PROYECTO

ONG

PROGRAMACIÓN

Mª del Carmen Jul Barreiro

Programación DAWA 1º

Alex Pernas. Dawa48

Índice

Descripción de la aplicación

Intentaremos realizar una aplicación de una organización no gubernamental, cuyo fin es poner en contacto a través de centros de formación y/o administrativos, diferentes proyectos de ayuda, bien sea energéticos (crear aldeas sostenibles, generadores fotovoltaicos, pequeños aerogeneradores), humanitarios (reconstruir aldeas, ayudar en crisis migratorias), o urbanistas(charlas con personas maltratadas, drogodependientes, talleres formativos, clases con jubilados o niños de familias sin recursos)…con los receptores de dichos proyectos.

Especificamos que:

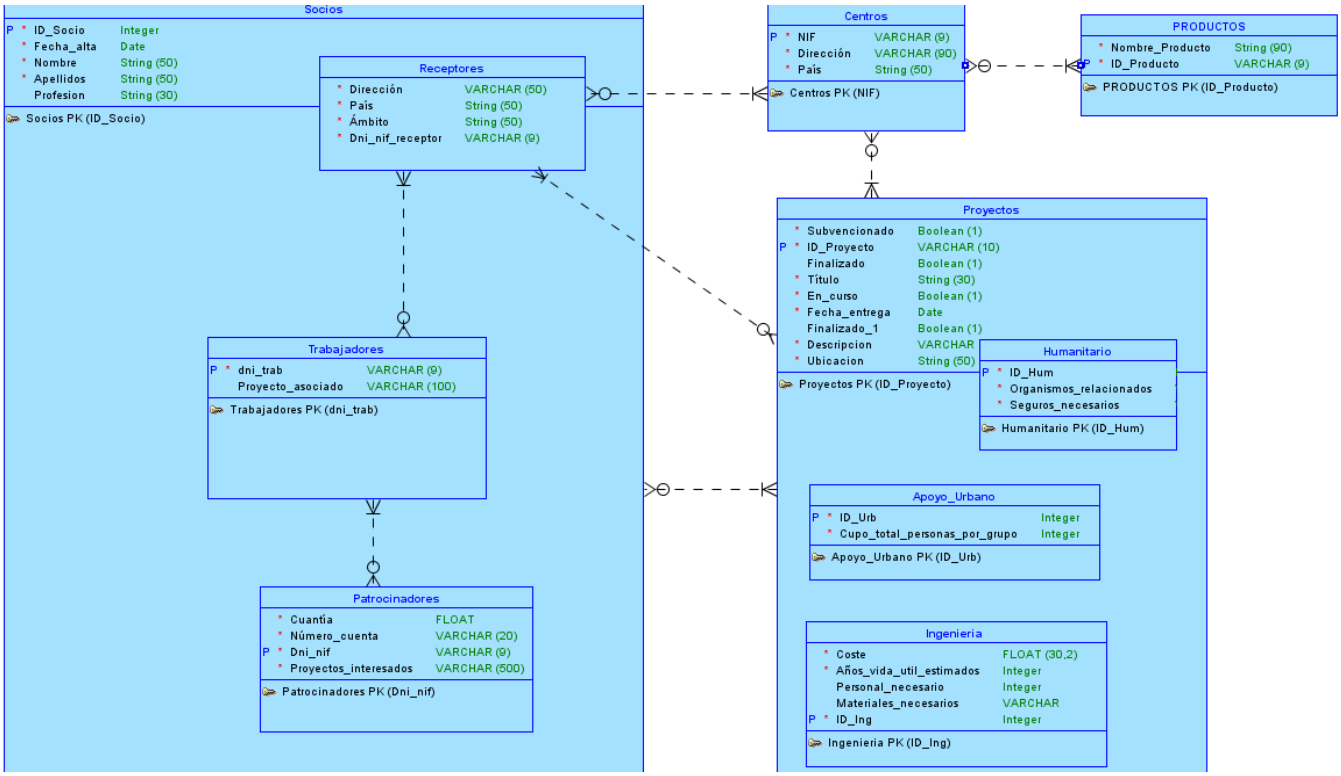
- Los socios deben de ser: trabajadores, patrocinadores o receptores, de ningún otro tipo; se puede dar el caso de un trabajador y receptor, o trabajador y patrocinador, pero nunca patrocinador y receptor, por lo que socios tendrá una relación solapada total con sus tres subentidades, (solapada indica que un ejemplar de la superentidad puede relacionarse con más de una subentidad; y total, que todos los ejemplares de la superentidad se relacionan con alguna subentidad).
- Los socios colaboran en los proyectos de diversas formas (esfuerzo y tiempo, con dinero, o siendo el encargado de recibirlo).
- Se reparte información de los proyectos a través de los distintos centros.
- Los proyectos pueden ser de los tres tipos mencionados anteriormente, pero cabe la posibilidad de que exista alguno en un futuro, que difiera de esa clasificación, por lo que definiremos esa relación, como exclusiva parcial, (exclusiva porque indica que un ejemplar de la superentidad sólo puede relacionarse con una subentidad; y parcial porque hay ejemplares de la superentidad que no se relacionan con ninguna subentidad).
- Los centros asociados pueden vender productos como tazas, camisetas o cupones, para ayudar a sufragar parte de los gastos. Lo haremos sin ánimo de lucro, y ese dinero será destinado a proyectos y mantenimiento de centros de trabajo.

Señalar que pusimos una cardinalidad (N,M), entre receptores y proyecto, en vez de (1,N), ya que podría darse el caso de que una misma persona(receptor), esté dentro de dos programas de ayuda (proyectos) a la vez, como por ejemplo en repoblar una aldea, y en grupo de apoyo maltrato/drogadicción.

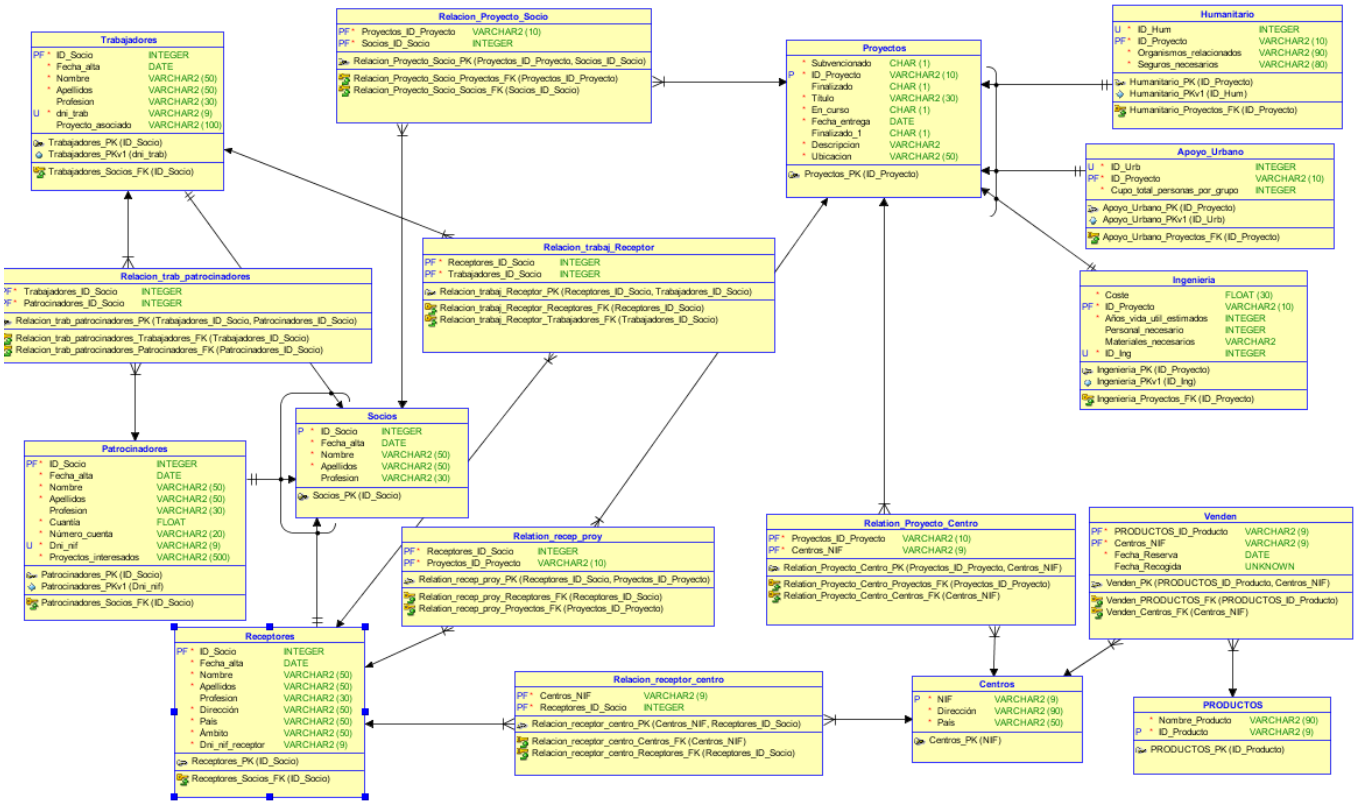
La relación venden, entre centros y productos, se hará con una cardinalidad (N,M), y le pondremos dos atributos a la relación, de tipo fecha, uno obligatorio (Fecha_reserva), y otro optativo (Fecha_recogida), ya que la segunda se utilizará sólo para compras por internet o teléfono. Esta cardinalidad dará origen a la relación venden, formada por las claves primarias de centros y productos, que propagan sus claves primarias, convirtiéndose en FK de dicha relación

