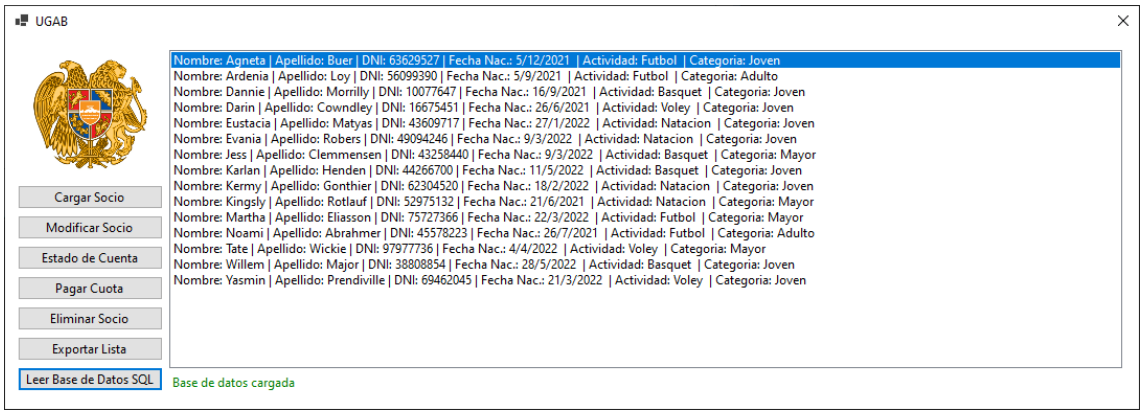
Tomás Pérez Ponisio, comisión 2E, primer cuatrimestre 2022

TP4 Final – Laboratorio 2: Gestor de socios del club UGAB

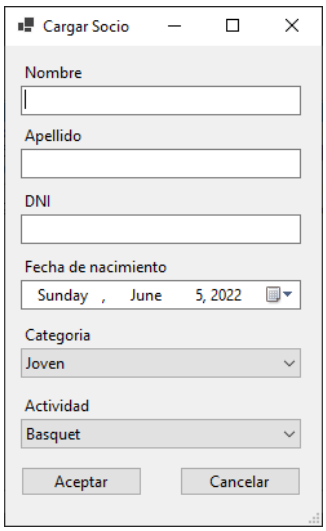
Siguiendo la consigna del TP el proyecto es un programa para gestionar a los socios de un club. En el inicio se pueden ver listados los socios del club, la lista estará vacía si no hay ningún socio o si no se ha cargado la base datos.



El menú tiene siete opciones con nombres descriptivos.

Pudiendo cargar un nuevo socio, modificar uno ya existente, ver el estado de cuenta de un socio (que cuotas pagó y si está al día), pagar una cuota del socio, eliminar un socio del club, guardar los datos de la lista de socios en formato XML y cargar la base de datos del club para gestionarlo.

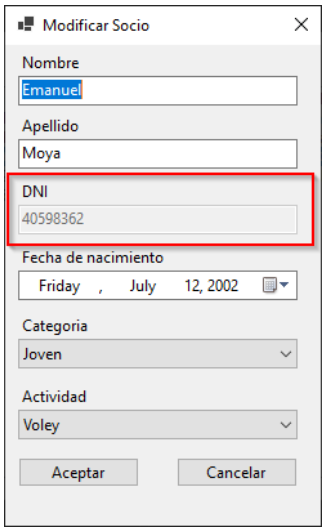
Cargar Socio



Aquí se puede cargar un nuevo socio al club ingresando todos los datos requeridos y en su correcto formato, por ejemplo, en el campo DNI solo se podrá ingresar números, que sean mayores a 0 y menores a 99999999, y que tengan 7 u 8 dígitos.

El campo DNI será el identificador de cada socio, siendo este dato único e irrepetible dentro del club.

