Programación Visual en C#.NET

Guion de la práctica 7

OBJETIVOS

- Aprender a crear y trabajar con bases de datos
- Aprender a crear y trabajar con controles enlazados a datos

TEMPORIZACIÓN

Recogida del enunciado:	Semana del 7 de noviembre
Desarrollo de la práctica:	**3 semanas**
Fecha de entrega:	Semana del 28 de noviembre
Fecha límite de entrega:	Semana del 5 de diciembre

PRÁCTICA 7
Bases de datos y controles enlazados a datos

TABLA DE CONTENIDOS:

11 Introducción

La finalidad de la mayoría de las aplicaciones es mostrar datos a los usuarios con el fin de que puedan editarlos. Esto sugiere pensar que habrá que transportar datos entre un origen y un destino; por ejemplo, entre el modelo de objetos obtenido del modelo de datos relacional de una base de datos y la interfaz gráfica de la aplicación que consume esos datos.

El enlace de datos es un proceso que establece una conexión entre la interfaz gráfica del usuario (IGU) de la aplicación y la lógica de negocio, para que cuando los datos cambien su valor, los elementos de la IGU que estén enlazados a ellos reflejen los cambios automáticamente, y viceversa.

Este proceso de transportar los datos adelante y atrás, si lo tenemos que implementar manualmente, requeriría escribir bastante código, pero utilizando los objetos que nos proporciona la biblioteca .NET, comprobaremos que se trata de un proceso sencillo.

Para la realización de esta práctica, deberá tener instalado *Microsoft Visual Studio*. Puede optar por crear una base de datos *SQL Server Express LocalDB* (motor de bases de datos predeterminado); esto no quiere decir que no pueda instalar y utilizar *Microsoft SQL Server xxxx Express* como motor de bases de datos (se aconseja instalar este servidor ya que será necesario en la práctica 8). Puede realizar la descarga de todos estos paquetes desde la página web de la asignatura.

11.1 Creación de la base de datos

Lo primero que vamos a hacer es crear la base de datos en la que se guardarán datos relacionados con personas famosas. Para facilitar esta tarea, se proporciona un *script* junto con el enunciado de esta práctica.

Si sólo tiene instalado *SQL Server Express LocalDB* (es la instalación predeterminada), sáltese los puntos 1 y 2 y continúe en el siguiente apartado: *Crear la aplicación*, para crear la aplicación y la base de datos de acuerdo con las especificaciones de ese *script*. Si instaló *Microsoft SQL Server Express*, siga los siguientes pasos para crear la base de datos utilizando el *script*:

1. Vaya a *Inicio* > *Ejecutar*, y abra la línea de órdenes (mediante la orden **cmd**) y escriba la siguiente orden:

   ```
   > sqlcmd –S localhost\SqlExpress
   ```

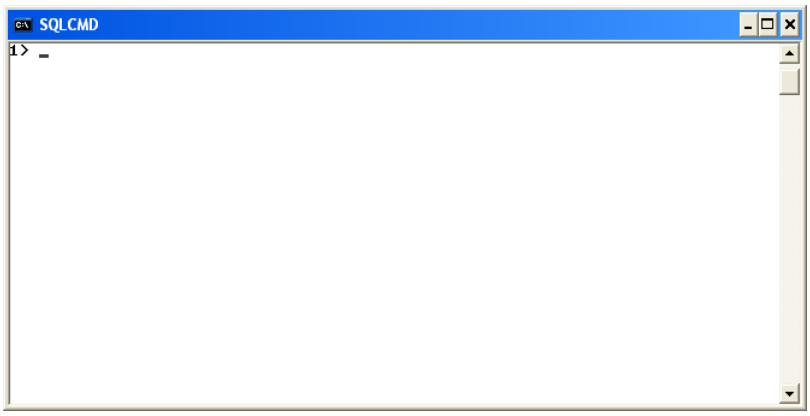
   Nota: **S** MAYÚSCULA. Pudiera darse el caso de que el ejecutable *SQLCMD.EXE* no se encontrara exactamente en la misma ruta en su máquina. Compruébelo.

