Conexión a BD desde c#

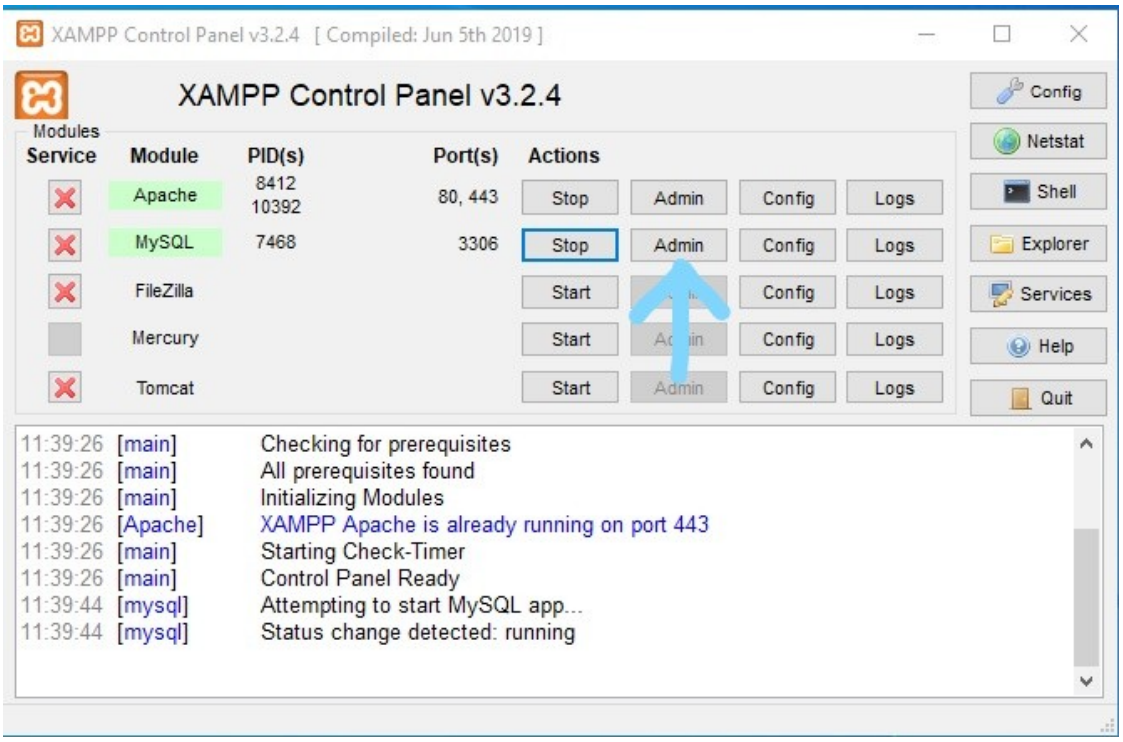
Índice

1 Introducción (instalación y despliegue de las herramientas)

Este material tiene como finalidad servir de ejemplo para el diseño y la implementación de un una aplicación de gestión con acceso a bases de datos utilizando para ello un entorno gráfico como Windows Forms

1.1 Instalación del SGBD

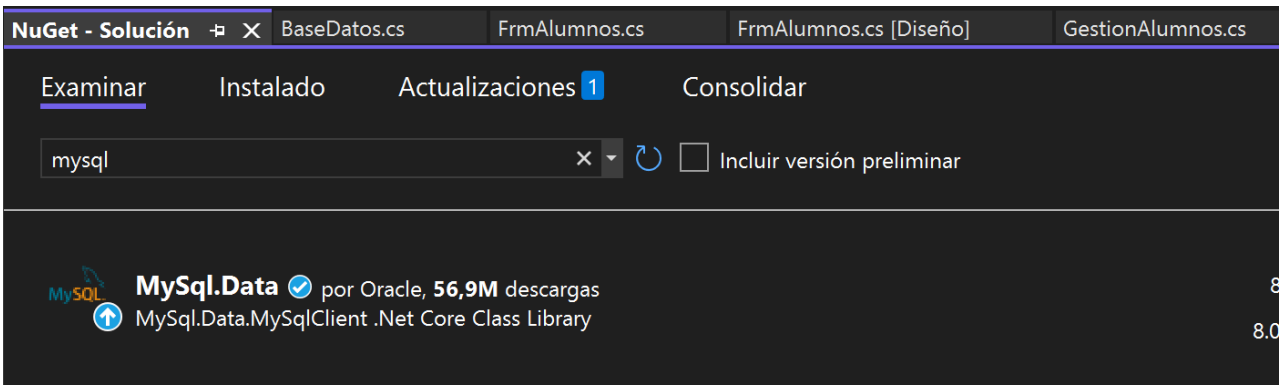
Para trabajar en este tema vamos a utilizar la herramienta XAMP que se puede descargar del siguiente enlace: https://www.apachefriends.org/es/index.html que nos permitirá trabajar más cómodamente.



