

Trabajo practico N°4 Laboratorio II



Nombre y apellido: Gabriel Alegre.

Nota para chequear los temas utilizados en el TP 3: la explicación de cómo y dónde implemente los temas vistos entre la clase 10 y 15 es a partir de la página 7, igualmente, se recomienda leer lo anterior para entender el funcionamiento del programa.

Nota para chequear los temas utilizados en el TP 4: la explicación de cómo y dónde implemente los temas vistos entre la clase 15 y 21 es a partir de la página 17, igualmente, se recomienda leer lo anterior para entender el funcionamiento del programa.

NetCom es una empresa de telecomunicaciones. Su oferta de servicios incluye la provisión de servicios de televisión por cable, telefonía fija e Internet.

Los planes disponibles que se les ofrecerá a los clientes cuando se quieran dar de alta son los siguientes:

Plan básico:

Cantidad de megas de internet: 50

Incluye Cable: Si

Incluye Telefonía fija: No

Incluye Fibra óptica: No

Precio final del plan: 2000

Beneficios por plan básico:

- pack HBO gratis

Plan intermedio:

Cantidad de megas de internet: 100

Incluye Cable: Si

Incluye Telefonía fija: Si

Incluye Fibra óptica: No

Precio final del plan: 3500

Beneficios por plan intermedio:

- Disney+ gratis

- Paramount plus gratis

Plan premium:

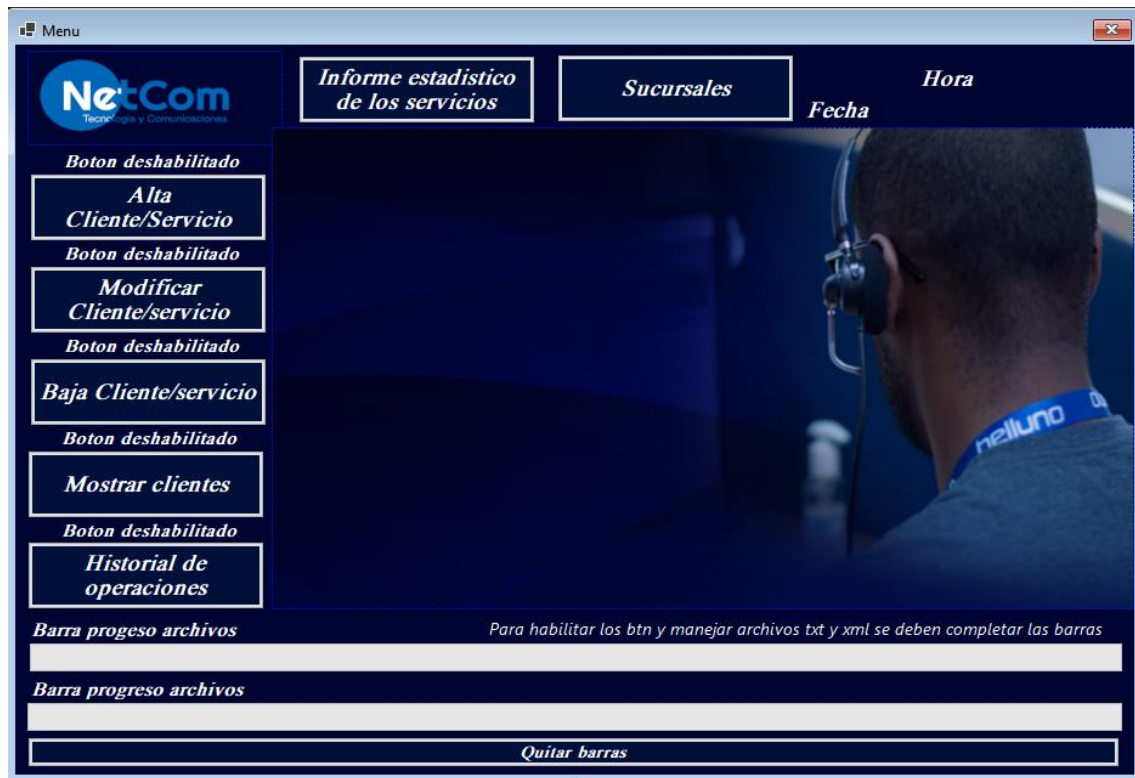
Cantidad de megas de internet: 1000

Incluye Cable: Si
Incluye Telefonía fija: Si
Incluye Fibra óptica: Si
Precio final del plan: 6000

Beneficios por contratar el plan premium:

- Netflix gratis
- Amazon prime video gratis
- Pack futbol gratis

Importante antes de arrancar: La especificación de la ruta donde están los archivos Xml, Json y txt está al final de la página 13.



Explicación de la funcionalidad del programa y como interactúa con cada botón:

Vamos a empezar por el Alta:

Al presionar el botón de alta nos pedirán el documento del cliente que se quiera dar de alta, al momento de ingresarlo (el documento) hay tres opciones:

- 1) Que el DNI corresponda a una persona que no esté en la lista de clientes, es decir, que sea un cliente totalmente nuevo y en tal caso se nos abrirá el siguiente form de alta para poder instanciar el cliente.

Alta de cliente

Numero de cliente asignado: 2

Nombre

Apellido

Direccion

Dni ingresado

Planes disponibles

☐ Plan basico

☐ Plan intermedio

☐ Plan premium

Detalles del plan basico

Cantidad de megas de internet: 50

Incluye Cable: Si

Incluye Telefonía fija: No

Incluye Fibra optica: No

Precio final del plan: 2000

Beneficios por plan basico:

- pack HBO gratis

Para dar de alta no debe haber campos vacios

Confirmar alta

Cancelar

2) Que el DNI corresponda a un cliente que ya activo, es decir, un cliente que ya este dado de alta en el sistema y en tal caso el programa nos avisara con el siguiente mensaje:

Aviso

Este cliente ya esta activo en el sistema con un plan adquirido, si quiere, lo puede modificar.

Aceptar

3) Que el DNI corresponda a un cliente que no activo (está en mi sistema, en mi lista de cliente pero con baja lógica), es decir, un cliente que en algún momento se dio de baja, y ahora quiere volver a darse de alta, en tal caso aparecerá el siguiente mensaje:

AVISO: Se espera respuesta

Esta persona ya fue cliente de NetCom en el pasado

- Seleccione SI: para darla de alta con lo mismos que tenia cuando era cliente
- Seleccione NO: para darla de alta con algun dato modificado

Sí

No

Cancelar


Si selecciona SI el cliente se dará de alta automáticamente con los mismos datos que tenía al momento de darse de baja. Al seleccionar NO, se abrirá un nuevo form para dar de alta al cliente permitiendo modificar alguno de sus atributos.

Botón Modificar:

Al presionar el botón modificar, se pedirá que se ingrese el DNI del cliente a modificar y verificara que dicho DNI ingresado realmente pertenezca a un cliente activo en el sistema (En la lista)

Si el DNI NO pertenece a un cliente activo, se avisará con el siguiente mensaje:

excepción (Propia) controlada

 excepción (propia) CONTROLADA:
- No existe ningun cliente activo con el dni ingresado, si quiere, lo puede dar de alta

Aceptar

En caso contrario, es decir, que el DNI ingresado pertenezca a un cliente activo del sistema, se abrirá el siguiente formulario autocompletado con los datos del cliente, una vez que se hayan modificado los datos correspondientes, a la derecha, esta la opción de ver como quedaría el usuario con dichas modificaciones.

