

TRABAJO PRÁCTICO 4 - LABORATORIO II
RENZO ORPELLI - DIV 2 E.
EMPRESA DE CIBERSEGURIDAD.

PARTE I:

Explicación del programa:

El programa se basa en un panel de administración de una empresa de ciberseguridad en el que se podrá cargar una lista de clientes desde un archivo o desde una base de datos, agregar clientes(cantidad máxima 300), modificar algunos de sus datos y también eliminarlos. Cada cliente tendrá una lista de servicios realizados y la posibilidad de imprimirlas una factura detallando qué servicios le fueron realizados.

Ubicación de la base de datos:

Dentro de la carpeta del proyecto, se encontrará una carpeta llamada BACKUP BASE DE DATOS, en la cual se encontrarán dos carpetas, una llamada QUERY, la cual tendrá el SCRIPT de sql listo para ejecutar.

La otra carpeta llamada BAK en la cual se encuentra el archivo de extensión .bak para ser restaurado dentro del programa "Microsoft SQL Server Management Studio 18".

Funcionamiento del programa:

Al iniciar la aplicación, nos encontraremos el formulario principal, donde

- 1) Nos llevará al panel de Administración de los clientes.
- 2) Nos sacará del programa.



Entrando en el panel "Administracion Clientes" nos encontraremos con distintas botones los cuales representan distintas acciones a realizar:

1)Alta Cliente:

Se mostrará por pantalla un nuevo formulario el cual permitirá cargar los datos de un nuevo cliente.

2) Modificar Cliente:

Nos mostrará por pantalla un nuevo formulario con los datos del cliente seleccionado previamente en la listbox "Listado Clientes" y nos permitirá modificar algunos datos del mismo.

3) Eliminar Cliente:

Removerá el cliente seleccionado de la listbox "Listado Clientes".

4) Salir:

Dará la oportunidad al usuario de salir del formulario.

5) Cargar Clientes desde archivo:

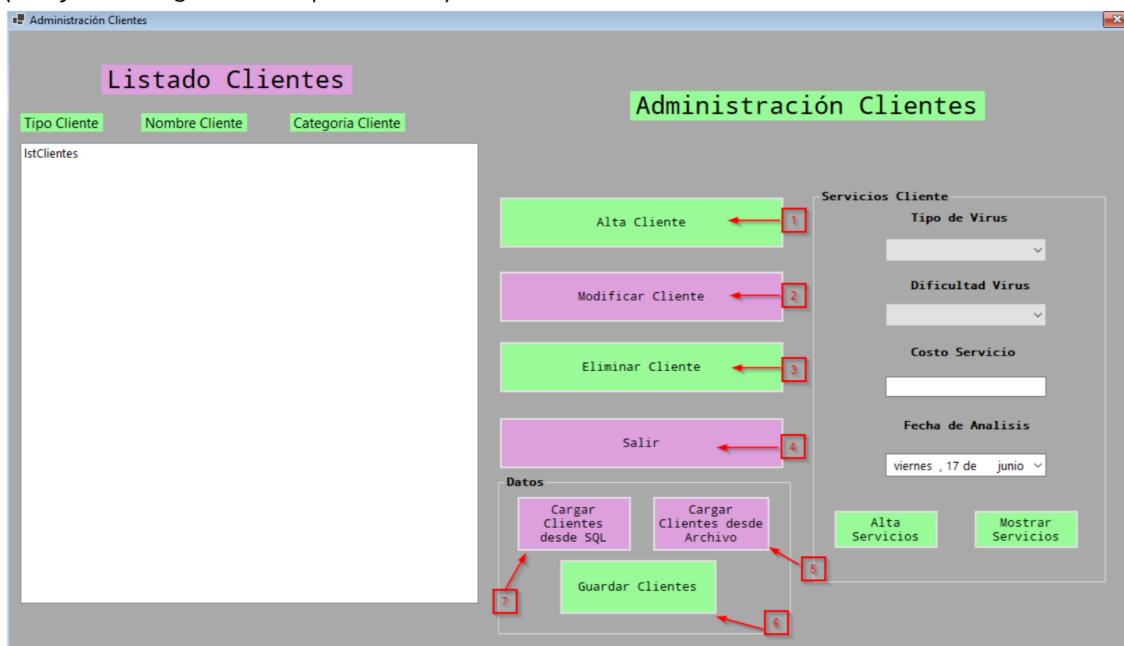
Cargará una lista de clientes en caso de que exista en la ruta especificada por el Gestor de Archivos.

