

Bienvenido a el programa de gestión de usuarios de IMA.

Introducción a la app:

En este caso nos encontramos con la obra social IMA, particularmente, con el programa que permite ingresar usuarios, modificarlos, darlos de baja y acceder a los datos de IMA. Se busca generar una app instructiva para el usuario para la facilidad de su uso.

Desarrollo de la app:

A continuación, explicaré paso a paso cómo funciona el programa:

1- En primer lugar, nos encontramos con el primer formulario FrmMenuIma. Este, en su evento load instancia al evento FrmSaludar, por lo cual, va a ser el primer formulario que vean.

FrmSaludar, da una bienvenida al usuario, presentando su nombre y el inicio de la jornada. Por un lado, utiliza la estructura DateTime.Now, para así obtener el horario de ingreso. Por otro lado, utiliza la clase Enviroment.UserName para obtener el nombre del usuario.

Debería

verse

así:

IMA Inicio Jornada



2- Una vez cerrado el formulario FrmSaludar, entramos finalmente en el FrmMenuIma. En este formulario nos encontraremos con una diversidad de botones, los cuales nos brindaran la información necesaria para atender a un futuro cliente. Al mismo tiempo, el formulario en su evento load, si es que el archivo existe, lee un archivo xml, con el método Leer y carga una lista de usuarios en la ruta AppDomain.CurrentDomain.BaseDirectory/**clientes.xml**;

Debería verse de la siguiente manera:

IMA

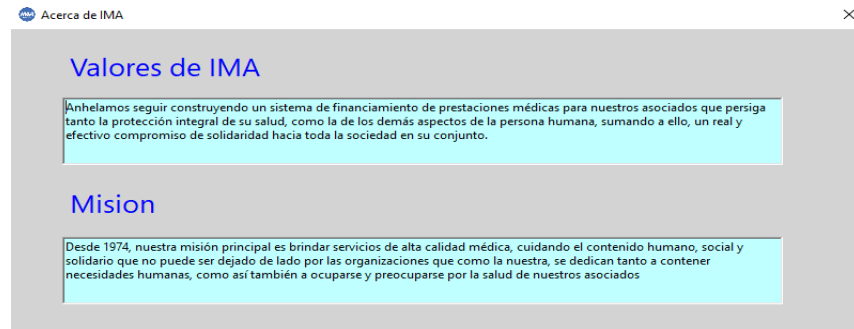


Explicaremos que función cumple cada uno de los botones:

A- Acerca de IMA (btnInfoIma): Instancia el formulario FrmAcercaIma.

Luego, lo presenta mediante el método ShowDialog(). Este formulario posee los métodos MostrarValoresIMA y MostrarMisionIMA, los cuales retornan una string con información de IMA. Estos 2 métodos son llamados en el evento Load del formulario y cargados en el rich text box correspondiente.

Debería verse así:



B- Planes (btnPlanes): Instancia el formulario FrmInfoPlanes. Luego, lo despliega utilizando el método ShowDialog().

Este formulario posee 4 botones. El primer botón, Plan clásico (btnPlanClasico) instancia un nuevo plan clásico y carga el rich text box con los datos del plan, mediante la sobrecarga del método ToString. El segundo botón, Plan Plus (btnPlanPlus) instancia un nuevo plan plus y carga el rich text box con los datos del plan, mediante la sobrecarga del método ToString. El tercer botón, Plan Premium (btnPlanPremium) instancia un nuevo plan premium y carga el rich text box con los datos del plan, mediante la sobrecarga del método ToString. Por último, el cuarto botón, cierra el formulario. Debería verse así:



C- El botón Normas de cobertura (btnCobertura): Instancia el formulario FrmNormasCobertura y lo despliega con el método ShowDialog().