1.2 Descarga del conector o driver

Para instalar el conector o driver lo podemos hacer de dos formas:

1. Desde el gestor de paquetes NuGet que lleva el propio Visual Studio



2. Desde la página web de MySQL https://dev.mysql.com/downloads/ y descargamos el que necesitemos, en este caso, el Connector/NET . Para hacerlo seguramente nos pedirá que nos registremos en Oracle

- C API (libmysqlclient)
- Connector/C++
- Connector/J
- Connector/NET
- Connector/Node.js
- Connector/ODBC
- Connector/Python
- MySQL Native Driver for PHP

1.3 Importar la BD para empezar a trabajar

Para empezar importaremos una base de datos sencilla con tres tablas haciendo click aquí. Para importar la base de datos debemos entrar primero en phpMyAdmin para ello pulsamos en admin de MySQL y luego creamos una base de datos nueva que se llamara ***academia, nos posicionamos en ella y luego pulsamos en*** Importar

2 Análisis de la aplicación

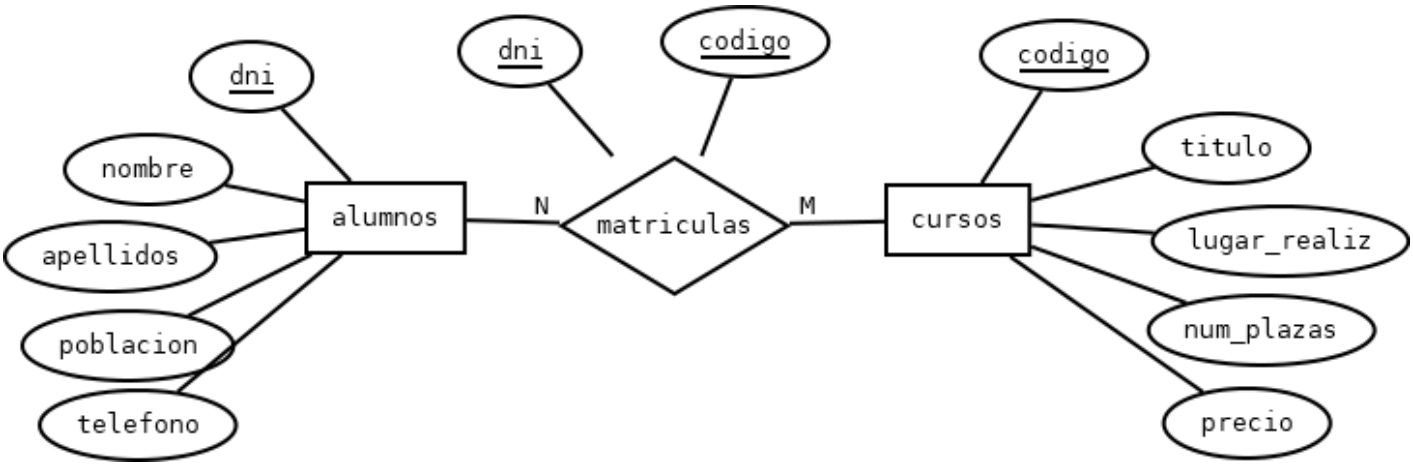
2.1 Análisis de Requerimientos

Se trata de una aplicación para control y gestión de una academia, tenemos alumnos, de los cuales guardaremos: dni, apellidos, nombre, población y teléfono y tenemos cursos a los cuales se pueden matricular los alumnos, un alumno se puede matricular en más de un curso, de los cursos almacenamos: código, título, lugar de realización,número de plazas y precio.

El cliente nos ha pedido una aplicación con ventanas, no de consola, por lo tanto habrá que realizarla con Windows Forms

2.2 Diseño de la BD academia

2.2.1 Modelo E/R



2.2.2 Modelo Relacional

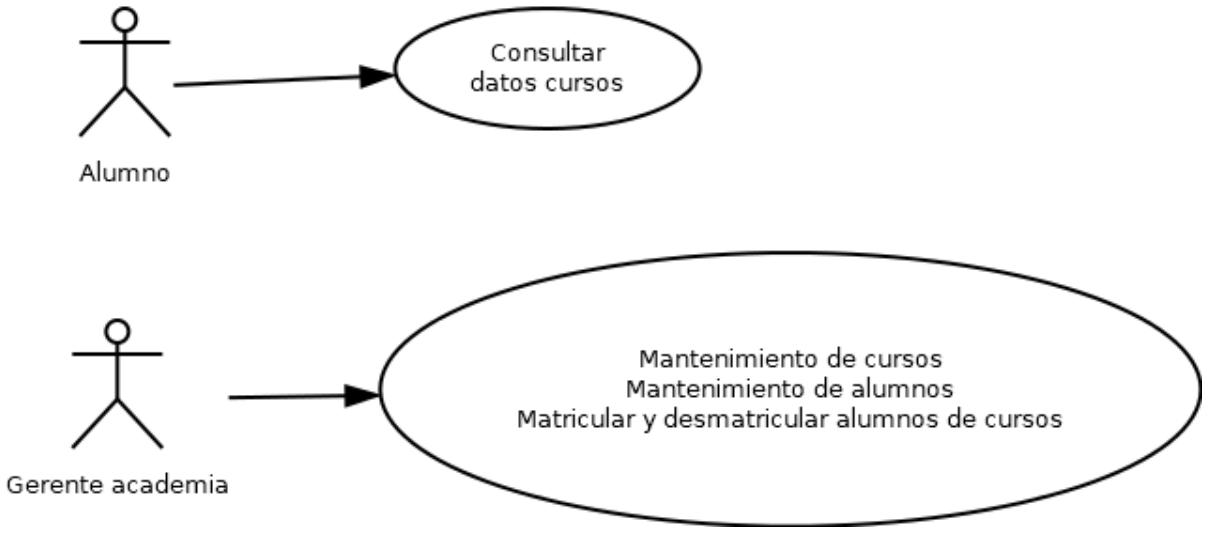
A partir del modelo E/R definimos el siguiente modelo relacional, definimos las claves principales con subrayado y las claves ajenas con fondo gris:

alumnos(dni,nombre,apellidos,poblacion, telefono)