Modificacion de datos

Estos son los datos actuales del cliente, modifique lo que corresponda

Nombre
Gabriel

Apellido
Alegre

Direccion
Av Mitre 750

Dni ingresado
43901903

El plan seleccionado actualmente es el que tenia el cliente, puede modificarlo

Planes disponibles

☐ Plan basico

☐ Plan intermedio

☐ Plan premium

Modificar

Cancelar

visibilidad

☐ Mostrar cliente con los datos originales

☐ Mostrar como quedaria el cliente con las modificaciones

Nombre: Gabriel
Apellido: Alegre
Domicilio: Av Mitre 750
DNI: 43901903
Numero de cliente: 3
Plan adquirido:
Detalles del plan premium
Cantidad de megas de internet: 1000
Incluye Cable: Si
Incluye Telefonía fija: Si
Incluye Fibra optica: Si
Precio final del plan: 6000

Beneficios por plan premium:
- Netflix gratis
- Amazon prime video gratis


Botón Baja:

Al presionar el botón de dar de Baja, se pedirá que se ingrese el DNI del cliente se quiere dar de baja y verificara que dicho DNI ingresado realmente pertenezca a un cliente activo en el sistema (En la lista)

Si el DNI NO pertenece a un cliente activo, se avisará con el siguiente mensaje:

excepción (Propia) controlada

×



excepción (propia) CONTROLADA:
- No existe ningun cliente activo con el dni ingresado, si quiere, lo puede dar de alta

Aceptar

En caso contrario, es decir, que el DNI ingresado pertenezca a un cliente activo del sistema, se abrirá el siguiente formulario, si se presiona el botón aceptar se dará de baja automáticamente al cliente.

Dar baja

×

Se dara el siguiente cliente de baja, esta seguro?

Nombre: Gabriel
Apellido: Alegre
Domicilio: Calle falsa 123
DNI: 43901903
Numero de cliente: 4
Plan adquirido:
Detalles del plan basico
Cantidad de megas de internet: 50
Incluye Cable: Si
Incluye Telefonía fija: No
Incluye Fibra optica: No
Precio final del plan: 2000

Beneficios por plan basico:
- pack HBO gratis

Aceptar

Botón Mostrar Clientes:

Al presionar el botón de mostrar clientes se abrirá el siguiente formulario, abajo, está la opción de mostrar los clientes activos o lo clientes no activos (Los dados de baja).

Listado de clientes

Listado de clientes activos:

Nombre: Daenerys

Apellido: Targaryen

Domicilio: Italia 874

DNI: 44902904

Numero de cliente: 1

Plan adquirido:

Detalles del plan premium

Cantidad de megas de internet: 1000

Incluye Cable: Si

Incluye Telefonía fija: Si

Incluye Fibra optica: Si

Precio final del plan: 6000

Beneficios por plan premium:

- Netflix gratis

- Amazon prime video gratis

- Pack futbol gratis

Nombre: Jon

Apellido: Snow

Domicilio: Av mitre 750

DNI: 45902905

Numero de cliente: 2

☐ Mostrar clientes activos☐ Mostrar clientes dados de baja**Aceptar****Botón Sucursal:**

Al presionar el botón sucursal se mostrará todas las sucursales que están cargadas en la base de datos a través de un data grid, también se dará la opción de poder filtrarlas por el nombre de la provincia, añadir una nueva sucursal a la base de datos o eliminarla. Esto estará mas desarrollado y especificado parte donde explico como y donde implemente SQL, pagina 17.

Sucursales que actualmente estan cargadas en la base de datos

IdSucursal	Provincia	Localidad	Direccion	Telefono
4	Chaco	Dock sud	Pieri 878	1143568790
5	Corrientes	Azul	California 464	1145675957
6	Chubut	Crucesita	San martin 2689	1146789875
8	Formosa	Pilcomayo	Brandsen 124	1123456789
12	Misiones	Iguazu	Suarez 356	1156896533
13	Neuquen	Catan Lil	Arenales 879	1166878877
14	Rio negro	Pichi Mahuida	Gral Paz 223	1146789875
15	Salta	Anta	French 454	1198752345
16	San Juan	Caucete	Lavalle 4256	1134578975
17	San Luis	Dupuy	Ameghino 232	1157987649
18	Santa Cruz	Ayacucho	Av. Roca 578	1198676464
19	Santa Fe	Ceres	Mones de Oca 986	1187907556
20	Santiago Del Estero	San Carlos	Helguera 234	1123644567
22	Tucuman	Monteros	Gutierrez 1212	1123644567

Filtrar por provincia

Todas

Añadir una sucursal**Quitar una sucursal****Salir**

Botón Historial de operaciones:

Al presionar el botón de historial de operaciones se abrirá el siguiente formulario que a partir de un archivo txt (la explicación de cómo y dónde lo implemente esta más adelante) que guarda todas las operaciones realizadas en forma de bitácora y cuando se presiona el botón y lo levanta, lo lee, y lo muestra:

■ Registro de operaciones ×

Historial/registro de operaciones

Adquirio el PlanBasico

domingo, 5 de junio de 2022 23:55hs - Se realizo el alta de un clienten:
Llamado: Mercedes
Con Dni: 47904907
Adquirio el PlanIntermedio

domingo, 5 de junio de 2022 23:58hs - Se realizo el alta de un clienten:
Llamado: Obi-Wan
Con Dni: 43334242
Adquirio el PlanPremium

lunes, 6 de junio de 2022 00:10hs - Se realizo la baja del cliente:
Llamado: Gabriel
Con Dni: 43901903

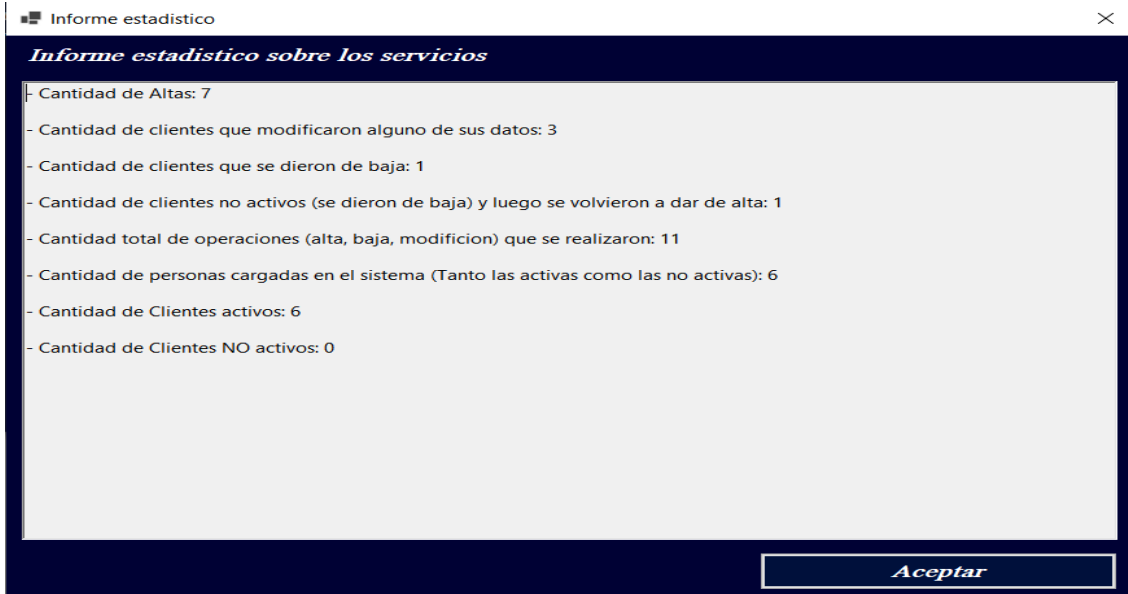
lunes, 6 de junio de 2022 00:10hs - Se realizo el alta de un cliente que anteriormente se habia dado de baja
Llamado: Gabriel
Con DNI: 43901903

lunes, 6 de junio de 2022 00:56hs - Se realizo una modificacion de datos al cliente:
Llamado: Daenerys
Con Dni: 44902904

Aceptar

Botón de los informes estadísticos:

Al presionar el botón se abre el siguiente formulario el informe estadístico del programa, cantidad de altas dadas, cantidad de modificaciones realizadas, cantidad de bajas, etc.



