

FRANCO DAVID MESSINA, DIV 2E

TP N4 LABORATORIO II

Introducción:

La aplicación se creó para un local de Ramos Mejias en el cual hacen servicios de Televisores, Controles(TV, Aire acondicionado) y Aire Acondicionado.

La aplicación administra una lista de clientes con sus datos respectivamente y los servicios pedidos.

Menú principal de la aplicación.

Menu

Nombre: Franco	Apellido: Messina	NumTel: 1139427762	Dni: 41195283
Nombre: Juan Martin	Apellido: Del Potro	NumTel: 1190807629	Dni: 34037812
Nombre: Lucas	Apellido: Rodriguez	NumTel: 1140700798	Dni: 39912092
Nombre: Esteban	Apellido: Prieto	NumTel: 1190762030	Dni: 35912893
Nombre: Mauricio	Apellido: Cerizza	NumTel: 1190129009	Dni: 30910905

Registrarse

Exportar Lista

Modificar

Recuperar Cliente

Eliminar

Mostrar Servicios

domingo junio 2022
20:02:13

Producto a reparar

Tv

Televisor

Marca

Sony

Modelo

T30Pulgadas

Tipo

LCD

Entrega

RetiroLocal

Fallas

☐ SinAudio

☐ PantallaRota

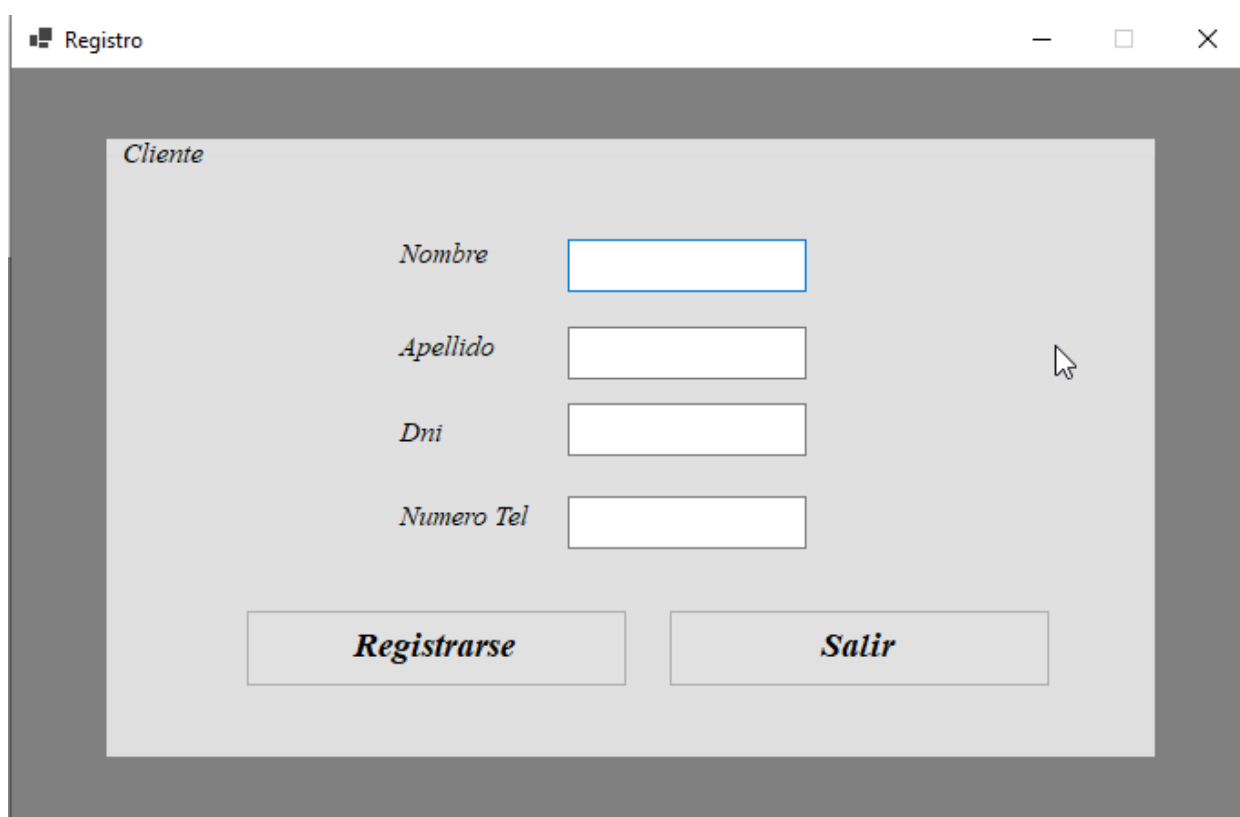
☐ SinImagen

Cargar Servicio

Consta de una listbox donde se puede visualizar los clientes registrados y que están dados de alta.

