



Manual: JAVA + SQLite

Borja García Barrera

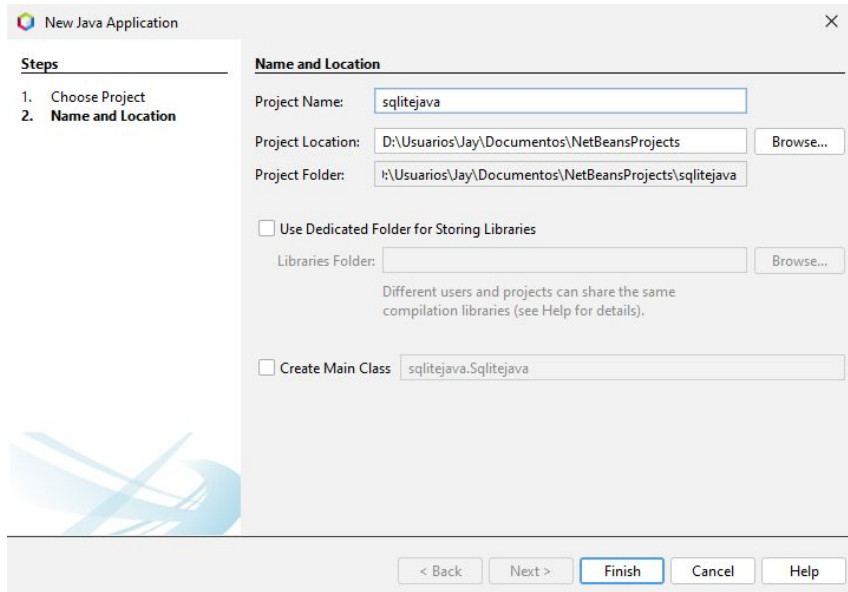
INTRODUCCIÓN



Aprenderemos a:

- **Conectarnos** a una base de datos desde Java
- **Obtener información** de una base de datos
- **Modificar y borrar** la información de la BD
- **Cerrar** la conexión

CREACIÓN DEL PROYECTO



New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name:

Project Location:

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☐ Create Main Class

< Back Next > **Finish** Cancel Help

→ Creamos un proyecto, desmarcando la opción “Create Main Class”.

¿Por qué?

En este caso, trabajaremos desde una interfaz gráfica, por lo que la clase Main no es necesaria.

DISEÑO DE LA INTERFAZ



→ Comenzamos con el diseño gráfico.

Utilizaremos:

- JButton
- JTable

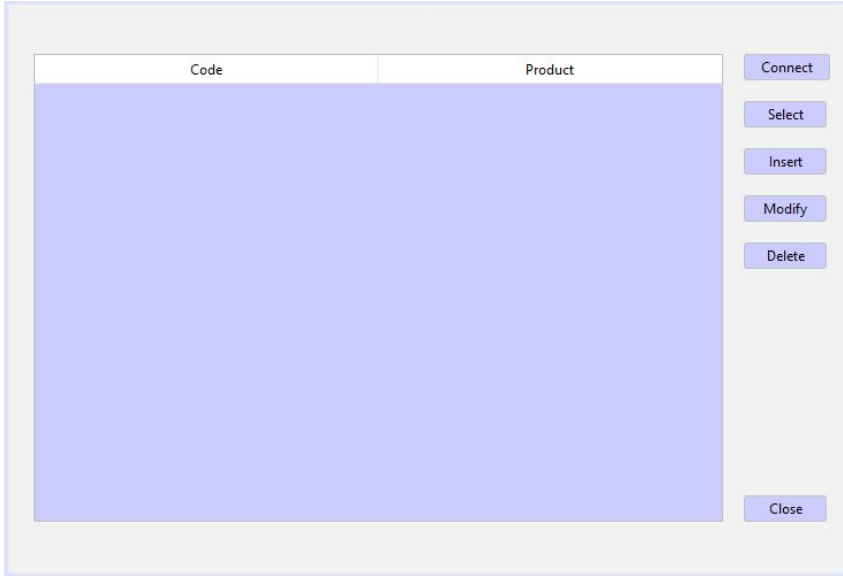
DISEÑO DE LA INTERFAZ



→ JBUTTON

- Connect → Conectarse a BD
- Select → Mostrar información de BD
- Insert → Añadir información a BD
- Modify → Modificar información de BD
- Delete → Borrar información de BD
- Close → Desconectarse de BD

DISEÑO DE LA INTERFAZ



→ JTABLE

- Dos columnas para mantenerlo simple
- Dejamos las **filas** (*rows*) en 0

LIBRERÍAS

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
import java.sql.SQLException;  
  
import javax.swing.table.DefaultTableModel;  
import javax.swing.JOptionPane;
```



sqlite-jdbc-3.41.2.1.jar

→ Importamos las librerías que vamos a utilizar en nuestro programa.

En lugar de importar toda la librería, importamos solamente lo necesario por motivos de optimización.

OBJETOS