Informe estadístico

Informe estadístico sobre los servicios

- Cantidad de Altas: 7
- Cantidad de clientes que modificaron alguno de sus datos: 3
- Cantidad de clientes que se dieron de baja: 1
- Cantidad de clientes no activos (se dieron de baja) y luego se volvieron a dar de alta: 1
- Cantidad total de operaciones (alta, baja, modificacion) que se realizaron: 11
- Cantidad de personas cargadas en el sistema (Tanto las activas como las no activas): 6
- Cantidad de Clientes activos: 6
- Cantidad de Clientes NO activos: 0

Aceptar

Temas vistos entre la clase 10 y 15 utilizados

Excepciones:

Primera excepción:

```
6 referencias
public class NoExisteClienteActivoConElDniIngresadoException : Exception
{
    /// <summary>
    /// </summary>
    /// <param name="message">Mensaje que describe el porque se produjo la excepcion</param>
    1 referencia
    public NoExisteClienteActivoConElDniIngresadoException(string message) : this(message, null)
    {
    }

    /// <summary>
    /// </summary>
    /// <param name="message">Mensaje que describe el porque se produjo la excepcion</param>
    /// <param name="innerException">Inner exception de la excepcion</param>
    1 referencia
    public NoExisteClienteActivoConElDniIngresadoException(string message, Exception innerException) : base(message, innerException)
    {
    }
}
```

Cuando se quiera modificar o dar de baja a un cliente, se pedirá el ingreso del DNI para confirmar la existencia de dicho cliente en el servidor, en caso de que el DNI ingresado NO pertenezca a ningún cliente activo, se lanzara dicha excepción

Método: donde se lanza la excepción: VerificarSiElClienteExisteYEstaActivoEnElSistema de la clase: Central administradora. Línea: 152


```

2 referencias
public static bool VerificarSiElClienteExisteYEstaActivoEnElSistema(int documento)
{
    bool estaEnElSistemaActivo;

    if (BuscarClienteActivoPorDni(documento) is not null)
    {
        estaEnElSistemaActivo = true;
    }
    else
    {
        throw new NoExisteClienteActivoConElDniIngresadoException("excepción (propia) CONTROLADA:\n- No existe ningun cliente activo con el dni");
    }

    return estaEnElSistemaActivo;
}

```

Esta excepción será capturada (Si es que corresponde, es decir, si ingresan un DNI que NO pertenezca a ningún cliente activo) En el formulario: FormIngresoDeDni. En el Método: VerificarQueElClienteEsteEnElSistema. Línea: 125

```

private void VerificarQueElClienteEsteActivoEnElSistema()
{
    try
    {
        if (CentralAdministradora.VerificarSiElClienteExisteYEstaActivoEnElSistema(dni))
        {
            this.DialogResult = DialogResult.OK;
        }
    }
    catch (NoExisteClienteActivoConElDniIngresadoException e)
    {
        this.DialogResult = DialogResult.Cancel;
        MessageBox.Show(e.Message, "excepción (Propia) controlada", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

Segunda excepción:

```

13 referencias
public class FallaDeArchivoException : Exception
{
    /// <summary>
    /// </summary>
    /// <param name="message">Mensaje que describe el porque se produjo la excepcion</param>
    7 referencias
    public FallaDeArchivoException(string message) : this(message, null)
    {
    }

    /// <summary>
    /// </summary>
    /// <param name="message">Mensaje que describe el porque se produjo la excepcion</param>
    /// <param name="innerException">Inner exception de la excepcion</param>
    1 referencia
    public FallaDeArchivoException(string message, Exception innerException) : base(message, innerException)
    {
    }
}

```

Esta excepción será lanzada al momento de querer guardar/serializar o leer/deserializar archivos y ocurra un fallo. Por ejemplo, una de las implementaciones es:

Clase: Central administradora. Métodos Guardar, Leer. Línea: 174 y 194 respectivamente.

```

2 referencias
public void Guardar(string ruta, string contenido)
{
    try
    {
        using (StreamWriter sw = new StreamWriter(ruta, true))
        {
            sw.WriteLine(contenido);
        }
    }
    catch (Exception)
    {
        throw new FallaDeArchivoException("Excepcion propia controlada:\n falla querer guardar el archivo txt del historial");
    }
}

/// <summary>
/// Metodo encargado de leer un archivo txt
/// </summary>
/// <param name="ruta">Ruta que señalan la ubicación del archivo</param>
/// <returns>El contenido del archivo en formato string, en caso contrario (que surja un error) lanza una excepción propia</returns>
2 referencias
public string Leer(string ruta)
{
    string contenidoDelArchivo = string.Empty;
    try
    {
        if (File.Exists(ruta) && new FileInfo(ruta).Length > 0)
        {
            using (StreamReader sw = new StreamReader(ruta))
            {
                contenidoDelArchivo = sw.ReadToEnd();
            }
        }
    }
    catch (Exception)
    {
        throw new FallaDeArchivoException("Excepcion propia controlada:\n falla al querer leer el archivo txt del historial");
    }
    return contenidoDelArchivo;
}

```

Esta excepción será capturada en el formClosing del form principal línea 238.

```

2 referencias
private void FormPrincipalMenu_FormClosing(object sender, FormClosingEventArgs e)
{
    try
    {
        serializadoraXml.Guardar(serializadoraXml.RutaArchivoXml, CentralAdministradora.ListaDeClientes);
        estadisticasDeLosServicios.Guardar(estadisticasDeLosServicios.RutaSerializarJson, estadisticasDeLosServicios);
        estadisticasDeLosServicios.calcularCantidadDeOperacionesQueSeRealizaron();
        estadisticasDeLosServicios.actualizarCantidadDeGenteActivaYnoActiva();
        MessageBox.Show("Datos guardados exitosamente", "Aviso", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (FallaDeArchivoException ex)
    {
        MessageBox.Show($"{ex.Message}", "Excepcion controlada", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"{ex.Message}", "Excepcion controlada", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

Test Unitarios:

En el proyecto “TestUnitarios” hay varios test, la mayoría para testear los métodos relacionados a la búsqueda de clientes.

Prueba	Duración	Resultado	Mensaje
TestUnitarios (3)	3 ms	✓	
TestUnitarios (3)	3 ms	✓	
TestUnitarios (3)	3 ms	✓	
BuscarClienteNoActivoEnLaList...	2 ms	✓	
VerificarSiClienteEstaEnElSistem...	< 1 ms	✓	
VerificarSiElClienteExisteYEstaA...	1 ms	✓	



TestUnitarios



Dependencias



TestUnitarios.cs

```
[TestMethod]
[ExpectedException(typeof(NoExisteClienteActivoConElDniIngresadoException))]
0 referencias
public void VerificarSiElClienteExisteYEstaActivoEnElSistema_CuandoElClienteNoExista_DeberiaLanzarNoExisteClienteActivoConElDniIngresadoException()
{
    // Arrange - instancio un cliente que no esta en mi sistema
    Cliente unCliente = new Cliente("Juan", "Perez", "Av. Mitre 1346", 895485765, new PlanPremium());

    // Act - Lo busco en el sistema, deberia lanzarme la excepcion xq dicho cliente no existe en mi sistema
    bool resultado = CentralAdministradora.VerificarSiElClienteExisteYEstaActivoEnElSistema(895485765);
}

[TestMethod]
0 referencias
public void VerificarSiClienteEstaEnElSistema_CuandoElClienteEsteEnElSistemaYActivo_DeberiaRetornarTrue()
{
    // Arrange - instancio un cliente que no esta en mi sistema y luego lo agrego
    Cliente unCliente = new Cliente("Juan", "Perez", "Av. Mitre 1346", 44965932, new PlanIntermedio());
    CentralAdministradora.ListaDeClientes.Add(unCliente);

    // Act - Verifico si el cliente esta en el sistema, deberia retornar true xq el si esta
    bool resultado = CentralAdministradora.VerificarSiClienteEstaEnElSistema(44965932);

    Assert.IsTrue(resultado);
}

[TestMethod]
0 referencias
public void BuscarClienteNoActivoEnLaLista_CuandoElClienteEsteActivo_DeberiaRetornarNull()
{
    // Arrange - instancio un cliente que no esta en mi sistema y luego lo agrego
    Cliente unCliente = new Cliente("Luis", "Miguel Hernandez", "Av. Belgrano 656", 43901903, new PlanIntermedio());
    unCliente.EstaActivo = true;
    CentralAdministradora.ListaDeClientes.Add(unCliente);

    // Act - Deberia retornar null xq el cliente esta activo
    Cliente resultado = CentralAdministradora.BuscarClienteNoActivoEnLaLista(43901903);

    Assert.IsNull(resultado);
}
```