El formulario en el constructor por defecto instancia 6 rutas ubicadas en:

```
this.rutaSistAtencion = $"{AppDomain.CurrentDomain.BaseDirectory}" + @"SistAtencion.txt";
this.rutaOrientacion = $"{AppDomain.CurrentDomain.BaseDirectory}" + @"Orientacion.txt";
this.rutaAtencionAmbulatoria = $"{AppDomain.CurrentDomain.BaseDirectory}" + @"AtencionAmbulatoria.txt";
this.rutaMedicamentos = $"{AppDomain.CurrentDomain.BaseDirectory}" + @"Medicamentos.txt";
this.rutaInternacion = $"{AppDomain.CurrentDomain.BaseDirectory}" + @"Internacion.txt";
this.rutaImprimir = $"{AppDomain.CurrentDomain.BaseDirectory}" + @"datosCobertura.txt";
```

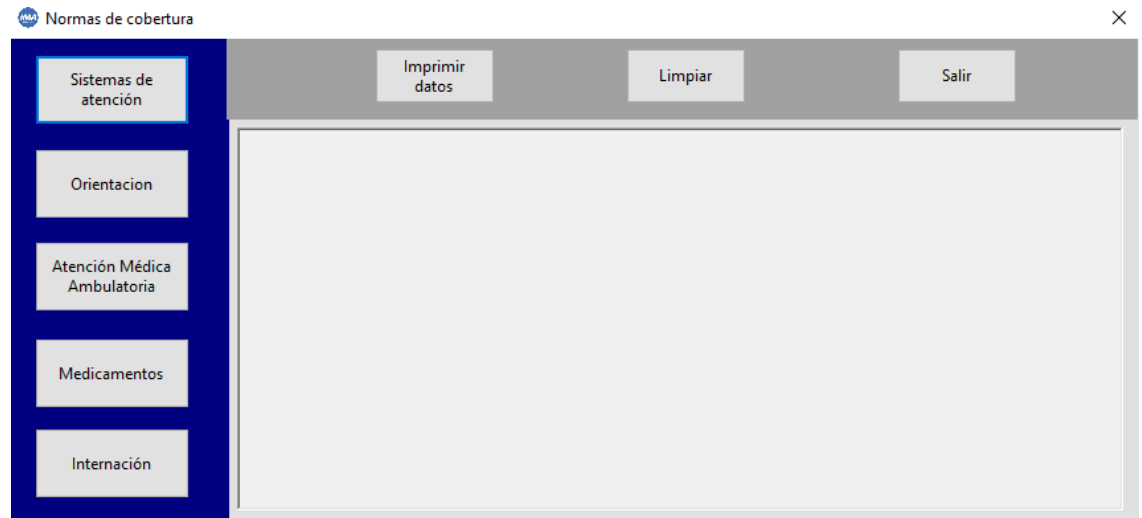
Los distintos eventos de click en los botones imprime en el rich text box los datos correspondientes leyendo el archivo txt que corresponda en cada caso.

El botón Imprimir datos, imprime la información con todos los datos en un archivo de tipo txt. Si el archivo existe, pisa los datos. Si el archivo no existe, lo crea.

El botón limpiar, carga el rich text box en string.Empty.

El botón salir cierra el formulario.

El formulario se ve de la siguiente manera:



D- El botón Acceder a contactos (btnContacto): Instancia el formulario FrmSucursales y lo despliega mediante el método ShowDialog().

Este formulario muestra en una data grid view las sucursales adheridas a IMA.

Dentro de este form tenemos 4 botones:

- 1- Mostrar/Actualizar sucursales(btnMostrar): Muestra todas las sucursales, y en caso de dar de alta una, actualiza para que aparezca la nueva sucursal.
- 2- Agregar sucursal (btnAgregar): Instancia el formulario FrmAltaSucursal. Allí, se ingresa localidad, contacto y dirección de la nueva sucursal. Al presionar cargar datos se guarda en la lista de sucursales y en la tabla de la base de datos IMA_SUCURSALES.
Dato extra: Cargar datos, sube los datos ya validados y crea la sucursal.
Limpiar datos, limpia todos los casilleros, los deja en blanco.
Salir, cierra el formulario de alta.
- 3- Filtrar ID (btnFiltrarId): Instancia el formulario FrmIdSucursal. Allí se ingresa el ID, de la sucursal que se desea ver (se verifica que exista). Si se confirma se busca la sucursal y se carga en la data grid, Si se cancela, se detiene la búsqueda.
- 4- Salir (btnSalir): Cierra el formulario.

