Instructivo de uso de la Aplicación

La aplicación tiene por objetivo simular un servicio de traumatología en un Hospital y generar una estadística de las cirugías realizadas y de la patología y procedimiento más prevalente.

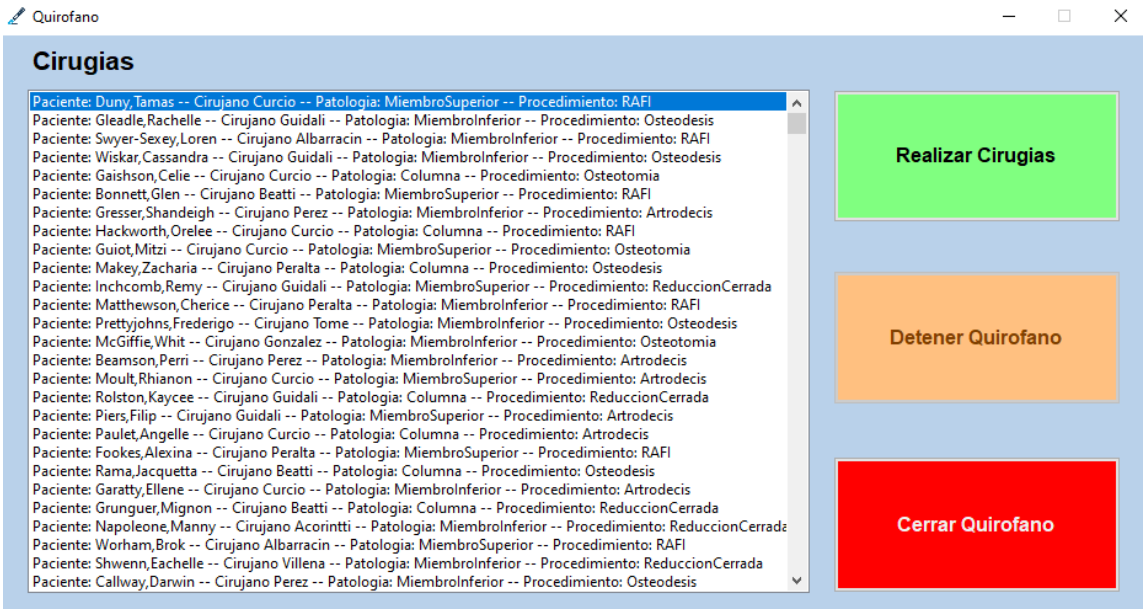
Desde el Menú Pacientes se puede cargar una nueva patología a un paciente ya existente o ingresar un nuevo paciente.

Desde el Menú Cirujano se puede cargar una nueva cirugía a un cirujano o agregar un nuevo cirujano.

Desde Quirófano→ se abre una simulación de quirófano que realiza las cirugías, elimina del listado de cirugías pendientes la cirugía realizada y actualiza la Estadística del Hospital.

