TRABAJO PRACTICO N2 LABORATORIO

Nombre: Agustin Matías Garcia Navas.

CONTEXTO DE NEGOCIO: CENTRO DE URGENCIAS MEDICAS

La aplicación fue desarrollada para lograr atender pacientes en un centro médico. Esta consta de varias partes. En primer lugar, tenemos un formulario donde se realizará la admisión de los pacientes para ingresarlos en una sala de espera, luego el medico podrá visualizar los pacientes en tiempo real, que esperan ser atendidos. Al momento de atender deberá realizar un diagnostico el cual se estará almacenando en la historia clínica del paciente, en caso de ser la primera atención se creará. Por otra parte, hay un formulario donde mostrara todos los registros. Los médicos y pacientes registrados o importados desde un archivo se almacenarán en una base de datos, también nuestra aplicación ofrece poder exportar e importar los pacientes mediante archivos JSON.

TEMAS APLICADOS

Excepciones:

Se crearon 3 excepciones para el proyecto:

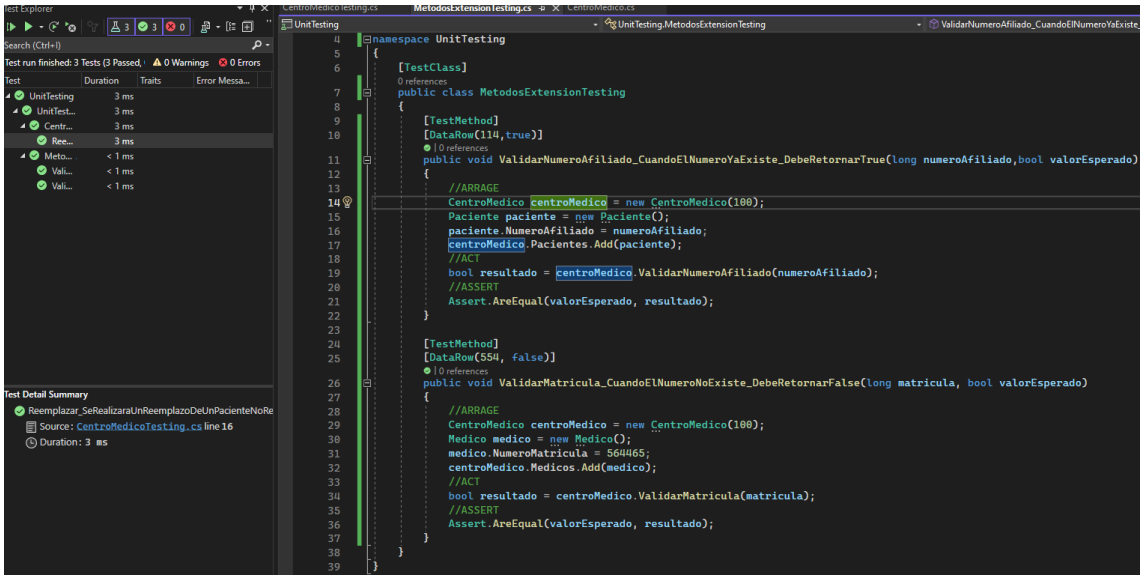
- CampoVacioException
- FalloGuardarRegistroException
- FalloModificarRegistroException

```
namespace Entidades.Excepciones
{
    9 references
    public class FalloGuardarRegistroException : Exception
    {

        2 references
        public FalloGuardarRegistroException(string msg) :base(msg)
        {
        }

        0 references
        public FalloGuardarRegistroException(string msg,Exception ex) : base(msg, ex)
        {
        }
    }
}
```

```
/// <summary>
/// Guarda un medico en la DB
/// </summary>
/// <param name="medico"></param>
/// <exception cref="FalloGuardarRegistroException"></exception>
1 reference
public static void Guardar(Medico medico)
{
    try
    {
        string sentencia = "INSERT INTO Medicos (nombre,apellido,dni,fecha_nacimiento,matricula,especialidad,fecha_modificacion)"
            "VALUES (@nombre,@apellido,@dni,@fecha_nacimiento,@matricula,@especialidad,@fecha_modificacion)";

        using (SqlConnection connection = new SqlConnection(ADOMedicos.stringConnection))
        {

            SqlCommand command = new SqlCommand(sentencia, connection);
            command.Parameters.AddWithValue("nombre", medico.Nombre);
            command.Parameters.AddWithValue("apellido", medico.Apellido);
            command.Parameters.AddWithValue("dni", medico.Dni);
            command.Parameters.AddWithValue("fecha_nacimiento", medico.FechaNacimiento);
            command.Parameters.AddWithValue("matricula", medico.NumeroMatricula);
            command.Parameters.AddWithValue("especialidad", medico.Especialidad.ToString());
            command.Parameters.AddWithValue("fecha_modificacion", medico.FechaModificacion);

            connection.Open();

            command.ExecuteNonQuery();
        }
    }
    catch
    {
        throw new FalloGuardarRegistroException("Error al guardar el medico en la DB");
    }

}
```