   Si se ha ejecutado correctamente salte al punto 2. Si la ejecución de la orden ha retornado un error pruebe a ejecutarla de nuevo desde la carpeta donde se encuentra el fichero. Teclee las siguientes ordenes:

   ```
   > cd "C:\Archivos de programa\Microsoft SQL Server\1??\Tools\binn"
   ```

```
> sqlcmd -S localhost\SqlExpress
```

2. Abra el *script sql* adjunto con un editor de texto cualquiera (puede modificar los datos de los famosos, si lo desea), copie su contenido en la línea de órdenes (que tendrá un aspecto como el de la ventana siguiente), y pulse *Entrar*:

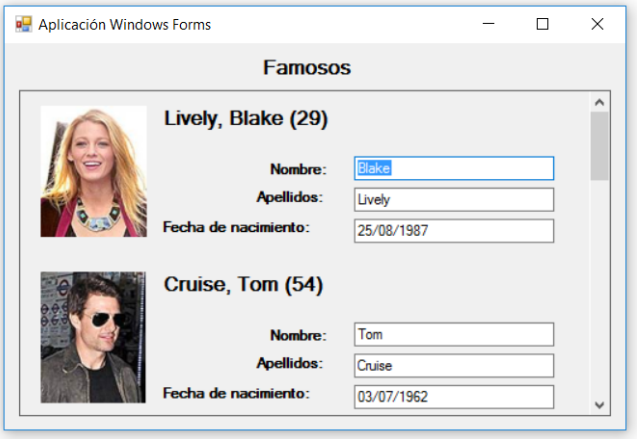


Una vez creada la base de datos observe que se llama ***bd_famosos*** y que contiene una tabla ***famoso*** con las siguientes columnas:

- ***ID***: es un entero que identifica al famoso entre otros (es la clave primaria).
- ***nombre***: es una cadena de caracteres de longitud 20 en la que se almacenará el nombre del famoso (p.ej. Jennifer).
- ***apellidos***: es una cadena de caracteres de longitud 30 en la que se almacenará los apellidos del famoso (p.ej. Lawrence).
- ***cumple***: es un campo de tipo fecha (***datetime***) en el que se almacenará la fecha de nacimiento del famoso (p.ej. dd/mm/aaaa).
- ***imagen***: es una cadena de caracteres de longitud 20. En ella, almacenará el nombre del archivo que contiene una foto del famoso (p.ej. famoso1.jpg, ...)

Crear la aplicación

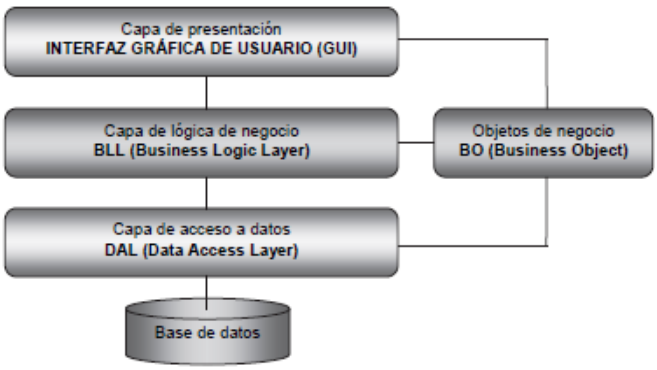
Una vez creada la base de datos, vamos a crear una aplicación con Visual Studio denominada *ApWinForms* para que, accediendo a la base de datos, muestre una interfaz como la que muestra la figura siguiente. Para ello, abra Visual Studio y cree un nuevo proyecto de tipo Visual C# > Windows > Aplicación de Windows Forms (.NET Framework) (se sugiere crear un directorio para la solución: desmarcar la casilla correspondiente).



En aplicaciones profesionales, el código de acceso a la base de datos está encapsulado en clases dedicadas a este tipo de operaciones. De esta forma, la aplicación, cliente de la base de datos, cuando tiene que ejecutar una operación contra la misma, crea un objeto de la clase adecuada y llama al método apropiado.