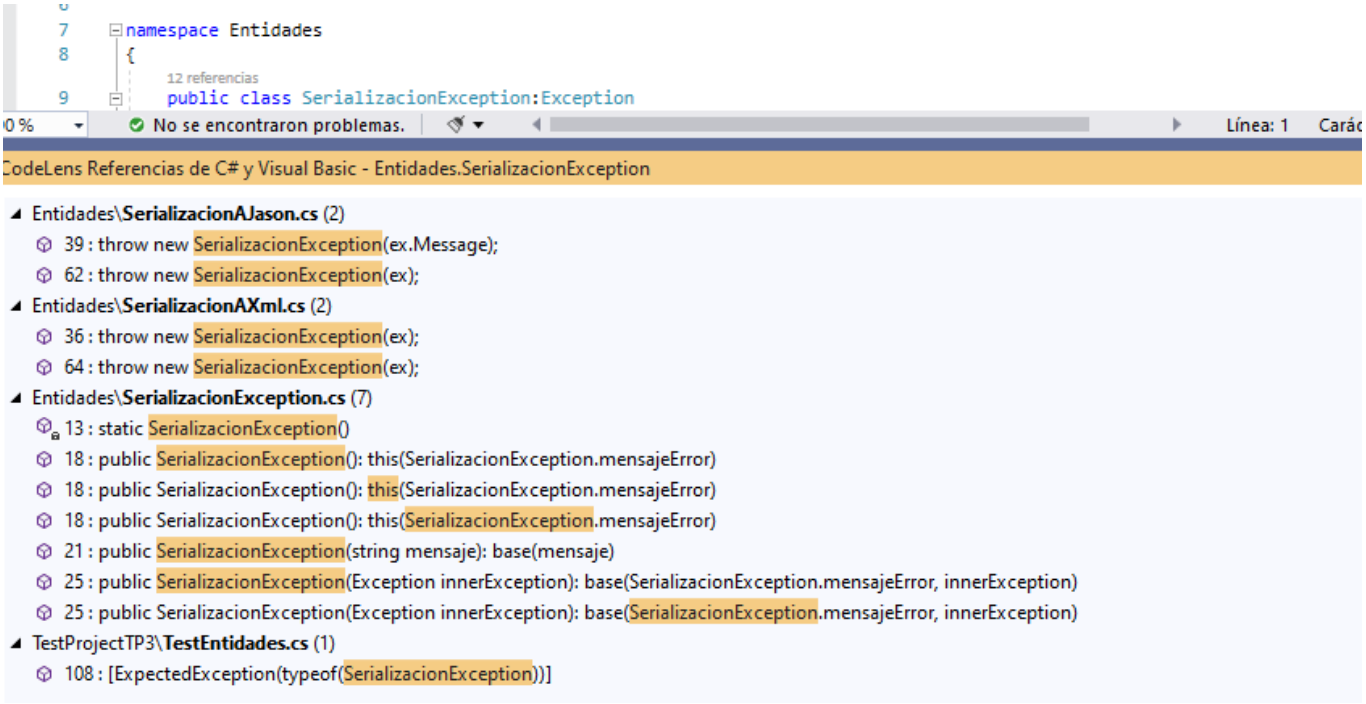
Desde →Ver Listado de Cirugías Realizadas→Acceder al listado de cirugías total y aplicar filtros para generar archivos y ver por Ejemplo las Cirugías realizadas por “x” Patología.

Desde →Estadísticas del Hospital se muestran él % de cirugías de cada cirujano, pacientes mayores de edad y patología y procedimiento más prevalentes.