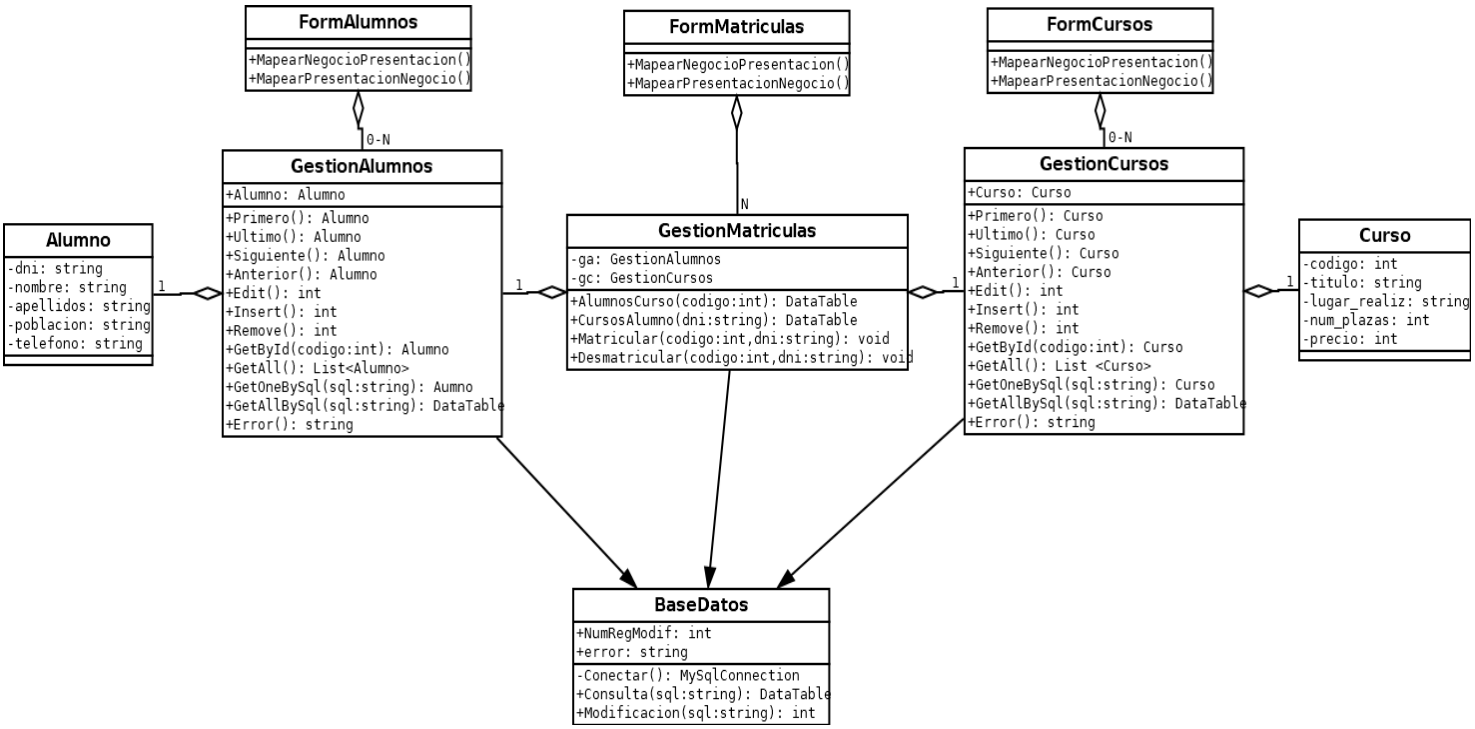
matriculas(dni, codigo)

cursos (codigo, titulo, lugar_realiz, num_plazas, precio)

