TP 4 – Laboratorio 2: “Software de gestión de gimnasios”

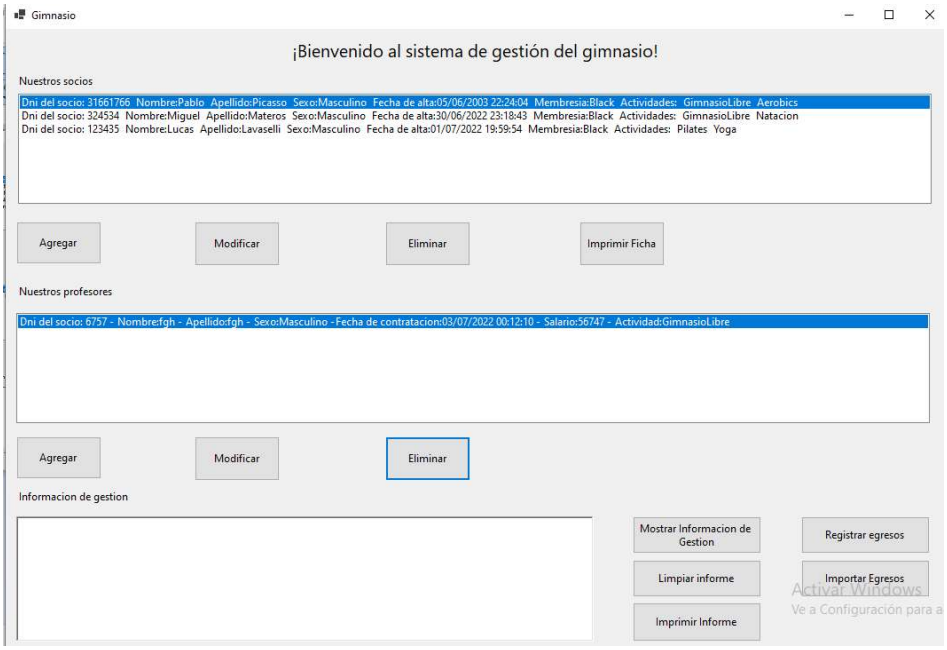
Aclaración importante

Para que la aplicación funcione correctamente, al usarla por primera vez se debe compilar para que los archivo “MockSocio.xml” y “SerializandoJson_listaEgresos.json” se copien en la carpeta de salida (en mi pc es la ruta “C:\Users\Marcos\Desktop\UTN tps\TP4\Zalazar.Marcos.2E.TP4\Vista\bin\Debug\net5.0-windows).

También es necesario ejecutar el script “GimnasioScript.sql” o restaurar la base de datos “Gimnasio_DB.bak, ambos archivos incluidos en el repositorio de GIthub.

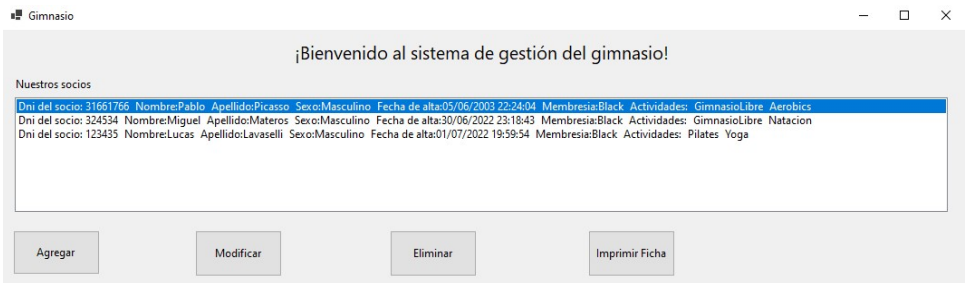
Modalidad de uso

Al ejecutar el programa, se abre la siguiente pantalla:

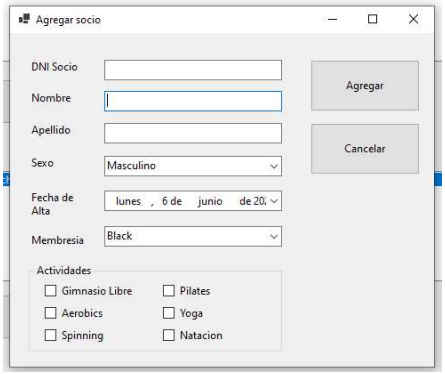


Como podemos ver, la pantalla principal está dividida en 3 secciones principales: listado de socios, listado de profesores e información de gestión.