Nombre: Franco	Apellido: Messina	NumTel: 1139427762	Dni: 41195283
Nombre: Juan Martin	Apellido: Del Potro	NumTel: 1190807629	Dni: 34037812
Nombre: Lucas	Apellido: Rodriguez	NumTel: 1140700798	Dni: 39912092
Nombre: Esteban	Apellido: Prieto	NumTel: 1190762030	Dni: 35912893
Nombre: Mauricio	Apellido: Cerizza	NumTel: 1190129009	Dni: 30910905

Registrar Cliente: El cliente deberá ingresar su nombre, apellido, DNI, y número de teléfono.



Registro

Cliente

Nombre

Apellido

Dni

Numero Tel

Registrarse **Salir**

Modificar Cliente: Se selecciona un cliente de la lista y cuando se aprieta el botón modificar se abre un nuevo formulario con todos los datos cargados del cliente seleccionado.



Modificar

Cliente

Nombre

Apellido

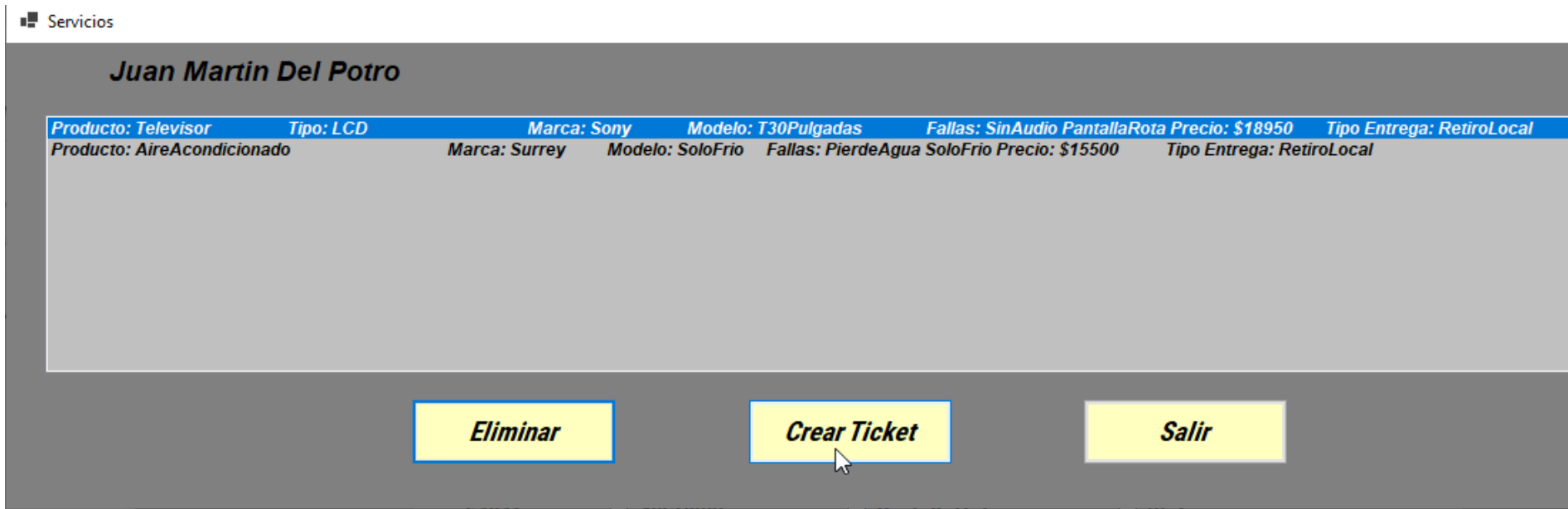
Dni

Numero Tel

Modificar **Salir**

Eliminar Cliente: Se selecciona un cliente de la lista y al apretar el botón eliminar se hace una baja lógica. Esto quiere decir que sigue existiendo en el sistema pero no se visualiza en la lista.