6) Guardar Clientes:

Tomará el estado actual de la lista y en caso de que esta no esté vacía, se la guardará en un archivo de tipo ".xml", mostrándole al usuario por un mensaje donde estará guardado el archivo.

7) Cargar Clientes desde SQL:

Establecerá una conexión con la base de datos y traerá todo el listado de clientes que se encuentre en ella. (El listado es distinto al del "Cargar clientes archivo").
(Abajo la imagen correspondiente).

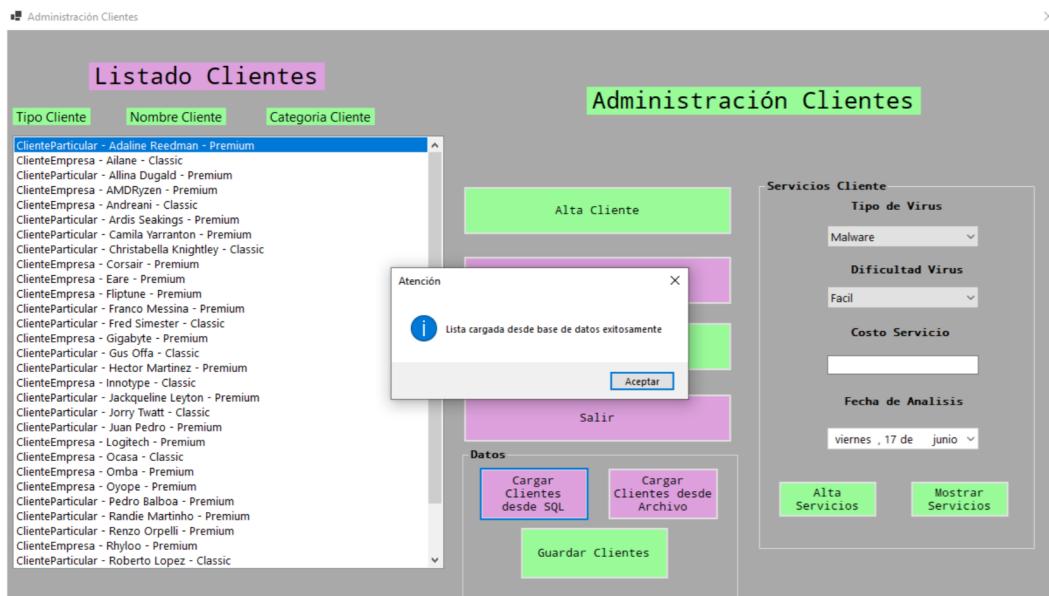
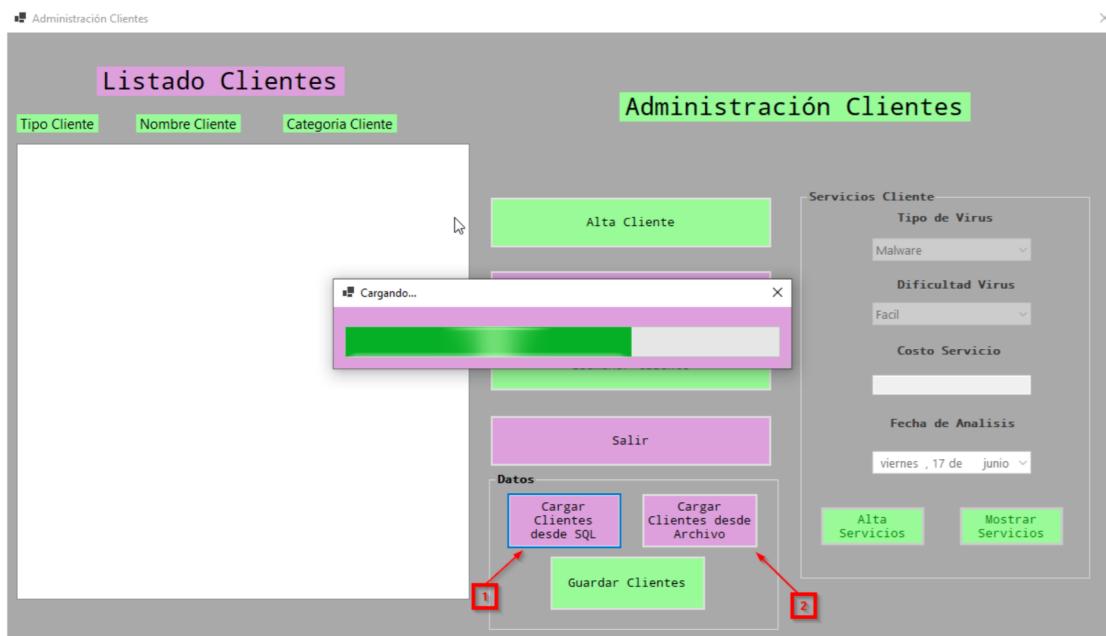


