Primera Aplicación. Parte 2

Nuestra aplicación consistirá en una interfaz que nos permita introducir los datos de un usuario. Lo datos que vamos a solicitar serán:

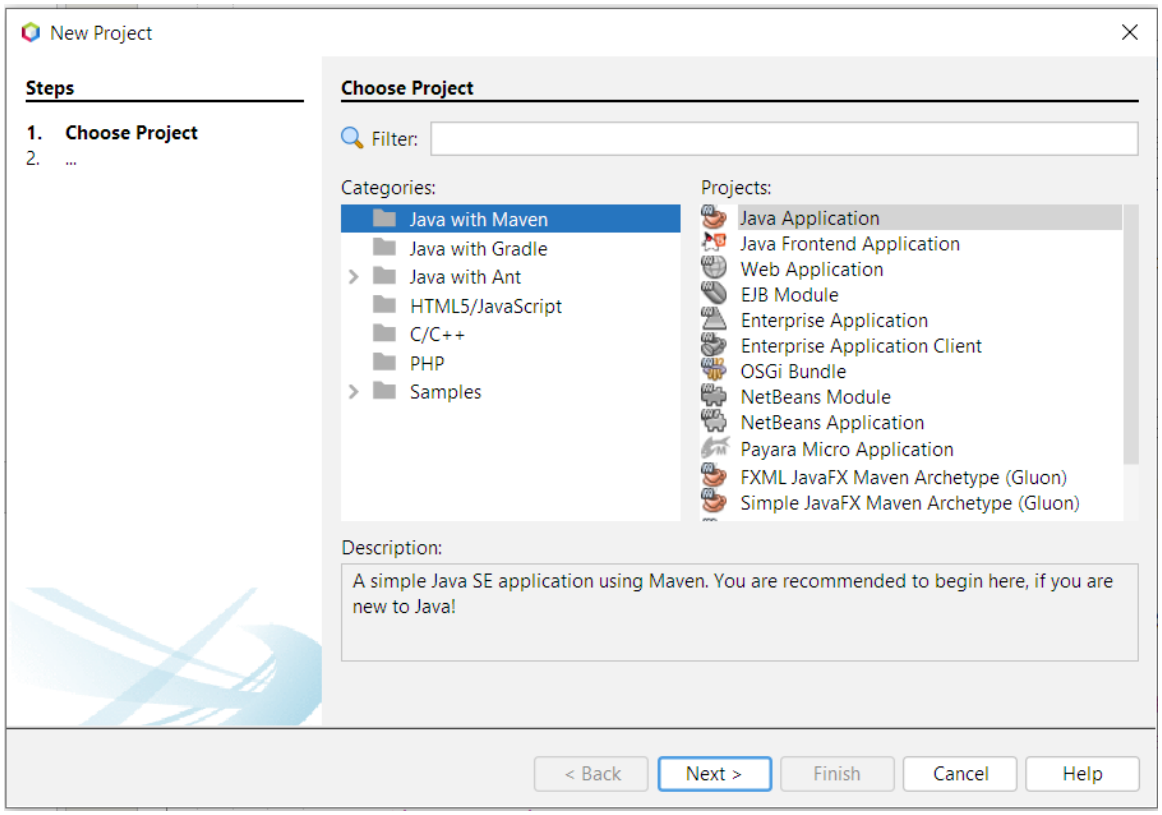
- DNI
- Nombre
- Apellidos
- E-mail
- Password
- Notas (cualquier tipo de anotación que consideremos oportuna sobre esa cuenta)