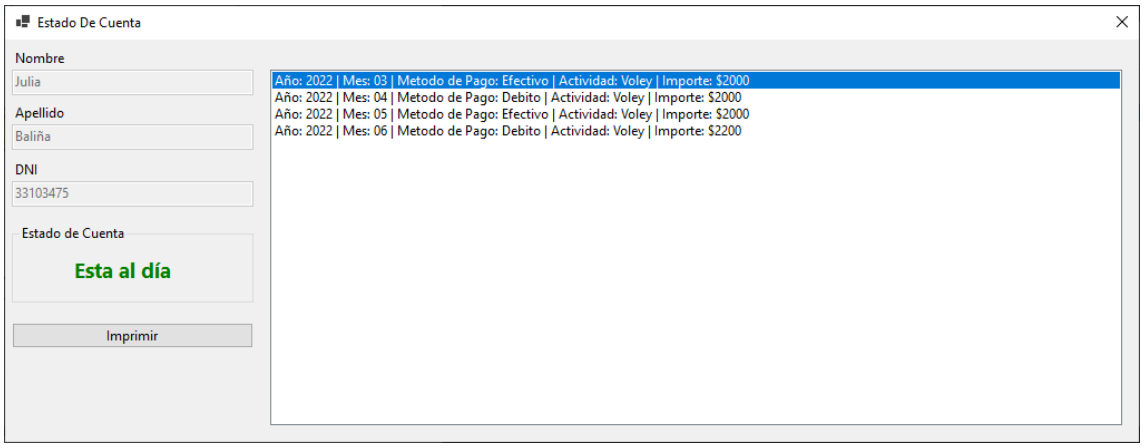
Modificar Socio

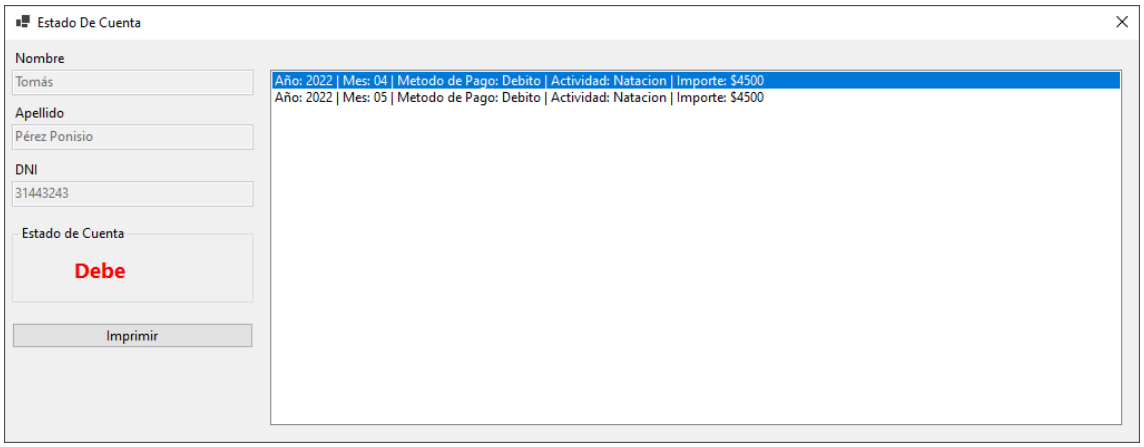


Con un socio seleccionado de la lista, se pueden modificar sus datos menos el de DNI, si se quiere modificar un DNI, hay que eliminar el socio y crearlo de nuevo.

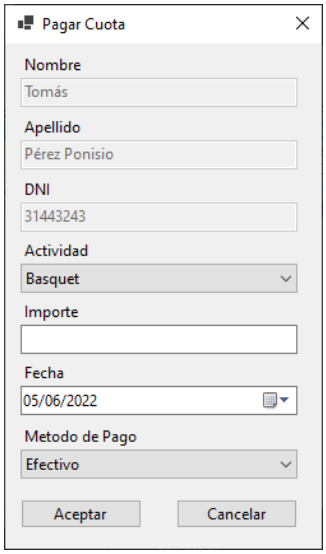
Estado de cuenta

Con un socio seleccionado se puede ver su estado de cuenta, esto es el historial de pagos de su cuota de socio y ver si debe o está al día. Un socio es deudor cuando no está paga la cuota correspondiente al último mes. Esta información se puede imprimir, se genera un archivo de texto con los datos del socio, el historial de pagos y el estado de la cuenta.





