Nombre completo: Álvarez, Martín Andrés
División: 2-C

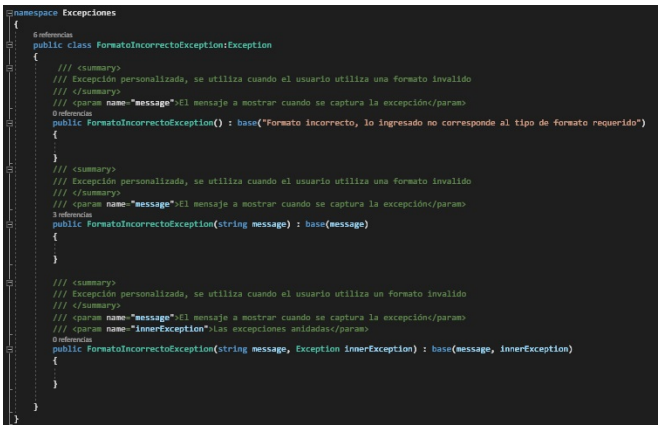
Numero de trabajo practico: 3

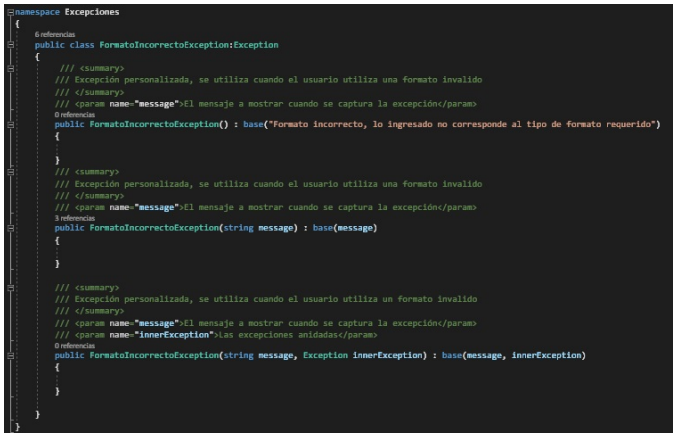
Contexto

El sistema esta pensado para un pequeño hospital rural, donde existen pacientes, médicos y atenciones. lo que se espera es que el médico pueda crear y documentar nuevas atenciones, utilizando la información necesaria de todas las entidades para brindar su diagnostico y a su vez poder compartir dicha información a un tercero. El programa no está pensado como un ABM debido a que el médico no cumple un rol administrativo, unicamente puede dar de alta atenciones y pacientes nuevos que deban ser atendidos de urgencia. El mismo está pensado para trabajar mediante SQL principalmente, pero al fallar la conexión se hará un backup de los datos y se le permitirá al usuario trabajar de manera offline mediante archivos, los cuales tendrán que ser enviados al soporte para analizarlos e ingresarlos al servidor SQL nuevamente.

Clase 10 - Excepciones

Se lanza cuando se intenta instanciar un valor incorrecto a un objeto.
Se define en la clase "FormatoIncorrectoException" de la Bibloteca "Exception"
Se la implementa en el método "VerificarInputsDelForm"(linea 115) de la clase "FormNuevoPaciente"





Clase 11 - Pruebas unitarias

1- Prueba el método Paciente.BuscarPacienteEnListaMedianteId, agregando un paciente a la lista y preguntando si está dentro de ella.
se define y prueba en la clase "MetodosPersonas" del test unitario "TestPersonas"