Carga de los datos:

Al seleccionar la carga se podrá observar que se encuentra los botones "Cargar datos desde archivo" el cuál cargará una lista desde un archivo xml y "Cargar datos desde sql" el cuál cargará la lista desde una base de datos. Estas listas son independientes una de la otra.

Cada acción que se quiera realizar sobre los elementos de la lista (Alta, Eliminación, Modificación, Alta de servicios, Mostrar servicios) afectará a la lista correspondiente de cada botón, es decir, la lista cargada desde el archivo o la lista cargada de la base de datos.

Por otro lado, al presionar algunos de estos botones se mostrará una animación simulando una carga. Aquí la imagen.



Sobre los servicios, para agregar un servicio a un cliente, se tendrá que seleccionar un cliente del listbox "Listado de clientes", para cargar un servicio. En el grupbox "Servicios Cliente" se tendrán que completar con todos los atributos del servicio, los cuales son el tipo de virus, la dificultad del virus, el costo del servicio y la fecha del análisis. Las acciones a realizar serán:

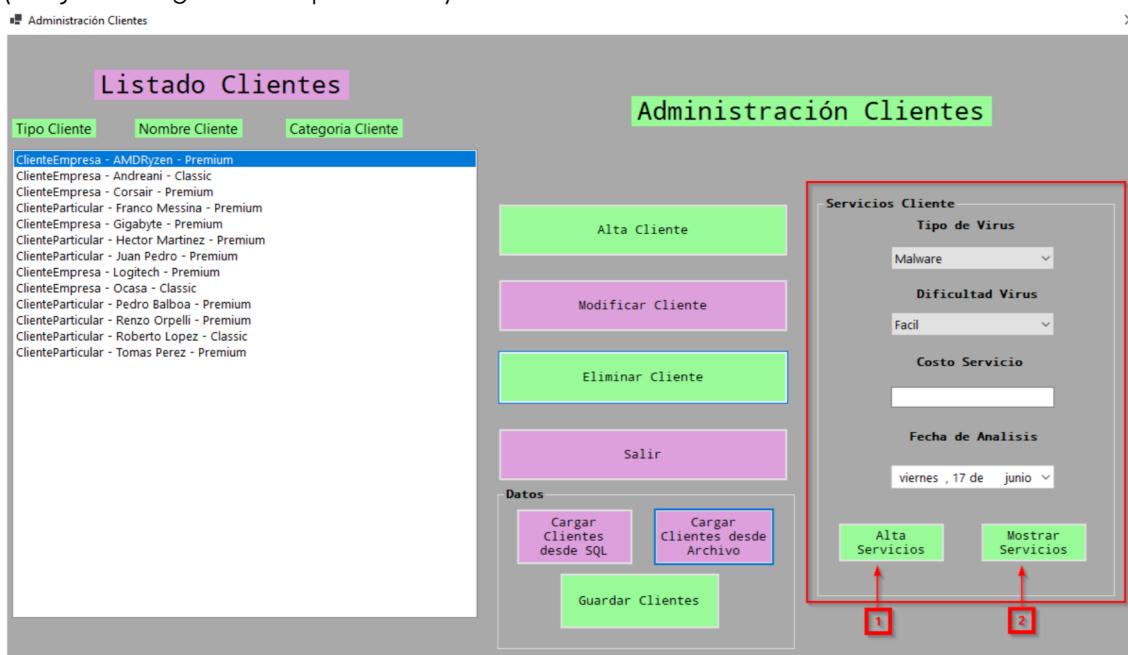
1) Alta servicios:

Dará de alta el servicio para el cliente previamente seleccionado y se lo agregara a su lista.