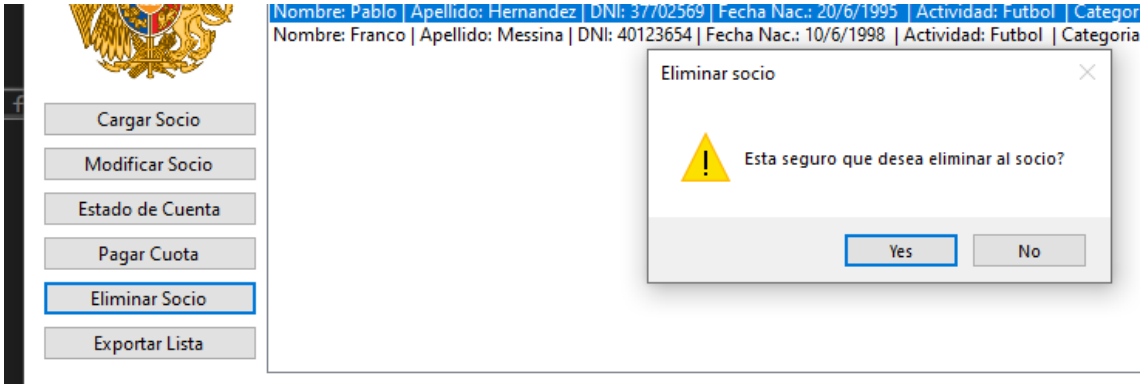
Pagar Cuota



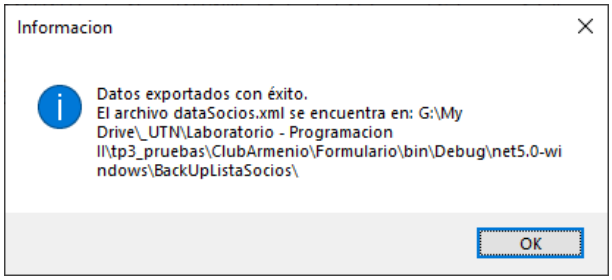
Aquí se puede pagar una cuota del socio seleccionado, se deberá ingresar el importe, fecha, la actividad y el método de pago y el registro se agregará automáticamente al historial de pagos (estado de cuenta) del socio. No se puede pagar dos veces el mismo mes, el sistema avisa con un mensaje.

Eliminar Socio

Con un socio seleccionado se puede eliminar del club, hay un pedido de confirmación previo.

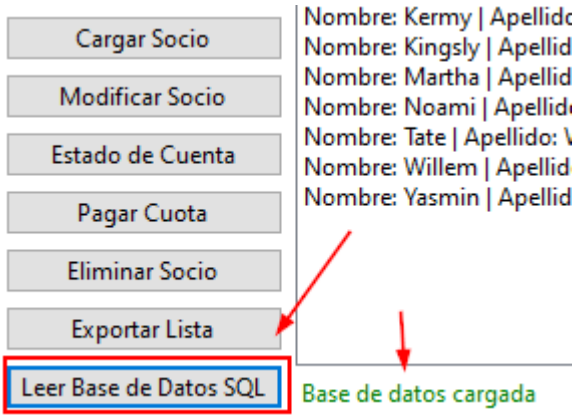


Exportar Lista



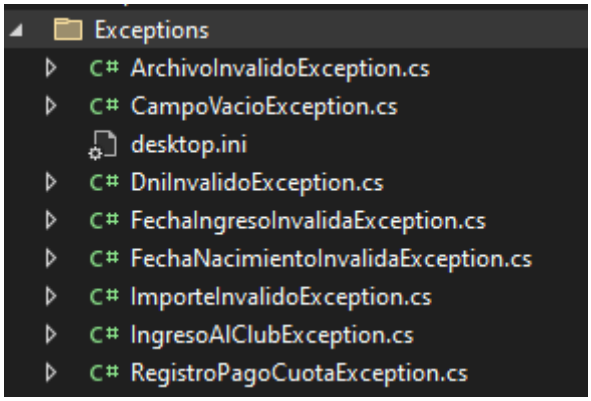
Exporta los datos del club en un archivo xml a modo de backup.

Leer Base de Datos SQL



Carga la base de datos de socios para comenzar a gestionar al club, en un label a su derecha se indica el estado de carga de los datos.

Tema 10 – Excepciones



Las siguientes excepciones fueron implementadas en distintas secciones del proyecto.