2- Prueba que un Paciente pueda ser serializado y deserealizado correctamente. Se define y prueba en la clase "MetodosPersonas" en método de test "SerializarUnaPersonaXml_SeEsperaCoincidienciaEnLosAtributosIdYDni"

```
/// <summary>
/// Prueba el metodo Paciente.BuscarPacienteEnListaMedianteId, agregando un paciente a la lista y preguntando si está dentro de ella.
/// </summary>
[TestMethod]
0 referencias
public void BuscarUnPacienteMedianteSuId()
{
    //ARRANGE
    Paciente nuevoPaciente = new(35353, 23, "Martín Álvarez", sexoEnum.Hombre, 1, new List<Atencion>(), false, "Ninguno");
    Paciente otroPaciente = new();
    List<Paciente> listaDePacientes = new();
    listaDePacientes.Add(nuevoPaciente);

    //ACT
    otroPaciente = Paciente.BuscarPacienteEnListaMedianteId(1, new List<Paciente>(listaDePacientes));

    //ASSERT
    Assert.AreEqual(nuevoPaciente.IdDePaciente, otroPaciente.IdDePaciente);
}
```

```
/// <summary>
/// Prueba la serialización de un Paciente a Xml, si los identificadores únicos de una Persona son iguales antes y despues de  ser serializada se aprueba el test
/// </summary>
[TestMethod]
0 referencias
public void SerializarUnaPersonaXml_SeEsperaCoincidienciaEnLosAtributosIdYDni()
{
    //ARRANGE
    Paciente nuevoPaciente = new(35353, 23, "Martín Álvarez", sexoEnum.Hombre, 1, new List<Atencion>(), false, "Ninguno");
    Paciente pacienteDeserealizado = new();

    //ACT

    using (XmlTextWriter xmlTextWriter = new XmlTextWriter($"{Environment.GetFolderPath(Environment.SpecialFolder.Desktop)}//PersonaTest.xml", Encoding.UTF8))
    {
        xmlTextWriter.Formatting = Formatting.Indented;
        XmlSerializer xmlSerializer = new XmlSerializer(typeof(Paciente));
        xmlSerializer.Serialize(xmlTextWriter, nuevoPaciente);
    }

    using (XmlTextReader xmlTextReader = new XmlTextReader($"{Environment.GetFolderPath(Environment.SpecialFolder.Desktop)}//PersonaTest.xml"))
    {
        XmlSerializer xmlSerializer = new XmlSerializer(typeof(Paciente));
        pacienteDeserealizado = (Paciente)xmlSerializer.Deserialize(xmlTextReader);
    }

    //ASSERT
    Assert.AreEqual(nuevoPaciente == pacienteDeserealizado, true);
}
```

Clase 12 - Tipos genéricos

Se la utiliza para poder exportar e importar objetos sin importar su tipo
Se la define en la clase "IArchivos" en la biblioteca "GestorArchivos"
Se la implementa en el evento "tlsmiArchivoExportarPacientes_Click"(linea 311) de la clase "FormMenuPrincipal"

```
public interface IArchivos<T>
{
    /// <summary>
    /// Interfaz utilizada para leer archivos
    /// </summary>
    /// <param name="path">La ruta donde el usuario elige donde guardar el archivo</param>
    /// <returns></returns>

    T Leer(string path);

    /// <summary>
    /// Interfaz utilizada para escribir archivos
    /// </summary>
    /// <param name="dato">El texto que se va a escribir en el archivo</param>
    /// <param name="path">La ruta donde el usuario elige donde guardar el archivo</param>
    8 referencias
    void Escribir(T dato, string path);
}
```

```
/// <summary>
/// Se serializa el campo listaPacientesSeleccionados, preguntando la ruta donde se guardará el archivo
/// </summary>
/// <param name="sender">emisor del evento</param>
/// <param name="e">Información de dicho evento</param>
1 referencia
private void tlsmiArchivoExportarPacientes_Click(object sender, EventArgs e)
{
    try
    {
        if (this.listaPacientesSeleccionados is not null)
        {
            Serializador<List<Paciente>> serializadorDePaciente = new Serializador<List<Paciente>>();
            serializadorDePaciente.EscribirPreguntandoRutaDelArhivo(this.listaPacientesSeleccionados);
        } else
            MessageBox.Show("No hay pacientes en la lista", "Error exportar paciente", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error al intentar exportar paciente", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Clase 13 - Interfaces

Se la utiliza para asegurar que las clases "ArchivosTexto" y "Serializador" implementen tipos genéricos y tengan un método para leer y escribir archivos.
Se la define en la clase "IArchivos" en la biblioteca "GestorArchivos"
Se la implementa en el evento "tlsmiArchivoExportarPacientes_Click"(linea 311) de la clase "FormMenuPrincipal"