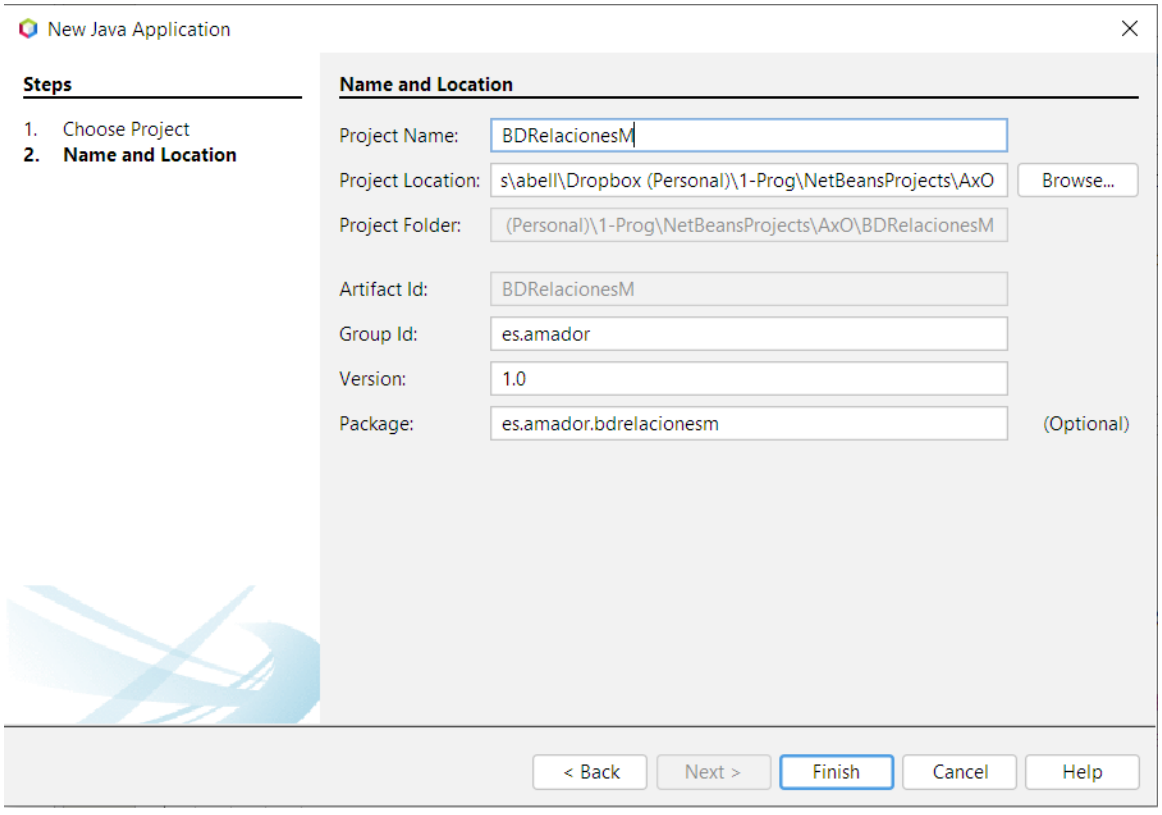
Todos los datos van a ser de tipo conjunto de caracteres, siendo DNI la clave primaria. Solo permitiremos valores nulos en el campo Notas

Desde el punto de vista de la aplicación JavaFX, en lo concerniente al tipo de control necesario para cada campo, estos serán de tipo **TextField** (una única línea de texto), salvo el Password que será **PasswordField** (sustituye los caracteres por puntos en la pantalla), y Notas que será una **TextArea** (nos permite introducir múltiples líneas de texto).

Como ya vimos en el tema 9 de gestión de SGBD relacionales, podríamos crear la tabla desde cualquier herramienta de gestión: PHPMyadmin, MySQL Workbench, el propio NetBeans desde la pestaña Services, directamente en modo texto en MySQL, …. Pero también programando desde Java. Esto último será lo que hagamos en este proyecto.

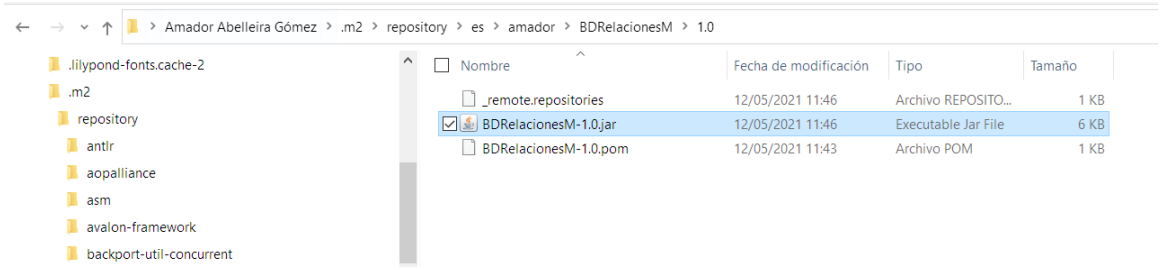
Como ya habíamos creado una librería con métodos para interaccionar con el SGBD, vamos a añadirla como dependencia en el **pom**. Pero para ello necesitamos antes volver a generarla mediante Maven:



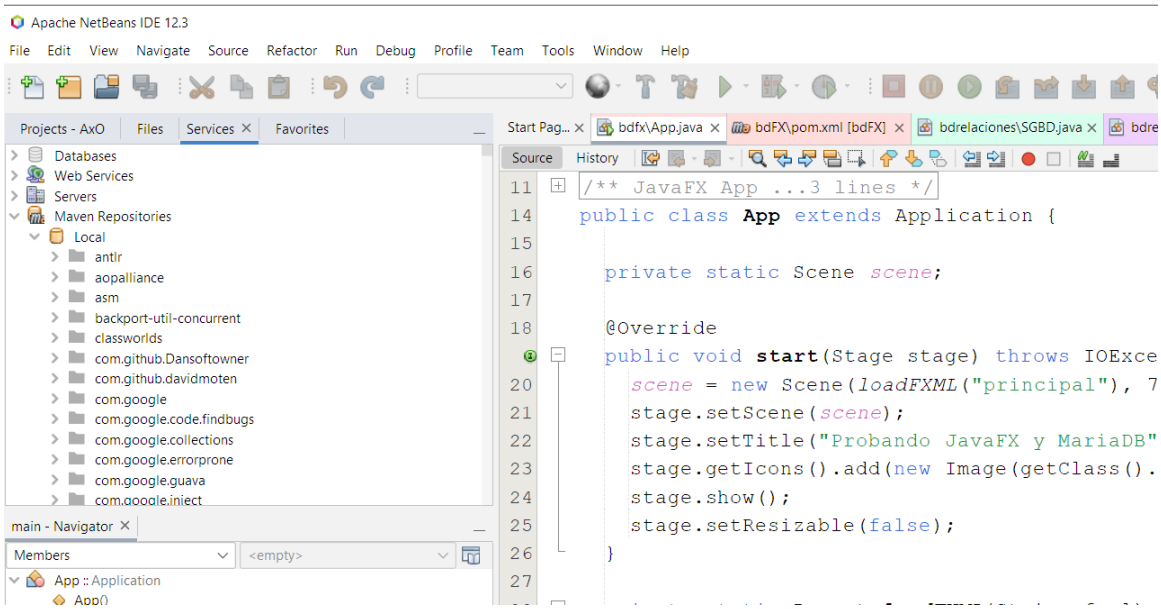


Copiamos la clase del proyecto Ant y la copiamos en el Maven, refactorizando.

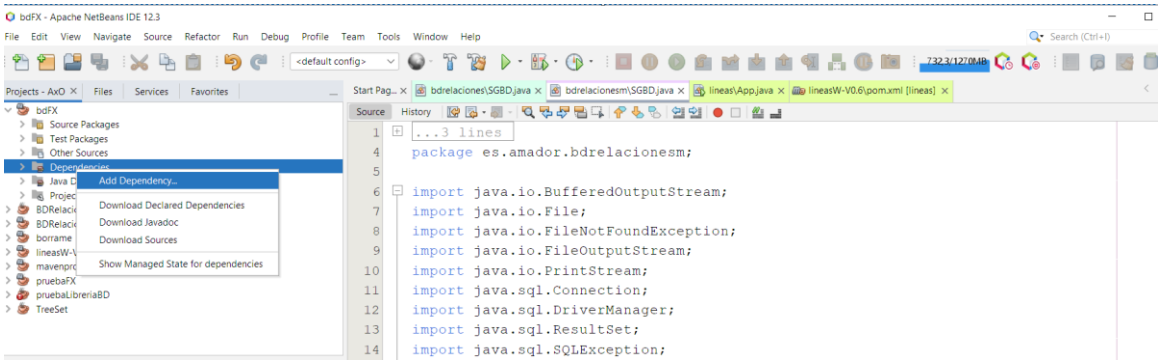
Hacemos un **Clean & Build** del proyecto, y se nos generará en **.jar**. OJO, con Maven no se crean los **.jar** en una carpeta **dist** dentro del proyecto, si no que se generan por defecto dentro del directorio del repositorio **.m2**, en su carpeta correspondiente:



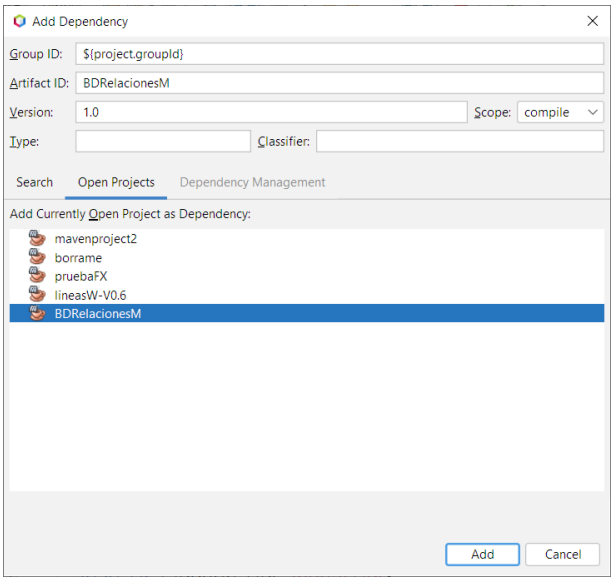
También lo podemos ver desde el propio Netbeans:



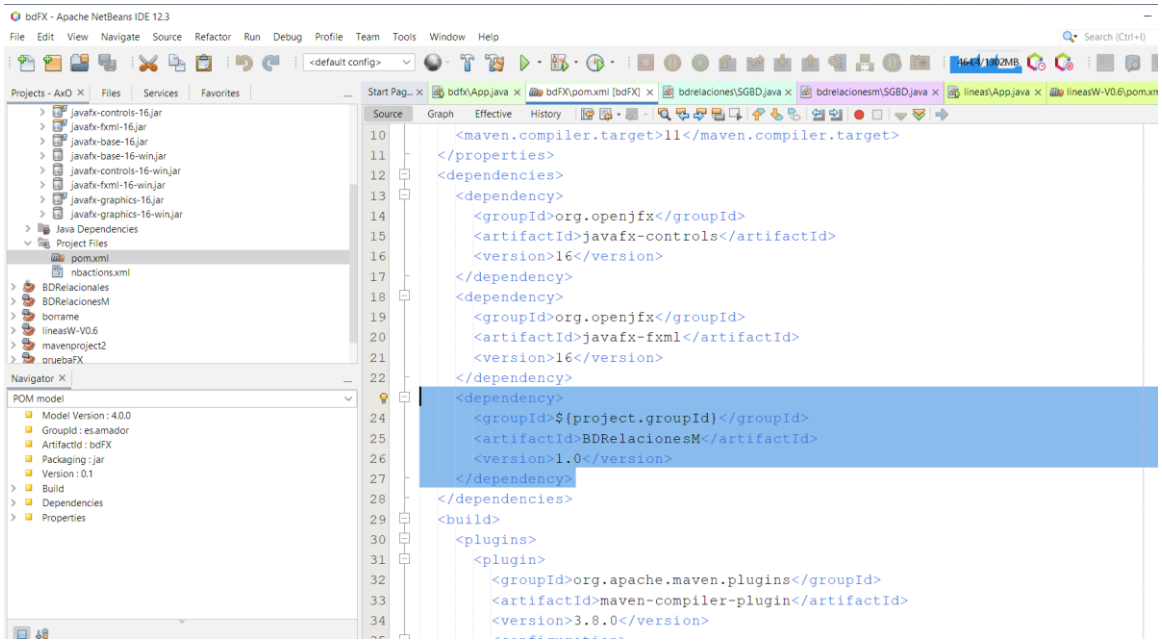
Ahora en nuestro proyecto **bdFX**, o bien editamos manualmente el fichero **.pom**, o bien, más cómodo, pulsamos botón derecho sobre **Dependencies**, y escogemos **Add Dependency**



Escogemos **Open Projects** y dentro de los que nos muestra, el de la librería de BD