2) Mostrar Servicios:

Abrirá un formulario mostrando todos los servicios que tiene dicho cliente y le dará la oportunidad de emitir una factura del cliente mostrando un mensaje con la ubicación del archivo.

(Abajo la imagen correspondiente).



PARTE II :

Implementaciones de temas utilizados para la realización del trabajo (Clase 10-20):

Excepciones:

Este tema fue aplicado varias veces en el proyecto con el fin de manejar y controlar posibles errores, un ejemplo de una implementación seria :

```

    /// <summary>
    /// Valida la cantidad de clientes de la lista no sea mayor a la cantidad de clientes permitidos
    /// </summary>
    /// <returns>true si no fue alcanzada, de lo contrario lanza una excepción de tipo CantidadDeClientesAlcanzadaException</returns>
1 referencia
private bool ValidarCantidadClientes()
{
    if (this.ListaClientes.Count == this.cantidadClientesMaxima)
    {
        throw new CantidadDeClientesAlcanzadaException("Cantidad de clientes máxima alcanzada");
    }
    return true;
}

    /// <summary>
    /// crea una excepcion con un mensaje
    /// </summary>
    /// <param name="message">mensaje de la excepcion</param>
1 referencia
public CantidadDeClientesAlcanzadaException(string message) : base(message)
{
}

    /// <summary>
    /// crea una excepcion con un emensaje y su innerException
    /// </summary>
    /// <param name="message">el mensaje de la excepcion</param>
    /// <param name="innerException">la innerException de la excepcion</param>
0 referencias
public CantidadDeClientesAlcanzadaException(string message, Exception innerException) : base(message, innerException)
{
}

```

Test Unitarios:

Este tema fue implementado con el fin de verificar la funcionalidad de ciertos métodos de la aplicación, un ejemplo de su aplicación de un test unitario sería:

```

[TestMethod]
0 referencias
public void AlAregarUnServicioAUnClienteConCategoriaPremium_DeberiaAplicarleUnDescuentoAlServicio()
{
    #region Arrange
    Cliente empresa = new ClienteEmpresa(Cliente.categoríaCliente.Premium, "Andreani", "123123123123", "lobos 123");
    Servicio servicio = new Servicio(20000, Servicio.dificultadVirus.Difícil, Servicio.tipoVirus.Adware, new System.DateTime(2020, 03, 23));
    double result = 0;
    double expected = 18000;
    #endregion

    #region Act
    if (empresa + servicio)
    {
        result = servicio.CostoServicio;
    }
    #endregion
    #region Assert
    Assert.AreEqual(expected, result);
    #endregion
}

```

Tipos Genericos:

Este tema fue implementado en la clase “Serializador” y en la interfaz “IArchivo” para adaptar el método “Escribir” y “Validar Extensión” y tener la misma funcionalidad aunque maneje distintos tipo de datos en las clases que se los implementa.

```

/// <summary>
/// metodo encargado de leer un archivo de tipo xml, deserializandolo y devolviendolo el objeto T
/// </summary>
/// <param name="nombreArchivo">string con el nombre del archivo</param>
/// <returns>retorna el objeto deserializado, caso contrario lanzara excepciones del tipo ArchivoNullException</returns>
/// <exception cref="ArchivoNullException"></exception>
1 referencia
public T Leer(string nombreArchivo)
{
    try
    {
        if (tipo == ETipo.ClienteXML)
        {
            if (ValidarExtension(nombreArchivo) && ExisteArchivo($"{rutaBase}\\{nombreArchivo}"))
            {
                using (XmlTextReader xmlTextReader = new XmlTextReader($"{rutaBase}\\{nombreArchivo}"))
                {
                    XmlSerializer serializer = new XmlSerializer(typeof(T));
                    return serializer.Deserialize(xmlTextReader) as T;
                }
            }
        }
        catch (ArchivoNullException ex)
        {
            throw ex;
        }
        catch (Exception ex)
        {
            throw new ArchivoNullException("Error al leer el archivo xml", ex);
        }
        return null;
    }
}

```

```

public interface IArchivo<T>
{
    /// <summary>
    /// Guarda un archivo del tipo Objeto Generico en la ruta indicada
    /// </summary>
    /// <param name="nombreArchivo">El nombre del archivo</param>
    /// <param name="contenido">el contenido del objeto generico</param>
    /// <returns>un string que contiene el resultado de la operacion</returns>
6 referencias
    string Escribir(string nombreArchivo, T contenido);

    /// <summary>
    /// valida la extension de un archivo
    /// </summary>
    /// <param name="nombreArchivo">nombre del archivo</param>
    /// <returns>true si es correcta, de lo contrario una excepcion de tipo ArchivoNullException</returns>
6 referencias | 1/1 pasando
    bool ValidarExtension(string nombreArchivo);
}

```

Interfaces:

Este tema fue implementado con la finalidad de agrupar métodos y que estos puedan ser utilizados en otras clases (Un ejemplo de implementación de las interfaces utilizadas se puede ver en la imagen de arriba).