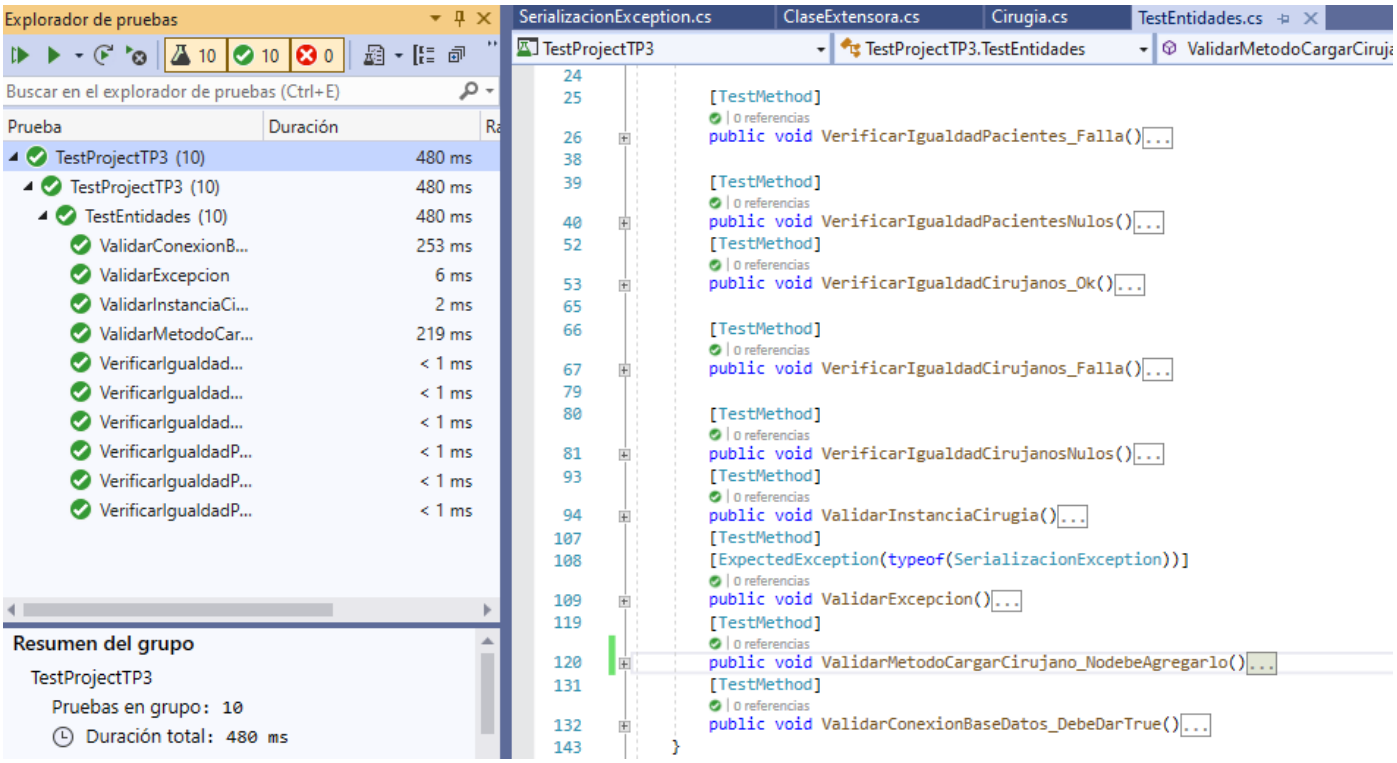
Excepciones

En biblioteca de clases ENTIDADES hay una excepción propia que se dispara cuando hay algún problema con la serialización.



Pruebas unitarias

Carpeta TestProjectTP3 se realizan pruebas donde se evalúa la correcta instancia de los objetos y la conexión a base de datos



Generics

En ENTIDADES SerializarAJson y SerializarAXml los métodos son Genéricos para poder serializar cualquier tipo de objeto.

```
22      public static void SerializarAJason<T>(string ruta, T obj) where T : class
23      {
24          try
25          {
26              if(obj is null)
27              {
28                  throw new Exception("objeto nulo");
29              }
30              JsonSerializerOptions jsonSerializerOptions = new JsonSerializerOptions { Converters = { new JsonStringEnumConverter() } };
31              jsonSerializerOptions.WriteIndented = true;
32
33              string objetoJson = JsonSerializer.Serialize(obj, jsonSerializerOptions);
34
35              File.WriteAllText(ruta, objetoJson);
36          }
37          catch(Exception ex)
38          {
39              throw new SerializacionException(ex.Message);
40          }
41      }
42      /// <summary>
43      /// Deserealiza un archivo Json
```