```
public interface IArchivos<T>
{
    /// <summary>
    /// Interfaz utilizada para leer archivos
    /// </summary>
    /// <param name="path">La ruta donde el usuario elige donde guardar el archivo</param>
    /// <returns></returns>
    T Leer(string path);

    /// <summary>
    /// Interfaz utilizada para escribir archivos
    /// </summary>
    /// <param name="dato">El texto que se va a escribir en el archivo</param>
    /// <param name="path">La ruta donde el usuario elige donde guardar el archivo</param>
    8 referencias
    void Escribir(T dato, string path);
}
```

```
/// <summary>
/// Se serializa el campo listaPacientesSeleccionados, preguntando la ruta donde se guardará el archivo
/// </summary>
/// <param name="sender">emisor del evento</param>
/// <param name="e">Información de dicho evento</param>
1 referencia
private void tlsmiArchivoExportarPacientes_Click(object sender, EventArgs e)
{
    try
    {
        if (this.listaPacientesSeleccionados is not null)
        {
            Serializador<List<Paciente>> serializadorDePaciente = new Serializador<List<Paciente>>();
            serializadorDePaciente.EscribirPreguntandoRutaDelArhivo(this.listaPacientesSeleccionados);
        } else
            MessageBox.Show("No hay pacientes en la lista", "Error exportar paciente", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error al intentar exportar paciente", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Clase 14 - Archivos y serialización

Se utiliza para escribir y leer archivos xml, json, csv y txt entre otros.
Se la define en el proyecto "GestorArchivos " en la clase "Serializador"(clase encargada de serializar una entidad de tipo genérico), utilizando el método "EscribirPreguntandoRutaDelArchivo" (linea 251)
Se la implementa en el evento tlsmiArchivoExportarPacientes_Click" en el form "FormMenuPrincipal" para serializar Personas (linea 311)

```
/// <summary>
///  Escribe un objeto pasado por parametro en un archivo .xml o .json
/// </summary>
/// <param name="dato">Lo que se va a guardar en el archivo</param>
6 referencias
public void EscribirPreguntandoRutaDelArhivo(T dato)
{
    try
    {
        using (SaveFileDialog rutaDelArchivoParaAbrir = new SaveFileDialog())
        {
            if (rutaDelArchivoParaAbrir.ShowDialog() == DialogResult.OK)
            {
                Escribir(dato, rutaDelArchivoParaAbrir.FileName);
            }
        }
    }
    catch (Exception ex)
    { throw new Exception("Se encontró un problema en el metodo EscribirPreguntandoRutaDelArchivo de la clase Serializador", ex); }
}
```