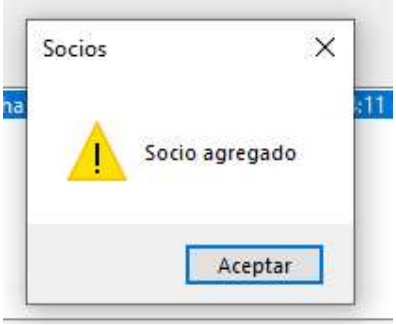
Listado de socios



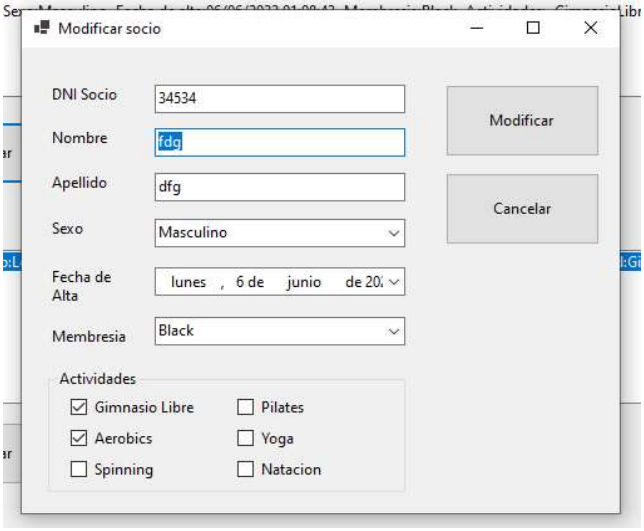
Esta sección tiene cuatro funcionalidades para gestionar los datos de los socios del club. Al desplegar la opción, se abre la siguiente pantalla:



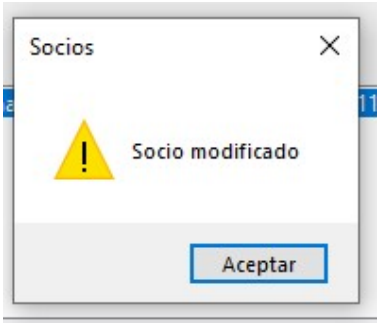
En ella, se pueden ingresar los datos necesarios para registrar al socio. Los campos que son completados por el usuario poseen validaciones que impiden que ingrese datos incorrectos, como, por ejemplo, ingresar letras en el campo DNI Socio o números en los campos de nombre y apellido. Tras oprimir el botón “aceptar”, el usuario recibirá una confirmación que el socio fue cargado con éxito. En caso de que lo desee, previo a realizar la carga puede cancelar la operación oprimiendo el botón correspondiente.



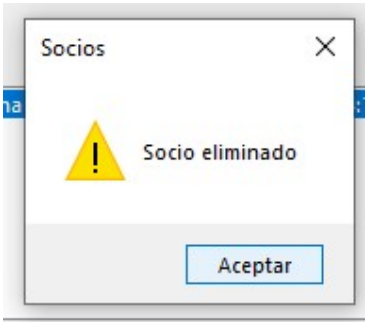
Si el usuario desea modificar los datos de un socio, debe seleccionar un socio del listado. Al hacerlo, una ventana similar a la anterior se va a desplegar, cargando los datos previamente registrados.



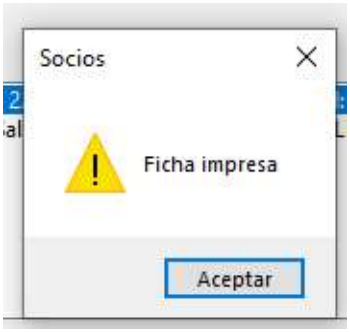
Luego de pulsar el botón “modificar”, los datos modificados podrán ser consultados en la lista.



Una situación similar se da al eliminar un usuario:



Finalmente, en caso de que se deseen consultar los datos de un usuario se podrá generar un archivo de su ficha de socio.