Modelo relacional

Modelo lógico:

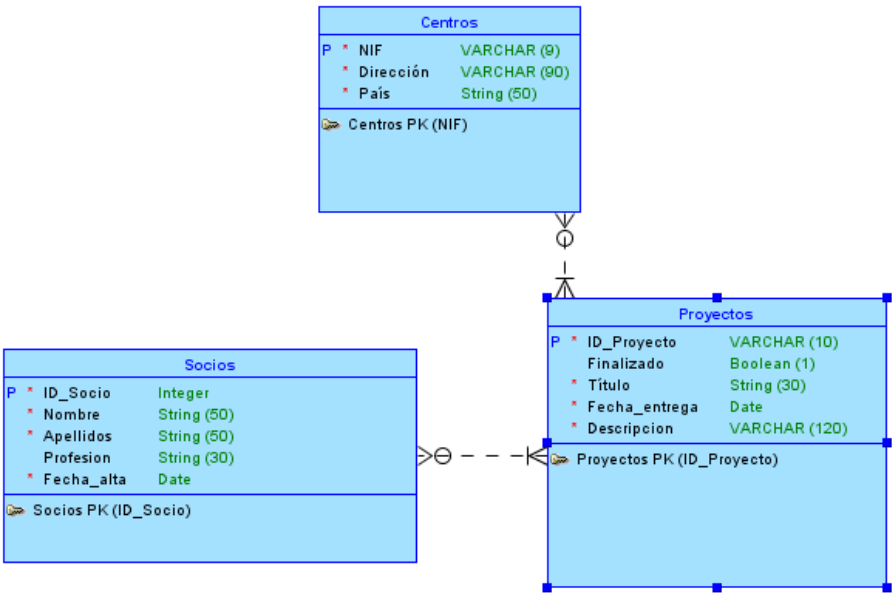


Modelo relacional:

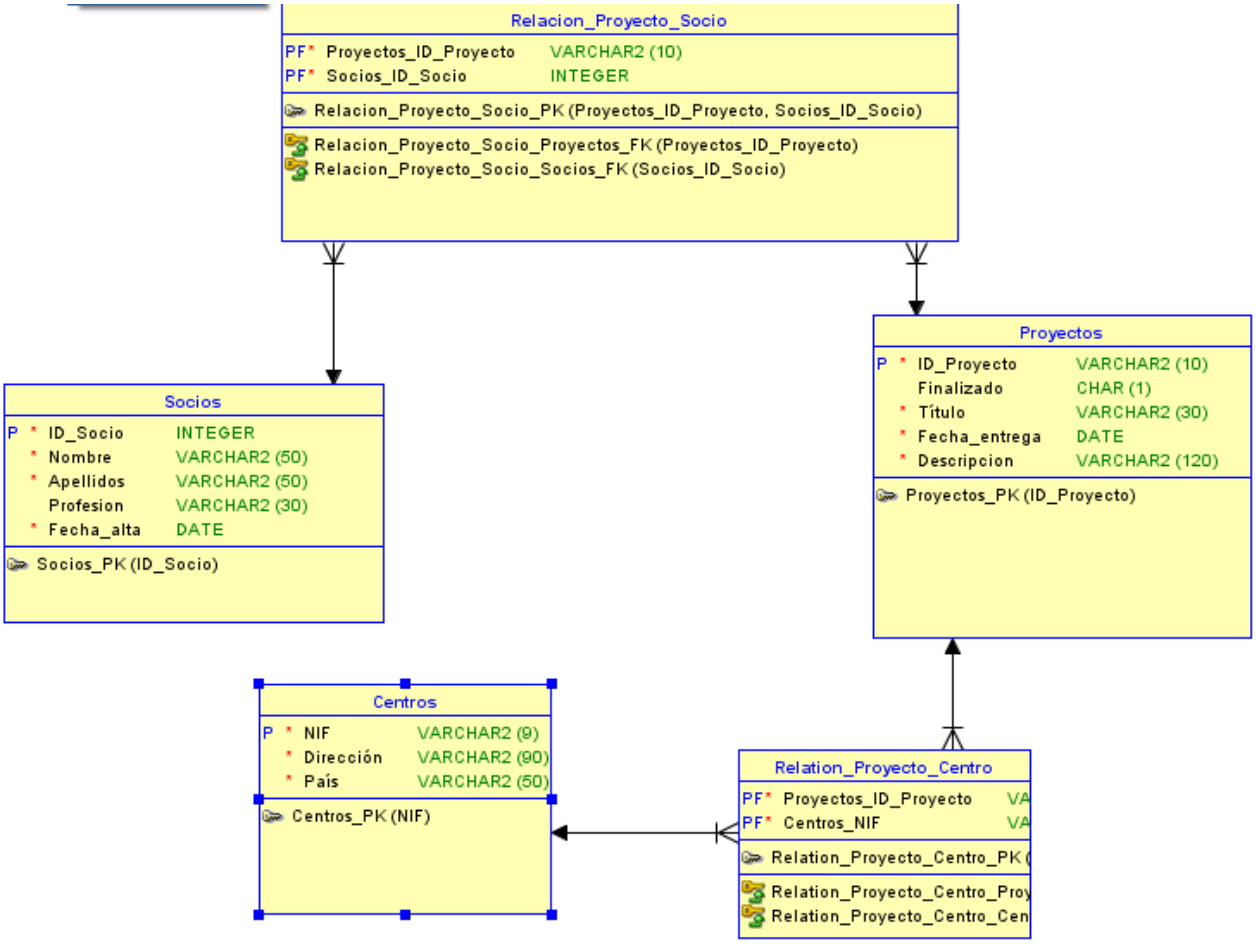


Escogeremos las tablas Socios, Centros y Proyectos, creando para ellas mediante java, una página de cada elemento CRUD para poder crear, leer, actualizar y eliminar los registros de nuestra base de datos.

Quedando, por lo tanto, para desarrollar este proyecto, un modelo lógico, tal que:



Convirtiéndolo al modelo relacional, obtenemos:



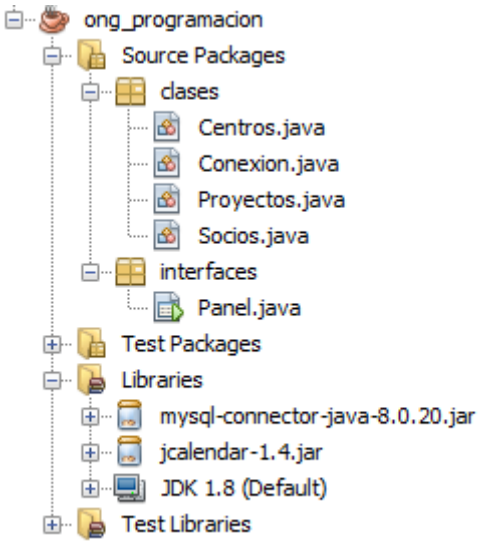
Manual de usuario

En este apartado explicaremos y describiremos el funcionamiento de las pantallas principales de nuestro proyecto.

Antes de nada, para realizar un CRUD, el requisito principal y obligatorio es tener instalado MySQL, además, de una herramienta gráfica que te permita administrar MySQL. Para este ejemplo utilizaremos MySQL Workbench.

Como bien dijimos antes, la finalidad del CRUD, es poder crear, listar, modificar y eliminar los registros de nuestra base de datos.

Vamos con nuestro proyecto, el cual consta de dos paquetes, clases e interfaces.



Dentro de clases, están los atributos y métodos básicos (getters, setters y constructores) de cada uno de los elementos de las tablas de nuestra base de datos

Dentro de interfaces, encontramos un JPanelForm, necesario para poder interactuar con la base de datos y hacer efectivo el CRUD, y poder interactuar con el usuario.

Pasamos a desglosar y explicar en profundidad cada elemento de ambos paquetes

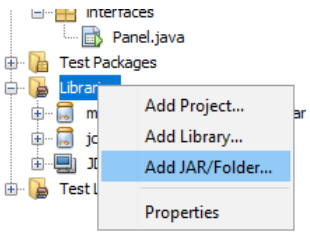
Paquete clases

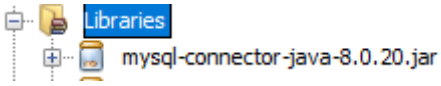
Compuesto por las siguientes java.classes:

- Conexión
- Centros
- Proyectos
- Socios

Conexión

Primero, debemos añadir el driver correspondiente en las librerías de nuestro proyecto, hemos de utilizar el .jar que está en C:\Program Files (x86)\MySQL\Connector J 8.0 y que es mysql-connector-java-8.0.20.jar .





Para probar su correcta conexión, necesitaremos el paquete java.sql que contiene todas las clases necesarias para el manejo de la BD (controladores, excepciones propias, interfaz con la aplicación). Para ello se utiliza Class.forName pasándole el driver que queremos que se cargue.

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

Después hemos de crear la conexión utilizando el método getConnection de la clase DriverManager.

La conexión devuelta por el método DriverManager.getConnection es una conexión abierta que se puede utilizar para crear sentencias JDBC que pasen nuestras sentencias SQL al controlador de la base de datos.