Interfaz y tipos genéricos

Interfaz genérica por un tema de practicidad ya que necesito guardar/serializar y leer/deserializar distintos objetos en distintas clases

```
3 referencias
public interface IArchivos<T>
{
    /// <summary>
    /// Guarda un objeto T en un archivo en una ruta específica, este metodo sera implemenado en las clases que implemeten esta interfaz.
    /// </summary>
    /// <param name="ruta">Ruta que señalan la ubicación del archivo</param>
    /// <param name="contenido">T Objeto a guardar dentro del archivo.</param>
    9 referencias
    void Guardar(string ruta, T contenido);

    /// <summary>
    /// Lee el archivo contenido en la ruta y devuelve un tipo de objeto T, este metodo sera implemenado en las clases que implemeten esta interfaz..
    /// </summary>
    /// <param name="ruta">Ruta que señalan la ubicación del archivo</param>
    /// <returns>Un objeto de tipo T</returns>
    8 referencias
    T Leer(string ruta);
}
```

Implementación de la interfaz en la clase Central Administradora

```
28 referencias
public class CentralAdministradora : IArchivos<string>
{
}
```

Con los respectivos métodos: en este caso quería leer y guardar archivos txt

Linea del método guardar: 174

Linea del método leer: 194

```
163 | 5 referencias
164 | public void Guardar(string ruta, string contenido)
165 | {
166 |     try
167 |     {
168 |         using (StreamWriter sw = new StreamWriter(ruta, true))
169 |         {
170 |             sw.WriteLine(contenido);
171 |         }
172 |     }
173 |     catch (Exception)
174 |     {
175 |         throw new FallaDeArchivoException("Excepcion propia controlada:\n falla querer guardar el archivo txt del historial");
176 |     }
177 | }
178 |
179 | /// <summary>
180 | /// Metodo encargado de leer un archivo txt
181 | /// </summary>
182 | /// <param name="ruta">Ruta que señalan la ubicación del archivo</param>
183 | /// <returns>El contenido del archivo en formato string, en caso contrario (que surga un error) lanza una excepción propia</returns>
184 | 2 referencias
185 | public string Leer(string ruta)
186 | {
187 |     string contenidoDelArchivo = string.Empty;
188 |     try
189 |     {
190 |         if (File.Exists(ruta) && new FileInfo(ruta).Length > 0)
191 |         {
192 |             using (StreamReader sw = new StreamReader(ruta))
193 |             {
194 |                 contenidoDelArchivo = sw.ReadToEnd();
195 |             }
196 |         }
197 |     }
198 |     catch (Exception)
199 |     {
200 |         throw new FallaDeArchivoException("Excepcion propia controlada:\n falla al querer leer el archivo txt del historial");
201 |     }
202 |     return contenidoDelArchivo;
203 | }
```

Implementación de la interfaz en la ClaseSerializadoraXml

```
3 referencias
public class ClaseSerializadoraXml : IArchivos<List<Cliente>>
{
```

Con los respectivos métodos: en este caso quería serializar y deserializar la lista de clientes en un archivo XML

Linea del método guardar: 33

Linea del método leer: 54

```
2 referencias
public virtual void Guardar(string ruta, List<Cliente> contenido)
{
    try
    {
        using (StreamWriter sw = new StreamWriter(ruta))
        {
            XmlSerializer xml = new XmlSerializer(typeof(List<Cliente>));
            xml.Serialize(sw, contenido);
        }
    }
    catch (Exception)
    {
        throw new FallaDeArchivoException("Excepcion. No se pudo serializar el archivo correctamente");
    }
}

/// <summary>
/// Metodo encargado de deserializar un archivo XML
/// </summary>
/// <param name="ruta">Ruta que señalan la ubicación del archivo XML</param>
/// <returns>La lista de cliente contenida en el archivo o null en caso que el archivo no exista o este vacio, en caso contrario (que surga un
3 referencias
public virtual List<Cliente> Leer(string ruta)
{
    try
    {
        //Agregue el File.Exists(ruta) si empieza a fallar es por eso
        if (File.Exists(ruta) && new FileInfo(ruta).Length > 0)
        {
            using (StreamReader sr = new StreamReader(ruta))
            {
                XmlSerializer xml = new XmlSerializer(typeof(List<Cliente>));
                List<Cliente> objeto = xml.Deserialize(sr) as List<Cliente>;
                return objeto;
            }
        }
        else
        {
            return null;
        }
    }
    catch (Exception)
    {
        throw new FallaDeArchivoException("Excepcion. No se pudo deserializar el archivo correctamente");
    }
}
```

Implementación de la interfaz en la clase EstadisticaServicios

```
11 referencias
public class EstadisticaServicios : IArchivos<EstadisticaServicios>
{
```

Con los respectivos métodos: en este caso quería serializar y deserializar en json

Linea del método guardar: 91

Linea del método leer: 112

```

2 referencias
public void Guardar(string ruta, EstadisticaServicios contenido)
{
    try
    {
        JsonSerializerOptions opciones = new JsonSerializerOptions();
        opciones.WriteIndented = true;

        string objJson = JsonSerializer.Serialize(contenido, opciones);
        File.WriteAllText(ruta, objJson);
    }
    catch (Exception)
    {
        throw new FallaDeArchivoException("Excepcion: Fallo la serializacion del archivo Json");
    }
}

/// <summary>
/// Metodo encargado de deserializar Json
/// </summary>
/// <param name="ruta">Ruta que señalan la ubicación del archivo Json</param>
/// <returns>Objeto de tipo EstadisticaServicios deserializado o null en caso que el archivo no exista o es
3 referencias
public EstadisticaServicios Leer(string ruta)
{
    try
    {
        if (File.Exists(ruta) && new FileInfo(ruta).Length > 0)
        {
            JsonSerializerOptions opciones = new JsonSerializerOptions();
            opciones.WriteIndented = true;
            string ContenidoArchivoJson = File.ReadAllText(ruta);
            return JsonSerializer.Deserialize<EstadisticaServicios>(ContenidoArchivoJson);
        }
        else
        {
            return null;
        }
    }
    catch (Exception)
    {
        throw new FallaDeArchivoException("Excepcion: Fallo la deserializacion del archivo Json");
    }
}

```

Archivos txt

El archivo txt lo utilice para escribir el historial de las operaciones realizadas. Cada vez que se realiza una operación como, por ejemplo, dar de alta a un cliente, modificar algún dato de los clientes o dar de baja se llamara al método guardar de la clase CentralAdministradora que se encarga de escribir el txt con la fecha, hora, y la operación realizada:

Clase y líneas en las cuales escribo el historial:

```

▲ Entidades\CentralAdministradora.cs (1)
  212 : Guardar(ruta, $"{DateTime.Now:f}hs - Se realizo el alta de un cliente:\nLlamado: {unCliente.Nombre}\nCon Dni: {unCliente.Dni}\nAdquirio el {unCliente.PlanEligido.GetType().Name}\n");

▲ Entidades\Archivos.cs (1)
  16 : void Guardar(string ruta, T contenido);

▲ Forms\FormIngresoDeDni.cs (1)
  196 : centralAdministradora.Guardar(centralAdministradora.Ruta, mensajeDelHistorial);

▲ Forms\FormPrincipalMenu.cs (2)
  77 : centralServicio.Guardar(centralServicio.Ruta, $"{DateTime.Now:f}hs - Se realizo una modificacion de datos al cliente:\nLlamado: {clienteParaModificar.Nombre}\nCon Dni: {clienteParaModificar.Dni}\n");
  106 : centralServicio.Guardar(centralServicio.Ruta, $"{DateTime.Now:f}hs - Se realizo la baja del cliente:\nLlamado: {clienteQueSeDaraDeBaja.Nombre}\nCon Dni: {clienteQueSeDaraDeBaja.Dni}\n");

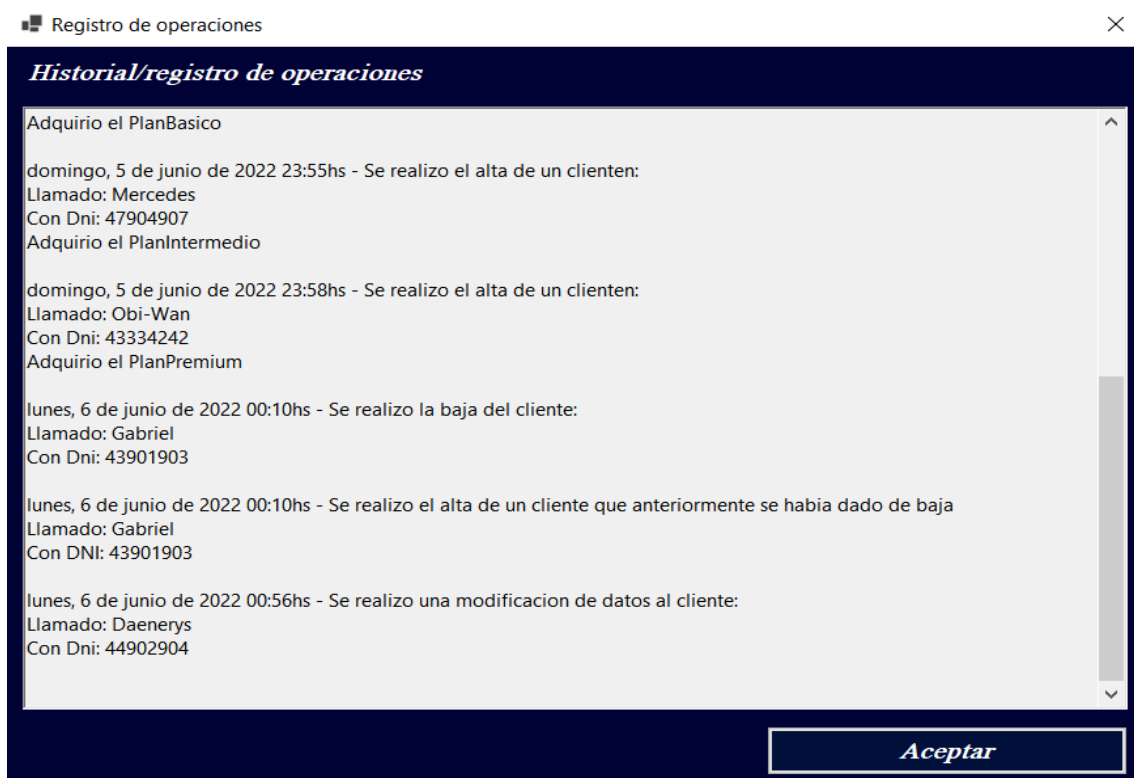
```

El archivo txt se levanta/lee cada en el evento click del btnHistorial del FormPrincipalMenu

Línea: 150

```
1 referencia
private void btnHistorial_Click(object sender, EventArgs e)
{
    try
    {
        string historialParaMostrar = centralServicio.Leer(centralServicio.Ruta);
        this.llamarFormMostrar("Registro de operaciones", "Historial/registro de operaciones", historialParaMostrar);
    }
    catch (FallaDeArchivoException ex)
    {
        MessageBox.Show(ex.Message, "Exception controlada", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Para acceder a dicho historial hay un botón en el form principal llamado: “Historial de operaciones” que mostrara el contenido del archivo txt:



Serializacion XML y Json

Importante: Los tantos archivos XML, Json y Txt serán encontrados en la siguiente ruta:

```
"${AppDomain.CurrentDomain.BaseDirectory}"
s\TP_[3]\Alegre.Gabriel.2A.TP3\Forms\bin\Debug
```

Xml: Utilizo serialización xml para serializar la lista de clientes, ya que cada cliente tiene un plan asociado y dicho plan tiene una relación de herencia.

Json: Utilizo la serialización Json para poder hacer un informe estadístico del programa, sobre la atención al público, ya que si no lo hacía con archivo (el informe estadístico) al momento de cerrar el programa iba a perder la cuenta de la cantidad de altas que se realizaron, cantidad de bajas, cuantas veces modificaron a un cliente etc.

Dichos archivos Xml y Json son levantados/Deserializados al momento de abrir la aplicación, en el evento load del FormPrincipalMenu. Línea: 53

```
1 referencia
private void FormPrincipalMenu_Load(object sender, EventArgs e)
{
    Deserializar();
}
```

El método encargado de Deserializar se encuentra en el FormPrincipalMenu. Línea: 199

```
1 referencia
private void Deserializar()
{
    try
    {
        if (serializadoraXml.Leer(serializadoraXml.RutaArchivoXml) is not null)
        {
            CentralAdministradora.ListaDeClientes = serializadoraXml.Leer(serializadoraXml.RutaArchivoXml);
        }