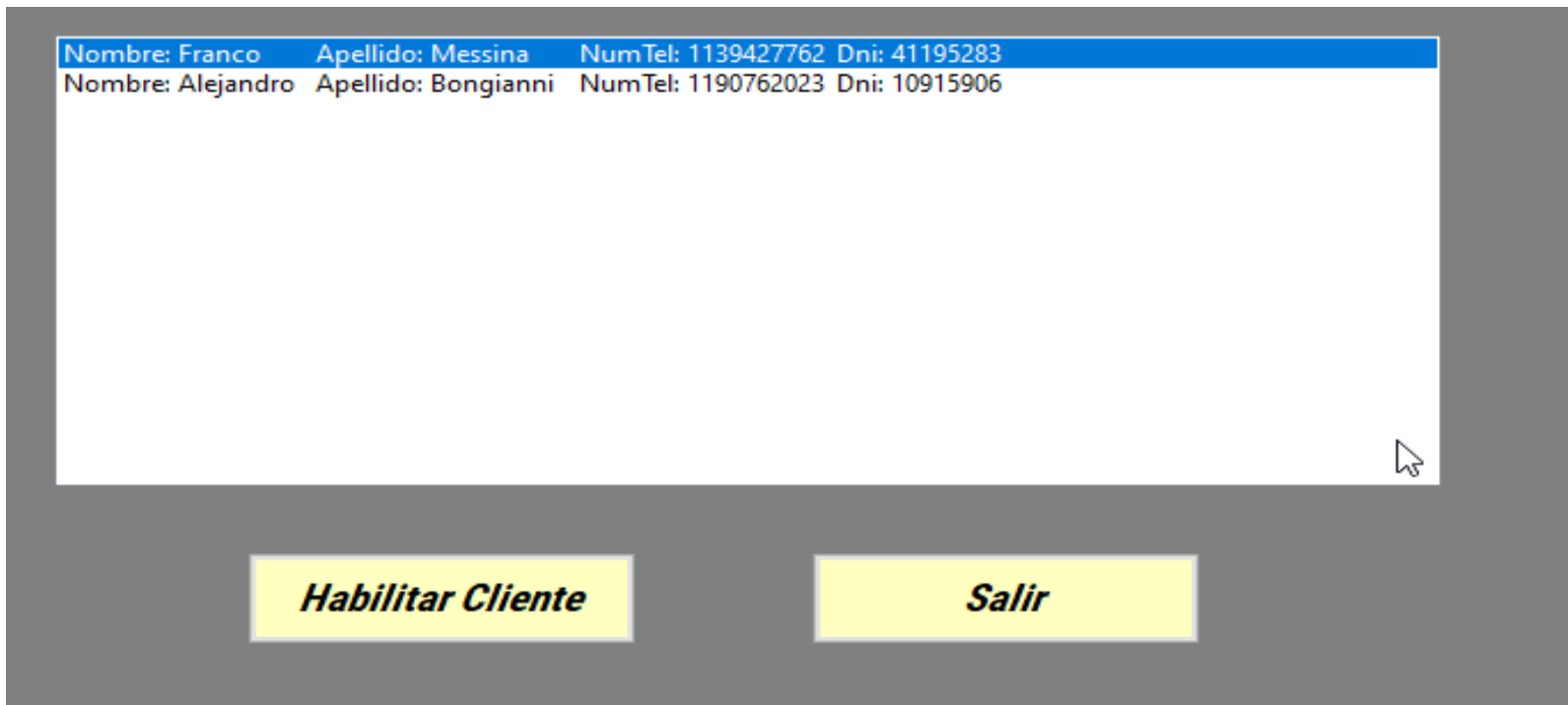
Mostrar Servicios: Se selecciona un cliente de la lista y al apretar el botón Mostrar se visualizará todos los servicios de ese cliente.



Eliminar Servicio: Se selecciona el servicio y al apretar el botón se eliminará.

Crear Ticket: Se selecciona un servicio y al apretar el botón va a generar un TXT en el escritorio->Archivos->TXT-> Nombre,Apellido del cliente y horario actual con todos los datos del servicio.

Recuperar Cliente: Se van a visualizar todos los clientes eliminados, es decir, que existieron alguna vez pero al día de hoy están dados de baja. Para recuperarlo se selecciona el cliente y se lo vuelve a habilitar.



Exportar Lista: Exporta la lista de clientes con todos sus datos personales y servicios en formato XML (en la ruta de escritorio->Archivos->XML->ListaDeClientes.XML).

Cargar Servicio: Se selecciona un cliente se elige el producto a reparar con todas sus características y al apretar el botón se carga el servicio en la lista de servicios del cliente seleccionado.

Lista de temas.