Pruebas Unitarias:

Realice 3 test unitarios en dos clases distintas.



Generics:

Contiene 1 clase genérica donde se manipularán los archivos JSON.

```
namespace Entidades
{
    7 references
    public class GestorArchivos<T> where T : class
    {
        private string rutaArchivo;
        private string objetoSerializado;
        private List<T> registros;

        2 references
        public GestorArchivos(string rutaArchivo)
        {
            this.rutaArchivo = rutaArchivo;
        }

        1 reference
        public GestorArchivos(string rutaArchivo, List<T> registros):this(rutaArchivo)
        {
            this.registros = registros;
        }

        0 references
        public string ObjetoDeserializado
        {
            get { return objetoSerializado; }
        }

        1 reference
        public List<T> Registros
        {
            get { return this.registros; }
        }

        /// <summary> Exporta una cadena en formato de archivo
        1 reference
        public void Exportar(string archivo)
        {
            try
```

Interfaces:

Se crearon dos interfaces, una para cuando el paciente ingresa a la guardia y otra para tener identificaciones de los registros en sql

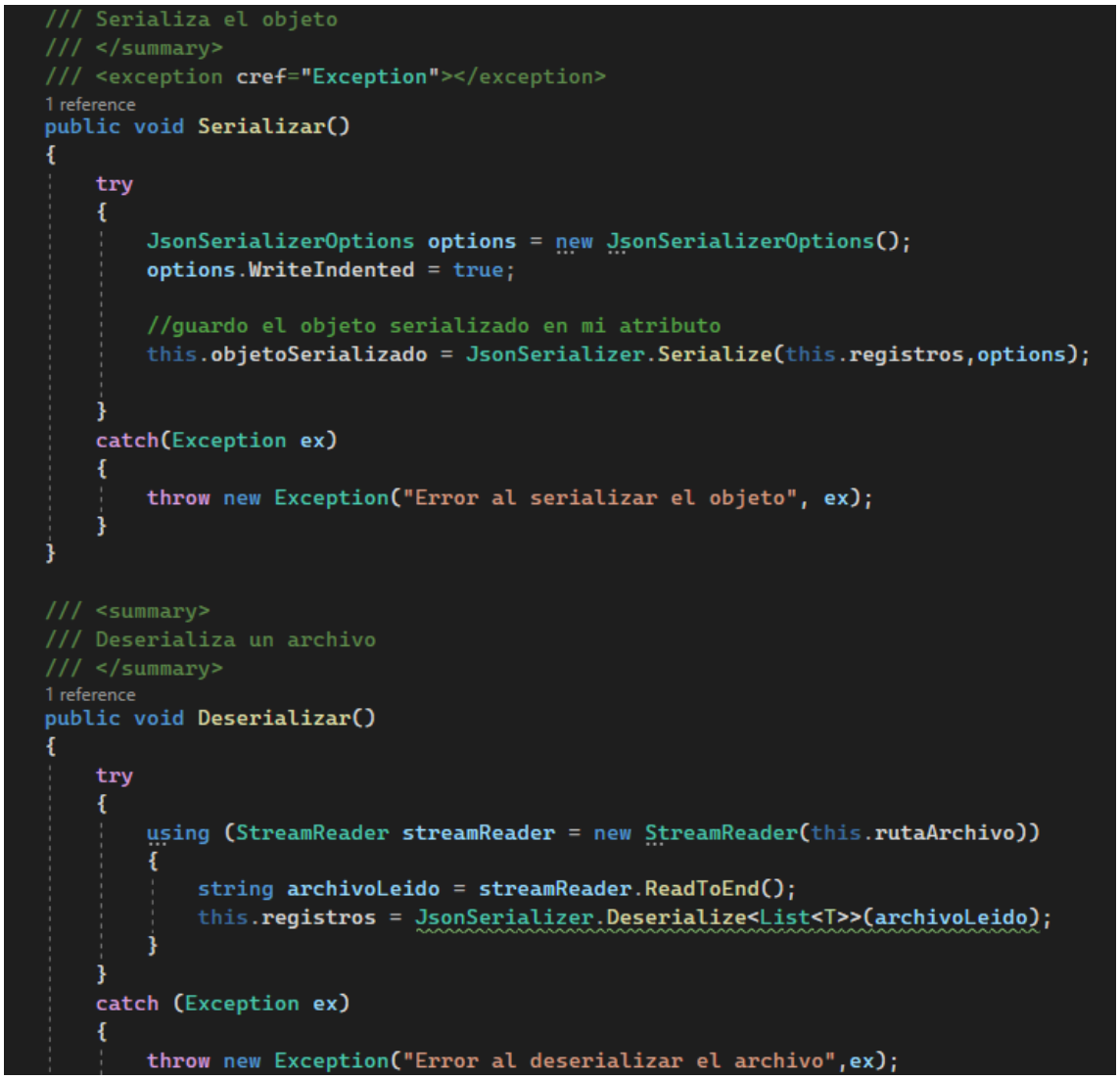
```
1 reference
public interface IGuardia
{
    14 references
    public bool EnEspera { get; set; }//sirve para consultar si el paciente se encuentra en espera
}
```

```
2 references
public interface IIdentificadorSql
{
    /// <summary>
    /// Identificara al objeto
    /// </summary>
    14 references
    public int Id { get; set; }

    /// <summary>
    /// Fecha que se crea dicho objeto
    /// </summary>
    10 references
    public DateTime FechaModificacion { get; set; }

}
```

Archivos y serialización:

Se creo una clase genérica y de instancia mostrada en el punto anterior “generics”, donde manipula los archivos tanto para importar y exportar de manera serializada.