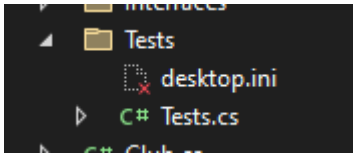
Por ejemplo, la *ArchivoInvalidoException* se dispara en todo lo que tenga que ver con manejo de archivos, en el *frmInicio* al leer o escribir el archivo con los datos de los socios, en *frmEstadoDeCuenta* al imprimir el archivo de texto.

```
17 references
public class ArchivoInvalidoException : Exception
{
    2 references
    public ArchivoInvalidoException(string message) : base(message)
    {
    }

    5 references
    public ArchivoInvalidoException(string message, Exception innerException) : base(message, innerException)
    {
    }
}
```

```
/// <summary>
/// Al cargar el formulario se busca si ya existe un archivo xml con datos de socios existentes, si lo encuentra se lee
/// y se crea un club con esa lista de socios encontrada. Si no hay archivo de datos se crea un club de cero.
/// Se actualiza la vista de la listbox.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void frmInicio_Load(object sender, EventArgs e)
{
    try
    {
        if (File.Exists($"{serializer.RutaDeEscritura()}\\dataSocios.xml"))
        {
            this.club = new Club("Club Armenio", this.serializer.Leer("dataSocios.xml"));
        }
        else
        {
            this.club = new Club("Club Armenio");
        }
        this.ActualizaListaSocios();
    }
    catch (ArchivoInvalidoException ex)
    {
        MessageBox.Show(ex.Message, "Error al leer el archivo de datos de socios", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Tema 11 - Pruebas unitarias



Respecto de los test unitarios se testean algunas de las funcionalidades del programa, por ejemplo, el método *EliminarSocio* de la clase socio que retorna un string informando el resultado.

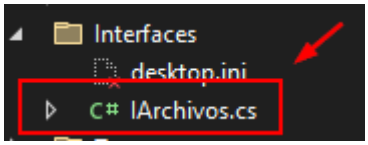
```
/// <summary>
/// Utilizando la sobrecarga del operador resta entre un club y un socio,
/// remueve el socio que recibe como paramtero de la lista del club
/// y retorna un mensaje en un string avisando si lo pudo remover o no
/// </summary>
/// <param name="socio"></param>
/// <returns>string</returns>
2 references | 1/1 passing
public string EliminarSocio(Socio socio)
{
    string mensaje = "El socio no se ha encontrado";
    if (this - socio)
    {
        mensaje = "Socio eliminado con exito";
    }
    return mensaje;
}
```

```
[TestMethod]
0 references
public void EliminarSocio_CuandoSeQuieraEliminarDeUnClubUnSocioQueNoExiste_DeberiaRetornarUnStringDiciendoQueElSocioNoSeHaEncontrado()
{
    //Arrange
    Club club = new Club("Test Club");
    Socio socio = new Socio("Juan", "Perez", 123, new DateTime(2000, 01, 01), Socio.EActividad.Basquet, Socio.ECategoria.Joven);
    string expected = "El socio no se ha encontrado";
    string actual;

    //Act
    actual = club.EliminarSocio(socio);

    //Assert
    Assert.AreEqual(expected, actual);
}
```

Tema 12 - Tipos genéricos y Tema 13 – Interfaces



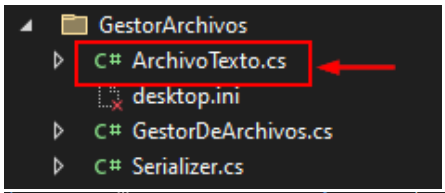
Por practicidad ambos se utilizan en la interfaz que usa el gestor de archivos.

```
public interface IArchivos<T> where T : class
{
    #region Metodos
    3 references
    T Leer(string nombreArchivo);
    4 references
    void Escribir(string nombreArchivo, T elemento);
    #endregion
}
```

```
3 references
public class ArchivoTexto : GestorDeArchivo, IArchivos<string>
{
    #region Constructor
    1 reference
```

```
3 references
public class Serializer<T> : GestorDeArchivo, IArchivos<T> where T : class, new()
{
```

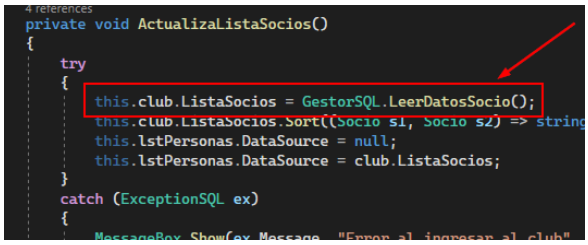
Tema 14 – Archivos



Archivos se usa en *frmEstadoDeCuenta* para imprimir un comprobante del estado de cuenta de un socio.