Vemos como se ha modificado en **pom.xml**

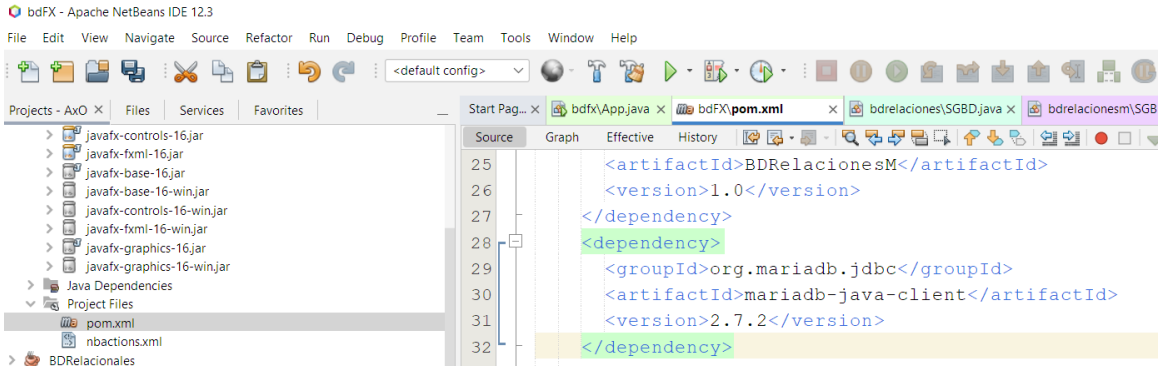


Conexión con MariaDB

En el tema 9, utilizando proyectos Ant, la forma de proceder era descargar el driver para MySQL/MariaDB, crear una nueva librería en NetBenas (Tools->libraries) en la que incluimos los .jar presentes en el paquete descargado, y posteriormente, para cado proyecto en el que queramos hacer uso de estos drivers, escogemos las propiedades del proyecto y en Libraries->Modulepath añadimos la librería que hayamos creado.

En Maven lo que haremos será añadir al fichero **pom.xml** una nueva dependencia indicando qué librería y en qué versión necesitamos emplear, y él se encarga de descargar todo lo necesario.

Editamos el fichero **pom.xml** y añadimos una nueva etiqueta de dependencias. En la siguiente imagen tenéis cuál sería su contenido (tiene autocompletado, con lo cual no hay que saberse de memoria todo el nombre, o la versión):



Le indicamos que necesitamos los drivers para MariaDB, en su versión 2.7.2 Cuando ejecutemos el proyecto, se descargarán los ficheros necesarios.

Como sabéis **MariaDB** es un fork de **MySQL**, así que, en lo que respecta a la programación en Java, no necesitamos cambiar nada en el código. Todos los métodos que habíamos creado en los proyectos del tema de BD, son perfectamente válidos.

Para comprobarlo vamos a realizar la conexión con la BD, y posteriormente la desconexión, de modo que se muestren mensajes por pantalla (no en la aplicación gráfica, sino en el **Output** de **NetBeans**), que nos indiquen si ha habido éxito o no.

Para ello vamos a modificar el fichero **App.java**. Este fichero es el que se corresponde con el método **main()**, o sea, donde comienza la ejecución del código (es un nombre por defecto, no es que teng que llamarse así necesariamente la clase). En JavaFX es un poco diferente a lo visto hasta ahora. La clase pública que alberga, tiene que heredar de la clase Application.

El método **main**() en principio se limita a lanzar la aplicación (launch())

Tenemos además otros tres métodos:

- **Start** donde podemos definir que escena se va a cargar, con qué tamaño por defecto (si no indicamos lo contrario, será reescalable), un título para la ventana,

fijarla y mostrarla. También se podrían cargar aquí los CSS, entre otras posibles opciones.