Archivos:

Este tema fue implementado con el fin de emitir la factura de cada cliente en formato .txt, con todos sus datos, todos los servicios, y el total a pagar del mismo.

```

/// <summary>
/// Recibe el nombre del archivo y el contenido a escribir del mismo de tipo string y lo guarda en un archivo de tipo txt
/// </summary>
/// <param name="nombreArchivo">el nombre del archivo</param>
/// <param name="contenido">el contenido del archivo en tipo string</param>
/// <returns>devuelve un mensaje con el resultado de la operacion</returns>
/// <exception cref="ArchivoNullException"></exception>
3 referencias
public string Escribir(string nombreArchivo, string contenido)
{
    string pudoLeer = "Error al escribir el archivo de texto";
    try
    {
        if (ValidarExtension(nombreArchivo))
        {
            using (StreamWriter streamWriter = new StreamWriter($"{base.rutaBase}\\{nombreArchivo}"))
            {
                streamWriter.WriteLine(contenido);
                pudoLeer = $"Archivo de Text Escrito con exito en {rutaBase}\\{nombreArchivo}";
            }
        }
    }
    catch (Exception ex)
    {
        throw new ArchivoNullException("Error al escribir archivo de texto", ex);
    }
    return pudoLeer;
}

```

Serialización:

Este tema fue implementado en la clase “Serializador” del tipo genérica la cual también utiliza la misma interfaz “IArchivo” del mismo tipo, para serializar o deserializar la lista de clientes.

Aquí debajo la imagen.

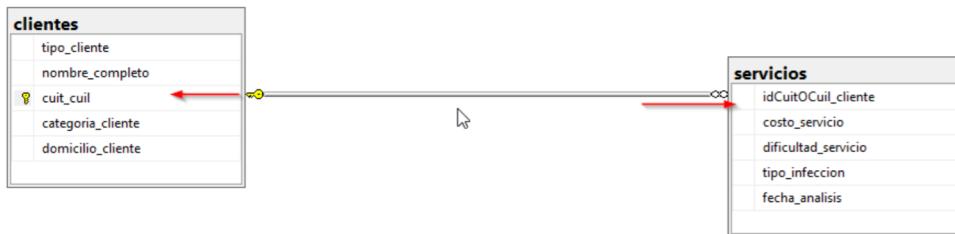
```
/// <summary>
/// método encargado de serializar la lista de clientes, recibe el nombre archivo y contenido de tipo genérico
/// lo serializara y guardara en la ruta especificada en la clase GestorDeArchivo
/// </summary>
/// <param name="nombreArchivo">string con el nombre del archivo</param>
/// <param name="elemento"></param>
/// <returns>un string con la ruta del archivo guardado o de caso contrario lanzara excepciones del tipo ArchivoNullException</returns>
/// <exception cref="ArchivoNullException"></exception>
3 referencias
public string Escribir(string nombreArchivo, T elemento)
{
    string resultado = "Error al Guardar";

    try
    {
        if (tipo == ETipo.ClienteXML)
        {
            if (ValidarExtension(nombreArchivo))
            {
                using (XmlTextWriter xmlTextWriter = new XmlTextWriter(${rutaBase}\{nombreArchivo}, Encoding.UTF8))
                {
                    xmlTextWriter.Formatting = Formatting.Indented;
                    XmlSerializer serializer = new XmlSerializer(typeof(T));
                    serializer.Serialize(xmlTextWriter, elemento);
                    resultado = $"Archivo Guardado Existosamente en {this.rutaBase}{nombreArchivo}";
                }
            }
        }
        catch (ArchivoNullException ex)
        {
            resultado = ex.Message;
        }
        catch (Exception ex)
        {
            throw new ArchivoNullException("Error al escribir el archivo xml", ex);
        }
    }
    return resultado;
}
```

Introducción a SQL:

Este tema se implementó en el diagrama de la base de datos, La misma cuenta con dos tablas, por un lado la tabla clientes que tendrá como Primary key(PK) al atributo cuit_cuil que estará referenciado como Foreign key (FK) al atributo idCuitOCuil_Cliente perteneciente a la tabla servicios.