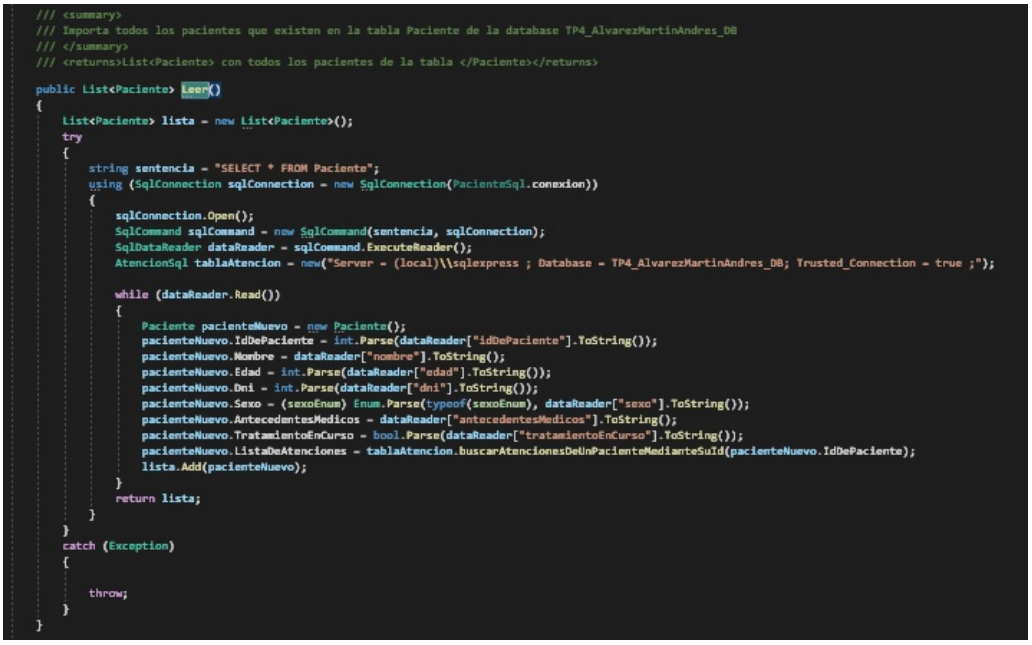
```
/// <summary>
/// Se serializa el campo listaPacientesSeleccionados, preguntando la ruta donde se guardará el archivo
/// </summary>
/// <param name="sender">emisor del evento</param>
/// <param name="e">Información de dicho evento</param>
1 referencia
private void tlsmiArchivoExportarPacientes_Click(object sender, EventArgs e)
{
    try
    {
        if (this.listaPacientesSeleccionados is not null)
        {
            Serializador<List<Paciente>> serializadorDePaciente = new Serializador<List<Paciente>>();
            serializadorDePaciente.EscribirPreguntandoRutaDelArhivo(this.listaPacientesSeleccionados);
        } else
            MessageBox.Show("No hay pacientes en la lista", "Error exportar paciente", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error al intentar exportar paciente", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Clase 15 y 16 - SQL

Se la utiliza para exportar e importar información del servidor SQL, verificar que exista una conexión al servidor o que un id sea único.

Se la define en el proyecto "GestorSql", cuenta con "MisCommandosSql" una clase que contiene instrucciones que no dependen de entidades, una interface creada para asegurarse que sea una entidad valida y tres clases con las entidades que existen en las tablas.

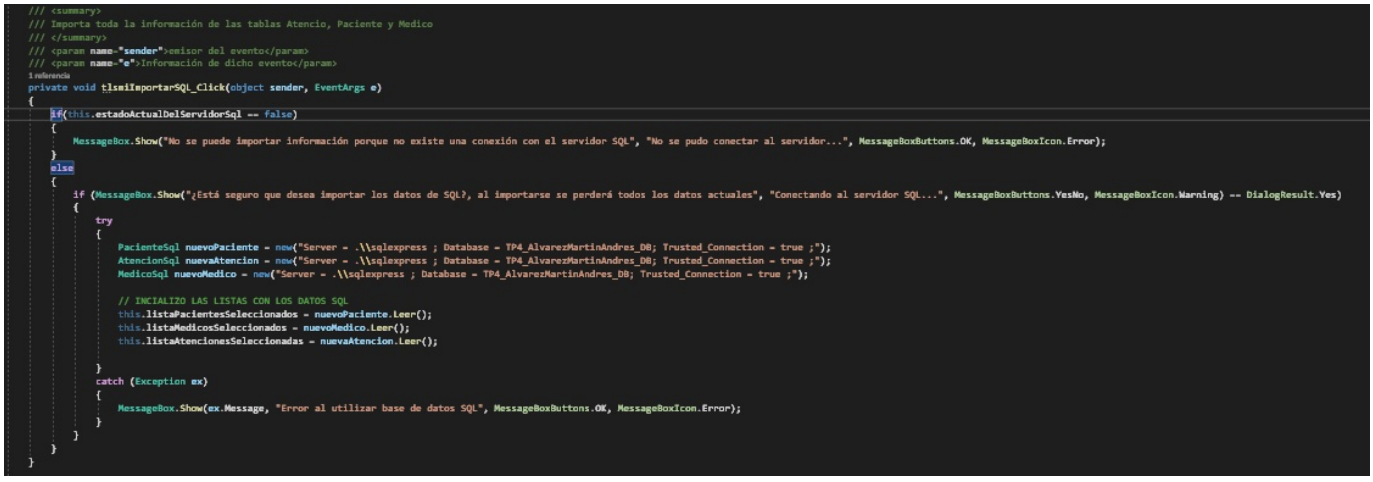
Se la implementa por ejemplo en el form "FormMenuPrincipal" en el evento "tlsmiImportarSQL_Click" (linea 733) para importar la información del servidor sql a las listas del form.