100 % No se encontraron problemas. Línea: 1 Carácter: 1

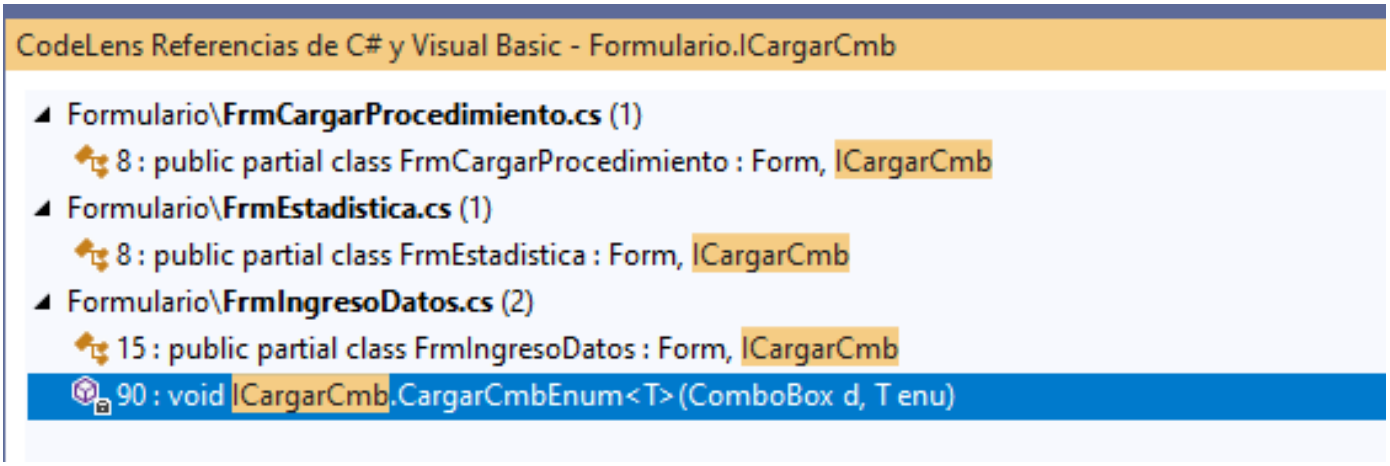
CodeLens Referencias de C# y Visual Basic - SerializacionAJason.SerializarAJason

- Entidades**Hospital.cs** (3)
- Formulario**FrmMostrarEstadistica.cs** (1)
  - 81 : SerializacionAJason.SerializarAJason(ruta, cirugias);
- TestProjectTP3**TestEntidades.cs** (1)

Interfaces

En Formularios se aplica la interfaz genérica para actualizar los comboBox de los distintos formularios

```
namespace Formulario
{
    3 referencias
    public interface ICargarCmb
    {
        6 referencias
        void CargarCmbLista<T>(ComboBox d, List<T> lista)
            where T : class;
        6 referencias
        void CargarCmbEnum<T>(ComboBox d, T enu)
            where T : Type;
    }
}
```



CodeLens Referencias de C# y Visual Basic - Formulario.ICargarCmb

- Formulario**FrmCargarProcedimiento.cs** (1)
  - 8 : public partial class FrmCargarProcedimiento : Form, ICargarCmb
- Formulario**FrmEstadistica.cs** (1)
  - 8 : public partial class FrmEstadistica : Form, ICargarCmb
- Formulario**FrmIngresoDatos.cs** (2)
  - 15 : public partial class FrmIngresoDatos : Form, ICargarCmb
  - 90 : void ICargarCmb.CargarCmbEnum<T>(ComboBox d, T enu)

Archivos y Serialización

Archivo de texto se genera un .txt con la estadística del Hospital desde el FrmEstadisticaHospital con Fecha.

```
1 referencia
31  public static void EscribirNuevoTxt(string ruta, string data)
32  {
33      try
34      {
35          using (StreamWriter streamWriter = new StreamWriter(ruta, false))
36          {
37              streamWriter.WriteLine(data);
38          }
39      }
40      catch (Exception)
41      {
42          throw;
43      }
44  }
1 referencia
```

00 % No se encontraron problemas.

CodeLens Referencias de C# y Visual Basic - Archivo.EscribirNuevoTxt

Formulario**FrmEstadisticaHospital.cs** (1)

35 : Archivo.EscribirNuevoTxt(ruta, this.rchInfoHospital.Text);

Serialización a Json y Xml: Las cirugías pendientes se cargan desde un Archivo Json.

En el FrmMostrarEstadistica se puede exportar a Json o Xml las cirugías que aparecen listadas.