```
namespace Entidades
{
    7 references
    public class GestorArchivos<T> where T : class
    {
        private string rutaArchivo;
        private string objetoSerializado;
        private List<T> registros;

        2 references
        public GestorArchivos(string rutaArchivo)
        {
            this.rutaArchivo = rutaArchivo;
        }

        1 reference
        public GestorArchivos(string rutaArchivo, List<T> registros):this(rutaArchivo)
        {
            this.registros = registros;
        }

        0 references
        public string ObjetoDeserializado
        {
            get { return objetoSerializado; }
        }

        1 reference
        public List<T> Registros
        {
            get { return this.registros; }
        }

        /// <summary> Exporta una cadena en formato de archivo
        1 reference
        public void Exportar(string archivo)
        {
            try
```



```
/// Serializa el objeto
/// </summary>
/// <exception cref="Exception"></exception>
1 reference
public void Serializar()
{
    try
    {
        JsonSerializerOptions options = new JsonSerializerOptions();
        options.WriteIndented = true;

        //guardo el objeto serializado en mi atributo
        this.objetoSerializado = JsonSerializer.Serialize(this.registros,options);

    }
    catch(Exception ex)
    {
        throw new Exception("Error al serializar el objeto", ex);
    }
}

/// <summary>
/// Deserializa un archivo
/// </summary>
1 reference
public void Deserializar()
{
    try
    {
        using (StreamReader streamReader = new StreamReader(this.rutaArchivo))
        {
            string archivoLeido = streamReader.ReadToEnd();
            this.registros = JsonSerializer.Deserialize<List<T>>(archivoLeido);
        }
    }
    catch (Exception ex)
    {
        throw new Exception("Error al deserializar el archivo",ex);
```