```
/// <summary>
/// Genera un documento de texto con los datos del socio, su historial de pago y si esta al dia con las cuotas o si debe el último mes.
/// Se abre el documento generado y se informa donde esta guardado. Si hay algun problema se arroja la exception correspondiente.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void btnImprimir_Click(object sender, EventArgs e)
{
    try
    {
        ArchivoTexto archivo = new ArchivoTexto();
        string nombreDeArchivo;
        archivo.Escribir($"{socio.Nombre}{socio.Apellido}.txt", socio.Imprimir());
        nombreDeArchivo = $"{archivo.RutaDeEscritura()}{socio.Nombre}{socio.Apellido}.txt";
        ProcessStartInfo proceso = new ProcessStartInfo
        {
            FileName = nombreDeArchivo,
            UseShellExecute = true
        };
        Process.Start(proceso);
        MessageBox.Show($"Impresión exitosa.\nRecibo en: {archivo.RutaDeEscritura()}", "Informacion", MessageBoxButtons.OK, MessageBoxIco
    }
    catch(ArchivoInvalidoException ex)
    {
        MessageBox.Show(ex.Message, "Error al escribir el archivo de texto", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error al escribir el archivo de texto", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Tema 15 – Serialización



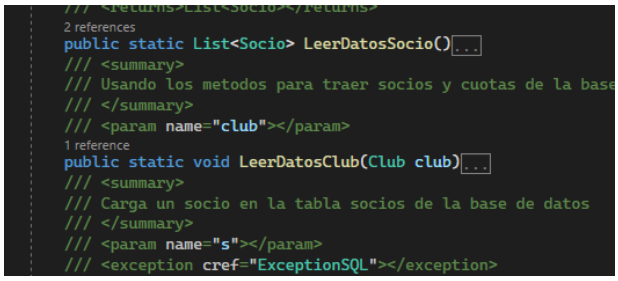
```
4 references
private void ActualizaListaSocios()
{
    try
    {
        this.club.ListaSocios = GestorSQL.LeerDatosSocio();
        this.club.ListaSocios.Sort((Socio s1, Socio s2) => string
        this.lstPersonas.DataSource = null;
        this.lstPersonas.DataSource = club.ListaSocios;
    }
    catch (ExceptionSQL ex)
    {
        MessageBox.Show(ex.Message, "Error al ingresar al club"
```

Usando un serializador en *frmInicio* para leer datos de socios desde un archivo xml y para exportar los datos del club a un archivo xml.

```
/// <summary>
/// Al cargar el formulario se busca si ya existe un archivo xml con datos de socios existentes, si lo encuentra se lee
/// y se crea un club con esa lista de socios encontrada. Si no hay archivo de datos se crea un club de cero.
/// Se actualiza la vista de la listbox.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void frmInicio_Load(object sender, EventArgs e)
{
    try
    {
        if (File.Exists($"{serializer.RutaDeEscritura()}\\dataSocios.xml"))
        {
            this.club = new Club("Club Armenio", this.serializer.Leer("dataSocios.xml"));
        }
        else
        {
            this.club = new Club("Club Armenio");
        }
        this.ActualizaListaSocios();
    }
    catch (ArchivoInvalidoException ex)
    {
        MessageBox.Show(ex.Message, "Error al leer el archivo de datos de socios", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

```
/// <summary>
/// Exporta los datos de la lista de socios a un archivo "dataSocios.xml" en una carpeta llamada XML en el escritorio.
/// Si el archivo ya existe lo sobreescribe
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 reference
private void btnExportar_Click(object sender, EventArgs e)
{
    try
    {
        this.serializer.Escribir("dataSocios.xml", this.club.ListaSocios);
        MessageBox.Show($"Datos exportados con éxito.\nEl archivo dataSocios.xml se encuentra en: {this.serializer.RutaDeEscritura()
    }
    catch (ArchivoInvalidoException ex)
    {
        MessageBox.Show(ex.Message, "Error al escribir el archivo de datos de socios", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Tema 16 y 17 – Introducción a bases de datos y SQL, Conexión a bases de datos



```
/// <returns>List<Socio></returns>
2 references
public static List<Socio> LeerDatosSocio()...
/// <summary>
/// Usando los metodos para traer socios y cuotas de la base
/// </summary>
/// <param name="club"></param>
1 reference
public static void LeerDatosClub(Club club)...
/// <summary>
/// Carga un socio en la tabla socios de la base de datos
/// </summary>
/// <param name="s"></param>
/// <exception cref="ExceptionSQL"></exception>
```