        if (estadisticasDeLosServicios.Leer(estadisticasDeLosServicios.RutaSerializarJson) is not null)
        {
            estadisticasDeLosServicios = estadisticasDeLosServicios.Leer(estadisticasDeLosServicios.RutaSerializarJson);
        }
    }
    catch (FallaDeArchivoException ex)
    {
        MessageBox.Show($"{ex.Message}", "Excepcion controlada", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"{ex.Message}", "Excepcion controlada", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Y estos archivos xml y json serán serializados al momento de cerrar la aplicación, en el evento formClosing del FormPrincipalMenu, Línea: 228

```
1 referencia
private void FormPrincipalMenu_FormClosing(object sender, FormClosingEventArgs e)
{
    try
    {
        serializadoraXml.Guardar(serializadoraXml.RutaArchivoXml, CentralAdministradora.ListaDeClientes);
        estadisticasDeLosServicios.Guardar(estadisticasDeLosServicios.RutaSerializarJson, estadisticasDeLosServicios);
        estadisticasDeLosServicios.calcularCantidadDeOperacionesQueSeRealizaron();
        estadisticasDeLosServicios.actualizarCantidadDeGenteActivaYnoActiva();
        MessageBox.Show("Datos guardados exitosamente", "Aviso", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (FallaDeArchivoException ex)
    {
        MessageBox.Show($"{ex.Message}", "Excepcion controlada", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"{ex.Message}", "Excepcion controlada", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```


SQL

Sql lo utilice para guardar y recuperar todas las sucursales que se encuentren en la base de datos, al presionar el botón “Sucursales” del form principal nos abrirá el siguiente formulario con todas las sucursales que se encuentren en la base de datos:

The screenshot shows a window titled 'FormSucursal' with a close button in the top right corner. The main content area is titled 'Sucursales que actualmente estan cargadas en la base de datos'. It contains a table with 5 columns: 'IdSucursal', 'Provincia', 'Localidad', 'Direccion', and 'Telefono'. The table lists 14 branches. To the right of the table is a dropdown menu labeled 'Filtrar por provincia' with 'Todas' selected. Below the table are three buttons: 'Añadir una sucursal', 'Quitar una sucursal', and 'Salir'.

IdSucursal	Provincia	Localidad	Direccion	Telefono
4	Chaco	Dock sud	Pieri 878	1143568790
5	Corrientes	Azul	California 464	1145675957
6	Chubut	Crucesita	San martin 2689	1146789875
8	Formosa	Pilcomayo	Brandsen 124	1123456789
12	Misiones	Iguazu	Suarez 356	1156896533
13	Neuquen	Catan Lil	Arenales 879	1166878877
14	Rio negro	Pichi Mahuida	Gral Paz 223	1146789875
15	Salta	Anta	French 454	1198752345
16	San Juan	Caucete	Lavalle 4256	1134578975
17	San Luis	Dupuy	Ameghino 232	1157987649
18	Santa Cruz	Ayacucho	Av. Roca 578	1198676464
19	Santa Fe	Ceres	Mones de Oca 986	1187907556
20	Santiago Del Estero	San Carlos	Helguera 234	1123644567
22	Tucuman	Monteros	Gutierrez 1212	1123644567

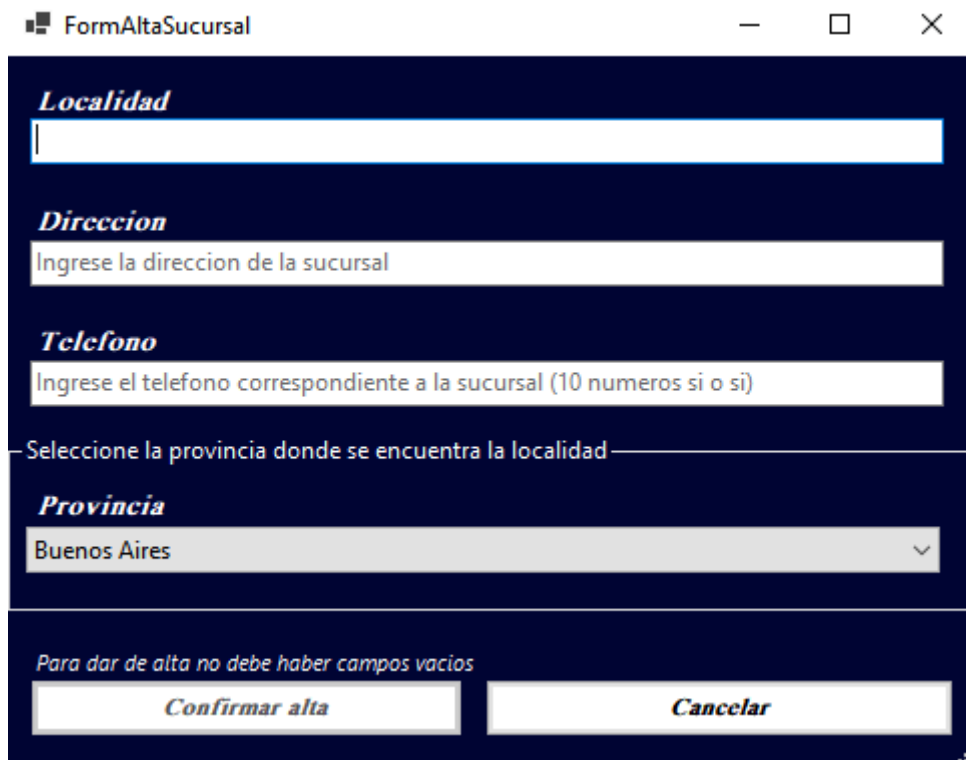
A la derecha tendremos la opción de filtrar por provincia:

This image is a close-up of the 'Filtrar por provincia' dropdown menu. The menu is open, showing a list of provinces. 'Todas' is selected at the top. The list includes: Buenos Aires, Catamarca, Chaco, Chubut, Cordoba, Corrientes, Entre Rios, Formosa, Jujuy, La Pampa, La Rioja, Mendoza, Misiones, Neuquen, Rio Negro, Salta, San Juan, San Luis, Santa Cruz, Santa Fe, Santiago Del Estero, Tierra Del Fuego, and Tucuman. The background shows parts of the table and buttons from the previous screenshot.

Al seleccionar una provincia se mostrará en el data grid todas las sucursales que se encuentre en la provincia seleccionada, si seleccionamos la opción de “Todas” se nos mostrará todas las sucursales independientemente de que provincia se encuentre.

El método que se encarga de recuperar (Todas las sucursales) y filtrar las sucursales por provincias que se encuentran en la base de datos se llama “Leer”, se encuentra en la clase “SqlSucursalesClass”, en la línea 88.

- Para agregar una sucursal en la base de datos se deberá presionar el botón “Anadir una sucursal”, Al presionarlo nos pedirá que ingresemos los datos de la sucursal que se desea añadir:



El método que se encarga de esto se llama “GuardarSucursal”, se encuentra en la clase “SqlSucursalesClass”, en la línea 32 y es el siguiente:

```
1 referencia
public static void GuardarSucursal(Sucursal unaSucursal)
{
    try
    {
        command.Parameters.Clear();
        connection.Open();
        command.CommandText = $"INSERT INTO Sucursales (provincia, localidad, direccion, telefono) VALUES (@Provincia, @Localidad, @Direccion, @Telefono)";
        command.Parameters.AddWithValue("@Direccion", unaSucursal.Direccion);
        command.Parameters.AddWithValue("@Localidad", unaSucursal.Localidad);
        command.Parameters.AddWithValue("@Provincia", unaSucursal.Provincia);
        command.Parameters.AddWithValue("@Telefono", unaSucursal.Telefono);
        command.ExecuteNonQuery();
    }
    catch (Exception)
    {
        throw;
    }
    finally
    {
        connection.Close();
    }
}
```

- Para eliminar una sucursal de la base de datos, primero se debe seleccionar una de la data grid, por ejemplo, en la siguiente foto seleccione la sucursal con ID 15 que se encuentra en la provincia de Salta, en la localidad Anta. Una vez seleccionada se tiene que presionar el botón “Quitar una sucursal”.

Sucursales que actualmente estan cargadas en la base de datos

IdSucursal	Provincia	Localidad	Direccion	Telefono
4	Chaco	Dock sud	Pieri 878	1143568790
5	Corrientes	Azul	California 464	1145675957
6	Chubut	Crucesita	San martin 2689	1146789875
8	Formosa	Pilcomayo	Brandsen 124	1123456789
12	Misiones	Iguazu	Suarez 356	1156896533
13	Neuquen	Catan Lil	Arenales 879	1166878877
14	Rio negro	Pichi Mahuida	Gral Paz 223	1146789875
15	Salta	Anta	French 41	198752345
16	San Juan	Caucete	Lavalle 4256	1134578975
17	San Luis	Dupuy	Ameghino 232	1157987649
18	Santa Cruz	Ayacucho	Av. Roca 578	1198676464
19	Santa Fe	Ceres	Mones de Oca 986	1187907556
20	Santiago Del Estero	San Carlos	Helguera 234	1123644567
22	Tucuman	Monteros	Gutierrez 1212	1123644567

Filtrar por provincia: Todas

Añadir una sucursal Quitar una sucursal Salir

El método encargado de eliminar la sucursal se llama “Eliminar”, se encuentra en la clase “SqlSucursalesClass”, en la línea 60 y es el siguiente:

```
1 referencia
public static int Eliminar(int id)
{
    int columnasAfectadas = 0;
    try
    {
        command.Parameters.Clear();
        connection.Open();
        command.CommandText = $"DELETE FROM Sucursales WHERE idSucursal = @Id";
        command.Parameters.AddWithValue("@Id", id);
        columnasAfectadas = command.ExecuteNonQuery();
    }
    catch (Exception)
    {
        throw;
    }
    finally
    {
        connection.Close();
    }