```
/// <summary>
/// Importa todos los pacientes que existen en la tabla Paciente de la database TP4_AlvarezMartinAndres_DB
/// </summary>
/// <returns>List<Paciente> con todos los pacientes de la tabla </Paciente></returns>
public List<Paciente> Leer()
{
    List<Paciente> lista = new List<Paciente>();
    try
    {
        string sentencia = "SELECT * FROM Paciente";
        using (SqlConnection sqlConnection = new SqlConnection(PacienteSql.conexion))
        {
            sqlConnection.Open();
            SqlCommand sqlCommand = new SqlCommand(sentencia, sqlConnection);
            SqlDataReader dataReader = sqlCommand.ExecuteReader();
            AtencionSql tablaAtencion = new("Server = (local)\\sqlexpress ; Database = TP4_AlvarezMartinAndres_DB; Trusted_Connection = true ;");

            while (dataReader.Read())
            {
                Paciente pacienteNuevo = new Paciente();
                pacienteNuevo.IdDePaciente = int.Parse(dataReader["idDePaciente"].ToString());
                pacienteNuevo.Nombre = dataReader["nombre"].ToString();
                pacienteNuevo.Edad = int.Parse(dataReader["edad"].ToString());
                pacienteNuevo.Dni = int.Parse(dataReader["dni"].ToString());
                pacienteNuevo.Sexo = (sexoEnum) Enum.Parse(typeof(sexoEnum), dataReader["sexo"].ToString());
                pacienteNuevo.AntecedentesMedicos = dataReader["antecedentesMedicos"].ToString();
                pacienteNuevo.TratamientoEnCurso = bool.Parse(dataReader["tratamientoEnCurso"].ToString());
                pacienteNuevo.ListaDeAtenciones = tablaAtencion.buscarAtencionesDeUnPacienteMediamteSuId(pacienteNuevo.IdDePaciente);
                lista.Add(pacienteNuevo);
            }
            return lista;
        }
    }
    catch (Exception)
    {

        throw;
    }
}
```



```
/// <summary>
/// Importa toda la información de las tablas Atencio, Paciente y Medico
/// </summary>
/// <param name="sender">emisor del evento</param>
/// <param name="e">Información de dicho evento</param>
1 referencia
private void tlsmiImportarSQL_Click(object sender, EventArgs e)
{
    if(this.estadoActualDelServidorSql == false)
    {
        MessageBox.Show("No se puede importar información porque no existe una conexión con el servidor SQL", "No se pudo conectar al servidor...", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    else
    {
        if (MessageBox.Show("¿Está seguro que desea importar los datos de SQL?, al importarse se perderá todos los datos actuales", "Conectando al servidor SQL...", MessageBoxButtons.YesNo, MessageBoxIcon.Warning) == DialogResult.Yes)
        {
            try
            {
                PacienteSql nuevoPaciente = new("Server = .\\sqlexpress ; Database = TP4_AlvarezMartinAndres_DB; Trusted_Connection = true ;");
                AtencionSql nuevaAtencion = new("Server = .\\sqlexpress ; Database = TP4_AlvarezMartinAndres_DB; Trusted_Connection = true ;");
                MedicoSql nuevoMedico = new("Server = .\\sqlexpress ; Database = TP4_AlvarezMartinAndres_DB; Trusted_Connection = true ;");