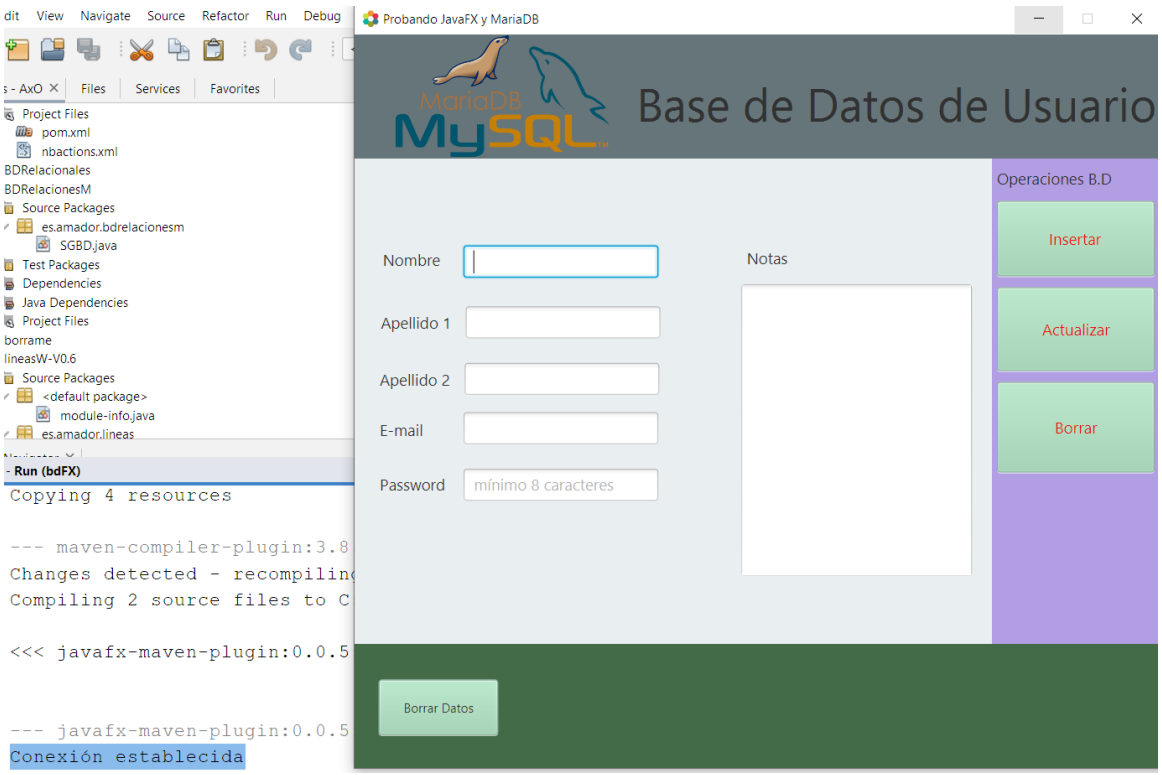
- **setRoot** que valdría para escoger cual sería el documento raíz. Si recordáis, en el proyecto que se genera por defecto, hay dos **.fxml** Aquí diríamos cual de los dos es el que va a arrancar. Como en este caso solo tenemos uno, podemos prescindir de este método.
- **loadFXML** se encarga de cargar el contenido del fxml indicado. En este caso, el único disponible

Prescindimos pues del método **setRoot**, modificamos **Start** para poner título a la ventana, sus dimensiones iniciales y un icono para la aplicación, y añadimos al **main** llamadas a los métodos para conectar y desconectar de la BD fijándonos en el valor devuelto (*false* en caso de fallo) para imprimir por la salida **Output** de NetBeans los mensajes apropiados. **NOTA**, *en esta ocasión utilizamos una versión de la librería sin métodos* ***static***

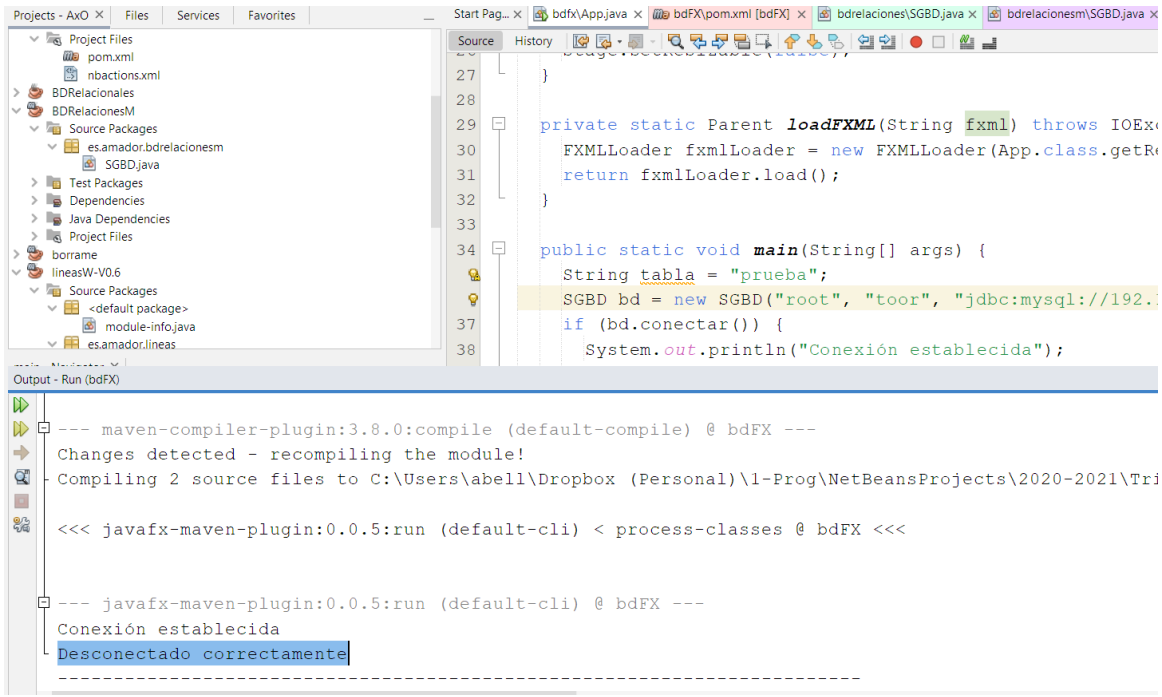
```
@Override
public void start(Stage stage) throws IOException {
  scene = new Scene(loadFXML("principal"), 770, 700);
  stage.setScene(scene);
  stage.setTitle("Probando JavaFX y MariaDB");
  stage.getIcons().add(new Image(getClass().getResourceAsStream("bdFX.png")));
  stage.show();
  stage.setResizable(false);
}
```

```
public static void main(String[] args) {
  String tabla = "prueba";
  SGBD bd = new SGBD("roor", "", "jdbc:mysql://192.168.0.200/java");
  if (bd.conectar()) {
    System.out.println("Conexión establecida");
  } else {
    System.out.println("Error en la conexión");
  }
  launch();
  if (bd.desconectar()) {
    System.out.println("Desconectado correctamente");
  } else {
    System.out.println("Error en desconexión");
  }
}
```

