Laboratorio II - TPF Final

Alumno: Claudio Matias Mansilla

Comisión: 2C 2023

Contexto: Aplicación que simula el funcionamiento de una unidad de atención de emergencias del "Hospital Público del Quemado".

Menú Principal:

Su único propósito es proporcionar al usuario un menú de opciones para realizar las consultas y acciones que desee. El usuario podrá:

- Realizar acciones de alta, baja y modificación de pacientes.
- Exportar datos registrados en el legajo del paciente a formato JSON
- Realizar el alta de nuevo medico mediante la importación de un archivo JSON
- Admisión guardia brinda las opciones necesarias para agregar a un paciente a la cola de espera, siempre y cuando el paciente exista en la base de datos.
- Atención paciente: Se pretende simular la atención de aquellos pacientes que estén cola de espera, asignando aleatoriamente un médico y un tratamiento a la herida del paciente.

AMB Paciente:

Form que pretende dar un contexto a un usuario (administrativo) para realizar acciones de alta, baja y modificación de usuarios:

- Radiobutton`s: Utilizados para direccionar los eventos y sus respectivos manejadores.
- Button "Importar desde archivo": Buscará en forma predeterminada (en el escritorio del usuario) si existe algún archivo llamado `"JsonPaciente.json"`, de existir ejecutará los métodos necesarios para deserializar el objeto y mostrar por pantalla los atributos del objeto, permitiendo modificarlo manualmente de ser necesario.
- Button "Buscar Paciente": Leerá el número de dni ingresado en el campo, en caso de existir un paciente en la base de datos, procederá a construir un nuevo objeto paciente inyectando los campos leídos, para acto seguido imprimir dicha información por pantalla.
- Button "Limpiar": No hace otra cosa que limpiar todos los textBox.
- Button "Guardar": Al momento de invocarse su evento click, en el cuerpo del método manejador se encuentran sentencias de control para direccionar el flujo del programa hacia la invocación de los eventos que correspondan cuando se cumpla la condición de "checked" de los RadioButton mencionados previamente.

Exportar Paciente:

Invoca al método InputBox para pedirle al usuario que ingrese un número de DNI, por detrás se harán las validaciones necesarias para construir una instancia de paciente, leyendo la información desde la base de datos. En caso de que la instancia de paciente posea un DNI válido, se ejecutará el método encargado de realizar la serialización del objeto y la exportación de datos hacia el escritorio del usuario.

Admisión Guardia:

Form que pretende dar un contexto a un usuario (administrativo) para realizar buscar un paciente en base de datos, mostrar por pantalla algunos datos de dicha instancia paciente.

Si la instancia de paciente es correcta y es la deseada por el usuario de la aplicación, entonces deberá de seleccionar una causa de herida en el combobox correspondiente y también podrá agregar al paciente a la cola de espera (realizando por detrás un INSERT a la base de datos, apuntando hacia la tabla que maneja dicha cola).

Al momento de instanciarse el form, se consultará a la base de datos a la tabla “emergencias”, con el fin de crear una Queue en tiempo de ejecución de aquellos pacientes que cumplan con la condición inQueue == true.

Atención de Pacientes:

Form que permite simular la atención de aquellos pacientes que se encuentran en la Queue de pacientes que se encuentran en cola.

En la ejecución del método Simular, se pretende:

- Mostrar por pantalla la información del próximo paciente mediante el método Peek.
- Instanciar un “paciente actual” mediante el método Dequeue de la Queue<Paciente>
- Instanciar un medico mediante el método Dequeue de la Queue<Medico>
- Manipular los objetos “paciente actual” y “medico” para asignar a paciente un “tratamiento” a su atributo ‘tratamiento’.
- Por último, el simulador realiza un nuevo Enqueue del medico, para así volverlo a utilizar y que la Queue<Medico> no quede vacía. Para finalizar la simulación, se invoca al método “DequeuePacienteDB” para realizar el UPDATE en la base de datos de dicho paciente, con el fin de registrar el DNI del médico que realizó la atención, el tratamiento aplicado y cambiar la condición de inQueue a false;

Breve reseña de la aplicación de temas vistos (líneas de código resumidas)

ARCHIVOS Y SERIALIZACION

Form "MainView.cs":

```
private void btnExportar_Click(object sender, EventArgs e)
{
    if (paciente.Dni > 0)
    {
        FileManager<Paciente>.ExportarArchivo(paciente);
    }
}
```

Form "ViewPaciente.cs":

```
private void btnImport_Click(object sender, EventArgs e)
{
    try
    {
        this.paciente = FileManager<Paciente>.ImportarArchivo();
        this.CargarCamposView(this.paciente);
    }
}
```

```
namespace Entidades.Archivos
{
    public static class JsonManager<T>
    {
        public static void SerializarJson(T obj, string path)
        {
            using (StreamWriter sw = new StreamWriter(path))
            {
                JsonSerializerOptions options = new JsonSerializerOptions();
                options.WriteIndented = true;

                string jsonFile = JsonSerializer.Serialize(obj, options);
                sw.WriteLine(jsonFile);
            }
        }

        public static T DeserializarJson(string path)
        {
            using (StreamReader sr = new StreamReader(path))
            {
                try
                {
                    string json = sr.ReadToEnd();
                    if (json != null)
                    {
                        return JsonSerializer.Deserialize<T>(json);
                    }
                }
            }
            return default(T);
        }
    }
}
```