2.3 Diagrama de Casos de Uso



2.4 Diagrama de clases



3 Capas de la programación

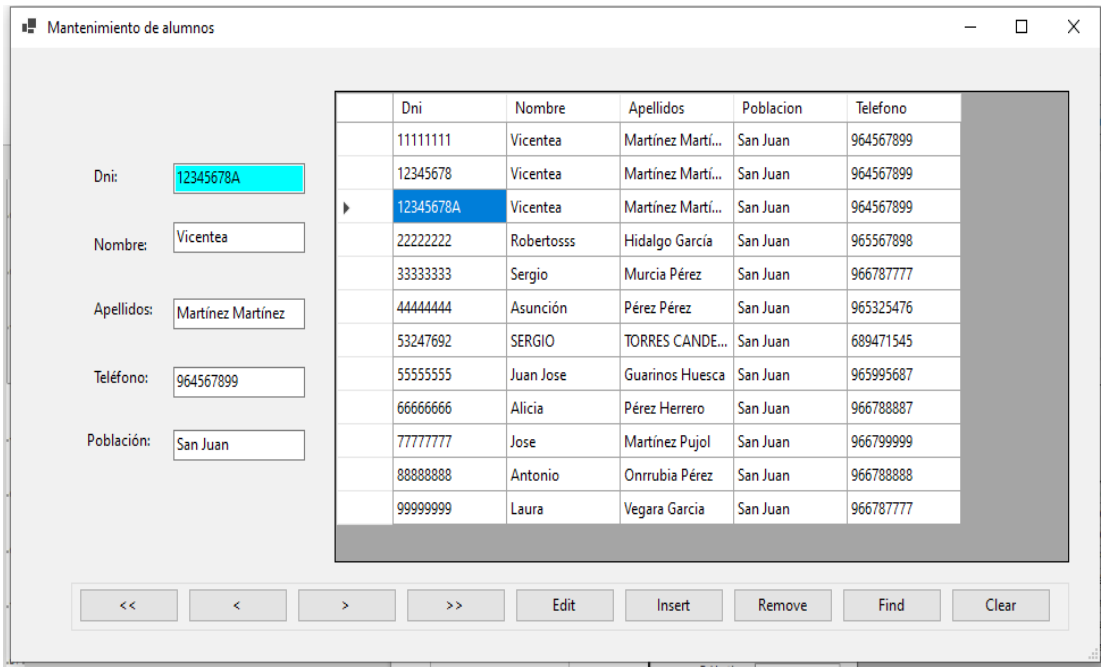
3.1 Clases de la capa de presentación (formularios)