Al presionar el botón salir, se cierra el formulario.

E- El botón Alta usuario (btnAltaUsuario): Instancia el formulario FrmAltaCliente y los despliega mediante el método ShowDialog(). A su vez, asigna a la administración ubicada en FrmMenuIma con el valor del retorno de la propiedad de solo lectura de FrmAltaCliente, para actualizar la lista constantemente.

Una vez desplegado el formulario de para dar de alta la cliente, se deben llenar los campos correspondientes, los cuales están validados según el tipo dato.

El formulario se debería ver de la siguiente manera:

Clientes

DNI:

Nombre:

Apellido:

Nacimiento: 1/ 6/2022

Edad: 0

Direccion:

Telefono:

Factor sangre: Tipo de sangre

**Si no selecciona un tipo, se asigna por defecto 0 negativo*

IMA
Medicina Privada

Genero
☒ Masculino
☐ Femenino
☐ No binario

Plan elegido
☒ Plan Clasico
☐ Plan Plus
☐ Plan Premiun

Cargar datos
Limpiar datos
Salir

Datos del plan:
PLAN SOLICITADO: Plan Clasico
VALOR DEL PLAN: 3500
SERVICIOS DEL PLAN:
 CONSULTAS DOMICILIARIAS: ST
 CONSULTAS POR CONSULTORIO: ST
 KINESIOLOGIA: SI
 FONOAUDIOLOGIA: NO
 INTERNACION CLINICA: NO
 INTERNACION QUIRURGICA: NO
 Datos extra: ST(Sin tope)

F- El botón Modificar Datos (btnModificarDatosUsuario) : Instancia el formulario FrmModificar y lo despliega con el método ShowDialog(). Se desplegará el siguiente formulario:

Modificar datos

Ingrese el DNI de la persona a la que desea modificar los datos:

Ingrese DNI

Confirmar Cancelar

Este formulario en el textbox recibe el DNI y valida el ingreso de datos según corresponda. Si la persona existe, al presionar confirmar, instancia el formulario FrmModificarDatos y lo despliega mediante el método ShowDialog.

Si el texto está en blanco o la persona no existe, se arrojará una excepción según corresponda. Si todo sale bien, el siguiente formulario se ve de la siguiente forma:

frmModificarDatos

DNI: 42417493

Nombre: Luca Nahuel

Apellido: Agnoli

Nacimiento: 22/ 2/2000

Edad: 22

Direccion: Moreno 1220

Telefono: 1122831588

Factor sangre: A Positivo

IMA
Medicina Privada

Genero
☒ Masculino
☐ Femenino
☐ No binario

Plan elegido
☐ Plan Clasico
☐ Plan Plus
☒ Plan Premiun

Cargar
Limpiar
Salir

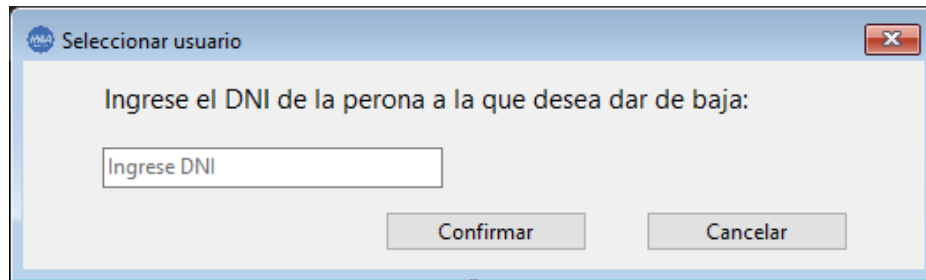
G- El formulario FrmModifcarDatos, se instancia ya cargado con los datos de la persona seleccionada. El DNI no se puede cambiar, los otros datos sí. Los text box están preparados para que no se puedan ingresar datos erróneos según corresponda.