                // INICIALIZO LAS LISTAS CON LOS DATOS SQL
                this.listaPacientesSeleccionados = nuevoPaciente.Leer();
                this.listaMedicosSeleccionados = nuevoMedico.Leer();
                this.listaAtencionesSeleccionadas = nuevaAtencion.Leer();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error al utilizar base de datos SQL", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
}
```

Clase 17 - Delegados y expresiones lambda

Delegados

Se los utiliza en el form "FormMenuPrincipal" para crear el evento "ComprobadorConexionSqlHandler"(linea 22)

```
public partial class FormMenuPrincipal : Form
{
    private List<Paciente> listaPacientesSeleccionados;
    private List<Medico> listaMedicosSeleccionados;
    private List<Atencion> listaAtencionesSeleccionadas;
    private bool estadoActualDelServidorSql;
    private bool estadoAnteriorDelServidorSql;
    public delegate void ComprobadorConexionSqlHandler();
    public event ComprobadorConexionSqlHandler OnCambioEnElEstadoDelServidorSql;
    private Thread threadVerificarConexionSql;
   private CancellationTokenSource cancellationToken;
```

Expresiones lambda

Se las utiliza en el form "FormMenuPrincipal" en el método "generarListaAtencionDePacientesYMedicos" (linea 694) para combinar las atenciones de los Médicos y los Pacientes, eliminando las repetidas.

```
/// <summary>
/// Retorna una lista de atenciones nuevas, la cual es una combinación de la lista de Medicos y Pacientes, eliminando los id de atenciones repetidos
/// </summary>
/// <returns>Una list<Atencion> con todas las atenciones, null si no está instanciada la lista de pacientes o medicos</Atencion></returns>
0 referencias
public List<Atencion> generarListaAtencionDePacientesYMedicos()
{
    if (this.listaPacientesSeleccionados is not null && this.listaMedicosSeleccionados is not null)
    {
        List<Atencion> auxAtencion = new();

        //CREO LA LISTA DE ATENCIONES
        this.listaPacientesSeleccionados.ForEach((paciente) => auxAtencion.AddRange(paciente.ListaDeAtenciones));
        this.listaMedicosSeleccionados.ForEach((medico) => auxAtencion.AddRange(medico.PacientesAtendidos));

        //LA ORDENO POR ID
        auxAtencion.Sort((a, b) => a.IdDeAtencion - b.IdDeAtencion);
        //AGRUPO POR EL ID Y AGREGO UNICAMENTE EL PRIMERO DE CADA ID
        return auxAtencion = auxAtencion.GroupBy((x) => x.IdDeAtencion).Select((y) => y.First()).ToList();
    }

    return null;
}
```

Clase 18 - Hilos.

Se lo utiliza para invocar al método manejador "c_ComprobarEstadoDelServidor()"(linea 60), dicho método revisará cada tres segundos que exista una conexión en el servidor SQL y hará una acción dependiendo el estado del servidor. Dicho hilo se cerrará al finalizar la aplicación.

```
/// <summary>
/// Evento load del form, deshabilita los menuStrip dependiendo si la lista de  Paciente, Medico y Atencion
/// </summary>
/// <param name="sender">emisor del evento</param>
/// <param name="e">Información de dicho evento</param>
1 referencia
private void FormMenuPrincipal_Load(object sender, EventArgs e)
{
    this.HarcodeoAtenciones();
    this.HarcodeoPaciente();
    this.HarcodeoMedicos();
    this.OnCambioEnElEstadoDelServidorSql += this.CambiarControlesFormDependiendoEstadoDeConexion;
    this.threadVerificarConexionSql = new Thread(()=> c_ComprobarEstadoDelServidor());
    threadVerificarConexionSql.Start();
}
```

Clase 19 - Eventos

Se dispara cuando se cambia el estado de la conexión de SQL a otro diferente.
Se le suscribe el manejador "c_CambiarControlesFormDependiendoEstadoDeConexion"en el evento "FormMenuPrincipal_Load" del form "FormMenuPrincipal"
Se lo invoca en el form "FormMenuPrincipal" en el método "ComprobarEstadoDelServidor" (linea 514)