Pantalla principal

frmAcademia

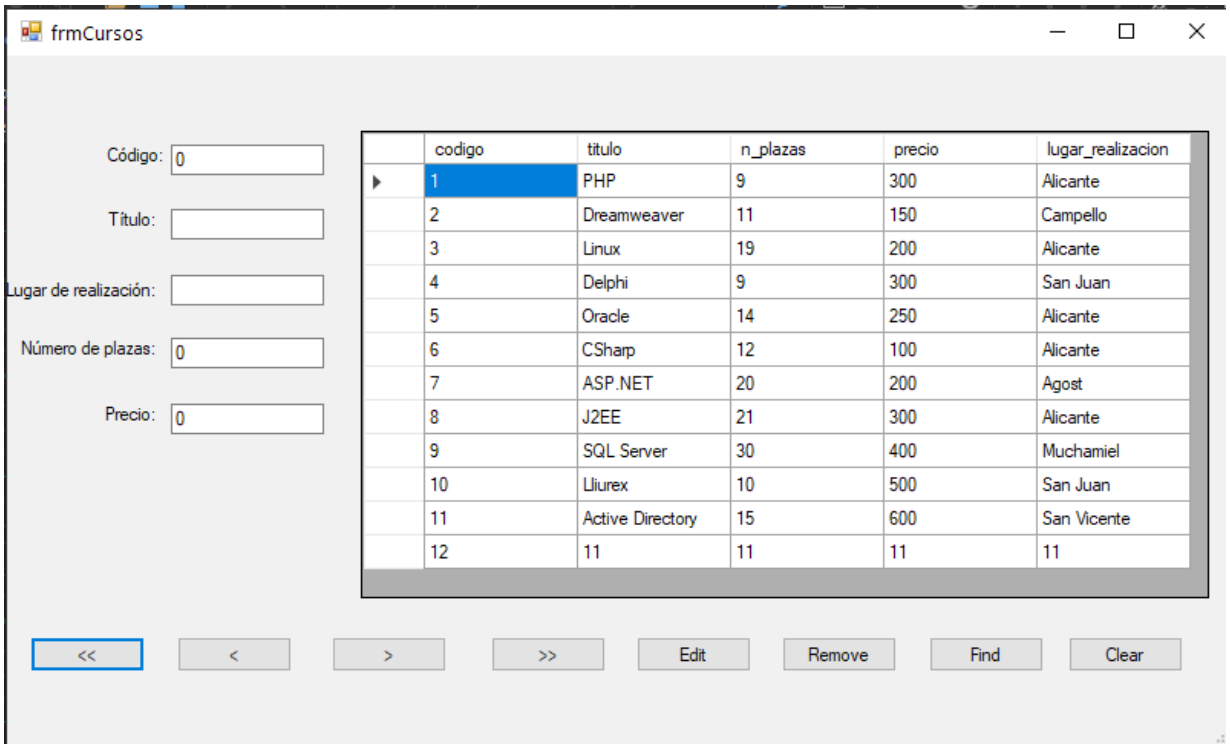
Mantenimiento de alumnos

frmAlumnos



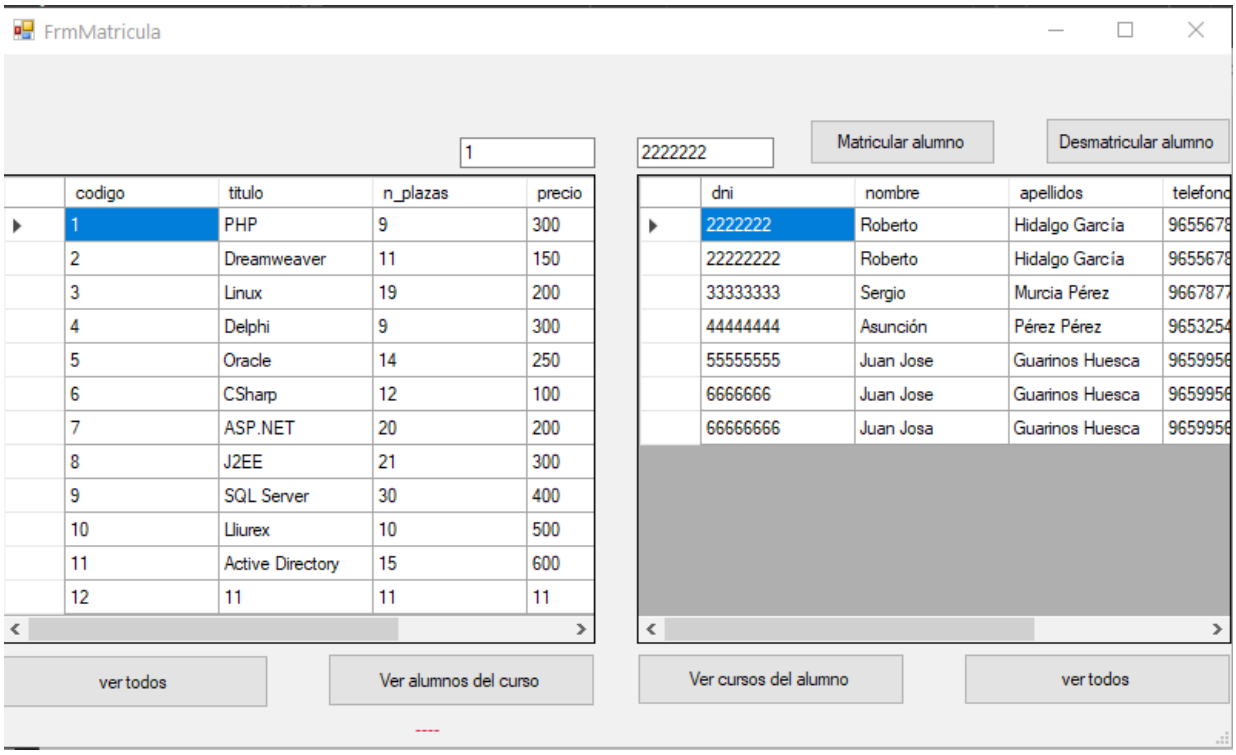
Mantenimiento de cursos

frmCursos



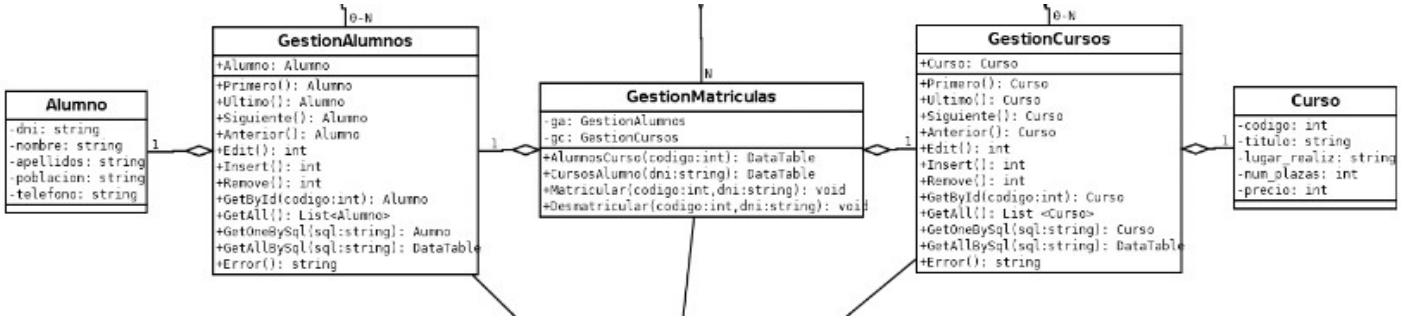
Gestión de matrículas

frmMatriculas

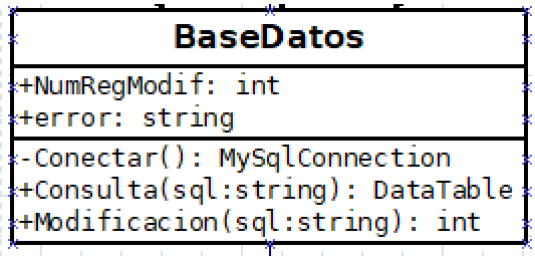


1.