Si se presiona confirmar se actualiza la lista de personas y se guardan los datos en el archivo clientes.xml. La ruta es la siguiente:

```
ruta = $"{AppDomain.CurrentDomain.BaseDirectory}" + @"clientes.xml";
```

H- El botón Baja usuario (btnBajaUsuario): Revisa que la lista este cargada, sino está cargada tira un mensaje con un error. Si la lista está cargada instancia un formulario FrmBajaUsuario y lo despliega con el método ShowDialog.

El formulario se debería ver de la siguiente manera:



Si el texto está en blanco o la persona no existe, arroja una excepción según corresponda. A su vez, valida que la entrada de datos sea acorde a un DNI. Si presiona confirmar y la persona es válida en el sistema, se cambia el atributo estadoPlan de la clase persona a false. Los datos de la persona siguen en el sistema, pero ya no se encuentra activa, es decir, no se realiza una baja lógica, se pretende que el sistema sea realista conteniendo los datos de la persona en un estado inactivo.

I- Si se presiona el botón Clientes activos (btnMostrarClientesActivos): Se instancia el formulario FrmClientesActivos y se despliega con el método ShowDialog(). En este formulario podremos observar un listado de todos los clientes con el estadoPlan en true. A su vez, podemos filtrar para buscar a un cliente en específico por DNI, la idea es que si alguien llama al operador/operadora pueda acceder a los datos de esta manera. Al mismo tiempo, si se desconoce el DNI, el formulario posee 2 botones para ordenar por apellido o DNI, en caso de ser necesario, para facilitar la búsqueda de la persona activa en el sistema. Este formulario SOLO muestra personas activas.

TEMAS APLICADOS

➤ Excepciones: Se crearon 3 excepciones para el programa. La primera, llamada ExcepcionArchivoInvalido, se lanza cuando el archivo no existe o no posee una ruta valida. Mostrando su mensaje mediante la sobrecarga del método ToString. Ejemplo de implementación:

```
1 referencia
public void Guardar(string ruta, string contenido)
{
    if (ValidarSiExisteElArchivo(ruta) && ValidarExtension(ruta))
    {
        GuardarArchivo(ruta, contenido);
    }
    else
    {
        throw new ExcepcionArchivoInvalido("La ruta o la extension no son validas", "Guardar", "PuntoTxt.cs");
    }
}
```

La segunda, llamada `ExcepcionCasillerosEnBlanco`, arroja una excepción si los casilleros de los text boxes no puedan ser espacios en blanco. Ejemplo de implementación:

```
private void AccionConfirmarClick()
{
    if (txtDniSolicitado.Text.Length == 8 && !string.IsNullOrEmpty(txtDniSolicitado.Text))
    {
        personaBaja = administracion.BuscarPorDni(administracion, Convert.ToInt32(txtDniSolicitado.Text));
        if (personaBaja is not null)
        {
            personaBaja.EstadoPlan = false;
            listaPersonasImprimir.Guardar($"{this.ruta}", administracion.ListaPersonas);
            MessageBox.Show($"Se ha dado de baja exitosamente! ", "Baja usuario",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
            Close();
        }
        else
        {
            throw new ExcepcionPersonaNula("La persona no se encuentra en el sistema o es nula", "FrmClientesActivos.cs", "btnBuscarPorDni_click");
        }
    }
    else
    {
        throw new ExcepcionCasillerosEnBlanco("Los campos no pueden estar en blanco o tener espacios", "FrmBajaUsuario", "AccionConfirmarClick()");
    }
}
```

La tercera, llamada `ExcepcionPersonaNula`, arroja una excepción si la persona tiene valor nulo o no pertenece a la nula. Ejemplo de implementación:

```
if (txtDniSolicitado.Text != string.Empty)
{
    if (txtDniSolicitado.Text.Length == 8)
    {
        personaBuscada = administracion.BuscarPorDni(administracion, Convert.ToInt32(txtDniSolicitado.Text));
        if (personaBuscada is not null)
        {
            if (personaBuscada.EstadoPlan)
            {
                FrmModificarDatos formModificar = new FrmModificarDatos(administracion, personaBuscada, listaPersonasImprimir);
                formModificar.ShowDialog();
            }
            else
            {
                MessageBox.Show($"La persona no se encuentra activa en el sistema ", "Error al modificar",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        else
        {
            throw new ExcepcionPersonaNula("Persona nula, no se encuentra en el sistema", "FrmModificar.cs", "AccionConfirmarClick");
        }
    }
}
```