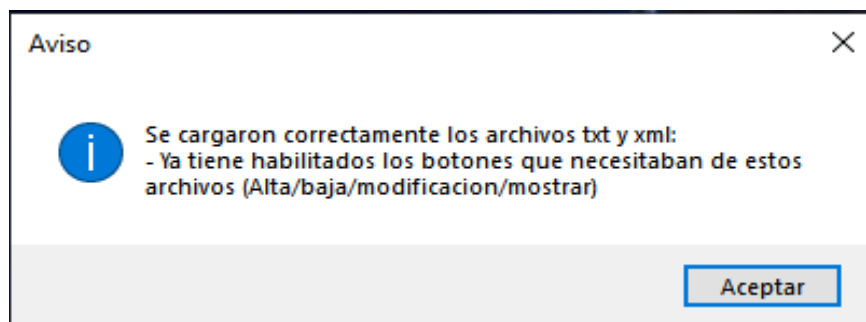
    return columnasAfectadas;
}
```

Delegados y expresiones lambda y Programación multi-hilo y concurrencia

Hilos: Utilice hilos para simular la carga de archivos XML y TXT a través de dos progressBar, lo hice con hilos ya que las progressBar tienen que correr en hilos paralelos, en subprocesos, para que no se bloquee al formulario principal, **mientras que las progressBar van incrementando estarán deshabilitadas las opciones de dar alta, baja, modificación, mostrar clientes e historial**, ya que estos requieren de los archivos txt y xml para funcionar. **Igualmente, mientras que las barras están cargando se puede seguir interactuando con el formulario**, por ejemplo, se puede ver el informe estadístico o ver, dar de alta, eliminar sucursales, etc. cómo se puede ver en la siguiente foto, que mientras que se están cargando las barras puedo acceder a las sucursales:

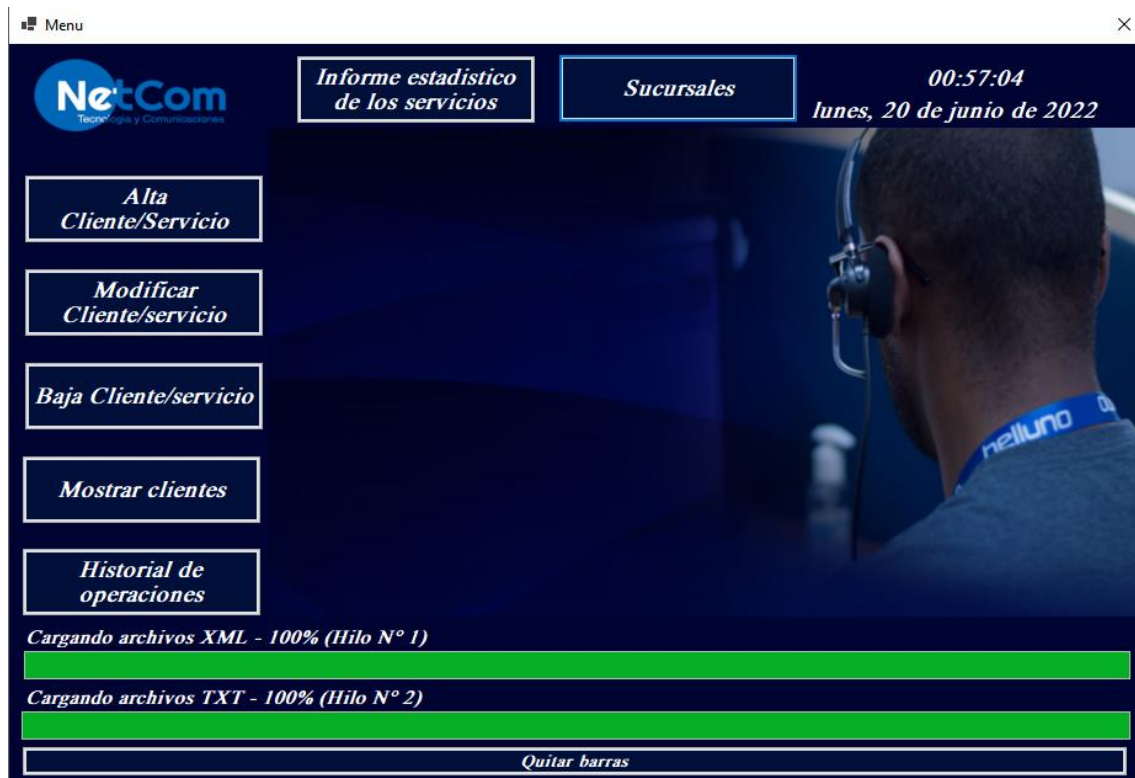
IdSucursal	Provincia	Localidad	Direccion	Telefono
4	Chaco	Dock sud	Pieri 878	1143568790
5	Corrientes	Azul	California 464	1145675957
6	Chubut	Crucesita	San martin 2689	1146789875
8	Formosa	Pilcomayo	Brandsen 124	1123456789
12	Misiones	Iguazu	Suarez 356	1156896533
13	Neuquen	Catan Lil	Arenales 879	1166878877
14	Rio negro	Pichi Mahuida	Gral Paz 223	1146789875
15	Salta	Anta	French 454	1198752345
16	San Juan	Caucete	Lavalle 4256	1134578975
17	San Luis	Dupuy	Ameghino 232	1157987649
18	Santa Cruz	Ayacucho	Av. Roca 578	1198676464
19	Santa Fe	Ceres	Mones de Oca 986	1187907556
20	Santiago Del Estero	San Carlos	Helguera 234	1123644567
22	Tucuman	Monteros	Gutierrez 1212	1123644567

Una vez que las progressBar estén cargadas al 100% se avisara con el siguiente mensaje:



Y se habilitaran todos los botones que estaban deshabilitados que eran para dar de alta, baja, modificación, mostrar clientes e historial, y se tendrá disponible la opción de poder quitar las

barras (Como se ve en la siguiente foto), ya que estas pueden resultar un poco molestas a la hora de atender al público.



Lambda: Usados en la línea 57 y 58 del “FormPrincipalMunu” en el evento load

```
private void FormPrincipalMenu_Load(object sender, EventArgs e)
{
    Deserializar();

    Task tareaBarraProgresoXlm = Task.Run(() => ComenzarProceso(progressBarXml, lblBarraProgresoXml, cancellationTokenSource), token);
    Task tareaBarraProgresoTxt = Task.Run(() => ComenzarProceso(progressBarTxt, lblBarraProgresoTxt, cancellationTokenSource), token);
}
```

Delegados: en el siguiente método: (AumentarBarra, línea 287 del “FormPrincipalMunu”)

```
2 referencias
private void AumentarBarra(ProgressBar barraDeProgreso, Label labelInformacionDelProgreso, string idHilo)
{
    if(InvokeRequired)
    {
        delegadoAumentarBarraProceso = AumentarBarra;
        object[] arrayObjParam = new object[] { barraDeProgreso, labelInformacionDelProgreso, idHilo };
        Invoke(delegadoAumentarBarraProceso, arrayObjParam);
    }
    else
    {
        barraDeProgreso.Increment(1);
        configuracionDeControles.Invoke();
        if(idHilo == "1")
        {
            labelInformacionDelProgreso.Text = $"Cargando archivos XML - {barraDeProgreso.Value}% (Hilo N° {idHilo})";
        }
        else
        {
            labelInformacionDelProgreso.Text = $"Cargando archivos TXT - {barraDeProgreso.Value}% (Hilo N° {idHilo})";
        }
    }
}
```

Necesito hacer una llamada recursiva, ya que al usar hilos, subprocesos, para las progressBar, Windows forms no te permite interactuar con controles del formulario (las progressBar y los labels) desde un hilo secundarios, entonces cree el método (El de la foto de arriba) para solucionar este problema, **si el invokeRequired da true** significa que estoy en el hilo secundario, entonces **tengo que llamar al mismo método** (AumentarBarra) desde el hilo principal, **para ello utilizo el método Invoke**, que **este necesita un delegado**, este delegado va a ser el que vamos a invocar desde el método principal, **que es el siguiente**:

Declarado en la línea 25 del "FormPrincipalMunu"

```
26 private delegadoProgressBar delegadoAumentarBarraProceso;
27
```

Utilizados en el método AumentarBarra, línea 291 y 293 del "FormPrincipalMunu"

```
293 delegadoAumentarBarraProceso = AumentarBarra;
294 object[] arrayObjParam = new object[] { barraDeProgreso, labelInformacionDelProgreso, idHilo };
295 Invoke(delegadoAumentarBarraProceso, arrayObjParam);
296
```

También, utilice delegados para los eventos, la clase "CentralAdministradora" en la línea 15:

```
10 30 referencias
11 public class CentralAdministradora : IArchivos<string>
12 {
13     private static List<Cliente> listaDeClientes;
14     private string ruta = $"{AppDomain.CurrentDomain.BaseDirectory}" + @"HistorialOperaciones.txt";
15     public delegate void DelegadoEventos(string mensaje);
16
17     public static event DelegadoEventos EventAvisadorQueClienteNoExiste;
18     public static event DelegadoEventos EventAvisadorQueClienteExisteVEstaActivo;
19     0 referencias
```

Eventos

Utilice tres eventos:

El primer evento para que mientras que la progressBar (que simula la carga de los archivos xml y txt) no este completa, es decir, que no llegue al 100% deshabilite los botones (de dar alta, baja, modificación, mostrar clientes) que requieren que dicha progressBar este llena, y cuando barra se complete los vuelve a habilitar y envía un mensaje en modo notficativo diciendo que los archivos txt y xml se cargaron sin problemas.

Dicho evento esta declarado en la línea 28 del "FormPrincipalMunu"

```
public delegate void delegadoConfiguracionControles();
public event delegadoConfiguracionControles configuracionDeControles;
```

Lo asocio con el metodo "ConfiguracionControles" en la línea 43 del "FormPrincipalMunu"

```
configuracionDeControles += ConfiguracionControles;
```

Y es invocado en el método "aumentarBarra" de la línea 298 del "FormPrincipalMunu"

```
{
    barraDeProgreso.Increment(1);
    configuracionDeControles.Invoke();
    if(idHilo == "1")
```

Los otros dos eventos los utilice para enviar un mensaje cuando se quiera dar de alta, modificar o dar de baja.

```
14
15     public delegate void DelegadoEventos(string mensaje);
16
17     public static event DelegadoEventos EventAvisadorQueClienteNoExiste;
18     public static event DelegadoEventos EventAvisadorQueClienteExisteYEstaActivo;
    0 referencias
```

En el evento “EventAvisadorQueClienteNoExiste”, declarado en la línea 17 de la clase “CentralAdministradora” enviara un mensaje en modo de notficativo si a la hora de dar de alta un cliente se cumple el caso de que este no se encuentre en el sistema.

Dicho evento es invocado en el metodo “VerificarSiCleinteEstaEnElSistema” de la clase “CentralAdministradora” en la línea 96

```
public static bool VerificarSiClienteEstaEnElSistema(int documento)
{
    bool estaEnElSistema = false;
    if (listaDeClientes != null)
    {
        foreach (Cliente unClienteDelSistema in listaDeClientes)
        {
            if (unClienteDelSistema.Dni == documento)
            {
                estaEnElSistema = true;
                break;
            }
        }
    }
    if(!estaEnElSistema)
    {
        EventAvisadorQueClienteNoExiste.Invoke("Excelente, el dni que ingreso NO corresponde a ningun cliente");
    }
}

return estaEnElSistema;
}
```

Y Lo asocio con el metodo “MensajeAvisado” en la línea 45 del “FormIngresoDni”

```
0 referencias
static FormIngresoDeDni()
{
    CentralAdministradora.EventAvisadorQueClienteNoExiste += MensajeAvisador;
```

En el evento “EventAvisadorQueClienteExisteYEstaActivo”, declarado en la línea 18 de la clase “CentralAdministradora” enviara un mensaje en modo de notficativo si a la hora de modificar o dar de baja se ingresa un dni que corresponda a un cliente que este en el sistema y activo.

Dicho evento es invocado en el metodo “VerificarSiElClienteExisteYEstaActivoEnElSistema” de la clase “CentralAdministradora” en la línea 159.

```

2 referencias
/// <returns>true: si se encontro un cliente activo con el dni pasado por parametro
public static bool VerificarSiElClienteExisteYEstaActivoEnElSistema(int documento)
{
    bool estaEnElSistemaActivo;

    if (BuscarClienteActivoPorDni(documento) is not null)
    {
        estaEnElSistemaActivo = true;
        EventAvisadorQueClienteExisteYEstaActivo.Invoke("Excelente, el dni que ingre
    }
    else
    {

```

Y Lo asocio con el metodo “MensajeAvisado” en la línea 45 del “FormIngresoDni”

```

44
45      CentralAdministradora.EventAvisadorQueClienteExisteYEstaActivo += MensajeAvisador;
46

```

Métodos de extensión

Hice tres métodos de extensión

El primero: llamado “IsNullOrEmptyMultiple” que se encuentra en la “ClaseExtendida” línea 18

Verifica que la instancia que invoca al método y los otros dos parámetros que recibe no sean null ni estén vacíos.

```

3 referencias
public static bool IsNullOrEmptyMultiple(this string primeraString, string segundaString, string terceraString)
{
    bool retorno = false;
    if (String.IsNullOrEmpty(primeraString) || String.IsNullOrEmpty(segundaString) || String.IsNullOrEmpty(terceraString))
    {
        retorno = true;
    }

    return retorno;
}

```

Y son utilizados en los siguientes formularios para chequear que no se deje ningún campo vacío:

```

▲ Forms\FormAlta.cs (1)
  107 : if (txtNombre.Text.IsNullOrEmptyMultiple(txtApellido.Text, txtDireccion.Text))
▲ Forms\FormAltaSucursal.cs (1)
  111 : if (txtDireccion.Text.IsNullOrEmptyMultiple(txtLocalidad.Text, txtTelefono.Text))
▲ Forms\FormModificar.cs (1)
  167 : if (txtNombre.Text.IsNullOrEmptyMultiple(txtApellido.Text, txtDireccion.Text))
Contraer todo

```

El segundo: llamado “EsDni” que se encuentra en la “ClaseExtendida” línea 34

verifica que la instancia que invoca al método tenga la cantidad de números correspondiente a un DNI, le permito que sean 7 números porque hay personas con DNI 9.000.000 millones por ejemplo.


```
1 referencia
public static bool EsDni(this string dni)
{
    return dni.Length == 7 || dni.Length == 8;
}
```

Y es utilizado en la línea 223 en el "txtDni_TextChanged" del "FormIngresoDeDni"

```
1 referencia
private void txtDni_TextChanged(object sender, EventArgs e)
{
    if (String.IsNullOrEmpty(txtDni.Text))
    {
        btnIngresarDni.Enabled = false;
        lblInformacionBoton.Visible = true;
    }
    else if (!txtDni.Text.EsDni())
    {
        btnIngresarDni.Enabled = false;
        lblInformacionBoton.Visible = true;
        lblInformacionBoton.Text = " El DNI debe tener 7 u 8 numeros";
    }
    else
    {
        btnIngresarDni.Enabled = true;
        lblInformacionBoton.Visible = false;
    }
}
```

El tercero: llamado "MostrarPlan" que se encuentra en la "ClaseExtendida" línea 45

Verifica a través del enumerado que pasan por parámetro cual es el plan seleccionado y retorna los datos del mismo

```
3 referencias
public static string MostrarPlan(this string datos, EPlanElegido planElegido)
{
    switch (planElegido)
    {
        case EPlanElegido.PlanBasico:
            datos = new PlanBasico().MostarDatosDelPlan();
            break;
        case EPlanElegido.PlanIntermedio:
            datos = new PlanIntermedio().MostarDatosDelPlan();
            break;
        case EPlanElegido.PlanPremium:
            datos = new PlanPremium().MostarDatosDelPlan();
            break;
    }
    return datos;
}
```

Es utilizado en el "FormAlta" en las siguientes líneas:

```
Forms\FormAlta.cs (3)
214 : rtbDatosDelPlan.Text = datosDelPlan.MostrarPlan(EPlanElegido.PlanBasico);
224 : rtbDatosDelPlan.Text = datosDelPlan.MostrarPlan(EPlanElegido.PlanIntermedio);
234 : rtbDatosDelPlan.Text = datosDelPlan.MostrarPlan(EPlanElegido.PlanPremium);
Contraer todo
```