SQL:

Realice dos clases una para pacientes donde tendrá diversos métodos (eliminar, modificar, agregar, consultar) y otra para médicos.

```
public class ADOMedicos
{
    private static string stringConnection;

    0 references
    static ADOMedicos()
    {
        ADOMedicos.stringConnection = "Server = .; Database = CentroMedicoTP; Trusted_Connection = True;";

    }

    /// <summary> Obtiene todos los medicos de la DB
    1 reference
    public static List<Medico> ObtenerMedicosTotales()
    {

        try
        {
            string sentencia = "SELECT * FROM Medicos";
            List<Medico> listaMedicos = new List<Medico>();

            using (SqlConnection connection = new SqlConnection(ADOMedicos.stringConnection))
            {

                SqlCommand command = new SqlCommand(sentencia, connection);
                connection.Open();

                SqlDataReader reader = command.ExecuteReader();

                while (reader.Read())
                {
                    //ACA APLICO LA LOGICA SEGUN LAS COLUMNAS, ME FALTA CREAR LA DB
                    Medico medico = new Medico();
```

```
19 references
public static class ADOPacientes
{
    private static string stringConnection;
    private static DateTime fechaComparacion;

    0 references
    static ADOPacientes()
    {
        ADOPacientes.stringConnection = "Server = .; Database = CentroMedicoTP; Trusted_Connection = True;";
        ADOPacientes.fechaComparacion = DateTime.Now;
    }

    /// <summary>
    /// Obtiene una lista del total de pacientes almacenados en la DB
    /// </summary>
    /// <returns></returns>
    /// <exception cref="Exception"></exception>
    1 reference
    public static List<Paciente> ObtenerLista()...

    /// <summary>
    /// Obtiene pacientes que hayan sido modificados recientemente
    /// </summary>
    /// <returns></returns>
    /// <exception cref="Exception"></exception>
    1 reference
    public static List<Paciente> ObtenerModificados()
    {
        try
        {
            string sentencia = "SELECT * FROM Pacientes WHERE fecha_modificacion > @fecha_modificacion";


            using (SqlConnection connection = new SqlConnection(ADOPacientes.stringConnection))
            {
                SqlCommand command = new SqlCommand(sentencia, connection);
```

Delegados y expresiones lambda:

Desarrolle un software donde tendrá un delegado encargado de referenciar métodos que actualicen ListBox en tiempo de ejecución y en conjunto con eventos e hilos. En los próximos puntos se vera a detalle. También se crearon métodos que contienen delegados del tipo FUNC esperando una función anónima como condición.

```
/// <summary>
/// Devuelve un paciente dependiendo el delagado pasado por parametro
/// </summary>
/// <param name="busqueda">El parametro sera un paciente y retornara un bool</param>
/// <returns></returns>
2 references
public Paciente ObtenerPaciente(Func<Paciente,bool> busqueda)
{
    foreach (Paciente item in this.Pacientes)
    {
        if (busqueda(item))
        {
            return item;
        }
    }

    return null;
}

/// <summary> Devuelve un medico dependiendo el delagado pasado por parametro
1 reference
public Medico ObtenerMedico(Func<Medico, bool> busqueda)
{
    foreach (Medico item in this.Medicos)
    {
        if (busqueda(item))
        {
            return item;
        }
    }

    return null;
}
```

```
namespace Entidades
{
    public delegate void ActualizaPacientes();

    12 references
    public enum EObrasSocial
    {
        OSECAC,
        OSMEDICA,
        UP,
        OSPRERA
    }
```