Ejecutamos el proyecto:



Y podemos comprobar que la conexión ha sido correcta. Cerramos ahora la ventana y nos fijamos en el mensaje de desconexión:



Correcta también. Ya podemos proceder a crera nuestra tabla.

Creación de la tabla de usuarios

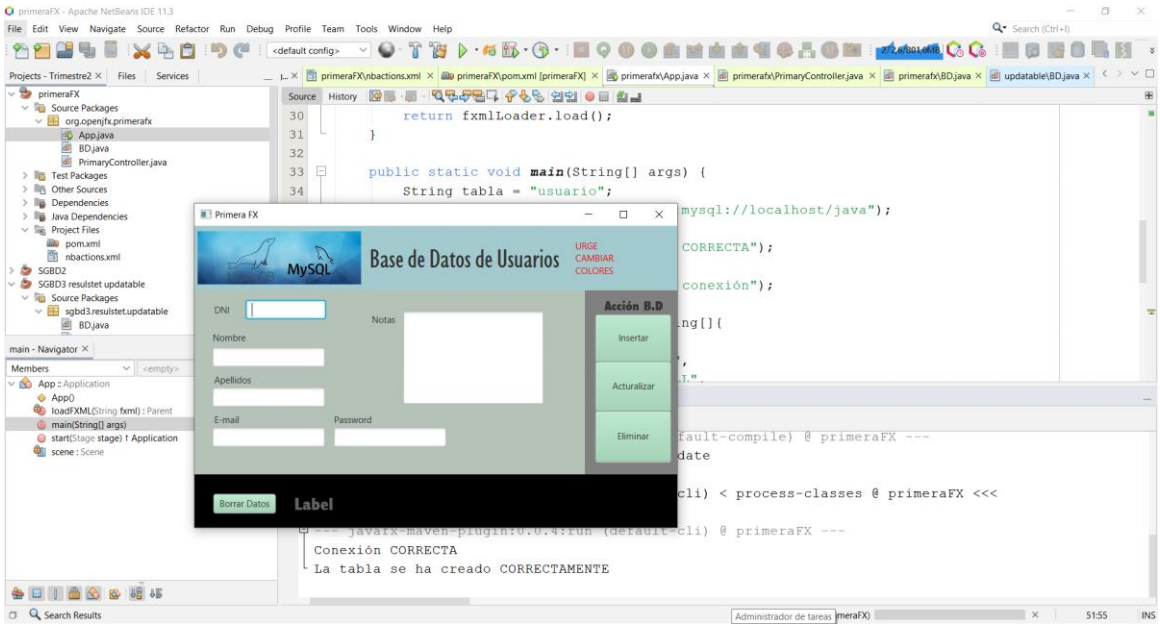
Vamos ahora a incluir el código necesario para crear la tabla de usuarios según las especificaciones dadas al principio del documento.