```
/// <summary>
/// Evento load del form, deshabilita los menuStrip dependiendo si la lista de  Paciente, Medico y Atenciones está vacia
/// </summary>
/// <param name="sender">emisor del evento</param>
/// <param name="e">Información de dicho evento</param>
1 referencia
private void FormMenuPrincipal_Load(object sender, EventArgs e)
{
    this.HarcodeoAtenciones();
    this.HarcodeoPaciente();
    this.HarcodeoMedicos();
    this.OnCambioEnElEstadoDelServidorSql += this.c_CambiarControlesFormDependiendoEstadoDeConexion;
    this.threadVerificarConexionSql = new Thread(()=> ComprobarEstadoDelServidor());
    threadVerificarConexionSql.Start();
}
```

```
/// <summary>
/// Modifica el form dependiendo del estado del servidor, si el servidor está offline se habilitan los archivos y muestra un groupbox notificando el error
/// </summary>
2 referencias
public void c_CambiarControlesFormDependiendoEstadoDeConexion()
{
    try
    {
        if (this.txtConexionSql.InvokeRequired)
        {
            ComprobadorConexionSqlHandler del = new ComprobadorConexionSqlHandler(this.c_CambiarControlesFormDependiendoEstadoDeConexion);
            if (this.threadVerificarConexionSql != null)
                this.txtConexionSql.Invoke(del);
        }
        else
        {
            if (this.estadoActualDelServidorSql == true)
            {
                grbSqlError.Hide();
                this.tlsmiArchivo.Available = false;
                this.estadoAnteriorDelServidorSql = this.estadoActualDelServidorSql;
            }
            if (this.estadoActualDelServidorSql == false)
            {
                grbSqlError.Show();
                this.tlsmiArchivo.Available = true;
                this.estadoAnteriorDelServidorSql = this.estadoActualDelServidorSql;
                this.CrearBackUpDeDatos();
            }
        }
    }
    catch (Exception)
    {
        throw;
    }
}
```

```
/// <summary>
/// Manejador del evento OnCambioEnElEstadoDelServidorSql, verifica cada 3 segundos que exista conexión con la data base TP4_AlvarezMartinAndres_DB
/// </summary>
1 referencia
public void ComprobarEstadoDelServidor()
{
    try
    {
        if (this.OnCambioEnElEstadoDelServidorSql is not null)
        {
            do
            {
                this.estadoActualDelServidorSql = MisComandosSql.VerificarConexionConSql("Server = .\\sqlexpress ; Database = TP4_AlvarezMartinAndres_DB; Trusted_Connection = true ;");
                if (this.estadoActualDelServidorSql != this.estadoAnteriorDelServidorSql)
                    this.OnCambioEnElEstadoDelServidorSql.Invoke();
                Thread.Sleep(3000);
            } while (threadVerificarConexionSql.IsAlive);

        }
    }catch(Exception ex)
    {
        MessageBox.Show(ex.Message, "Error al revisar conexión con el servidor", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

}
```

¿Clase 20 - Métodos extensión? (no aparece en el classroom pero si en los apuntes)

Se la utiliza en la clase "ListExtension" de la bibloteca "Extension" para Organizar una lista de atenciones, eliminando id repetidos y ordenándolos por id

```
namespace Extension
{
    0 referencias
    public static class ListExtension
    {
        1 referencia
        public static List<Atencion> OrganizarListaDeAtenciones(this List<Atencion> lista)
        {
            return  lista.GroupBy((x) => x.IdDeAtencion).Select((y) => y.First()).ToList();
        }

    }
}
```