La conexión a la BD se hace con el método *getConnection()*

Establecemos una conexión estática, y utilizaremos el siguiente comando, para que no nos lance ningún error de zona horaria en la conexión:

jdbc:mysql://localhost/db?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC

Es necesario utilizar un try-catch, para evitar que nos lance un error. También hemos de tener en cuenta la *ClassNotFoundException* porque nos ocurre lo mismo con la carga del driver.

Escribiríamos el mensaje de error que queremos que nos salte, en caso que suceda algún fallo, y solo restaría cerrar la conexión, con otro try-catch.

La clase Conexión, quedaría de la siguiente forma:

```java
package clases;

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


public final class Conexion {
    private static Connection conexion;
    private static final String MIC="jdbc:mysql://localhost/ong_programacion?useUnicode="
            + "true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC";

    public static Connection getConexion(){
       try{
        Class.forName("com.mysql.cj.jdbc.Driver");
      // conexion =DriverManager.getConnection("jdbc:mysql://localhost/ong","alex", "a");
       //conexion normal la anterior, si no queremos tener problemas con la time_zone:
        conexion =DriverManager.getConnection(MIC,"alex", "a");
       System.out.println("bien la conexion"); //por si nos falla al principio
        }
        catch (ClassNotFoundException | SQLException e)
        {
            System.out.println("error al conectar a la base de datos "); //por si nos falla al ppio
        }
          return conexion;
      }
    public static void cerrar_conexion(){
            try{
                conexion.close();
                }
            catch(SQLException e){
            }
    }


}
```

Centros

Aquí simplemente creamos los objetos y sus atributos, junto a los métodos constructor, getters y setters

```java
package clases;

public class Centros {

    private String nif;
    private String dirección;
    private String país;
    private String id_proyecto;

    public Centros() {
    }

    public Centros(String nif, String dirección, String país, String id_proyecto) {
        this.nif = nif;
        this.dirección = dirección;
        this.país = país;
        this.id_proyecto = id_proyecto;
    }

    public String getNif() {
        return nif;
    }
    public void setNif(String nif) {
        this.nif = nif;
    }

    public String getDirección() {
        return dirección;
    }
    public void setDirección(String dirección) {
        this.dirección = dirección;
    }

    public String getPaís() {
        return país;
    }
    public void setPaís(String país) {
        this.país = país;
    }

    public String getId_proyecto() {
        return id_proyecto;
    }
    public void setId_proyecto(String id_proyecto) {
        this.id_proyecto = id_proyecto;
    }
}
```

Proyectos

Al igual que el anterior, creamos los objetos y sus atributos, junto a los métodos constructor, getters y setters

```
package clases;

public class Proyectos {
    private String ID_Proyecto;
    private String Finalizado;
    private String Título;
    private String Fecha_entrega;
    private String Descripción;

    public Proyectos() {
    }
    public Proyectos(String ID_Proyecto, String Finalizado, String Título, String Fecha_entrega, String Descripción) {
        this.ID_Proyecto = ID_Proyecto;
        this.Finalizado = Finalizado;
        this.Título = Título;
        this.Fecha_entrega = Fecha_entrega;
        this.Descripción = Descripción;
    }
    public String getID_Proyecto() {
        return ID_Proyecto;
    }
    public void setID_Proyecto(String ID_Proyecto) {
        this.ID_Proyecto = ID_Proyecto;
    }
    public String getFinalizado() {
        return Finalizado;
    }
    public void setFinalizado(String Finalizado) {
        this.Finalizado = Finalizado;
    }
    public String getTítulo() {
        return Título;
    }
    public void setTitulo(String Título) {
        this.Título = Título;
    }
    public String getFecha_entrega() {
        return Fecha_entrega;
    }
    public void setFecha_entrega(String Fecha_entrega) {
        this.Fecha_entrega = Fecha_entrega;
    }
    public String getDescripción() {
        return Descripción;
    }
    public void setDescripción(String Descripción) {
        this.Descripción = Descripción;
    }
}
```

Socios

Como en los dos anteriores, creamos los objetos y sus atributos, junto a los métodos constructor, getters y setters

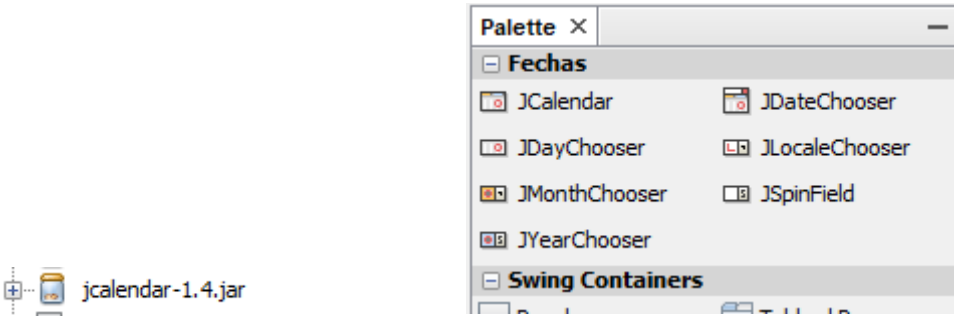
```java
package clases;

import java.time.LocalDate;

public class Socios {

    private String ID_Socio;
    private String Nombre;
    private String Apellidos;
    private String Profesión;
    private String Fecha_alta;
    private String nif_centros;

    public Socios() {
    }

    public Socios(String ID_Socio, String Nombre,String Apellidos, String Profesión, String Fecha_alta,  String nif_centros) {
        this.ID_Socio = ID_Socio;
        this.Nombre = Nombre;
        this.Apellidos = Apellidos;
        this.Profesión = Profesión;
        this.Fecha_alta = Fecha_alta;
        this.nif_centros= nif_centros;
    }

    public String getID_Socio() {
        return ID_Socio;
    }

    public void setID_Socio(String ID_Socio) {
        this.ID_Socio = ID_Socio;
    }

    public String getNombre() {
        return Nombre;
    }

    public void setNombre(String Nombre) {
        this.Nombre = Nombre;
    }
```

```
    public void setNombre(String Nombre) {
        this.Nombre = Nombre;
    }

    public String getApellidos() {
        return Apellidos;
    }

    public void setApellidos(String Apellidos) {
        this.Apellidos = Apellidos;
    }

    public String getProfesión() {
        return Profesión;
    }

    public void setProfesión(String Profesión) {
        this.Profesión = Profesión;
    }

    public String getFecha_alta() {
        return Fecha_alta;
    }

    public void setFecha_alta(String Fecha_alta) {
        this.Fecha_alta = Fecha_alta;
    }

    public String getNif_centros() {
        return nif_centros;
    }

    public void setNif_centros(String nif_centros) {
        this.nif_centros = nif_centros;
    }


}
```

Paquete interface

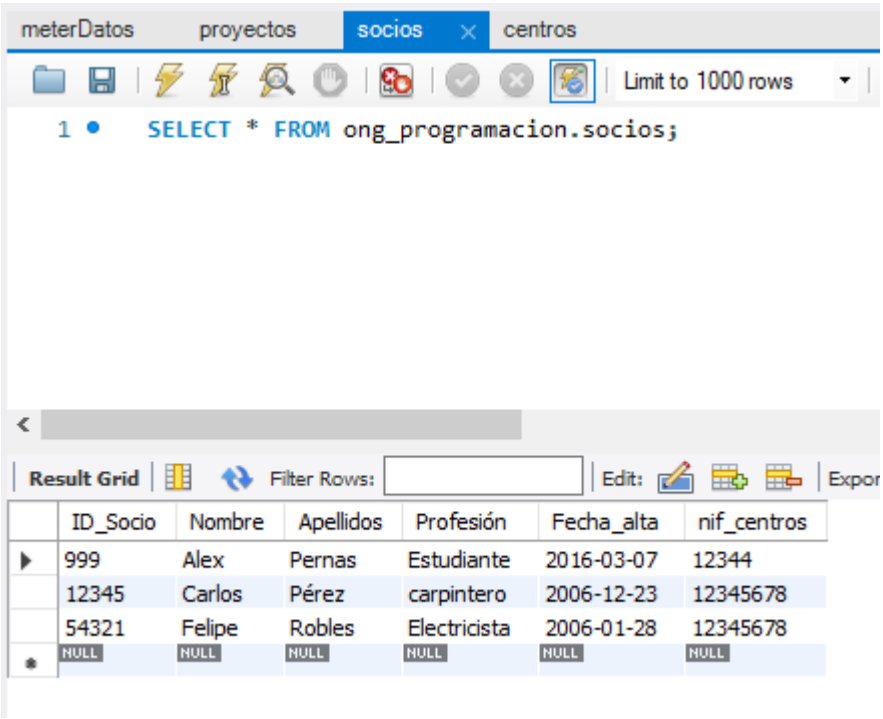
JPanel se trata de una parte del paquete Java Swing, es un contenedor que puede almacenar un grupo de componentes. La tarea principal de JPanel es organizar los componentes, se pueden establecer varios diseños en JPanel que proporcionan una mejor organización de los mismos.

Aquí tuvimos que añadir en la "Paleta de Componentes", otro jar, para poder insertar un calendario para las fechas



Diseñamos nuestro formulario creando 3 tablas, cada una para que se sincronice con nuestra base de datos, es decir, una pestaña con socios, otra con proyectos y la última con centros.