Entidades**Hospital.cs** (3)

145 : SerializacionAJason.SerializarAJason(ruta, pacientes);

153 : SerializacionAJason.SerializarAJason(ruta, cirugiasPendientes);

161 : SerializacionAJason.SerializarAJason(ruta, cirujanos);

Formulario**FrmMostrarEstadistica.cs** (1)

81 : SerializacionAJason.SerializarAJason(ruta, cirugias);

TestProjectTP3**TestEntidades.cs** (1)

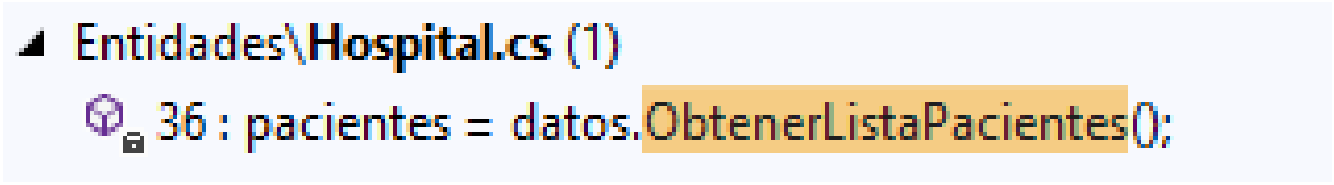
115 : SerializacionAJason.SerializarAJason(ruta, algo);

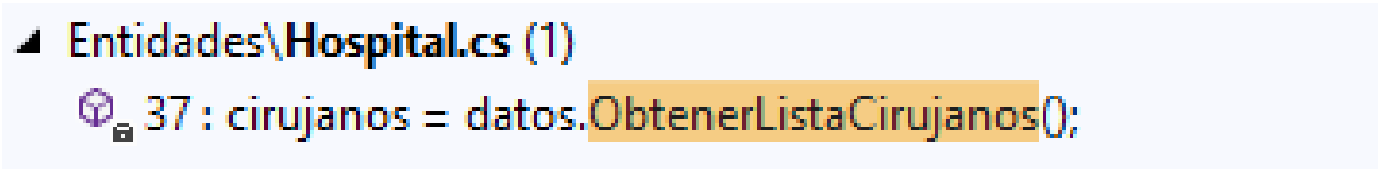
Formulario**FrmMostrarEstadistica.cs** (1)

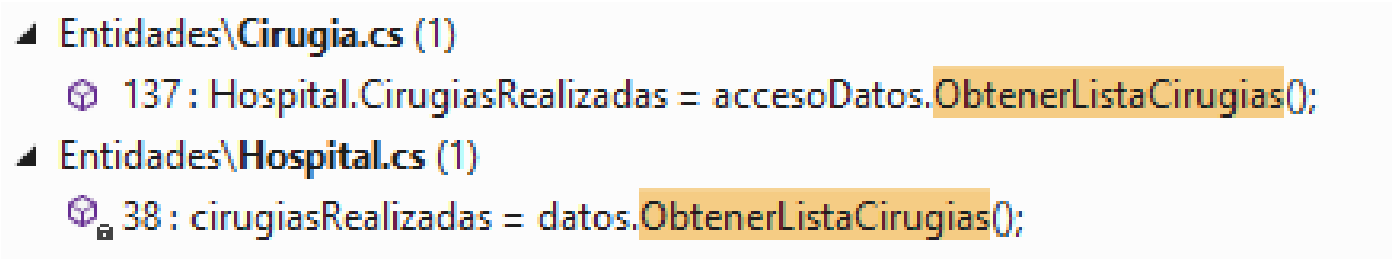
98 : SerializacionAXml<List<Cirugia>>.SerializarAXmlLista(ruta, cirugias);

Base de datos

En la Biblioteca de Clases ENTIDADES la clase AccesoDatos maneja la conexión con la base de datos del Hospital (TpFinalCurcioOrnela) y al ejecutarse el programa trae: Las cirugías realizadas, los pacientes y los cirujanos que se encuentran en la base de datos.