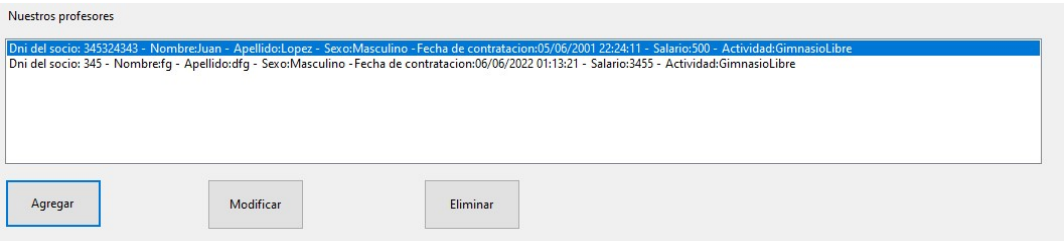
Ficha de PabloPicasso.txt: Bloc de notas

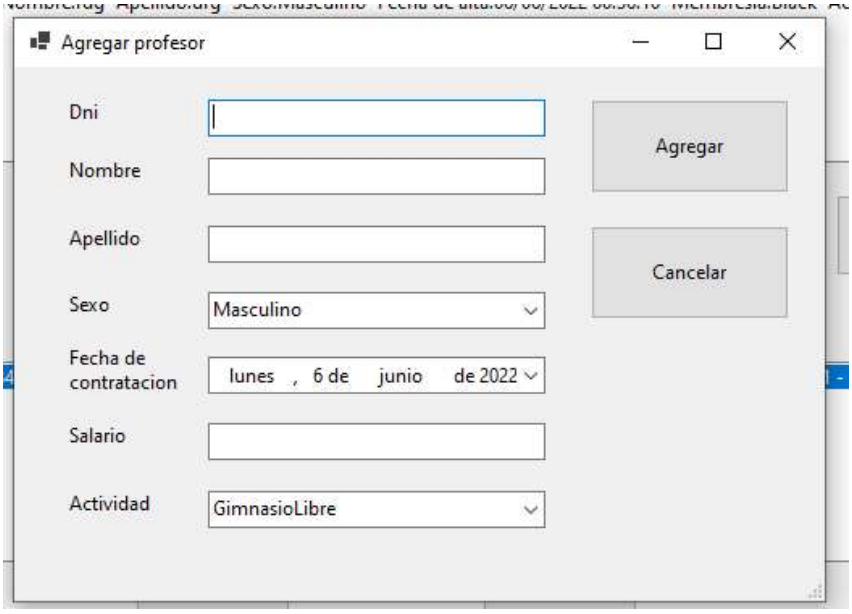
Archivo Edición Formato Ver Ayuda

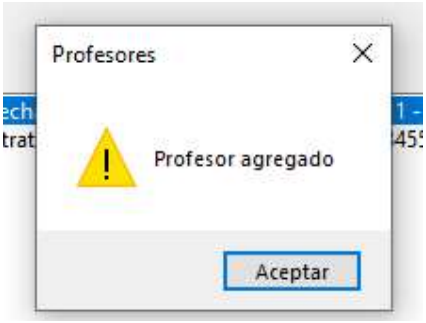
```
FICHA DEL SOCIO
-------------------

Dni del socio: 31661766
Nombre:Pablo
Apellido:Picasso
Sexo:Masculino
Fecha de alta:05/06/2003 22:24:0
Membresia:Black
Actividades:
 GimnasioLibre
 Aerobics

-------------------
```

Listado de profesores

La funcionalidad de esta sección es similar a la anterior contando con tres operaciones: alta, baja y modificación de los profesores contratados.



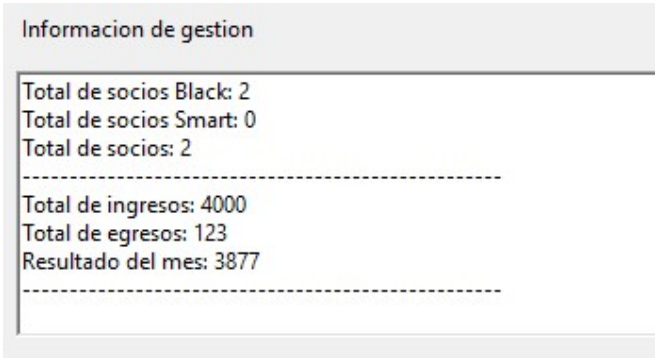




Para evitar repeticiones, no se brindarán detalles sobre su uso, remitiendo al lector a la sección anterior.