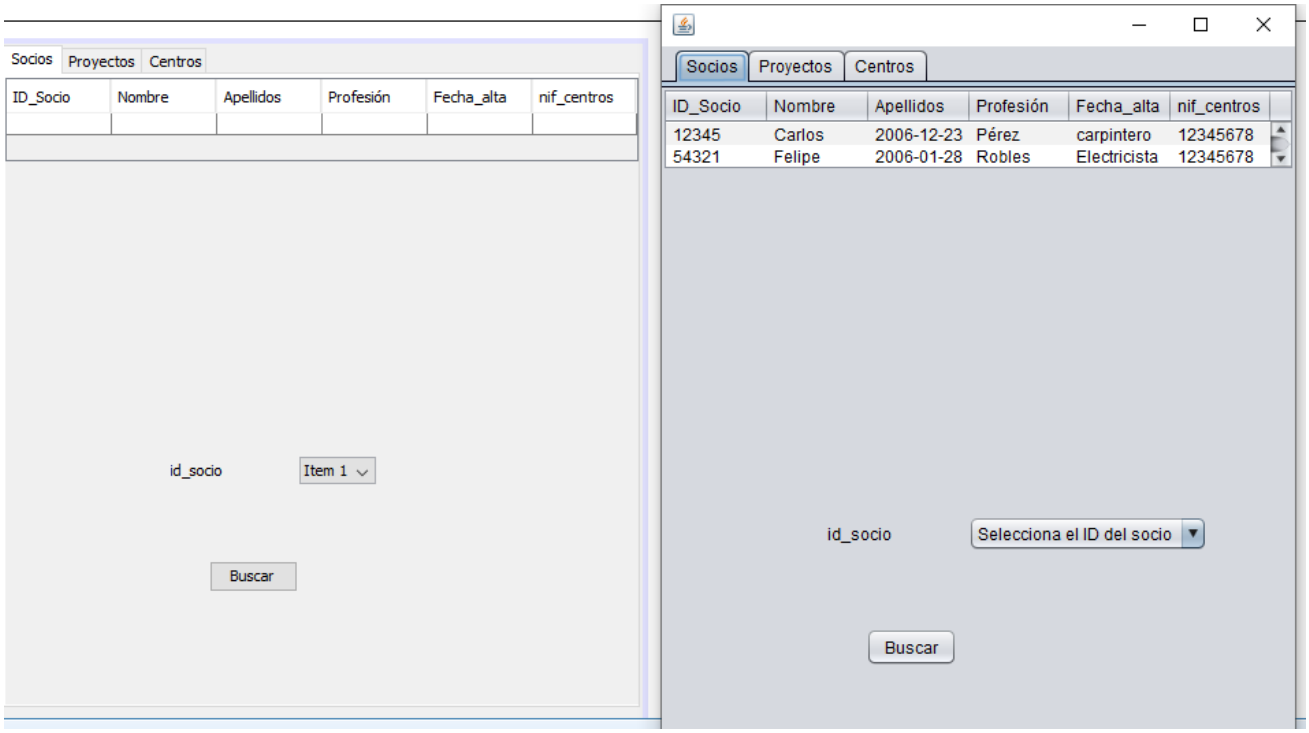
Tenemos en nuestra pequeña base de datos, los siguientes socios:



ID_Socio	Nombre	Apellidos	Profesión	Fecha_alta	nif_centros
999	Alex	Pernas	Estudiante	2016-03-07	12344
12345	Carlos	Pérez	carpintero	2006-12-23	12345678
54321	Felipe	Robles	Electricista	2006-01-28	12345678
NULL	NULL	NULL	NULL	NULL	NULL

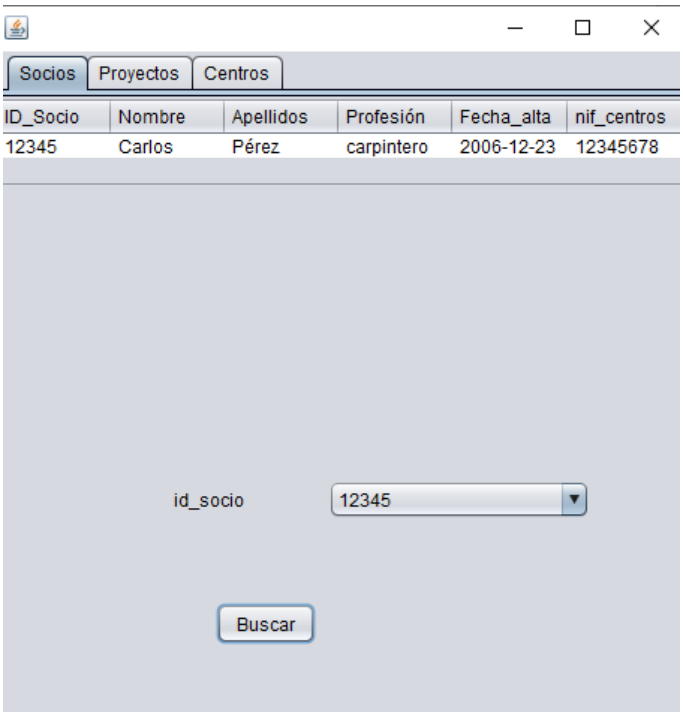
Pestaña socios

El diseño de nuestro J Panel Form, queda de la siguiente manera, a la izquierda el panel sin ejecutar (el diseño), y a la derecha, cuando ejecutamos:



Comentar aquí, que según lo lanzamos, nos lista todos los usuarios que disponemos en nuestra tabla socios.

En esta pestaña incluimos un scroll para visualizar todas las tablas, y la parte del CRUD de listar nuestra bd, el campo mediante el que identificamos un socio es mediante su PK, es decir, su id_socio, ponemos, por ejemplo, el ID_socio 12345



Se comprueba cómo se lista ese usuario en concreto

El código necesario para hacerlo funcionar:

```
package interfaces;

import clases.*;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
//todos esos pueden ser java.sql.*
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Calendar;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;

public class Panel extends javax.swing.JFrame {

    Connection connection=null;

    public Panel() {
        initComponents();
        setLocationRelativeTo(null);

        connection=Conexion.getConexion();
        if (connection!=null)
        {
        mostrarSociosTabla();
        mostrarProyectosTabla();
        mostrarCentrosTabla();
        }
        else
            JOptionPane.showMessageDialog(null, "Error al conectar", "Mensaje de error", JOptionPane.ERROR_MESSAGE);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code
```

Primero importamos todos los paquetes y colecciones necesarios para su puesta en marcha, posteriormente creamos la clase pública panel, que gracias a la cláusula *extends*, indica la clase base de la cuál queremos heredar. Al heredar de una clase base heredaremos tanto los atributos como los métodos. Javax.swing se trata de una versión extendida de java.awt.Frameeso, la cual agrega soporte para la arquitectura de componentes JFC / Swing. Es un entorno de pantallas múltiples y contiene todos los componentes que no son de menú, mostrados por JFrame.

Hacemos uso de public, una de las funciones principales en Java para ejecutar una clase, para que cualquier clase en cualquier paquete, puede acceder al método.

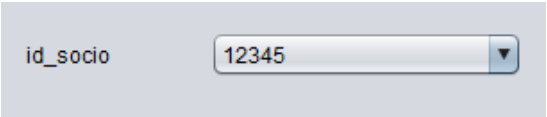
Inicializamos los componentes, conectamos e invocamos los tres métodos mostrar, en caso de que la conexión sea correcta, de no serlo, nos debería aparecer el mensaje de error.

Para que nos funcione la pestaña socios, vista anteriormente, mostraremos a continuación, el código utilizado:

```
public ArrayList<Socios> getListaSocios() {
    ArrayList<Socios> listaSocios = new ArrayList<>();

    String query = "SELECT * FROM socios";

    try {
        Statement st = connection.createStatement();
        ResultSet rs = st.executeQuery(query);
        jComboBoxID_socio.removeAllItems();
        jComboBoxID_socio.addItem("Selecciona el ID del socio");
        Socios socio;
        while (rs.next()) {
            socio = new Socios (rs.getString("ID_Socio"), rs.getString("Nombre"), rs.getString("fecha_alta"),
                    rs.getString("Apellidos"), rs.getString("Profesión"), rs.getString("nif_centros"));
            jComboBoxID_socio.addItem(rs.getString("ID_Socio"));
            listaSocios.add(socio);
               }
        st.close();
        rs.close();
    } catch (SQLException e) {
        System.out.println(e);
    }

    return listaSocios;
}

public void mostrarSociosTabla() {
    ArrayList<Socios> listaSocios = getListaSocios();
    DefaultTableModel modelo = (DefaultTableModel) jTableSocios.getModel();

    modelo.setRowCount(0);
    Object col[] = new Object[6];

    for (int i = 0; i < listaSocios.size(); i++) {
        col[0] = listaSocios.get(i).getID_Socio();
        col[1] = listaSocios.get(i).getNombre();
        col[2] = listaSocios.get(i).getApellidos();
        col[3] = listaSocios.get(i).getProfesión();
        col[4] = listaSocios.get(i).getFecha_alta();
        col[5] = listaSocios.get(i).getNif_centros();

        modelo.addRow(col);
    }
}
```

En el primer método, recogemos en el array *listaSocios*, los datos del query mediante un select, es decir, listamos de nuestra base de datos en la tabla socios, y mediante un try-catch, para evitar errores, recogemos esos valores en dicho array.

Con remove all ítems y additem, definimos el comportamiento del combobox creado en la pestaña socios



Por último cerraremos la conexión, y nos devolverá los resultados de nuestra bd, al array listaSocios.

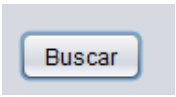
Utilizaremos el void, para poder invocar una función o método de la que no nos interese que devuelva un valor.