Un buen diseño sugiere crear una clase para cada tabla de la base de datos o para un grupo lógicamente relacionado de tablas, implementar un método por cada operación de inserción, borrado, modificación o selección, y utilizar procedimientos almacenados para cada operación sobre la base de datos.

La figura siguiente describe el modelo de aplicación que estamos proponiendo:



11.2 Interfaz para el acceso a los datos de la base de datos

Un enfoque adecuado en el desarrollo de aplicaciones es separar la capa de acceso a datos (DAL) de la capa de presentación. La DAL típicamente se implementa como un proyecto *Biblioteca de clases*. Entonces, como siguiente paso, añada un nuevo proyecto de tipo *Biblioteca de clases (.NET Framework)* a la solución *ApWinForms*, denominado, por ejemplo, *BDFamosos*, y establezca este proyecto como proyecto de inicio.

Si aún no ha creado la base de datos porque sólo tiene instalado *SQL Server Express LocalDB*, es el momento de crearla. Para ello, añada al proyecto *BDFamosos* un nuevo elemento, *bd_famosos.mdf*, del tipo *Base de Datos basada en servicio*. En la ventana de propiedades, cambie el valor de *Copiar en el directorio* a *Copiar si es posterior*. A continuación, puede continuar añadiendo las tablas, los datos de las tablas, las relaciones entre las tablas, etc. Para ello haga doble click sobre el nodo *bd_famosos.mdf* para que se muestre el *Explorador de Servidores*; (tiene un ejemplo en el apartado

Ejercicios resueltos del capítulo *Acceso a una base de datos* del libro referenciado en la bibliografía). (Opcionalmente podría utilizar la ventana *Explorador de Objetos de SQL Server* para realizar esta tarea).

Nota: como alternativa, utilizando *SQL Server Express LocalDB* también se podría haber creado la base de datos local al proyecto según explica el punto 1. No se hizo por motivos puramente didácticos. En este caso, escribiríamos la siguiente orden:

```
> sqlcmd –S (localdb)\MSSQLLocalDB
```

La base de datos se creará en la carpeta del usuario.
Nota: el nombre del objeto SQL Server que tiene que utilizar puede ser `V11.0`, `MSSQLLocalDB`, `ProyectsV12`, etc., dependiendo de su instalación. Para averiguarlo, abra el menú *Ver* de VS y seleccione la opción *Explorador de objetos SQL Server*.

Operaciones contra la base de datos

Obtener todas las filas de la tabla *famoso*. Esta operación será realizada por el siguiente procedimiento almacenado:

```
CREATE PROCEDURE dbo.stproObtenerFilasFamosos
AS
    SELECT ID, nombre, apellidos, cumple, imagen
    FROM famoso
```

Modificar una fila de la tabla *famoso*. Esta operación será realizada por el siguiente procedimiento almacenado:

```
CREATE PROCEDURE dbo.stproActualizarFamosos
    @ID int,
    @nombre varchar(20),
    @apellidos varchar(30),
    @cumple datetime,
    @imagen varchar(20)
AS
    UPDATE famoso
    SET nombre = @nombre, apellidos = @apellidos,
        cumple = @cumple
    WHERE (ID = @ID)
```

Para añadir estos procedimientos a la base de datos, desde el EDI Visual Studio, añada una conexión a la base de datos en el explorador de servidores o explorador de objetos de SQL Server, expanda la base de datos y localice el nodo *Procedimientos almacenados* donde se deberán añadir los procedimientos almacenados.

Objetos de negocio

Para facilitar el intercambio de datos entre la capa de presentación y la capa de acceso a datos vamos a crear una clase *CFamosoBO* que proporcione todas las columnas de la tabla *famoso* como propiedades públicas, a las que añadiremos algunas más; por ejemplo, una, `modificado`, para seguir la pista a las modificaciones realizadas sobre el conjunto de datos recuperado.