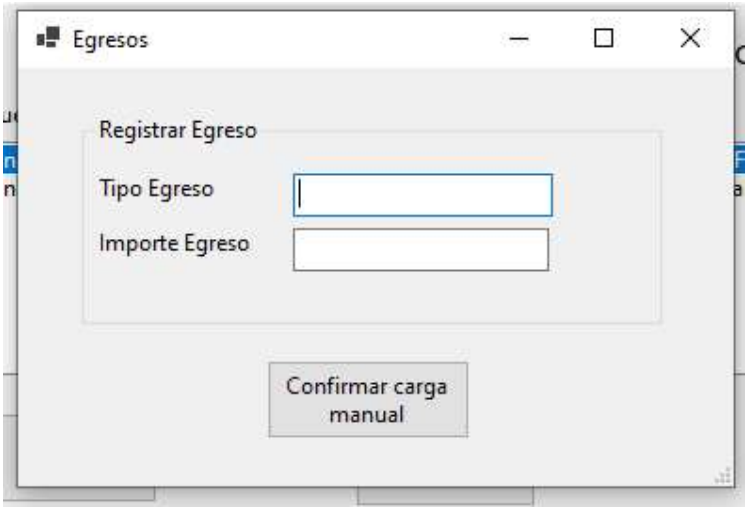
Información de gestión

Esta sección informa sobre los datos de gestión más relevantes. Al oprimir el botón “mostrar información de gestión” se despliegan los principales indicadores del negocio.

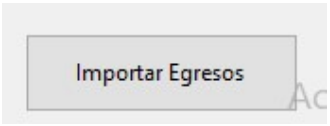


El sistema calcula la cantidad de socios que tiene el gimnasio, clasificados por su membresía. También informa el total de ingresos, en función de las cuotas cobradas a sus socios. En el caso de los egresos, se importar los gastos traídos de otro sistema en formato json mediante el botón “importar egresos”. En caso de que se desee agregar algún gasto adicional a este json, se utilizará la opción “registrar egreso”. En caso de que se desee modificar algún egreso, en forma manual se debe editar el archivo json antes de importarlo.

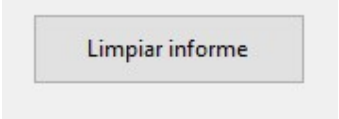
Al oprimir el primer botón, se despliega la siguiente pantalla



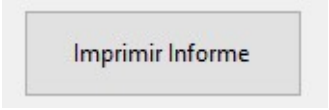
En el campo tipo de egreso se ingresa el concepto del egreso, por ejemplo, luz, gas, teléfono. En el otro campo disponible, el importe del egreso. Tras confirmar la carga el sistema informa que la operación fue realizada con éxito. Al hacerlo, se guarda en un archivo json esta información. En un mes posterior, el usuario podrá optar por importar estos gastos registrados en el archivo json, oprimiendo el botón “importar egresos”.



Luego de consultada la información, se podrá borrar el informe en pantalla tocando el botón “Limpiar informe”.



También podrá guardarse este informe en un archivo txt oprimiendo el botón “Imprimir Informe”



Conceptos aplicados

- **Excepciones**: Se creo la clase “CargaFormException” y “PersonaException” para prevenir problemas en la carga del formulario:

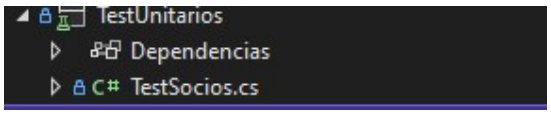
```
namespace GimnasioException
{
    13 referencias
    public class CargaFormException : Exception
    {
        0 referencias
        public CargaFormException()
        {

        }

        10 referencias
        public CargaFormException(string message) : base(message)
        {

        }
```

```
13 referencias
public class PersonaException : Exception
{
    0 referencias
    public PersonaException()
    {

    }

    4 referencias
    public PersonaException(string message) : base(message)
    {

    }
```

En distintas partes del código se utilizaron bloques try-catch para atrapar posibles excepciones.

```
try
{
    if (!Directory.Exists(path))
    {
        Directory.CreateDirectory(nombreFile);
    }

    using (StreamWriter streamWriter = new StreamWriter(nombreArchivo))
    {
        XmlSerializer xmlSerializer = new XmlSerializer(typeof(T));
        xmlSerializer.Serialize(streamWriter, datos);
    }

}
catch (Exception e)
{
    throw new Exception($"Error en el archivo ubicado en {path}", e);
}
```

- **<u>Pruebas unitarias</u>**: Se realizaron algunos test unitarios de la clase socios.