```
public formal() {  
    initComponents();  
    model=(DefaultTableModel) this.jTable1.getModel();  
}
```

```
DefaultTableModel model;  
String url = "jdbc:sqlite:src/database/progsql.db";  
Connection connect;
```

→ Creamos los objetos necesarios para que el programa sea funcional.

- **DefaultTableModel** *model*
- **String** *url*
- **Connection** *connect*

OBJETOS



```
public formal() {  
    initComponents();  
    model=(DefaultTableModel) this.jTable1.getModel();  
}
```

```
DefaultTableModel model;  
String url = "jdbc:sqlite:src/database/progsql.db";  
Connection connect;
```

→ **DefaultTableModel** *model*

- Permitirá que nos comuniquemos con la JTable.
- Igualamos nuestro objeto model al modelo de la JTable, casteándolo.

OBJETOS



```
public formal() {  
    initComponents();  
    model=(DefaultTableModel) this.jTable1.getModel();  
}
```

```
DefaultTableModel model;  
String url = "jdbc:sqlite:src/database/progsql.db";  
Connection connect;
```

→ **String url**

- Indicamos al programa la dirección de nuestra base de datos.

OBJETOS



```
public formal() {  
    initComponents();  
    model=(DefaultTableModel) this.jTable1.getModel();  
}
```

```
DefaultTableModel model;  
String url = "jdbc:sqlite:src/database/progsq1.db";  
Connection connect;
```

→ **Connection** *connect*

- A través de este objeto nos podremos comunicar con la base de datos.

CÓDIGO: BOTONES → CONNECT



```
private void ConnectActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
  
    try{  
        connect = DriverManager.getConnection(url);  
        if(connect!=null){  
            JOptionPane.showMessageDialog(parentComponent: null, message: "Connected.");  
        }  
    }catch(Exception x){  
        JOptionPane.showMessageDialog(parentComponent: null, message: x.getMessage().toString());  
    }  
}
```

- Configuramos el Try/Catch
- Llamamos a **getConnection** mediante **DriverManager**, indicando la **URL** de nuestra BD.

CÓDIGO: BOTONES → SELECT

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    model.setRowCount(0);  
    ResultSet result=null;  
  
    try{  
        PreparedStatement st= connect.prepareStatement("select codigo, nombre from producto");  
        result=st.executeQuery();  
  
        while(result.next()){  
            model.addRow(new Object[]{result.getInt("codigo"), result.getString("nombre")});  
        }  
    }catch (Exception x){  
        JOptionPane.showMessageDialog(parentComponent: null, message: x.getMessage().toString());  
    }  
}
```

- En primer lugar, inicializamos las **columnas** a 0
- Creamos el **objeto result**, donde depositaremos el resultado de la **query**
- Con el **objeto PreparedStatement**, indicamos la **instrucción SQL** que ejecutará el botón
- Creamos un **bucle** para leer toda la información contenida en nuestro **objeto result**, añadiéndola a las columnas

CÓDIGO: BOTONES → INSERT

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try{  
        int code;  
        String name;  
  
        code = Integer.parseInt(JOptionPane.showInputDialog(message: "Product code: "));  
        name = JOptionPane.showInputDialog(message: "Product name: ");  
  
        PreparedStatement st = connect.prepareStatement("insert into producto(codigo, nombre) values('" + code + "', '" + name + "')");  
        st.execute();  
        JOptionPane.showMessageDialog(parentComponent: null, message: "Data succesfully saved.");  
  
    }catch (Exception x){  
        JOptionPane.showMessageDialog(parentComponent: null, message: x.getMessage().toString());  
    }  
    refreshTabla();  
}
```

- Creamos **variables** para almacenar la información que nos de el usuario.
- Mediante un **objeto PreparedStatement**, lanzamos la **query**.

CÓDIGO: BOTONES → MODIFY

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    try{  
        int code;  
        String name;  
  
        code = Integer.parseInt(JOptionPane.showInputDialog(message: "Product code: "));  
        name = JOptionPane.showInputDialog(message: "New product name: ");  
  
        PreparedStatement st = connect.prepareStatement("update producto set nombre = '"+name+"' where codigo = '"+code+"'");  
        st.execute();  
        JOptionPane.showMessageDialog(parentComponent: null, message: "Data succesfully modified.");  
    } catch (Exception x){  
        JOptionPane.showMessageDialog(parentComponent: null, message: x.getMessage().toString());  
    }  
    refreshTabla();  
}
```

- Creamos **variables** para almacenar la información que el usuario quiere **modificar**.
- Mediante un **objeto PreparedStatement**, lanzamos la **query**.