Esta clase implementa la interfaz **INotifyPropertyChanged** para notificar a los clientes enlazados de los cambios que se produzcan en sus propiedades.

```
using System;
using System.ComponentModel;

namespace BDFamosos
{
  public class CFamosoBO : INotifyPropertyChanged
  {
    // Propiedades relacionadas con la tabla famoso
    private int ID;
    public int Id
    {
      get { return ID; }
      set
      {
        ID = value;
        modificado = true;
        OnPropertyChanged(new PropertyChangedEventArgs("Id"));
      }
    }

    // Añadir el resto de las propiedades

    // Constructores
    public CFamosoBO() {}
    public CFamosoBO(int id, string nom, string ape, DateTime cum, string ima = null)
    {
      Id = id; Nombre = nom; Apellidos = ape; Cumple = cum; Imagen = ima;
    }

    // Otras propiedades
    private bool modificado;
    public bool Modificado
    {
      get { return modificado; }
      set { modificado = value; }
    }

    public string Edad
    {
      get // complete el código
      {
        DateTime hoy = ...;
        int edad = ...;
        if (...) --edad;
        return String.Format("{0}", edad);
      }
    }

    public string NombreAprellidosEdad
    {
      get
      {
        // Devolver un string de la forma, p.ej.: Lawrence, Jennifer (22)
      }
    }

    // Implementación de la interfaz INotifyPropertyChanged
    public event PropertyChangedEventHandler PropertyChanged;
    public void OnPropertyChanged(PropertyChangedEventArgs e)
    {
      if (PropertyChanged != null)
      {
        PropertyChanged(this, e); // generar evento
      }
    }
```

```
  }
}
```

Complete el código en los lugares donde aparece "...".

Para trabajar con colecciones de objetos *CFamosoBO*, añada una clase *ColCFamosos* derivada de **BindingList**<*T*>:

```
using System.ComponentModel;

namespace BDFamosos
{
  public class ColCFamosos : BindingList<CFamosoBO> { }
}
```

Capa de acceso a datos

La capa de acceso a datos implementará una clase denominada *CFamosoDAL* que defina un método por cada procedimiento almacenado que definimos anteriormente:

```
using System;
using System.Data.SqlClient;
using System.Data;

namespace BDFamosos
{
  public class CFamosoDAL
  {
    private string strConexion;

    public CFamosoDAL()
    {
      // Obtener la cadena de conexión: elija la adecuada para su instalación
      //strConexion = @"Data Source=.\sqlexpress;" +
      //  @"Initial Catalog=bd_famosos; Integrated Security=True";

      //strConexion = @"Data Source=(LocalDB)\MSSQLLocalDB;" +
      //  @"AttachDbFilename=|DataDirectory|\bd_famosos.mdf; " +
      //  @"Integrated Security=True";

      // Si no es válido MSSQLLocalDB pruebe con v11.0
    }

    public ColCFamosos ObtenerFilasFamosos()
    {
      try
      {
        using (SqlConnection Conexion = new SqlConnection(strConexion))
        {
          SqlCommand OrdenSql =
            new SqlCommand("stproObtenerFilasFamosos", Conexion);
          OrdenSql.CommandType = CommandType.StoredProcedure;
          // Crear una colección para todos los famosos
          ColCFamosos colFamosos = new ColCFamosos();
          // Abrir la base de datos
          Conexion.Open();
          SqlDataReader lector = OrdenSql.ExecuteReader();
          while (lector.Read())
          {
            CFamosoBO fila = new CFamosoBO((int)lector["ID"],
              (string)lector["nombre"], (string)lector["apellidos"],
              (DateTime)lector["cumple"], (string)lector["imagen"]);
            colFamosos.Add(fila);
          }
```

```
          return colFamosos;
        }
      }
      catch (SqlException err)
      {
        throw new ApplicationException("Error SELECT famoso");
      }
    }

    public void ActualizarFamosos(CFamosoBO bo)
    {
      try
      {
        using ...
        {
          ...
          // Parámetros
          OrdenSql.Parameters.AddWithValue("@ID", bo.Id);
          ...
          // Abrir la base de datos
          Conexion.Open();
          OrdenSql.ExecuteNonQuery();
        ...
    }
  }
}
```