La cuarta y última, llamada `ExcepcionSucursalNula` arroja una excepción si la sucursal tiene valor nulo o no pertenece a la lista de sucursales. Ejemplo de implementación

```

private void AccionBotonConfirmar()
{
    int idIngreado;
    if (txtId.Text != string.Empty)
    {
        idIngreado = Convert.ToInt32(txtId.Text);
        sucursalBuscada = idIngreado.LeerDatosPorId();
        if (sucursalBuscada is not null)
        {
            MessageBox.Show("Sucursal encontrada con exito", "Filtrar sucursal", MessageBoxButtons.OK, MessageBoxIcon.Information);
            Close();
        }
        else
        {
            throw new ExcepcionSucursalNula("La sucursal no existe", "FrmIdSucursal.cs", "AccionBotonConfirmar");
        }
    }
    else
    {
        throw new ExcepcionCasillerosEnBlanco("El texto no puede estar en blanco", "FrmModificar.cs", "AccionConfirmarClick");
    }
}

```

➤ **Pruebas unitarias:** Las pruebas unitarias validan que los archivos utilizados en formato txt y xml tengan una ruta y extensión valida. Ejemplos de implementación en el caso de formato xml:

```

[TestMethod]
0 referencias
public void PuntoXml_DeberiaRetornarTrue_SiLaExtensionEsTipoXml()
{
    //Arrange
    string ruta = $"{Environment.GetFolderPath(Environment.SpecialFolder.Desktop)}\\MiArchivo.xml";
    PuntoXml<string> xml = new PuntoXml<string>();
    bool rutaValidacion = xml.ValidarExtension(ruta);
    // string
    Assert.IsTrue(rutaValidacion);
}

[TestMethod]
[ExpectedException(typeof(ExcepcionArchivoInvalido))]
0 referencias
public void PuntoXml_DeberiaLanzarMiExcepcionArchivoInvalido_SiElArchivoNoExiste()
{
    //Arrange
    string ruta = $"{Environment.GetFolderPath(Environment.SpecialFolder.Desktop)}\\ArchivoArchivo.xml";
    PuntoTxt<string> xml = new PuntoTxt<string>();
    if (!xml.ValidarSiExisteElArchivo(ruta))
    {
        throw new ExcepcionArchivoInvalido("Archivo invalido", "Error", "Test puntoJson");
    }
}

```

➤ **Tipos genéricos:** Las clases PuntoTxt y PuntoXml son genéricas, para poder imprimir cualquier tipo de dato que se requiera. En este caso se utiliza para imprimir personas, strings y contactos. Se crearon genéricas en caso de a futuro, para el tp4, sea necesario imprimir mayor cantidad de datos de distintas clases.

➤ **Interfaces:** La interfaz implementada se llama IDatosContacto, obliga a las clases que la implementen a utilizar los campos contenidas en ella. Es utilizada en las clases contactoInterno, Persona y Sucursales.

```

public interface IDatosContacto
{
    /// <summary>
    /// Propiedad de lectura, que a su vez, permite asignar un valor al Nombre.
    /// Obliga a las clases que la implementen a utilizarla.
    /// </summary>
    8 referencias
    string Nombre
    {
        get; set;
    }
    /// <summary>
    /// Propiedad de lectura, que a su vez, permite asignar un valor al NumeroTelefonico.
    /// Obliga a las clases que la implementen a utilizarla.
    /// </summary>
    8 referencias
    double NumeroTelefonico
    {
        get; set;
    }
    /// <summary>
    /// Propiedad de lectura, que a su vez, permite asignar un valor a la Direccion.
    /// Obliga a las clases que la implementen a utilizarla.
    /// </summary>
    8 referencias
    string Direccion
    {
        get; set;
    }
}

