizada exitosamente!");
  } else {
    // Realizamos la operación en la base de datos.
    boolean operacionExitosa = this.baseDeDatos.insertarPerro(new Perro(this.campoNombre.getText(),
        this.campoRaza.getText(), Integer.parseInt(this.campoEdad.getText()), this.campoGenero.getText()));

    // Verificamos si la operción fue exitosa.
    if (operacionExitosa) {
      JOptionPane.showMessageDialog(null, "¡Perro registrado exitosamente!");
    } else {
      JOptionPane.showMessageDialog(null, "Ocurrió un error al realizar el registro, intente de nuevo más tarde.");
    }
  }
}
```

Finalmente, la ejecución del programa, en su estado inicial, luce así:

Y una vez realizada la conexión se muestra el siguiente mensaje:

Al llenar los campos y hacer click en enviar se muestra también el siguiente mensaje:

Podemos verificar que la operación fue exitosa observando el dato en la base de datos directamente:



Conclusión

Las bases de datos son uno de los elementos fundamentales en el desarrollo de sistemas informáticos modernos. En esta práctica se pudo apreciar de forma cercana el proceso que conlleva realizar la conexión en el lenguaje de programación Java mediante el paquete JDBC usando el *Driver* de PostgreSQL y nos permite visualizar los distintos pasos y consideraciones que debemos tener al momento de programar este tipo de procesos.

Una de las cuestiones más relevantes de la práctica fue entender el proceso para integrar paquetes externos así como realizar la compilación cuando se depende de uno de éstos. Es importante que sepamos las especificidades del lenguaje ya que, aunque la implementación final puede resultar muy simple, debemos tener presente las estructuras de control que debemos (y podemos) emplear, así como los distintos errores y complicaciones que pueden surgir durante el desarrollo así como durante la ejecución de éste programa.

Si bien, el programa es muy simple en funcionalidad, la integración completa me permitió trabajar en un entorno en el que no había desarrollado este tipo de integración por lo que me parece sumamente interesante el modo en que Java maneja esta funcionalidad.