Capa de lógica de negocio

Opcionalmente, podemos añadir otra capa de lógica de negocio que interactúe con la capa de presentación y con la capa de acceso a datos. Cada clase de esta capa, normalmente, contendrá la misma interfaz pública que su clase análoga en la capa de acceso a datos, y sus métodos realizarán, si es preciso, funciones de validación y de filtrado de datos antes de llamar a sus homólogos en la capa de acceso a datos. A modo de ejemplo, vamos a añadir una clase *CFamosoBLL* cuyos métodos, en principio, simplemente llamarán a sus homólogos de la capa de acceso a datos:

```
namespace BDFamosos
{
  public class CFamosoBLL
  {
    private CFamosoDAL bd = new CFamosoDAL();

    public ColCFamosos ObtenerFilasFamosos()
    {
      ColCFamosos coTfnos = bd.ObtenerFilasFamosos();
      return coTfnos;
    }

    public void ActualizarFamosos(CFamosoBO bo)
    {
      bd.ActualizarFamosos(bo);
    }
  }
}
```

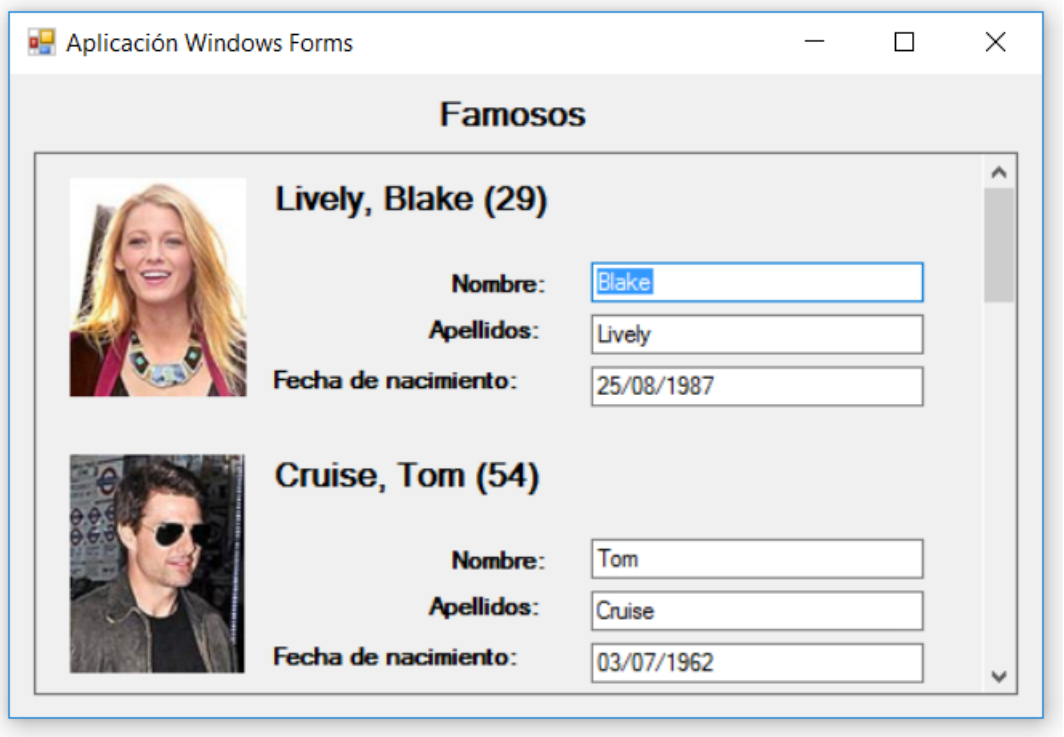
Compile y ejecute para ver el resultado.