HILOS Y EVENTOS:

La clase CentroMedico contiene un atributo Task para iniciar el subproceso, también los atributos correspondientes para cancelar el hilo cuando se desee. Por otra parte, también contiene un evento que es del tipo de delegado anterior. La funcionalidad del hilo es actualizar la lista de pacientes siempre que la base de datos tenga una modificación.

```
public class CentroMedico
{

    private List<Paciente> pacientes;
    private List<Medico> medicos;
    public event ActualizaPacientes OnActualizarLista;
    private CancellationTokenSource cancellationTokenSource;
    private CancellationToken cancellation;
    private Task actualizacion;
    private int invervaloTiempo;


    4 references | 3/3 passing
    public CentroMedico(int invervaloTiempo)
    {
        this.pacientes = new List<Paciente>();
        this.medicos = new List<Medico>();
        this.invervaloTiempo = invervaloTiempo;

    }
```

```
1 reference
public void IniciarActualizacion()
{
    if(this.actualizacion is null)
    {
        this.cancellationTokenSource = new CancellationTokenSource();
        this.cancellation = this.cancellationTokenSource.Token;

        //instancio el hilo y le agrego el metodo que quiero en segundo plano
        this.actualizacion = new Task(this.ActualizacionPacientes, this.cancellation);
    }

    //VALIDAR QUE ONDA ACAAAA
    if (!(this.actualizacion.Status == TaskStatus.Running || this.actualizacion.Status == TaskStatus.WaitingForActivation))
    {
        //inicio la tarea en segundo plano
        this.actualizacion.Start();
    }

}

1 reference
public void CancelarActualizacion()
{
    //cancelo el hilo
    if (this.cancellationTokenSource is not null)
    {
        this.cancellationTokenSource.Cancel();
    }

}
```

Eventos

```
/// <summary> Actualiza la lista de pacientes mediante un evento
1 reference
public void ActualizacionPacientes()
{
    try
    {
        while (!cancellation.IsCancellationRequested)
        {
            if (this.OnActualizarLista is not null)
            {
                List<Paciente> pacientesModificados = ADOPacientes.ObtenerModificados();
                if (pacientesModificados is not null)
                {
                    //ACA ESTA EL ERROR
                    //this.pacientes.AddRange(pacientesModificados);
                    this.ExtenderListaPacientes(pacientesModificados, true);
                    this.OnActualizarLista.Invoke();

                }
            }

            Thread.Sleep(this.invervaloTiempo);
        }
    }
    catch (Exception ex)
    {
        throw new Exception("Error al obtener los pacientes modificados de la DB en el hilo secundario", ex);
    }

}
```

METODOS DE EXTENSION:

Por ultimo y se creo una clase con métodos de extensión hacia la clase “CentroMedico” agregando 5 métodos.

```
namespace Entidades.MetodosDeExtension
{
    0 references
    public static class CentroMedicoExtension
    {
        /// <summary> Agregara una lista de pacientes a una lista ya existente, en caso  ...
        1 reference
        public static void ExtenderListaPacientes(this CentroMedico centroMedico,List<Paciente> lista)...

        /// <summary> Agrega una lista de pacientes a una ya existente, si ya existe el  ...
        1 reference
        public static void ExtenderListaPacientes(this CentroMedico centroMedico, List<Paciente> lista, bool reemplazar)...

        2 references | ● 1/1 passing
        public static bool ValidarNumeroAfiliado(this CentroMedico centroMedico, long numero)...

        2 references | ● 1/1 passing
        public static bool ValidarMatricula(this CentroMedico centroMedico, long numero)...

        1 reference
        public static string GenerarHistoriaClinica(this CentroMedico centroMedico,string diagnostico,Paciente paciente,Medico medico)...

    }
}
```