La clase *GestorSQL* tiene siete métodos para leer y escribir en la base de datos SQL del programa, con ella se maneja el alta, modificación o eliminación de socios, como el registro de sus cuotas pagas. Después de cada interacción en el programa la base de datos es actualizada para no perder información.

```
4 references
private void ActualizaListaSocios()
{
    try
    {
        this.club.ListaSocios = GestorSQL.LeerDatosSocio();
        this.club.ListaSocios.Sort((Socio s1, Socio s2) => string
        this.lstPersonas.DataSource = null;
        this.lstPersonas.DataSource = club.ListaSocios;
    }
    catch (ExceptionSQL ex)
    {
        MessageBox.Show(ex.Message, "Error al ingresar al club"
```

```
1 reference
private void btnAceptar_Click(object sender, EventArgs e)
{
    try
    {
        this.ValidarCampos();
        this.ValidarDniExistente(int.Parse(this.txtDni.Text));

        string nombre = this.txtNombre.Text;
        string apellido = this.txtApellido.Text;
        int dni = int.Parse(this.txtDni.Text);
        DateTime fechaNacimiento = this.dtpFechaNacimiento.Value;
        Socio.ECategoria categoria = (Socio.ECategoria)this.cmbEnum1.SelectedItem;
        Socio.EActividad actividad = (Socio.EActividad)this.cmbEnum2.SelectedItem;

        Socio socio = new Socio(nombre, apellido, dni, fechaNacimiento, actividad, categoria);
        string mensaje = string.Empty;

        if (this.IngresarSocio(this.club, socio))
        {
            GestorSQL.AltaSocio(socio);
            MessageBox.Show("Socio agregado con éxito", "Informacion", MessageBoxButtons.OK, MessageBoxIcon.Information);
            this.Close();
        }
    }
    catch (DniInvalidoException ex)
```

Tema 18, 19 y 29 – Delegados y expresiones lambda, Programación multi-hilo y concurrencia, Eventos

Usando expresiones lambda como parámetro del método Sort() se ordenan las listas de socios y de cuotas.

```
vate void ActualizaListaSocios()

try
{
    this.club.ListaSocios = GestorSQL.LeerDatosSocio();
    this.club.ListaSocios.Sort((Socio s1, Socio s2) => string.Compare(s1.Nombre, s2.Nombre));
    this.lstPersonas.DataSource = null;
    this.lstPersonas.DataSource = club.ListaSocios;
}
```

Y usando delegados, eventos e hilos se realiza la carga de la base de datos, todo se puede ver en el código del *frmInicio*.

En las líneas 25-29 declaro mis delegados y creo 2 eventos, uno que informará el proceso de la carga de la base y el otro que informará cuando ya se haya cargado la base de datos.

```
25      public delegate void CargandoBaseDeDatos(int tiempo);
26      public delegate void FinCargaBaseDeDatos();
27
28      public event CargandoBaseDeDatos InformeCargando;
29      public event FinCargaBaseDeDatos FinDeCarga;
```

En el evento click del botón Leer SQL, se subscriben los métodos a los eventos, y en un hilo separado se lanza el evento para iniciar la carga de la base de datos.

```
223     private void btnLeerSql_Click(object sender, EventArgs e)
224     {
225         this.InformeCargando += CargandoBase;
226         this.FinDeCarga += FinDeCargaBase;
227
228         Task task = Task.Run(IniciarCarga);
229     }
```

Luego en las líneas 259 en adelante se pueden ver los métodos mencionados e invocados. Primero el de *IniciarCarga()* que define el tiempo de espera, y verificando que *InformeCargando()* tenga algún subscriptor, lo invoca pasando el tiempo como parámetro. Aquí el método *CargandoBase()* es el que actualiza el texto del label informando el tiempo faltante, y se va actualizando hasta que el tiempo se acabe. Ahí de vuelta en *IniciarCarga() FinDeCarga* invoca al método que informa que la base de datos fue cargada actualizando el label.

Tema 21 – Métodos de extensión

Con el método *CantidadDeDigitos()* de la clase extendida, puedo obtener la cantidad de dígitos de un entero, eso se usa en la validación del ingreso del dni de los socios.

```
public static class ClaseExtendida
{
    /// <summary>
    /// Metodo que extiende la clase int, retorna la cantidad de digitos de un entero
    /// </summary>
    /// <param name="numero"></param>
    /// <returns>un entero, la cantidad de digitos del numero</returns>

    public static int CantidadDeDigitos(this int numero)
    {
        return numero.ToString().Length;
    }
}
```

```
        throw new DniInvalidoException("El DNI debe ser un número entre 0 y 99.999.999");
    }
    else if (int.Parse(this.txtDni.Text).CantidadDeDigitos() < 7 || int.Parse(this.txtDni.Text).CantidadDeDigitos() > 8)
    {
        throw new CantidadDeDigitosException("El DNI debe tener 7 u 8 dígitos");
```