Aquí la imagen.



Conexión a base de datos:

Este tema se implementó en las clases PersonaDao y ServiciosDao, estableciendo en cada una de ellas una cadena de conexión a la base de datos.

```

/// <summary>
/// constructor estatico encargado de instanciar la cadena de conexion a la base de datos
/// </summary>
0 referencias
static ClienteDAO()
{
    cadenaConexion = "Server=.;Database=CLIENTES_DB;Trusted_Connection=True;";
}

```

También en cada uno de los métodos de estas clases se implementan instancias de objetos que permiten la interacción del usuario con la base de datos.
Aquí debajo una imagen

```

/// <summary>
/// metodo encargada de Leer la lista de Clientes de la base de datos.
/// </summary>
/// <returns>retornara una lista de clientes, caso contrario lanzara una excepcion de tipo ConexionSQLEception </returns>
/// <exception cref="ConexionSQLEception"></exception>
1 referencia
public static List<Cliente> LeerDatos()
{
    List<Cliente> listaClientes = new List<Cliente>();

    try
    {
        string queryClientes = "SELECT * FROM clientes";
        using(SqlConnection connection = new SqlConnection(ClienteDAO.cadenaConexion))
        {
            SqlCommand cmd = new SqlCommand(queryClientes, connection);
            connection.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            while(reader.Read())
            {
                Cliente.categoriaCliente categoriaCliente = (Cliente.categoriaCliente)reader.GetInt32(3);
                string nombreCliente = reader.GetString(1);
                string domicilioCliente = reader.GetString(4);
                string cuitOCuil = reader.GetString(2);
                int tipoCliente = reader.GetInt32(0);
                if(tipoCliente == 1)
                {
                    Cliente cliente = new ClienteEmpresa(categoriaCliente, nombreCliente, cuitOCuil, domicilioCliente);
                    listaClientes.Add(cliente);
                }
                else
                {
                    Cliente cliente = new ClienteParticular(categoriaCliente, nombreCliente, cuitOCuil, domicilioCliente);
                    listaClientes.Add(cliente);
                }
            }
        }
    }catch (Exception ex)
    {
        throw new ConexionSQLEception("Error al leer datos desde la base de datos", ex);
    }
    return listaClientes;
}

```

Delegados y Expresiones Lambda:

Delegados:

Este tema fué implementado en varias ocasiones con el objetivo que hagan referencia a métodos o definan un tipo de evento un ejemplo de implementación fué aquí donde el delegado `CargarBarraEventHandler` podrá hacer referencia a métodos o eventos que asocien a un método que no devuelva nada (`void`) y que reciba un entero (`int`) como parámetro :

```

public delegate void CargarBarraEventHandler(int porcentaje);
public delegate void BarraCargada();
6 referencias
public partial class ProgressBarForm : Form
{
    public event CargarBarraEventHandler cargarBarra;
}

```

Expresiones Lambda:

Al igual que delegado fué implementado en varias ocasiones, un ejemplo de implementación fué para definir el criterio de ordenamiento de la lista:

```

/// <summary>
/// Metodo encargado de actualizar la lista de clientes y habilitar la carga de servicios si la lista
/// cuenta con clientes cargados, utilizara un delegado del tipo Comparison para ordenar la lista por orden alfabetico de los nombres
/// </summary>
5 referencias
private void ActualizarListaClientes()
{
    try
    {
        Comparison<Cliente> comparador = (Cliente c1, Cliente c2) => c1.NombreCliente.CompareTo(c2.NombreCliente);
    }
}

```

Programación Multi-hilo y Concurrency:

Este tema también fué implementado varias veces, un ejemplo de implementación fué en para ejecutar el formulario Alta Clientes, el cuál se ejecutará en un hilo secundario permitiendo al usuario ver la actualización de la lista a la vez que se están cargando los clientes . Aquí su implementación.

```

/// <summary>
/// metodo perteneciente al evento Click del btnAltaCliente
/// instanciará un nuevo formulario de alta cliente el cual se ejecutara en un hilo secundario y permitira al usuario agregar clientes a la lista
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 referencia
private void btnAltaCliente_Click(object sender, EventArgs e)
{
    try
    {
        FrmAltaCliente altaCliente = new FrmAltaCliente(administracion, cancellationToken, cancellationTokenSource, ActualizarLista);
        Task.Run(() => IniciarAltaClientes(altaCliente, cancellationToken));
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Atencion", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}

/// <summary>
/// metodo encargado mostrar el formulario que se esta ejecutando en un hilo secundario
/// preguntara si el el manipulador del control del formulario se creo en otro hilo y como la primera vez
/// dara true invocara el delegado que referencia al metodo con los argumentos para permitir el control
///
/// </summary>
/// <param name="altaCliente"></param>
2 referencias
private void IniciarAltaClientes(Form altaCliente)
{
    if (this.InvokeRequired)
    {
        AbrirFormulario form = IniciarAltaClientes;
        object[] args = { altaCliente };
        this.BeginInvoke(form, args);
    }
    else
    {
        altaCliente.Show();
    }
}

```

Eventos:

Un ejemplo de la implementación de este tema fué en el método IniciarCargaBarra() del form “ProgressBarForm” en el cual invoca a los eventos cargarBarra y barraCargada los cuales están referenciados a dos métodos.

```

public delegate void CargarBarraEventHandler(int porcentaje);
public delegate void BarraCargada();
6 referencias
public partial class ProgressBarForm : Form
{
    public event CargarBarraEventHandler cargarBarra;
    public event BarraCargada barraCargada;
}

```

```

cargarBarra += CargandoBarra;
barraCargada += FinCargaBarra;

```

```

/// <summary>
/// metodo encargado de asignar un porcentaje a la propiedad value de la progressbar, al ser una tarea ejecutada en un hilo secundario
/// se llamará a la propiedad invokeRequired para poder ejercer cambios a un control del form desde un hilo secundario
/// </summary>
/// <param name="porcentajeCarga"></param>
1 referencia
private void CargandoBarra(int porcentajeCarga)
{
    if (this.InvokeRequired)
    {
        object[] args = { porcentajeCarga };
        this.BeginInvoke(cargarBarra, args);
    }
    else
    {
        this.progressBar1.Value = porcentajeCarga;
    }
}

/// <summary>
/// metodo encargado de controlar el estado de la barra, esta tarea al estar siendo ejecutada en un hilo secundario
/// se llamará a la propiedad InvokeRequired para que pueda ser ejecutada desde otro hilo y después si la barra se
/// cargo correctamente llamará al metodo close que cancelará el hilo
/// </summary>
1 referencia
private void FinCargaBarra()
{
    if (this.InvokeRequired)
    {
        this.BeginInvoke(horraCargada);
    }
    else
    {
        if (progressBar1.Value == 100)
        {
            this.Close();
        }
    }
}

```

y finalmente estos son invocados a través del método `IniciarCargaBarra()` del mismo form

```

/// <summary>
/// metodo encargado de simular el estado de porcentaje completado de la progressbar
/// sera el encargado de llamar a los eventos cargarBarra y barra cargada
/// </summary>
1 referencia
private void IniciarCargaBarra()
{
    int barraCompleta = 0;
    while(barraCompleta < 100)
    {
        barraCompleta += 10;
        if(this.cargarBarra is not null)
        {
            this.cargarBarra.Invoke(barraCompleta);
            Thread.Sleep(200);
        }
    }
    if(barraCargada is not null)
    {
        barraCargada.Invoke();
    }
}

```

Métodos de Extensión:

Este tema fue implementado en para la extensión del tipo de dato DOUBLE, realizando un método que permite sacar un porcentaje a una instancia de este tipo. Aquí debajo una imagen:

```
public static class MetodosDeExtension
{
    /// <summary>
    /// metodo encargado de calcular un porcentaje sobre un tipo de dato double
    /// </summary>
    /// <param name="precio">la instancia de la variable que lo llama</param>
    /// <param name="porcentaje">es el porcentaje sobre el cual se quiere hacer el calculo</param>
    /// <returns>retorna el porcentaje</returns>
    public static double CalcularPorcentaje(this double precio, double porcentaje)
    {
        return precio * porcentaje / 100;
    }
}
```

Orpelli Renzo División 2E.