CÓDIGO: BOTONES → DELETE

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    int code;  
  
    code = Integer.parseInt(JOptionPane.showInputDialog(message: "Introduce the code you want to delete: "));  
    try{  
        PreparedStatement st = connect.prepareStatement("delete from producto where codigo = '"+code+"'");  
        st.execute();  
        JOptionPane.showMessageDialog(parentComponent: null, message: "Successfully deleted.");  
    }catch(Exception e){  
        JOptionPane.showMessageDialog(parentComponent: null, message: e.getMessage().toString());  
    }  
    refreshTabla();  
}
```

- Creamos una **variable** para almacenar el código del producto que se desea **eliminar**.
- Mediante un **objeto PreparedStatement**, lanzamos la **query**.

CÓDIGO: BOTONES → CLOSE



```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
  
    try{  
        connect.close();  
    }catch (Exception x){  
        JOptionPane.showMessageDialog( parentComponent:null, message:x.getMessage().toString());  
    }  
}
```

→ Utilizamos el **objeto connect** para **cerrar** la conexión con la base de datos.

Es **importante** cerrar siempre la conexión para evitar posibles problemas, como **fallos** al insertar o modificar.

CÓDIGO: REFRESCAR UI

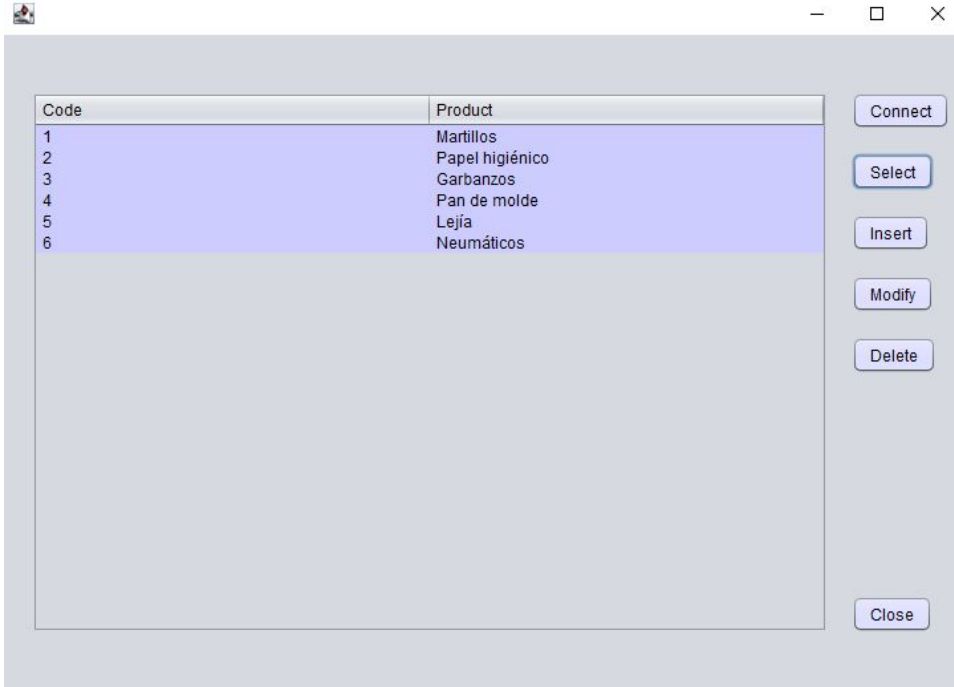


```
public void refreshTabla() {  
    model.setRowCount( rowCount: 0);  
    ResultSet resultado = null;  
  
    try{  
        PreparedStatement st = connect.prepareStatement( string: "select codigo, nombre from producto");  
        resultado = st.executeQuery();  
  
        while(resultado.next()){  
            model.addRow(new Object[]{resultado.getInt( string: "codigo"), resultado.getString( string: "nombre")});  
        }  
    }  
    catch(Exception e){  
        JOptionPane.showMessageDialog( parentComponent: null, message: e.getMessage().toString());  
    }  
}
```

→ Este método es un clon del código del botón **SELECT**.

Añadiéndolo al final de cada botón que modifique la información, refrescamos la UI para que el cambio aparezca reflejado automáticamente.

¡FINALIZADO!



→ Tras seguir todos estos pasos, ya tendremos un programa en Java completamente funcional para gestionar una tabla de una base de datos.

GITHUB



Borja García Barrera

JayBGB · he/him

Estudiante de Desarrollo de Aplicaciones
Multiplataforma.

→ <https://github.com/JayBGB/ProgDB>