Con la función *mostrarSociosTabla*, definimos cómo nos debe mostrar los resultados conseguidos en el array antes mencionado, es decir, mostrarlo en la tabla que creamos, y le indicamos a qué columna pertenece cada elemento del array mediante los número de *col[nº]*, 0 para la primera, 1 para la segunda, y así hasta el final

ID_Socio	Nombre	Apellidos	Profesión	Fecha_alta	nif_centros
12345	Carlos	Pérez	carpintero	2006-12-23	12345678

ID_Socio vendría a ser la columna 0.

A continuación mostramos el código de funionamiento del botón buscar



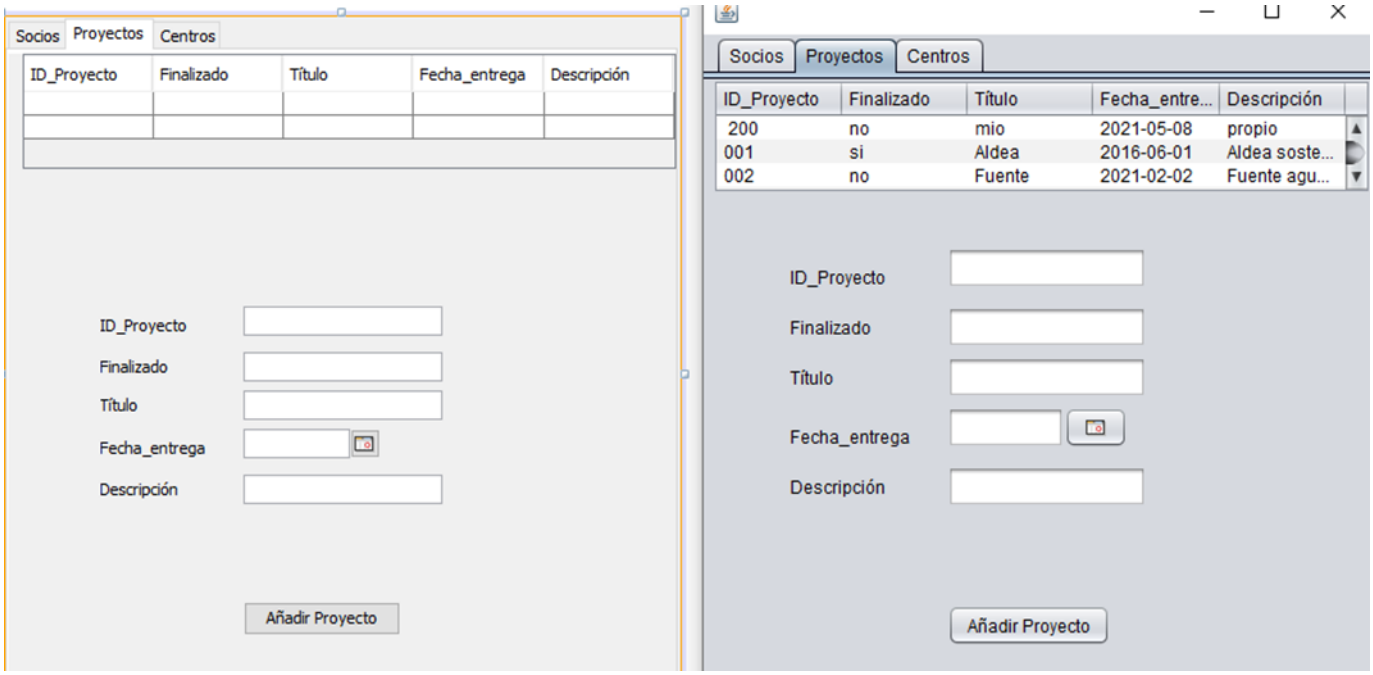

```
private void Buscar1ActionPerformed(java.awt.event.ActionEvent evt) {

    try{
        //De esta forma si quisiésemos ponerle el nombre a las columnas:
        //  String buscarPrepared="select dni as 'DNI', nombre as 'NOMBRE', apellidos as
        String buscarPrepared="select * from socios where ID_Socio LIKE ?";
        PreparedStatement psB = connection.prepareStatement(buscarPrepared);

        psB.setString (1, (String) jComboBoxID_socio.getSelectedItem()+"%");

        //Aquí guardamos el resultado de la consulta
        ResultSet resultadoB=psB.executeQuery();//Crea tabla virtual, el resultset

        ResultSetMetaData resulMetaData = resultadoB.getMetaData(); //encabezado columna
        int col=resulMetaData.getColumnCount();//Contamos nº de columnas
        DefaultTableModel tablaSociosB = new DefaultTableModel();

        //De esta forma nos pone el nombre de las columnas igual al de la tabla:
        for (int i=1;i<=col;i++){
            tablaSociosB.addColumn(resulMetaData.getColumnLabel(i));
        }
        while(resultadoB.next()){
            String fila []=new String [col];
            for(int j=0;j<col;j++){
                fila[j]=resultadoB.getString(j+1);
            }
            tablaSociosB.addRow(fila);
        }

        jTableSocios.setModel(tablaSociosB);
        psB.close();
        resultadoB.close();
    }catch (SQLException e){

        System.out.println("Fallo al conectar");
    }
    //se podría usar algo como:
    // DaoSocios.buscar_socios(connection,jTextFieldl.getText());
}
```

Definimos mediante un select, que nos liste todas los elementos de nuestra tabla socios de la bd, utilizando para ello el parámetro ID_Socio, recogido mediante un combobox, el resultest crea la tabla virtual.

En java, las clases DataBaseMetaData y ResultSetMetaData permiten, respectivamente, analizar la estructura de una base de datos (qué tablas tiene, que columnas cada tabla, de qué tipos, etc) o de un ResultSet de una consulta, para averiguar cuántas columnas tiene dicho ResulSet, de qué columnas de base de datos proceden, de qué tipo son, etc.

Mencionar por último, que en esta pestaña enlazamos dos tablas de nuestra base de datos, es decir, la tabla completa de socios, junto a la PK de centros, nif_centros, para relacionar ambas.

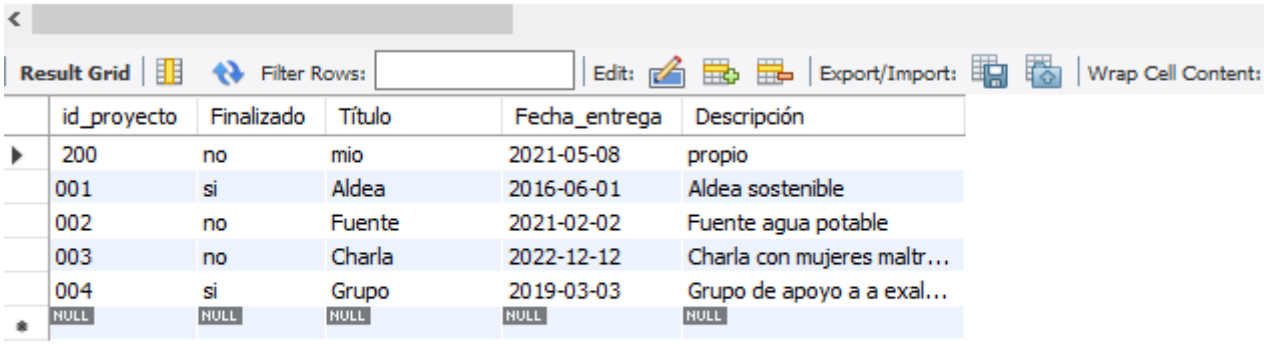
Pestaña proyectos



Al igual que comentamos en el apartado anterior, según lo lanzamos, nos lista todos los usuarios de los que disponemos en nuestra tabla proyectos.

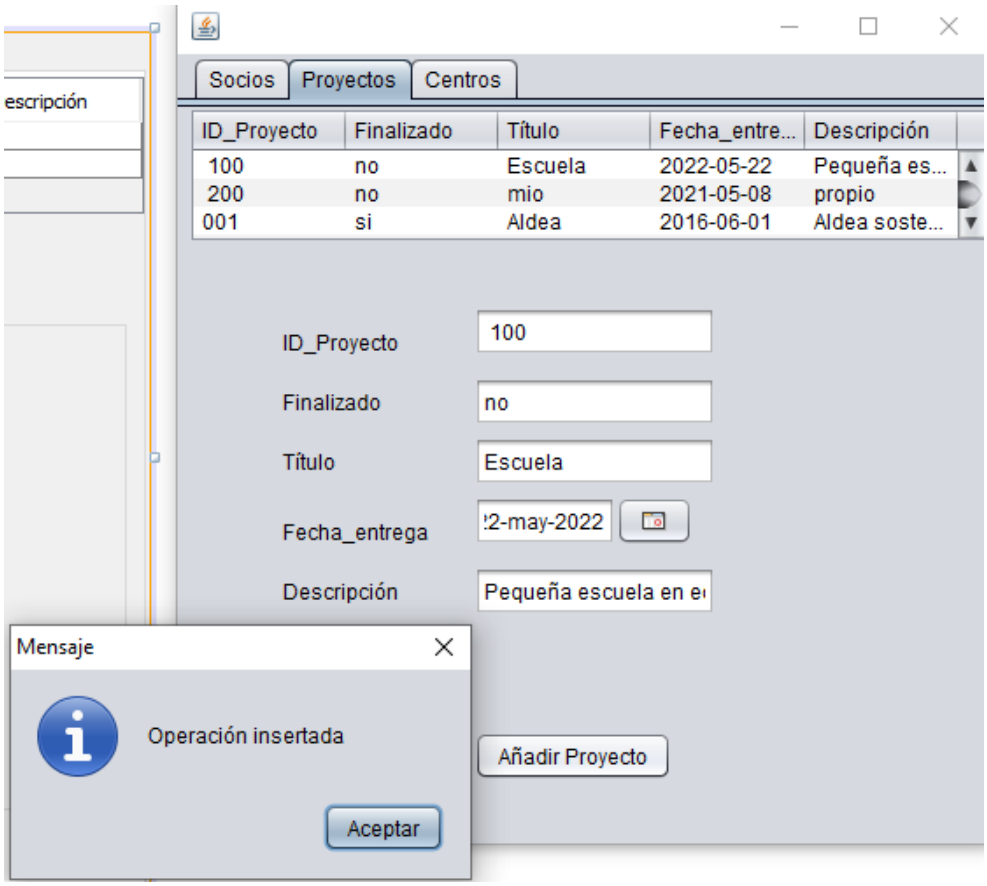
En esta pestaña incluimos un scroll para visualizar todas las tablas, y la parte del CRUD de crear en nuestra bd.