```

➤ **Archivos y serialización:** Se imprimen y leen datos en formato txt y xml. Se creo una clase padre Archivo, de la cual heredan las hijas. Estas son, PuntoXml y PuntoTxt. Según el caso lo amerite, se imprimen y leen datos en dichos formatos. Ejemplo de la clase PuntoXml:


```

0 referencias
public void Guardar(string ruta, T contenido)
{
    if (ValidarSiExisteElArchivo(ruta) && ValidarExtension(ruta))
    {
        Serializar(ruta, contenido);
    }
    else
    {
        throw new MiExcepcionArchivoInvalido("La ruta o la extension no son validas", "Guardar", "PuntoXml.cs");
    }
}

/// <summary>
/// Si el archivo no existe, guarda los datos y crea una carpeta en la ruta seleccionada.
/// Caso contrario arroja MiExcepcionArchivoInvalido.
/// </summary>
/// <param name="ruta">ruta del archivo</param>
/// <param name="contenido">contenido del archivo</param>
0 referencias
public void GuardarComo(string ruta, T contenido)
{
    if (ValidarExtension(ruta))
    {
        Serializar(ruta, contenido);
    }
    else
    {
        throw new MiExcepcionArchivoInvalido("La ruta o la extension no son validas", "GuardarComo", "PuntoXml.cs");
    }
}

```

Archivos cargados:

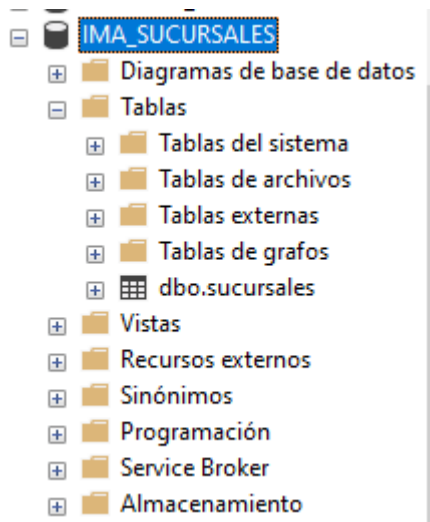
Trabajo practicos II > tps_laboratorio_ii > TP3 > Agnoli.Luca.2A.TPFinal > FrmMenuImla > bin > Debug > net5.0-windows >				
Acceso rápido	Nombre	Fecha de modificación	Tipo	Tamaño
OneDrive - Personal	ref	5/6/2022 06:15	Carpeta de archivos	
Este equipo	AtencionAmbulatoria.txt	4/6/2022 16:24	Documento de te...	5 KB
Descargas	BibliotecalMA.dll	5/6/2022 16:18	Extensión de la ap...	18 KB
Documentos	BibliotecalMA.pdb	5/6/2022 16:18	Program Debug D...	19 KB
Escritorio	clientes.xml	5/6/2022 23:09	XML Document	7 KB
Imágenes	datosCobertura.txt	5/6/2022 21:23	Documento de te...	8 KB
Música	datosSucursales.json	5/6/2022 20:14	Archivo de origen ...	1 KB
Objetos 3D	FormsTP3.deps.json	5/6/2022 06:16	Archivo de origen ...	1 KB
Videos	FormsTP3.dll	5/6/2022 23:16	Extensión de la ap...	426 KB
Disco local (C:)	FormsTP3.exe	5/6/2022 23:16	Aplicación	123 KB
Datos (T:)	FormsTP3.pdb	5/6/2022 23:16	Program Debug D...	30 KB
Red	FormsTP3.runtimeconfig.dev.json	5/6/2022 06:15	Archivo de origen ...	1 KB
	FormsTP3.runtimeconfig.json	5/6/2022 06:15	Archivo de origen ...	1 KB
	Internacion.txt	4/6/2022 16:24	Documento de te...	2 KB
	Medicamentos.txt	4/6/2022 16:24	Documento de te...	1 KB
	Orientacion.txt	4/6/2022 16:24	Documento de te...	1 KB
	SistAtencion.txt	5/6/2022 19:58	Documento de te...	2 KB

- **SQL:** Se realizó una lectura de todos los datos de la tabla, una filtración por id y un dar de alta (Utilizados y explicados en FrmSucursal). La conexión es la siguiente:

```

/// <summary>
/// Inicializa la cadenaConexion a la base de datos correspondiente.
/// </summary>
0 referencias
static GestorSQL()
{
    GestorSQL.cadenaConexion = "Server=.;Database=IMA_SUCURSALES;Trusted_Connection=True";
}