3.2 Clases de la capa de negocio



3.3 Clases de la capa de datos (acceso a la base de datos)

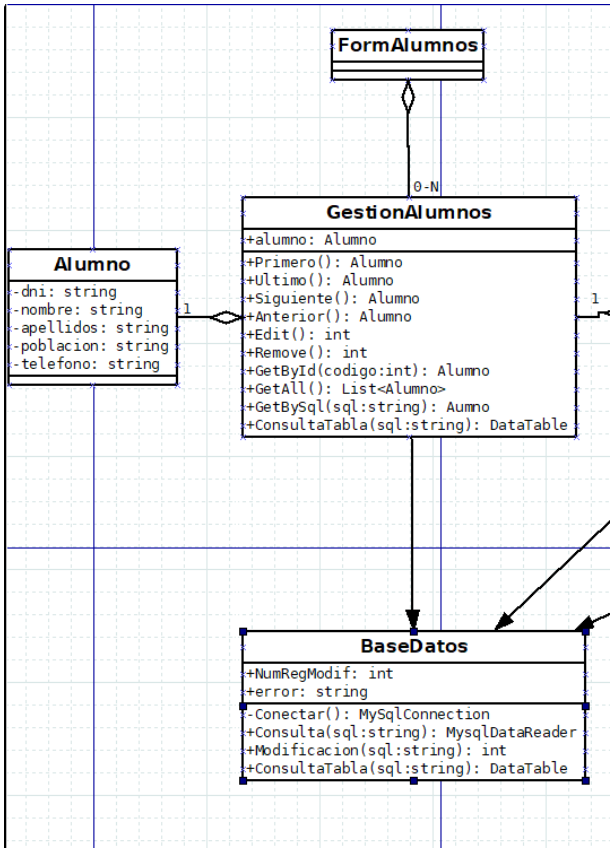


4 Implementación

Para empezar la implementación lo haremos en sentido inverso a las flechas que son las que indicas que una clase esta acoplada con otra (usa o tiene o hereda).

En este orden:

1. BaseDatos
2. Alumno
3. GestionAlumnos
4. FrmAlumnos



4.1 Capa de datos

Empezaremos a programar el sistema por la clase menos acoplada del sistema (menos acoplada significa que no llama o utiliza ninguna otra clase del sistema, ver diagrama de clases) que se encuentra en la capa de datos, la clase BaseDatos que tiene un método privado para conectar con la base de datos y dos métodos públicos estáticos:

- Consulta() Que devuelve un objeto DataTable o null dependiendo de que la consulta devuelva o no datos
- Modificación() Que devuelve el número de registros afectados en un Insert, Delete o Update

4.1.1 la clase *BaseDatos*

```
using System;
using System.Data;
using MySql.Data.MySqlClient;
using System.Configuration;

namespace academia_03data  {
    static class BaseDatos       {
        static public string Error { get; set; }
        static public int NumRegModif { set; get; }

        // método que conecta con la BD
        static private  MySqlConnection Conectar() {
            string cad = "datasource=127.0.0.1; port=3306; username=root; password=; database=academia;";
            MySqlConnection conexionBD = new MySqlConnection(cad);
            return (conexionBD);
        }

        //consulta de tipo SELECT que devuelve un DataTable (tabla en memoria)
        static public DataTable Consulta(string sql) {
            MySqlConnection conexionBD = Conectar();
            MySqlDataAdapter adapter = new MySqlDataAdapter(sql, conexionBD);
            try   {
                DataSet ds = new DataSet();
                adapter.Fill(ds);
                NumRegModif = 0;
                Error = "";
                return (ds.Tables[0]);
            }catch (Exception e)    {
                Error = e.Message;
                NumRegModif = -1;
                return (null);
            }finally { conexionBD.Close(); }
        }

        //consulta de acción INSERT, UPDATE Y DELETE se devuelve el número de registros afectados
        static public int Modificacion(string sql)   {
            MySqlConnection conexionBD = Conectar();
            try   {
                MySqlCommand comando = new MySqlCommand(sql, conexionBD);
                conexionBD.Open();
                NumRegModif =  comando.ExecuteNonQuery();
                Error = "";
                return NumRegModif;
            }catch (MySqlException ex)   {
                Error =ex.Message;
                NumRegModif = -1;
                return NumRegModif;
            }finally  { conexionBD.Close();  }
```

```
        }
    }
}
```

Como podemos observar en caso de error:

- almacena -1 en la propiedad numRegModif
- almacena el texto del error en la propiedad Errror
- devuelve null

Esto se hace para manejar el error desde las capas de arriba. Si mostráramos el error en esta capa por consola o con un formulario con MessageBox.Show(“texto”) o algo parecido, estaríamos acoplando esta capa con la de presentación que no es nada conveniente.

4.2 Capa de negocio

4.2.1 La clase Alumno

Una clase que contiene todas las propiedades de un alumno, junto con sus geters y seters y los constructores, uno sin propiedades y otro completo con todas las propiedades.