Si todo ha ido bien, observará que en la ventana de resultados no se muestran errores. En este caso, para continuar con la aplicación, añada al proyecto *ApWinForms* una referencia a la biblioteca de clases *BDFamosos*, establezca *ApWinForms* como proyecto de inicio y añada a la clase *Form1* de *ApWinForms* la línea siguiente:

```
using BDFamosos;
```

11.3 Diseño de la capa de presentación

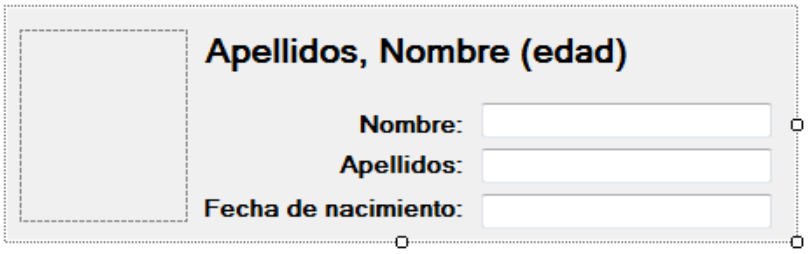
La ventana principal del proyecto *ApWinForms* presentará una interfaz gráfica como la siguiente:



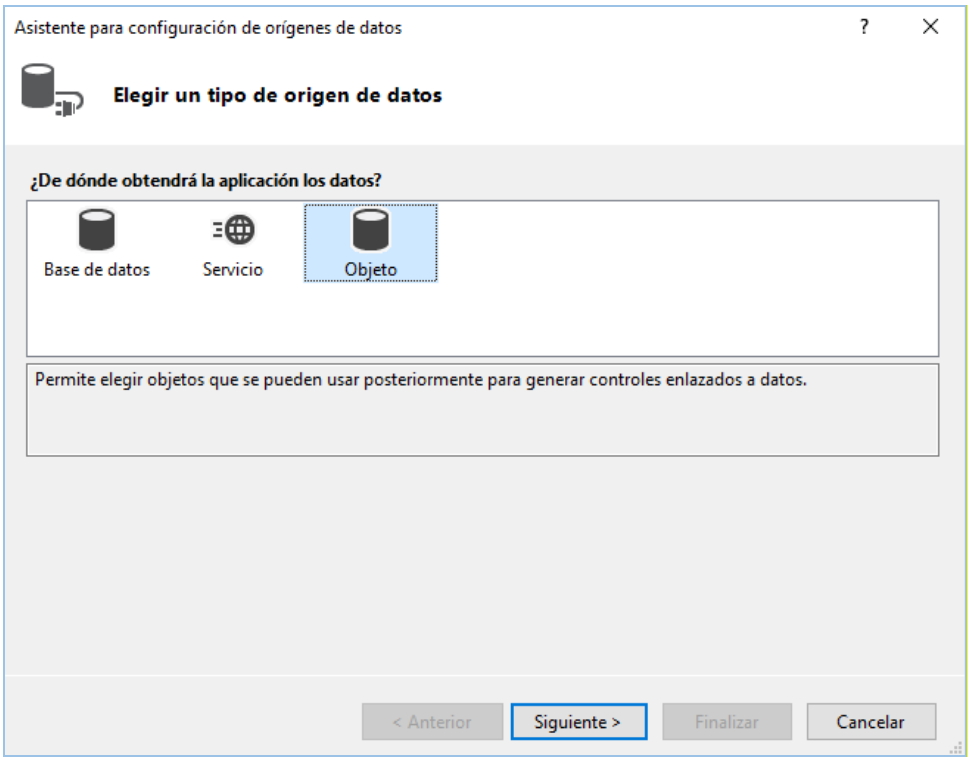
Observamos que la ventana muestra una lista donde cada elemento se corresponde con un objeto de la clase *CFamosoBO*.