Nuestra base de datos:



id_proyecto	Finalizado	Título	Fecha_entrega	Descripción
200	no	mio	2021-05-08	propio
001	si	Aldea	2016-06-01	Aldea sostenible
002	no	Fuente	2021-02-02	Fuente agua potable
003	no	Charla	2022-12-12	Charla con mujeres maltr…
004	si	Grupo	2019-03-03	Grupo de apoyo a a exal…
NULL	NULL	NULL	NULL	NULL

Ejemplo para comprobar su funcionamiento:



Su código:

```
public ArrayList<Proyectos> getListaProyectos() {
    ArrayList<Proyectos> listaProyectos = new ArrayList<>();

    String query = "SELECT * FROM proyectos";
    Statement st;
    ResultSet rs;

    try {
        st = connection.createStatement();
        rs = st.executeQuery(query);
        Proyectos proyectos;

        while (rs.next()) {
                proyectos = new Proyectos ( rs.getString("ID_Proyecto"),
                        rs.getString("Finalizado"), rs.getString("Título"),rs.getString ( "Fecha_entrega"),
                        rs.getString("Descripción"));
            listaProyectos.add(proyectos);
        }
        st.close();
        rs.close();

    } catch (SQLException e) {
        System.out.println(e);
    }
    return listaProyectos;
}


public void mostrarProyectosTabla() {
    ArrayList<Proyectos> listaProyectos = getListaProyectos();
    DefaultTableModel modelo = (DefaultTableModel) jTableProyectos.getModel();

    modelo.setRowCount(0); //ponemos a cero las filas
    Object col[] = new Object[5];

    for (int i = 0; i < listaProyectos.size(); i++) {

        col[0] = listaProyectos.get(i).getID_Proyecto();
        col[1] = listaProyectos.get(i).getFinalizado();
        col[2] = listaProyectos.get(i).getTítulo();
        col[3] = listaProyectos.get(i).getFecha_entrega();
        col[4] = listaProyectos.get(i).getDescripción();

        modelo.addRow(col);
    }
}
```

Obviaremos repetir lo explicado en los apartados anteriores, y pasaremos al funcionamiento del botón

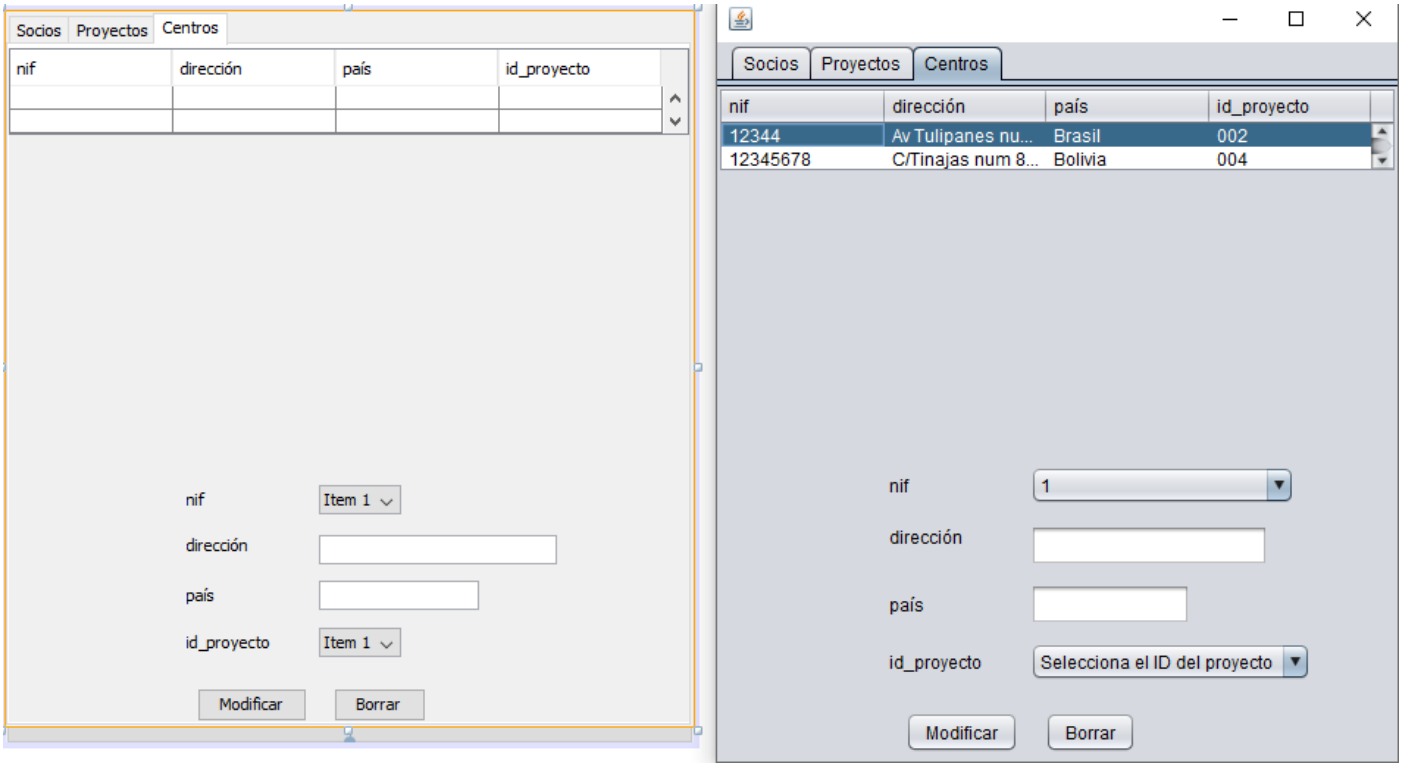
```
private void jButtonInsertarActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        String consulta = "INSERT INTO proyectos (ID_Proyecto,Finalizado, Título,Fecha_entrega, Descripción ) VALUES (?,?,?,?,?)";
        PreparedStatement ps = connection.prepareStatement(consulta);

        if (comprobarCajasProyectos()) {
            ps.setString(1, jTextFieldID_proyecto.getText());
            ps.setString(2, jTextFieldFinalizado.getText());
            ps.setString(3, jTextFieldTitulo.getText());
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
            String addDate = dateFormat.format(jDateChooserFecha_entrega.getDate());
           //ps.setString(3, addDate);
            //una forma de guardar la fecha tomándola como String o como tipo fecha,
            //en el segundo caso hay que utilizar java.sql.Date para cambiar de tipo Date a sql.date
            ps.setDate(4, java.sql.Date.valueOf(addDate));
            ps.setString(5, jTextFieldDescripcion.getText());
            ps.executeUpdate();

            mostrarProyectosTabla();
            JOptionPane.showMessageDialog(null, "Operación insertada");
            limpiarCajasProyectos();
        }
        ps.close();
    } catch (Exception e) {

        JOptionPane.showMessageDialog(null, "Error al insertar", "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

Mediante la instrucción insert into, agregamos los registros que nosotros queramos, a la tabla proyectos, ésto se conoce como una consulta de datos anexados.

El funcionamiento es el mismo que buscar, explicado anteriormente, salvo que ahora agregamos en vez de listar.

Aquí tenemos algo nuevo, que es el *jDateChooser*, creado gracias al .jar calendar, que nos permite, en vez de escribir la fecha, insertarla mediante el lanzamiento de un pequeño calendario

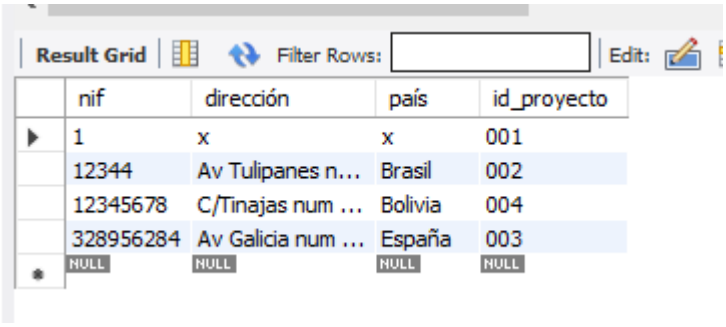
Pestaña centros



Como en los apartados anteriores, según lo lanzamos, nos lista todos los usuarios de los que disponemos en nuestra tabla proyectos.

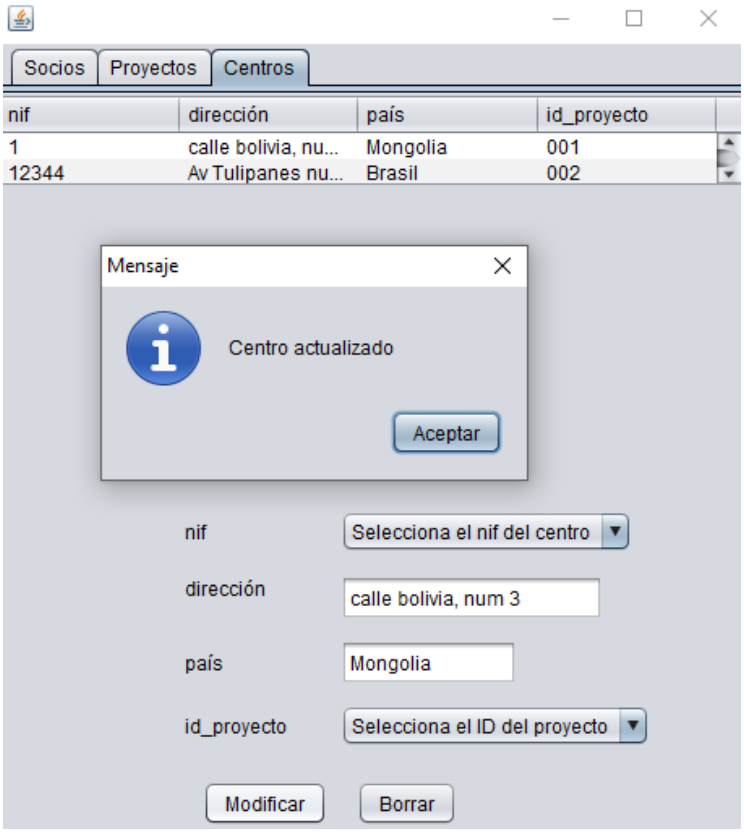
Se trata de la parte del CRUD, que tiene la finalidad de modificar y eliminar registros de nuestra bd.

Nuestra base de datos:

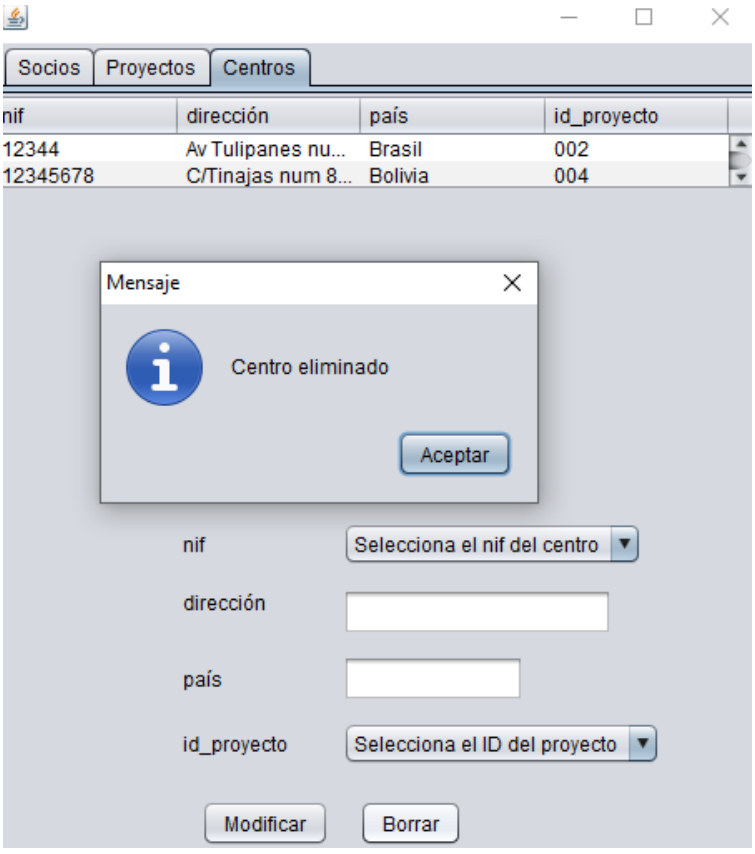


Señalar que en esta pestaña, también enlazamos diferentes tablas, ya que el parámetro id_proyecto, es la clave primaria de la tabla proyectos

Ejemplo para comprobar su funcionamiento, cambiaremos el primero por:



Y ahora pasaremos a eliminarlo, simplemente poniendo el nif



El código que permite su funcionamiento:

```
public void mostrarCentrosTabla() {
    ArrayList<Centros> listaCentros = getListaCentros();
    DefaultTableModel modelo = (DefaultTableModel) jTableCentros.getModel();

    modelo.setRowCount(0);
    Object col[] = new Object[4];

    for (int i = 0; i < listaCentros.size(); i++) {
        col[0] = listaCentros.get(i).getNif();
        col[1] = listaCentros.get(i).getDirección();
        col[2] = listaCentros.get(i).getPaís();
        col[3] = listaCentros.get(i).getId_proyecto();

        modelo.addRow(col);
    }
}


public ArrayList<Centros> getListaCentros() {
    ArrayList<Centros> listaCentros = new ArrayList<>();

    String query = "SELECT * FROM centros";
    Statement st;
    ResultSet rs;

    try {
        st = connection.createStatement();
        rs = st.executeQuery(query);
        jComboBoxNIF.removeAllItems();
        jComboBoxNIF.addItem("Selecciona el nif del centro");
        jComboBoxID_proyectoCentro.removeAllItems();
        jComboBoxID_proyectoCentro.addItem("Selecciona el ID del proyecto");
        Centros centros;
        while (rs.next()) { //String nif, String dirección, String país, String id_proyecto
            centros = new Centros(rs.getString("nif"), rs.getString("dirección"), rs.getString("país"), rs.getString("id_proyecto"));
            jComboBoxNIF.addItem(rs.getString("nif"));
            jComboBoxID_proyectoCentro.addItem(rs.getString("id_proyecto"));
            //aprovechamos para ir rellenando el combo
            listaCentros.add(centros);
        }
        st.close();
        rs.close();
    } catch (SQLException e) {
        System.out.println(e);
    }
    return listaCentros;
}
```

En esta parte, omitiremos la explicación, ya que es idéntica a la del primer apartado

- Botón modificar:

```
private void jButtonModificarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    if (comprobarCajasCentros() && !jComboBoxNIF.getSelectedItem().equals("")) {

        try {
            String consulta = "UPDATE centros SET nif=?, dirección=?, pais=?, id_proyecto=? WHERE nif=?";
            PreparedStatement ps = connection.prepareStatement(consulta);

            ps.setString(1, (String) jComboBoxNIF.getSelectedItem());
            ps.setString(2, jTextFieldDireccion.getText());
            ps.setString(3, jTextFieldPais.getText());
            ps.setString(4, (String) jComboBoxID_proyectoCentro.getSelectedItem());
            ps.setString(5, (String) jComboBoxNIF.getSelectedItem());
            ps.executeUpdate();

            mostrarCentrosTabla();
            JOptionPane.showMessageDialog(null, "Centro actualizado");
            limpiarCajasCentros();
            ps.close();
        } catch (SQLException e) {