- **<u>Tipos genéricos</u>**: Utilizada principalmente en las clases serializadoras de XML y Json, y también la clase “Gestión”.

```
namespace Serializacion
{
    12 referencias
    public static class ClaseSerializadora<T>
    {
        static string path;
        0 referencias
```

```
3 referencias
public class ClaseSerializadoraJson<T>
{
    static string path;
```

```
namespace EntidadesNegocio
{
    public class Gestion<T, U> where T : class where U : class
    {
        private List<T> egresos;
        private List<U> ingresos;
```

- **Interfaces**: Se creo la interface “IValidadorCampos” implementada por los formularios de alta y modificación de socios y profesores.

```
2 referencias
public interface IValidadorCampos
{
    /// <summary>
    /// Verifica que un campo determinado no esté vacío
    /// </summary>
    /// <param name="campo"> campo a evaluar</param>
    /// <returns></returns>
    9 referencias
    public bool ElCampoNoEstaVacio(string campo);

    /// <summary>
    /// Valida que se haya elegido un valor para el campo sexo
    /// </summary>
    /// <returns></returns>
    4 referencias
    public bool ValidarEleccionSexo();
```

```
6 referencias
public partial class FrmAgregarModificarProfesores : Form, IValidadorCampos
{
    private Profesor profesor;
```

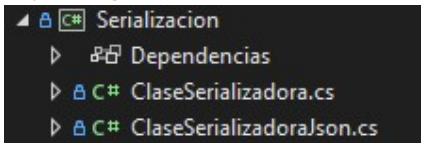
```
namespace Vista
{
    6 referencias
    public partial class FrmAgregarModificarSocio : Form, IValidadorCampos
    {
        private Socio socio;

        2 referencias
```

- **Archivos**: Se crea un txt con la ficha del socio en el método imprimirFicha y se imprime un informe de egresos en txt con el método imprimirInforme.

```
private void btnImprimirFicha_Click(object sender, EventArgs e)
{
    Socio socioSeleccionado = (Socio)this.lstSocios.SelectedItem;

    if (socioSeleccionado is not null)
    {
        string contenido = this.gimnasio.ImprimirFichaDelSocio(socioSeleccionado);
        string nombreFicha = "Ficha de "+socioSeleccionado.Nombre +" "+ socioSeleccionado.Apelli
        ClaseSerializadora<string>.EscribirEnTxt(nombreFicha, contenido);
        MessageBox.Show("Ficha impresa", "Socios", MessageBoxButtons.OK, MessageBoxIcon.Exclamat
    }
```

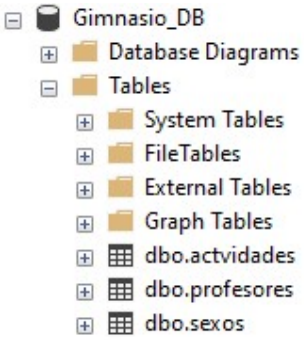
```
private void btnImprimirInforme_Click(object sender, EventArgs e)
{
    try
    {
        Task tareaDeImpresion = Task.Run(() =>
        {
            string contenido = this.gimnasio.InformacionGestion();
            string nombreInforme = $"Informe de gestión de fecha {DateTime.Now.ToString("dd-mm-yy
            MessageBox.Show("Este podía demorar algunos segundos.Aguarde mientras se imprime el i
            Thread.Sleep(4000);
            ClaseSerializadora<string>.EscribirEnTxt(nombreInforme, contenido);
            MessageBox.Show("Informe de gestión impreso", "Informe de gestión", MessageBoxButtons
        });
```

- **Serialización**: se implementaron dos clases serializadoras de archivos XML y Json. Se serializa y deserealiza a XML el objeto socios. Se serializan y deserealizan a JSON los objetos egresos