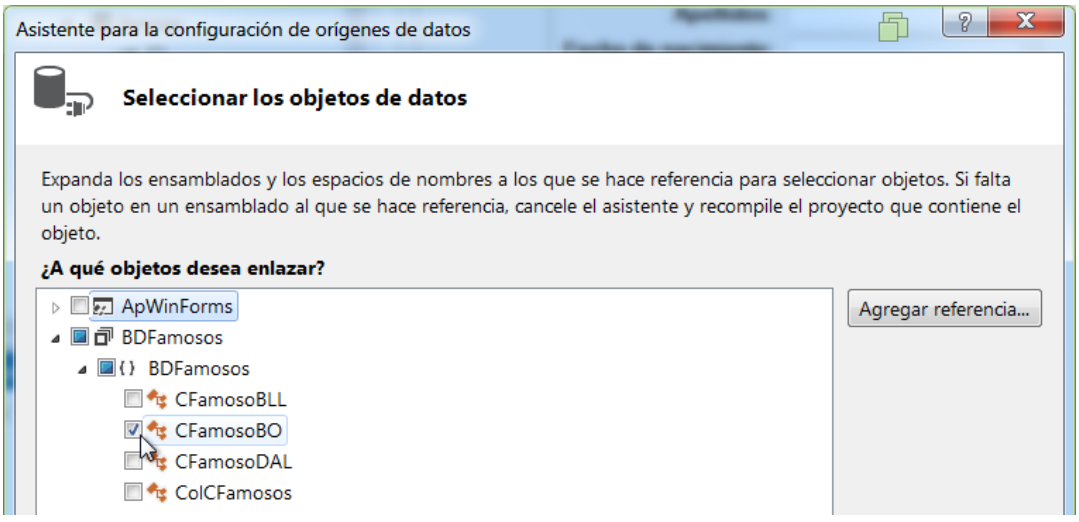
Cuando el usuario de la aplicación edite el nombre, los apellidos o la fecha de nacimiento de cualquiera de los elementos de la lista y mueva el foco a otro campo, el nombre completo más la edad de la persona en cuestión, que se muestra en la parte superior del elemento, será actualizado para reflejar los cambios. Esto indica que los controles estarán enlazados a las propiedades de los objetos *CFamosoBO*.

Los elementos de la lista que muestra el formulario serán objetos de una clase, *ElementoLista*, derivada de **UserControl** (un control de usuario es un control reutilizable). Añadamos entonces al proyecto *ApWinForms* un nuevo control de usuario denominado *ElementoLista*. Después, completaremos el diseño para que incluya los controles que vemos en la figura: un control **PictureBox** y controles **Label** y **TextBox** (denomínelos, *imgFamoso*, *etNomApEdad*, *txtNombre*, *txtApellidos* y *txtFechaNaci*). Añada también un control **ToolTip** denominado *ttElementoLista*.

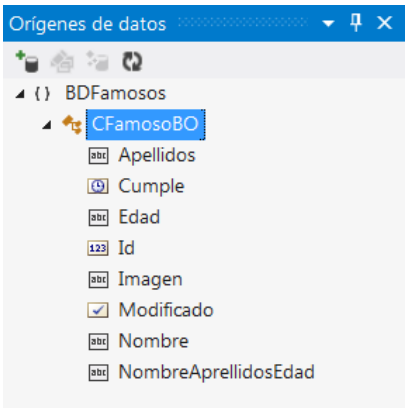


A continuación, añada al control de usuario *ElementoLista* un origen de datos generado a partir del modelo de datos. Para ello, utilice la ventana *Orígenes de datos* y seleccione *Objeto* como origen de datos.





El resultado será el origen datos siguiente:



Añada un objeto **BindingSource**, denominado *famosoBindingSource*, asigne a su propiedad **DataSource** el origen de datos *CFamosoBO*, y vincule cada propiedad del origen de datos *famosoBindingSource* con el control del objeto *ElementoLista* que debe mostrarla. Esto se puede hacer fácilmente desde la ventana de propiedades, estableciendo la propiedad **DataBindings** de cada uno de los controles; el código que se generará por cada uno, será análogo al siguiente:

```
this.txtNombre.DataBindings.Add(
  new Binding("Text", this.famosoBindingSource, "Nombre", true));
```

Después, complete la implementación del control de usuario añadiendo un conversor, *StringToBitmap*, que permita obtener la imagen (objeto **Bitmap**) desde la ruta (objeto **string**) del fichero que la contiene y, una propiedad, *ObjetoVinculado*, que permita, por una parte acceder al objeto *CFamosoBO* vinculado con el control de usuario, y por otra, vincular un objeto *CFamosoBO* con el control de usuario y establecer una descripción abreviada (objeto **ToolTip**), asociada con la imagen, que se corresponda con el ID de ese objeto.