            JOptionPane.showMessageDialog(null, "Centro no actualizado", "Mensaje de error", JOptionPane.ERROR_MESSAGE);
        }
    } else {
        JOptionPane.showMessageDialog(null, "Uno o más campos están vacíos o son incorrectos", "Mensaje de error", JOptionPane.ERROR_MESSAGE);
    }
}
```

En esta parte, hacemos uso de la sentencia update, gracias a la cual actualiza nuestra base de datos, seguida de la tabla a modificar, y de los elementos que deseamos cambiar. Gracias a la cláusula where, permite especificar los criterios que se deben cumplir, para que los registros que contengan los valores, se incluyan en los resultados de la consulta

Señalar la diferencia apreciable entre usar un textfield, y un combobox, mientras que en la primera se recoge lo escrito mediante un *getText(),* en la segunda debemos pasarlo a String, y utilizar un *getSelectedItem*().

- Botón borrar:

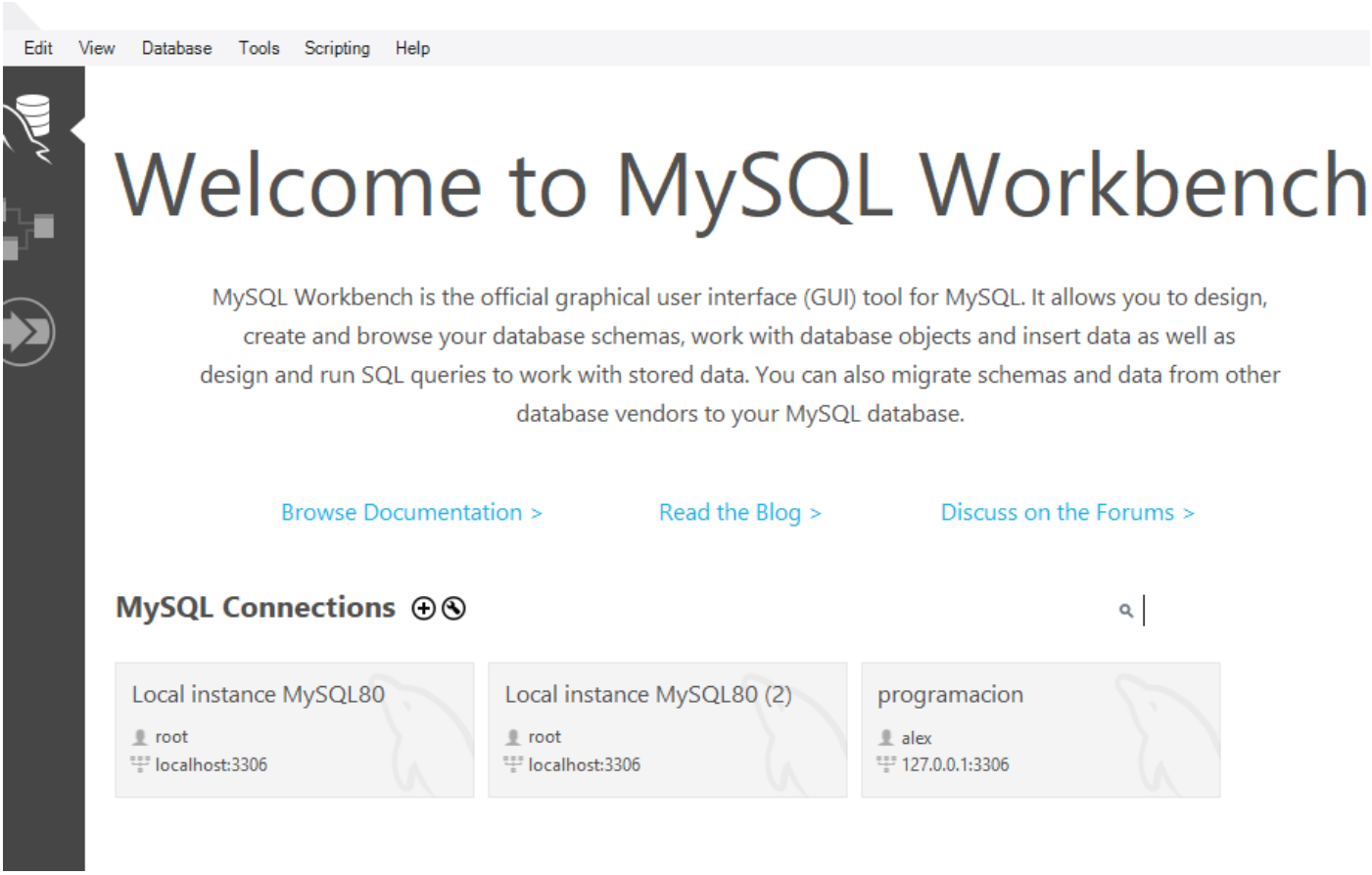
```
private void jButtonBorrarActionPerformed(java.awt.event.ActionEvent evt) {
    if (!  jComboBoxNIF.getSelectedItem().equals("")) {
        try {
            String consulta = "DELETE FROM centros WHERE nif = ?";
            PreparedStatement ps = connection.prepareStatement(consulta);
            int id = Integer.parseInt((String)jComboBoxNIF.getSelectedItem());
            ps.setInt(1, id);
            ps.executeUpdate();

            mostrarCentrosTabla();
            JOptionPane.showMessageDialog(null, "Centro eliminado");
            limpiarCajasCentros();
            ps.close();
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "Centro no eliminado", "Mensaje de error", JOptionPane.ERROR_MESSAGE);
        }
    } else {
        JOptionPane.showMessageDialog(null, "Centro no eliminado : No hay nif para borrar", "Mensaje de error", JOptionPane.ERROR_MESSAGE);
    }
}
```

Gracias a la instrucción delete, podemos eliminar un registro de nuestra base de datos, mediante la cláusula where, de la que podemos hacer uso para señalar el criterio mediante el cual se borrará, en nuestro caso, es el parámetro nif.

Manual del programador

Comentaré en este apartado los pasos que seguí para poder acceder al workbench, y mostraré el código común que no señalé en las partes anteriores.

Gracias a la tecnología JDBC (Java Database Connectivity), un API de Java para poder acceder a nuestras BD, podremos acceder a mysql. Primero creamos una nueva conexión llamada programación, con el usuario alex, y un password de nuestra elección, conectándonos a localhost mediante la ip 127.0.0.1, y mediante el puerto 3306, que es el que viene por defecto



Creamos un nuevo usuario con permisos básicos de selección, inserción, borrado y modificación, los necesarios para realizar luego nuestro CRUD. La ventaja es que dicho usuario creado de esta forma, puede acceder desde cualquier equipo, y por cuestiones de seguridad, es conveniente utilizar un usuario con privilegios limitados y no el usuario root.

El código main, señalar simplemente *eventQueue.invokeLater.* El procesamiento completo de Swing se realiza en un subproceso llamado EDT (Subproceso de envío de eventos). El camino a seguir aquí, es procesar su cálculo dentro de un hilo diferente, para que su GUI responda. *EventQueue.invokeLater,* publica un evento (su Runnable) al final de la lista de eventos de Swings y se procesa después de que se procesen todos los eventos de GUI anteriores

```
/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
     Look and feel setting code (optional)
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Panel().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton BuscarI;
```

```
private void jTableProyectosMouseClicked(java.awt.event.MouseEvent evt) {
    //al pinchar en la tabla y sobre la fecha me aparece en el DateChooser el valor actual
    Calendar calendario=Calendar.getInstance();
    SimpleDateFormat formato=new SimpleDateFormat("yyyy-MM-dd");
    try{
        String cadena=jTableProyectos.getValueAt(jTableProyectos.getSelectedRow(), jTableProyectos.getSelectedColumn()).toString();
        calendario.setTime(formato.parse(cadena));
        jDateChooserFecha_entrega.setDate(calendario.getTime());
    }
    catch(Exception e)
    {}
}

private void jScrollPane2MouseClicked(java.awt.event.MouseEvent evt) {

}

public boolean comprobarCajasProyectos() {
    if (jTextFieldID_proyecto.getText().isEmpty() || jTextFieldFinalizado.getText().isEmpty() || jTextFieldDescripcion.getText().isEmpty() ||
        jDateChooserFecha_entrega.getDate() == null || jTextFieldTitulo.getText().isEmpty()) {
        JOptionPane.showMessageDialog(null, "Uno o más campos están vacíos o son incorrectos", "Mensaje de error", JOptionPane.ERROR_MESSAGE);
        return false;

    }
    return true;
}


public boolean comprobarCajasCentros() {
    if (jTextFieldDireccion.getText().isEmpty() || jTextFieldPais.getText().isEmpty())
        return false;
    else return true;
}
```

```
public void limpiarCajasProyectos() {
        jTextFieldFinalizado.setText("");
        jTextFieldDescripcion.setText("");
        jDateChooserFecha_entrega.setCalendar(null);
        jTextFieldTitulo.setText("");
        jComboBoxID_proyectoCentro.setSelectedIndex(0);
    }

    public void limpiarCajasCentros() {
        jTextFieldDireccion.setText(null); //con null o con ""
        jTextFieldPais.setText(null);

    }


    private void jTableCentrosMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        int i = jTableCentros.getSelectedRow();
        TableModel modelo = jTableCentros.getModel();
        jTextFieldDireccion.setText(modelo.getValueAt(i, 0).toString());
        jTextFieldPais.setText(modelo.getValueAt(i, 1).toString());
    }
```

Conclusión

En realidad, no sé muy bien qué poner en este apartado, que me costó un huevo y parte de otro llegar a completar el proyecto, y pienso, que parte del poco pelo que me queda, también lo perdí, como en las películas cuando alguien sufre un estrés traumático y se le queda la pelambrera blanca…

Que hubiera apostado en mi contra que no llegaba a realizar un proyecto parecido al del ejemplo, pero ahora, echando la vista unos pocos días atrás, me siento más sabio (igual en verano pierdo ese conocimiento, no sé, pero ahora me siento un poco más)

Que sin ti y simplemente tirando de internet, ni de lejos lo hubiera sacado adelante, y a pesar de la infinidad de correos que te envié, obtuve respuesta, por lo que te estaré eternamente agradecido.

Que era la asignatura que más ganas tenía de aprender, y sin embargo, la que más me costó sin ninguna duda.

Que del proyecto que tenía en mente, al proyecto entregado, es un mundo de diferencia, y que lo que me queda por seguir aprendiendo de Java, es una galaxia..

Y poco más, que al final, como esto se está pareciendo más a una carta que a un apartado, pues me despido sin posdata, con poco que decir, y con menos dicho.

Bibliografía

https://www.geeksforgeeks.org/java-swing-jpanel-examples/

http://laurel.datsi.fi.upm.es/_media/docencia/cursos/java/2012/guis_en_java-1pp_2012_.pdf

https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=595:paquete-javautil-api-java-interfaces-y-clases-stringtokenizer-date-calendar-hashset-treeset-cu00916c&catid=58&Itemid=180

https://docs.oracle.com/javase/10/docs/api/javax/swing/JFrame.html

https://www.mundojava.net/la-herencia-en-java.html?Pg=java_inicial_4_4_6.html

http://www.chuidiang.org/java/mysql/ResultSet-DataBase-MetaData.php

https://sites.google.com/site/yuvalzisummaries/home/plang/jvm/java/gui/responsivenes/java-awt-eventqueue-invokelater-explained

PDF´s:

- CONEXIÓN ENTRE JAVA Y MYSQL
- EJERCICIO HOSPITAL
- Interfaces_Graficas_Actividade
- Ejemplo Hospital

Y sobre todo y más que nada, mucho intercambio de correos, para aprovechar la sabiduría de MªCarmen