GENERICS

```
namespace Entidades.Archivos
{
    public static class FileManager<T>
    {
        public static T ImportarArchivo()
        {
            return JsonManager<T>.DeserializarJson(pathImport);
        }

        public static void ExportarArchivo(T obj)
        {
            JsonManager<T>.SerializarJson(obj,pathExport);
        }
    }
}
```

Toda la clase “ColaEspera.cs” trabaja sus métodos con <T>

```
namespace Entidades.Modelos
{
    public class ColaEspera<T> where T : class, new()
    {
        private Queue<T> personasEnCola;
    }
}
```

```
namespace Entidades.Interfaces
{
    public interface IEmergencia<T> where T: class, new()
    {
        public bool TratarQuemadura(T param);
        public bool Vendar(T param);
        public bool Suturar(T param);
        public bool Amputar(T param);
    }
}
```

BASE DE DATOS SQL

El manejo de base de datos fue aplicado a través de las clases, invocadas desde las clases que modelan los objetos relacionados a tales métodos.

- ADOColaEspera.cs
- ADOMedicos.cs
- ADOPacientes.cs
- DBManager.cs

METODOS DE EXTENSION

El manejo de base de datos fue aplicado a través de las clases:

```
namespace Entidades.MetodosExtencion
{
    public static class PersonaExtension
    {
        public static int CalcularEdad(this Persona persona)
        {
            int hoy = DateTime.UtcNow.Year;
            int nac = persona.FechaNac.Year;
            int edad = hoy - nac;
            return edad;
        }

        public static int CastearStrToInt(this string dniStr)
        {
            if (!string.IsNullOrWhiteSpace(dniStr))
            {
                int dni = 0;
                int.TryParse(dniStr, out dni);
                return dni;
            }
            return 0;
        }

    }
}
```

INTERFACES

Interface IAtencion implementada por la clase Paciente, cumpliendo con el contrato de IAtencion debiendo implementar sus métodos (en este caso, sus propiedades).

```
namespace Entidades.Interfaces
{
    public interface IAtencion
    {
        public string SangreGrupo { get; set; }
        public string SangreFactor { get; set; }
        public string Tratamiento { get; set; }
        public string CausaHerida { get; set; }
    }
}
```

Interface IEmergencia implementada por la clase Medico, cumpliendo con el contrato de IEmergencia debiendo implementar sus métodos (con el fin de setear el atributo “tratamiento” de la instancia Paciente recibida).

```
namespace Entidades.Interfaces
{
    public interface IEmergencia<T> where T: class, new()
    {
        public bool TratarQuemadura(T param);
        public bool Vendar(T param);
        public bool Suturar(T param);
        public bool Amputar(T param);
    }
}
```

EXCEPCIONES

El manejo de excepciones fue aplicado a través de las siguientes clases, con el fin de capturar cualquier posible excepciones producto de la conexión con la base de datos.

- DBManagerException.cs
- ReaderIsNullException.cs
- FileManagerException.cs

EVENTOS, DELEGADOS, CONCURRENCIA Y EXPRESIONES LAMBDA

```
namespace View
{
    public partial class ViewGuardia : Form
    {
        public event DelegadoVoid OnCargarMedicos;
        public event DelegadoVoid OnCargarPacientes;
        public event DelegadoABM OnGuardar;
        public event DelegadoABM OnAtender;

        private CancellationTokenSource cancellation;

        private void ViewGuardia_Load(object sender, EventArgs e)
        {
            this.OnCargarMedicos += this.medico.GetMedicos;
            this.OnCargarPacientes += this.pacienteActual.GetColaPacientes;
            this.OnAtender += this.medico.Atender;
            this.medico.OnMedicos += InicializarColaMedicos;
            this.pacienteActual.OnColaEspera += InicializarColaPacientes;

            this.OnCargarMedicos.Invoke();
            this.OnCargarPacientes.Invoke();
        }
        private void btnComenzar_Click(object sender, EventArgs e)
        {
            this.Simular();
        }

        private void btnDetener_Click(object sender, EventArgs e)
        {
            this.cancellation.Cancel();
        }


        private void ImprimirProxPaciente()
        {
            if (this.InvokeRequired) { this.BeginInvoke(() =>
this.ImprimirProxPaciente()); }
            else
            {
                this.txtBoxPacienteProx.Text =
this.DatosPaciente(this.pacienteProximo);
            }
        }
```

```
        private void Simular()
        {
            Task.Run(() =>
            {
                this.cancellation = new CancellationTokenSource();

while (!this.colaEspera.ColaIsEmpty() &&
!this.cancellation.IsCancellationRequested)
                {
                    this.pacienteProximo = this.colaEspera.VerProximaPersona;
                    this.ImprimirProxPaciente();
                    Thread.Sleep(5000);

                    this.pacienteActual = this.colaEspera.ProximaPersona;
                    this.pacienteProximo = this.colaEspera.VerProximaPersona;
                    this.medico = this.medicos.ProximaPersona;
                    this.medicos.NuevaPersona = this.medico;
                    this.ImprimirProxPaciente();
                    this.ImprimirPacienteActual();
                    this.ImprimirMedicoActual();

                    this.OnAtender.Invoke(this.pacienteActual);
                    this.ImprimirUltimoPaciente();
                }
            });

        }
    }
}
```