Esta clase alumno representa los datos que vemos en el formulario de un alumno

4.2.2 La clase GestionAlumnos

```
class GestionAlumnos
   {
        public Alumno Alumno { set; get; }  //El alumno con el que trabajaremos

        public GestionAlumnos()
        {
            Alumno = new Alumno();
        }
        public string Error()
        {
           //Devuelve el error si lo hay de la clase BaseDatos
        }
        public Alumno Primero()
        {
            //Busca el primer alumno y si lo encuentra lo asigna a la propiedad Alumno
        }
        public Alumno Ultimo()
        {
           //Busca el último alumno y si lo encuentra lo asigna a la propiedad Alumno
        }
        public Alumno Siguiente()
        {
           //Busca el siguiente alumno al actual (this.Alumno.dni) y si lo encuentra lo asigna a la
propiedad Alumno
        }
        public Alumno Anterior()
        {
            //Busca el anterior alumno al actual (this.Alumno.dni) y si lo encuentra lo asigna a la
propiedad Alumno
        }

       //Este método actualiza los datos de alumno cargado, hace un select con la clave principal
Aluno.Dni y si existe en la base de datos hace un update, en caso de no existir devuelve -1 para que la
capa de encima pueda sacar un mensaje
        public int Edit()
        {
            string sql = "select * from alumnos where dni = '" + this.Alumno.Dni+"'";
            if (BaseDatos.Consulta(sql).Rows.Count > 0)
            {
                sql = "update alumnos set  nombre = '" + this.Alumno.Nombre + "', apellidos =  '"
                    + this.Alumno.Apellidos + "', telefono = '" + this.Alumno.Telefono +
                   "', poblacion = '" + this.Alumno.Poblacion + "' where dni = '" + this.Alumno.Dni +
"'";
                return (BaseDatos.Modificacion(sql));
            }
            return -1;
        }
```

```
//Este método inserta un nuevo alumno cuyos datos tenemos cargados en la propiedad Alumno, para ello
primero mira que el alumno no exista en la base de datos, en caso de que ya exista devuelve -1
        public int Insert()
        {
            //Este método hace intenta insertar el alumno que tenemos en la propiedad
       GestionAlumnos.Alumno:
              - si no existe su DNI en la base de datos lo inserta y devuelve 1
              - Si existe su DNI o hay algún error devuelve  -1
        }



        public int Remove()
        {
            //Este método borra de la base de la base de datos el alumno con la clave principal:
Alumno.dni. Habrá que sacar una  ventana de advertencia antes de borrarlo
        }

//Este método obtiene el alumno cuyo dni se pasa como parámetro

        public Alumno GetById(string dni)
        {
            DataTable dt = BaseDatos.Consulta("select * from alumnos where dni = '" + dni+ "'");
            if (dt != null && dt.Rows.Count > 0 )
            {
                Alumno a = new Alumno(dt.Rows[0]["dni"].ToString(), dt.Rows[0]["nombre"].ToString(),
dt.Rows[0]["apellidos"].ToString(), dt.Rows[0]["telefono"].ToString(), dt.Rows[0]
["poblacion"].ToString());
                return (a);
            }
            return (null);
        }

//Este método obtiene una lista con todos los alumnos

        public List<Alumno> GetAll()
        {
            List<Alumno> alumnos = new List<Alumno>();
            DataTable dt = BaseDatos.Consulta("select * from alumnos order by dni");

            for(int i = 0; dt != null && i < dt.Rows.Count; i++)
            {
                alumnos.Add(new Alumno(dt.Rows[i]["dni"].ToString(), dt.Rows[i]["nombre"].ToString(),
dt.Rows[i]["apellidos"].ToString(), dt.Rows[i]["telefono"].ToString(), dt.Rows[0]
["poblacion"].ToString()));
            }
            return (alumnos);
        }

//Este método obtiene un alumno (el primero en orden) a partir de una consulta SQL

        public Alumno GetOneBySql(string sql)
        {
            DataTable dt = BaseDatos.Consulta(sql);
            if (dt != null && dt.Rows.Count > 0 )
            {
                Alumno a = new Alumno(dt.Rows[0]["dni"].ToString(), dt.Rows[0]["nombre"].ToString(),
dt.Rows[0]["apellidos"].ToString(), dt.Rows[0]["telefono"].ToString(), dt.Rows[0]
["poblacion"].ToString());
                return (a);
            }
            return (null);
        }
//Este método devuelve un DataTable con el resultado de la consulta que se le pase
```

```
        public DataTable GetAllBySql(string sql)
        {
            return BaseDatos.Consulta(sql);
        }
    }
}
```

4.3 Capa de presentación

Clases:

1. FrmAlumnos

Esta capa de presentación se va construyendo poco a poco para ir probando las clases de abajo. Esta clase tiene un objeto GestionAlumnos y tiene varios métodos importantes:

- MapearNegocioPresentacion()

  Carga las propiedades de un objeto Alumno a los cuadros de texto del formulario, por ejemplo:

  ```
  this.txtDni.Text = this.gestionAlumnos.Alumno.Dni;
  ```

- MaperarPresentacionNegocio()

  Carga las propiedades de los textbox del fornulario a las propiedades del objeto Alumno, por ejemplo:

  ```
  this.gestionAlumnos.Alumno.Dni = this.txtDni.Text;
  ```

- Métodos para contestar a los eventos del ratón al hacer click sobre los botones
  - `btnPrimero_Click()`
  - `btnUltimo_Click()`
  - `btnAnterior_Click()`
  - `btnSiguiente_Click()`
  - `btnEdit_Click()`
  - `btnInsert_Click()`
  - `btnRemove_Click()`
  - `btnFind_Click()`
  - `btnClear_Click()`