- **Introducción a SQL y conexión a bases de datos**: se creo la clase GestorSQL para poder gestionar la conexión con la base de datos "Gimnasio_DB" en donde se guarda información de la clase profesores.

```
10 referencias
public class GestorSQL
{
    private static string cadenaConexion;
    0 referencias
    static GestorSQL()
    {
        GestorSQL.cadenaConexion = "Server=.;Database=Gimnasio_DB;Trusted_Connection=True;";
    }

    1 referencia
    public static List<Profesor> LeerDatosProfesor()
    {
        List<Profesor> listaProfesoresDB = new List<Profesor>();

        string query = "select * from profesores";
        using (SqlConnection connection = new SqlConnection(GestorSQL.cadenaConexion))
```



- **Delegados, expresiones lambda, hilos y eventos**: En la clase gimnasio se creo el delegado NofiticadorNecesidadDeProfesores y el evento NotificacionEnviada. El método EvaluarNecesidadDeProfesores compara la cantidad de socios activos con la cantidad de profesores y en caso de que el número de profesores sea insuficiente, lanza el evento NotificacionEnviada. El método InformarNecesidadDeProfesores será el manejador de este evento, informando la necesidad de profesores mediante una etiqueta roja en el formulario. Cuando se contraten profesores o se reduzca el número de alumnos, esta etiqueta desaparecerá.

```
public class Gimnasio
{
    protected string nombre;
    public List<Profesor> listaProfesores;
    public List<Socio> listaSocios;
    public Gestion<Egreso,Ingreso> periodoComercial;
    public delegate void NotificadorNecesidadDeProfesores(string mensaje);
    public event NotificadorNecesidadDeProfesores NotificacionEnviada;
```

```
public void EvaluarNecesidadDeProfesores()
{
    Task nuevaTarea = Task.Run(() =>
    {
        int cantidadSocios = ContarSociosBlack() + ContarSociosSmart();
        int cantidadProfesores = 0;

        foreach (Profesor profesor in this.listaProfesores)
        {
            if (profesor.ProfesorActivo == true)
            {
                cantidadProfesores++;
            }
        }
        if (cantidadSocios > (cantidadProfesores * 3))
        {
            NotificacionEnviada?.Invoke("Cantidad de profesores insuficiente. Para m
        }
```

```
    this.primerArchivoSocio=AppDomain.CurrentDomain.BaseDirectory + "Moc
    this.gimnasio.NotificacionEnviada += InformarNecesidadDeProfesores;
}
```

```
private void InformarNecesidadDeProfesores(string mensajeRecordatorio)
{
    if (lblRecordatorioProfesores.InvokeRequired)
    {
        Action<string> delegado = InformarNecesidadDeProfesores;
        object [] parametro= new object[] {mensajeRecordatorio};
        lblRecordatorioProfesores.Invoke(delegado, parametro);
    }
    else
    {
        Thread.Sleep(2000);
        lblRecordatorioProfesores.Visible = true;
        lblRecordatorioProfesores.Text = mensajeRecordatorio;
    }
}
```

Adicionalmente, en otro hilo secundario también se realiza la impresión del informe de gestión para que el usuario pueda continuar con otras tareas sin esperar hasta la conclusión de esta tarea.

```
1 referencia
private void btnImprimirInforme_Click(object sender, EventArgs e)
{
    try
    {
        Task tareaDeImpresion = Task.Run(() =>
        {
            string contenido = this.gimnasio.InformacionGestion();
            string nombreInforme = $"Informe de gestión de fecha {DateTime.Now.T
            MessageBox.Show("Este podia demorar algunos segundos.Aguarde mientra
            Thread.Sleep(4000);
```

- **<u>Métodos de extensión</u>**: se creo la clase StringExtendido que extiende a la clase String

```
public static class StringExtendido
{
    /// <summary>
    /// Método que valida si el número ingresado es un dni válido
    /// </summary>
    /// <param name="dniAValidar"> Dni a validar</param>
    /// <returns></returns>

    public static bool ValidarDni(this string dniAValidar)
    {
        int auxDniInt = int.Parse(dniAValidar);

        if ((auxDniInt > 0 && auxDniInt <= 99999999) && (dniAValidar.Length == 7 || dniAVal
        {
            return true;
        }
        else
        {
            throw new CargaFormException("Dni inválido, verifique la carga");
```