Como ya teníamos codificado un método que nos permitía la creación de tablas, vamos a utilizarlo ahora. Por ahora seguimos utilizando System.out.println para comprobar la corrección, pero después, o bien, lo mostraremos sobre la componente gráfica o dirigiremos a ficheros de log.

```
public static void main(String[] args) {
  String tabla = "usuario";
  SGBD bd = new SGBD("root", "toor", "jdbc:mysql://192.168.0.200/java");
  bd.redirigeSalidaError("logBdFX");
  if (bd.conectar()) {
    System.out.println("Conexión establecida");
  } else {
    System.out.println("Error en la conexión");
    Platform.exit();
    System.exit(0);
  }
  if (bd.crearTabla(tabla, new String[]{
    "dni VARCHAR(9) PRIMARY KEY",
    "nombre VARCHAR(15) NOT NULL",
    "apellido VARCHAR(30) NOT NULL",
    "email VARCHAR(15) NOT NULL",
    "password VARCHAR(15) NOT NULL",
    "notas VARCHAR(500)"
  })) {
    System.out.println("La tabla se ha creado correctamente");
  }
  else {
    System.out.println("ERROR en la creación de la tabla");
  }
  launch();
```

```
  if (bd.desconectar()) {
    System.out.println("Desconectado correctamente");
  } else {
    System.out.println("Error en desconexión");
  }
}
```

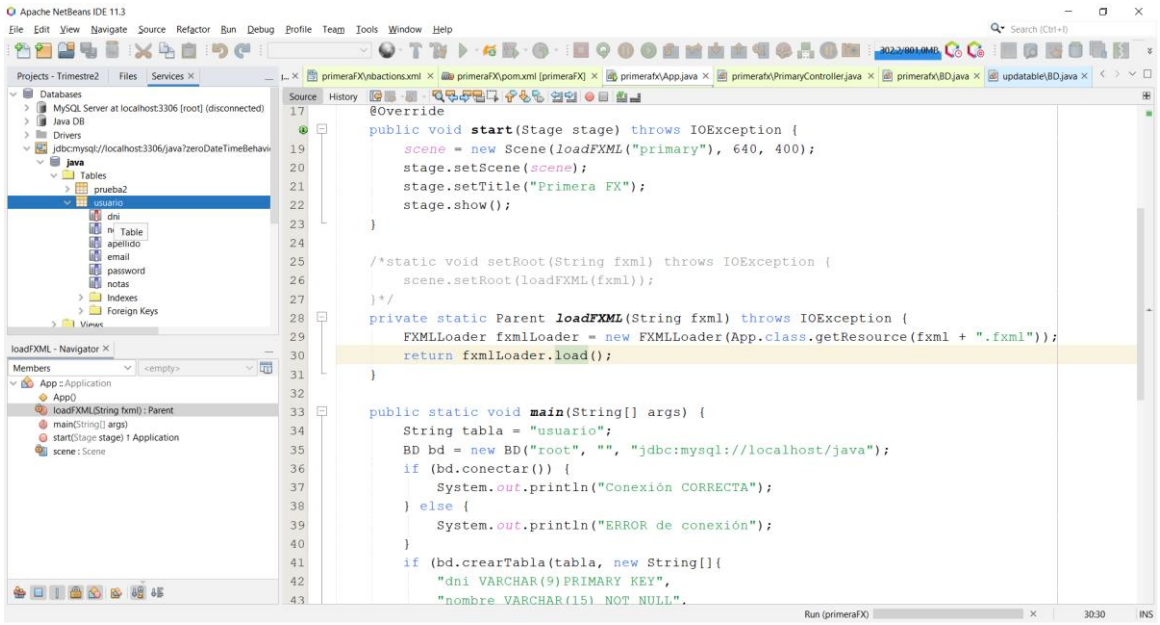
Y ejecutamos el proyecto



Como vemos, la tabla se ha creado correctamente. Bueno, más que creado correctamente, lo correcto sería decir que no se produjo ningún error, porque crearse solo se crea en la primera ejecución. Ejecuciones sucesivas no darán error, pero no la crearán, ya que el método **crearTabla** incluye:

String sql = "CREATE TABLE **IF NOT EXISTS** " + tabla + "(";

Comprobamos que la tabla existe desde NetBeans en la pestaña Services:



Y aquí finaliza esta segunda parte. En la tercera y última, haciendo usor del paradigma de programación orientada a eventos, añadiremos el código que nos permitirá insertar, modificar y borrar usuarios.