- Eventos al cargar la pantalla y al hacer clic en el DataGridView
  - `FrmAlumnos_Load()`
  - `dgvListaAlumnos_CellContentClick()`

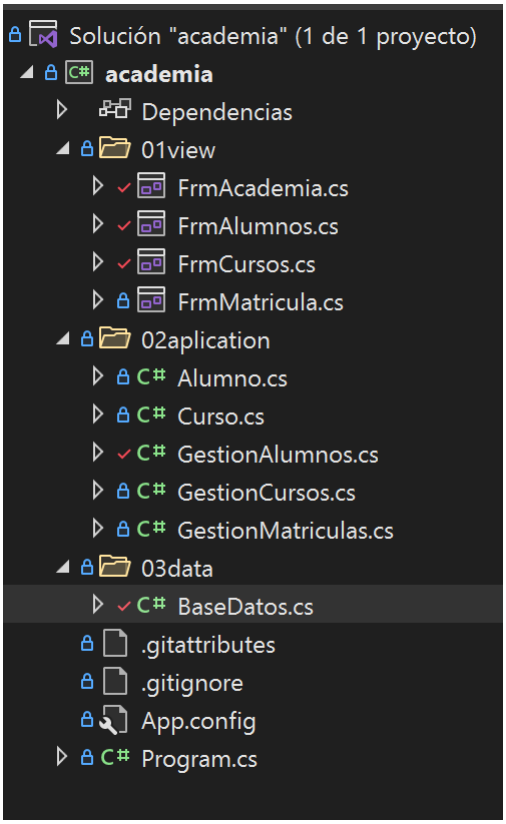
5 Ejercicios propuestos

A partir de los ejemplos de código y del análisis propuesto, desarrolla la aplicación de la Academia completa.

Primero haz que funcione la parte del mantenimiento de alumnos. En segundo lugar implementa el mantenimiento de cursos y de forma opcional implementa la parte de matriculas.

Una vez creado el proyecto, una buena idea para mantener la separación entre capas es tener las clases pertenecientes a cada una de ellas en carpetas diferentes, en el ejemplo tenemos:

- 01view → para las vistas, es decir los formularios
- 02aplication → para las clases de la lógica de negocio
- 03data → para las clases de acceso a datos



5.1.1 Creación del esqueleto de la aplicación y añadir las clases BaseDatos y Alumno en sus carpetas correspondientes

5.1.2 Creación del formulario de alumnos frmAlumnos y poner como propiedad del formulario la clase GestionAlumnos. Según diagrama de clases

En el formulario hay que cambiar de nombre todos objetos siguiendo la siguiente nomenclatura:

tres letras del tipo de objeto que es y luego a que objeto afecta, por ejemplo:

- Labels: lblDni, lblNombre,...lblMensajes (para sacar los errores de la base de datos)

- TextBox: txtDNI, txtNombre…
- DataGridView: dgvAlumnos (para sacar todos los alumnos)

5.1.3 En el formulario frmAlumnos realizar los métodos:

- MaperarNegocioPresentacion() //del objeto Alumno a los textBox
- MapearPresentacionNegocio() //de los textbox al objeto Alumno
- En el eveto Load del formulario pondremos lo siguiente para cargar todos los alumnos:
  - dgvAlumnos.DataSource = this.GestionAlumnos.GetAll() //para cargar la lista de alumnos en el DataGridView

5.1.4 En la clase GestiónAlumnos hacer los métodos:

- Error() // devuelve BaseDatos.Error
- Primero(), Ultimo() // Podemos llamar al método:
  - GestionAlumnos.GetOneBySql(**sql**) que devuelve el primer alumno que aparece en la consulta o null si no lo encuentra,
- Siguiente() y Anterior() //debemos tener un alumno cargado en la propiedad GestionAlumnos.Alumno y si es así lo usamos para sacar el siguiente, en caso contrario no hacemos nada
- Podéis basaros en el método Update() que ya está hecho para hacer los siguientes métodos:
  - Remove() //Borra el Alumno de la BD si existe
  - Insert() //Inserta Alumno en la BD si no existe
- Buscar()
  - Cuando pulsemos buscará por los datos que haya cargados en los TextBox:
    - txtDNI (que sea igual). Si este campo está vacío busca por el siguiente
    - txtNombre (busca por subcadena) . Si este campo está vacío busca por el siguiente.
    - txtApellidos (busca por subcadena). Si este campo está vacío busca por el siguiente.
    - txtTelefono (que sea igual) Si este campo está vacío busca por el siguiente.
    - txtPoblación (por subcadena).
  - El resultado se mostrará en el DataGridView

5.1.5 Enlazar el formulario frmAlumnos con la clase GestionAlumnos

- Evento click de los botones Primero, Último, Siguiente y Anterior, llamarán al método correspondiente de la clase GestionAlumnos
- Evento click de los botones Editar, Insertar, Borrar,
- Evento click del botón Buscar
- Evento `CellContentClick` que se produce al pulsar en un elemento del DataGridVier y que provocará que el alumno seleccionado se cargue el objeto GestionAlumnos.Alumno y se mapea en los textBox del formulario