Las imágenes las suponemos añadidas en una carpeta *Imagenes* de la carpeta desde la que se ejecuta la aplicación (fichero *.exe*).

```
public partial class ElementoLista : UserControl
{

  public ElementoLista()
  {
    InitializeComponent();
    // Obtener el enlace con la propiedad Image del control imgFamoso
    // para gestionar su evento Format
    Binding bImagen = this.imgFamoso.DataBindings["Image"];
    bImagen.Format += this.StringToBitmap;
  }

  private void StringToBitmap(object sender, ConvertEventArgs e)
  {
    // Format ocurre siempre que haya que mostrar en imgFamoso
    // el valor de la propiedad Imagen (la ruta de la imagen)
    if (e.DesiredType != typeof(Image)) return;
    // Convertir la ruta de la imagen a un Bitmap
    e.Value = Bitmap.FromFile(ObtenerRutaImagen(e.Value as string));
  }

  public CFamosoBO ObjetoVinculado
  {
    get { return this.famosoBindingSource.DataSource as CFamosoBO; }
```

```
    set
    {
      // Vincular objeto
      ...
      // Establecer un tooltip en la imagen que muestre el ID correspondiente
      // al objeto CFamosoBO
      if (value != null)
      {
        string msj = "ID famoso: " + value.Id;
        ...
      }
    }
  }

  private string ObtenerRutaImagen(string nombreImagen)
  {
    string carpeta =
      Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
    carpeta = Path.Combine(carpeta, "Imagenes");
    return Path.Combine(carpeta, nombreImagen);
  }
}
```

Compile y ejecute para ver el resultado.

Finalmente, realizamos el diseño del formulario principal de la aplicación. Para mostrar el conjunto de elementos *ElementoLista* vamos a utilizar un control **FlowLayoutPanel**. Este control organiza su contenido, referenciado por su propiedad **Controls**, en una dirección de flujo horizontal o vertical. La dirección del flujo se especifica estableciendo su propiedad **FlowDirection** (*BottomUp*, *LeftToRight*, *RightToLeft* y *TopDown*); el valor predeterminado es *LeftToRight*: los elementos fluyen desde el borde izquierdo de la superficie de diseño al borde derecho. Y el contenido del control puede ajustarse o recortarse estableciendo su propiedad **WrapContents** (**true** si debe ajustarse; **false** en caso contrario); el valor predeterminado es **true**.

Añada entonces un control **FlowLayoutPanel** al formulario y asigne a su propiedad **FlowDirection** el valor *TopDown* y a su propiedad **WrapContents** el valor **false**. Asigne también a su propiedad **AutoScroll** el valor **true**.

El panel será llenado cuando el usuario ejecute la aplicación. Esto es, por cada objeto *CFamosoBO* de la colección, crearemos un control *ElementoLista* (que invocará a su método *ObjetoVinculado* para vincularlo con el objeto *CFamosoBO* en curso y establecer la descripción abreviada) que añadiremos al panel.

```
public partial class Form1 : Form
{
  ColCFamosos colFamosos;
  CFamosoBLL bd;

  public Form1()
  {
    InitializeComponent();

    bd = new CFamosoBLL();
    colFamosos = bd.ObtenerFilasFamosos();

    // Agregar al panel un control ElementoLista por cada objeto CFamosoBO
    foreach (CFamosoBO obj in colFamosos)
    {
      ElementoLista elemento = new ElementoLista();
      elemento.ObjetoVinculado = obj;
```

```
      ...
    }
  }
}
```

Cuando se cierre el formulario, la base de datos deberá actualizarse con los cambios que se hayan realizado:

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
  for (int i = 0; i < colFamosos.Count; i++)
  {
    // Actualizar la base de datos con los cambios realizados
    ...
  }
}
```

Compile y ejecute para ver el resultado.

¿Es necesario que la clase *CFamosoBO* implemente la interfaz **INotifyPropertyChanged**? Compruébelo y razone la respuesta.