Hilos

El quirófano corre en un hilo secundario.

```
private void btnRealizarCirugias_Click(object sender, EventArgs e)
{
    try
    {
        Task.Run(() => ActualizarLista(Hospital.Cts.Token));
        btnDetenerQuirofano.Enabled = true;
        btnRealizarCirugias.Enabled = false;
    }
    catch(Exception ex)
    {
        ex.MostrarMensajeError();
    }
}

private void ActualizarLista(CancellationToken cts)
{
    foreach (Cirugia item in cirugias)
    {
        if (cts.IsCancellationRequested)
            return;

        if (item.Paciente.Patologia is not null)
        {
            item.RealizarOperacion();
        }

        if (this.lstPacientes.InvokeRequired)
        {
            this.lstPacientes.BeginInvoke((MethodInvoker)delegate ()
            {
                lstPacientes.DataSource = null;
                lstPacientes.DataSource = Hospital.CirugiasPendientes;
            });
            Thread.Sleep(2000);
        }
        else
        {
            lstPacientes.DataSource = null;
            lstPacientes.DataSource = Hospital.CirugiasPendientes;
        }
    }
}
```

Eventos

El delegado DelegadoCirugia se encarga de Nullear la lista de patologías del paciente para que figure operado, marcar la cirugía como realizada, actualizar la estadística del hospital y la base de datos. El evento Operar se dispara en el quirófano al Realizar Cirugías.

```
public delegate void DelegadoCirugia();
53 referencias
public class Cirugia
{
    #region Atributos

    private Cirujano cirujano;
    private Paciente paciente;
    private EPatologia patologia;
    private EProcedimiento procedimiento;
    private DateTime fecha;
    bool operado;
    public event DelegadoCirugia operar;
    #endregion

    #region Constructores
    1 referencia
    public Cirugia():this(new Paciente(),DateTime.Now,new Cirujano(),EPatologia.Columna,EProcedimiento.Artrodecis)
    {
    }
    /// <summary> Constructor con parametros
    4 referencias | 1/1 pasando
    public Cirugia(Paciente paciente, DateTime fecha, Cirujano cirujano,
                   EPatologia patologia, EProcedimiento procedimiento)
    {
        this.paciente = paciente;
        this.fecha = fecha;
        this.cirujano = cirujano;
        this.patologia = patologia;
        this.procedimiento = procedimiento;
        operar += paciente.PacienteOperado;
        operar += CirugiaRealizada;
        operar += ActualizarEstadisticaHospital;
        operar += CargarEnBaseDatos;
```

Métodos de extensión

La ClaseExtensora se ubica en el Namespace de Formulario y extiende la clase Excepción para mostrar un MessageBox si ocurrió algún problema en la aplicación.

También se genera un hilo paralelo que genera un txt para luego revisar todos los erros que ocurrieron en la ejecución del programa.

```
public static class ClaseExtensora
{
    13 referencias
    public static void MostrarMensajeError(this Exception ex)
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendLine($"Error : {ex.Message}");
        sb.AppendLine(ex.StackTrace);
        MessageBox.Show(sb.ToString(), "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);
        try
        {
            Task tsk = new Task(() => ClaseExtensora.GuardarErrores(ex));
            tsk.Start();
        }
        catch (Exception exArchivoLog)
        {
            MessageBox.Show(exArchivoLog.Message, "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    1 referencia
    private static void GuardarErrores(Exception ex)
    {
        string ruta = Archivo.GenerarRuta("Errores.txt");
        StringBuilder sb = new StringBuilder();
        sb.AppendLine("Fecha y hora del error:");
        sb.AppendLine(DateTime.Now.ToString());
        sb.AppendLine("Descripcion:");
        sb.AppendLine(ex.Message);
        sb.AppendLine(ex.StackTrace);
        sb.AppendLine("-----------------------------------------------------------------------------------");
        Archivo.EscribirAgregarTxt(ruta, sb.ToString());
    }
```