```



DESKTOP-3UP6QCF.I...S - dbo.sucursales			
	Nombre de columna	Tipo de datos	Permitir val...
▶	id	int	<input type="checkbox"/>
	localidad	varchar(40)	<input type="checkbox"/>
	numeroTelefono	float	<input type="checkbox"/>
	direccion	varchar(40)	<input type="checkbox"/>
			<input type="checkbox"/>

- **Programación Multi-Hilo y eventos:** Se realizó una tarea en segundo plano en la cual, cuando el empleado/a ingresa a la aplicación se inicia un conteo que va descontando el tiempo en el cual la jornada termina (Se puso 120 segundos, lo ideal es 8 hs, por cuestiones de facilitar el tiempo de corrección). Al mismo tiempo, que esta tarea trabaja en un hilo secundario, cuando la barra llegue al final se iniciaría un evento que despliega un MessageBox avisando que la jornada laboral terminó. Este se ve de la siguiente forma:





- **Delegados y expresiones Lambda:** Se instancia el delegado, DelegadoCargarDatos, al cual se lo cargo con 2 métodos, CargarLista y CargarGridView. Este se llama en el form:

```
1 referencia
public FrmSucursales()
{
    InitializeComponent();
    this.listaSucursales = new List<Sucursales>();
    miDelegado = CargarLista;
    miDelegado += CargarGridView;
}

/// <summary>
/// Llama a los metodos CargarRichTextBoxs.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 referencia
private void frmSucursales_Load(object sender, EventArgs e)
{
    miDelegado.Invoke();
}
```

Expresiones Lambda de utilizo de la siguiente manera (también en muchas propiedades de las biblitecas):

```
2 referencia
public void ComenzarCronometro(int segundosTiempoJornada)
{
    //int numeroBarra = 0;
    try
    {
        CancellationTokensource cancellation;
        cancellation = new CancellationTokensource();
        CancellationToken token = cancellation.Token;
        Task.Run(() => AccionCronometro(cancellation, segundosTiempoJornada), token);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"{ex.Message}\n{ex.StackTrace}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

- **Método de extensión:** Se realizó un método de extensión en la clase GestorSQL, la cual es estática, que se ve de la siguiente manera:

```
public static Sucursales LeerDatosPorId(this int id)
{
    Sucursales sucursal = null;
    try
    {
        string query = "SELECT * FROM sucursales WHERE id=@id";
        using (SqlConnection connection = new SqlConnection(GestorSQL.cadenaConexion))
        {
            SqlCommand cmd = new SqlCommand(query, connection);
            cmd.Parameters.AddWithValue("id", id);
            connection.Open();
            SqlDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                string localidad = reader.GetString(1);
                double numeroTelefonico = reader.GetDouble(2);
                string direccion = reader.GetString(3);

                sucursal = new Sucursales(id, localidad, numeroTelefonico, direccion);
            }
        }
    }
}
```

El id pasa a ser de extensión, gracias a ello, puedo invocar el método a partir de una int id. Se ve de la siguiente manera:

```
1 referencia
private void AccionBotonConfirmar()
{
    int idIngresado;
    if (txtId.Text != string.Empty)
    {
        idIngresado = Convert.ToInt32(txtId.Text);
        sucursalBuscada = idIngresado.LeerDatosPorId();
        if (sucursalBuscada is not null)
        {
            MessageBox.Show("Sucursal encontrada con éxito", "Filtrar sucursal", MessageBoxButtons.OK, MessageBoxIcon.Information);
            Close();
        }
        else
        {
            throw new ExcepcionSucursalNula("La sucursal no existe", "FrmIdSucursal.cs", "AccionBotonConfirmar");
        }
    }
    else
    {
        throw new ExcepcionCasillerosEnBlanco("El texto no puede estar en blanco", "FrmModificar.cs", "AccionConfirmarClick");
    }
}
```

Gracias por su tiempo. Saludos de parte del equipo IMA.