15,16) Conexión a bases de datos (SQL):

Al atributo `cadenaConexion` le asignamos la conexión en un constructor estático para luego reutilizar ese atributo.

```
8 referencias
public static class ClienteDAO
{
    private static string cadenaConexion;

    0 referencias
    static ClienteDAO()
    {
        ClienteDAO.cadenaConexion = "Server=.;Database=TP4;Trusted_Connection=True;";
    }
}
```

LeerDatosClientes: Lista los clientes con sus respectivos datos. Solamente los datos de los clientes, es decir, no se le asignan sus servicios.

```
11 referencias
private static List<Cliente> LeerDatosClientes()
{
    try
    {
        List<Cliente> lista = new List<Cliente>();
        string query = "select * from clientes";
        using (SqlConnection connection = new SqlConnection(cadenaConexion))
        {
            SqlCommand command = new SqlCommand(query, connection);
            connection.Open();
            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
                string dni = reader.GetString(0);
                string nombre = reader.GetString(1);
                string apellido = reader.GetString(2);
                string numeroTel = reader.GetString(3);
                bool dadoDeAlta = reader.GetBoolean(4);

                Cliente auxCliente = new Cliente(dni, nombre, apellido, numeroTel);
                auxCliente.DadoDeAlta = dadoDeAlta;
                lista.Add(auxCliente);
            }
        }
        return lista;
    }
    catch (Exception)
    {
        throw new CargaInvalidaSQLException("Error al leer los datos de los clientes");
    }
}
```

LeerServiciosDeCliente: lee los servicios de un cliente en específico, es decir, le asigna a su lista de servicios todos los servicios que le corresponden.

```

private static void LeerServiciosDeCliente(Cliente cliente)
{
    try
    {
        string query = $"Select Servicios.precio,Servicios.tipoEntrega,Servicios.dni_cliente,Servicios.nombre, Servicios.marca, Servicios.modelo";
        List<Servicio> lista = new List<Servicio>();

        using (SqlConnection connection = new SqlConnection(cadenaConexion))
        {
            SqlCommand cmd = new SqlCommand(query, connection);
            connection.Open();
            SqlDataReader reader = cmd.ExecuteReader();

            while (reader.Read())
            {
                float precio = (float)reader.GetDouble(0);
                ETipoEntrega tipoEntrega = (ETipoEntrega)reader.GetInt32(1);
                string dni_cliente = reader.GetString(2);
                string nombreProducto = reader.GetString(3);
                string marca = reader.GetString(4);
                string modelo = reader.GetString(5);
                string fallaUno = reader["fallaUno"] is not DBNull ? reader.GetString(6) : string.Empty;
                string fallaDos = reader["fallaDos"] is not DBNull ? reader.GetString(7) : string.Empty;
                string fallaTres = reader["fallaTres"] is not DBNull ? reader.GetString(8) : string.Empty;
                List<String> listFallas = new List<String>();
                listFallas.Add(fallaUno);
                listFallas.Add(fallaDos);
                listFallas.Add(fallaTres);
            }
        }
    }
}

```

```

        if (nombreProducto == "Televisor")
        {
            ETipoTv tipoTv = (ETipoTv)reader.GetInt32(9);
            Televisor televisor = new Televisor(marca, modelo, listFallas, tipoTv);
            lista.Add(new Servicio(tipoEntrega, televisor, precio, dni_cliente));
        }
        else if (nombreProducto == "Control")
        {
            ETipoControl tipoControl = (ETipoControl)reader.GetInt32(9);
            Control control = new Control(marca, modelo, listFallas, tipoControl);
            lista.Add(new Servicio(tipoEntrega, control, precio, dni_cliente));
        }
        else if (nombreProducto == "AireAcondicionado")
        {
            AireAcondicionado aire = new AireAcondicionado(marca, modelo, listFallas);
            lista.Add(new Servicio(tipoEntrega, aire, precio, dni_cliente));
        }
    }
    cliente.ListaServicios = lista;
}
catch (Exception)
{
    throw new CargaInvalidaSQLException("Error al leer los datos de los servicios");
}
}

```

ListarClientes: retorna lista de clientes con sus servicios asignados.

```

public static List<Cliente> ListarClientes()
{
    try
    {
        List<Cliente> listaCliente = LeerDatosClientes();
        foreach (Cliente cliente in listaCliente)
        {
            LeerServiciosDeCliente(cliente);
        }
        return listaCliente;
    }
    catch (Exception)
    {
        throw new CargaInvalidaSQLException("Error al listar todos los clientes");
    }
}

```

AltaCliente da de alta un cliente y lo inserta en la tabla de Clientes.

```
1 referencia
public static void AltaCliente(Cliente cliente)
{
    string query = "Insert into Clientes (dni,nombre,apellido,numeroTel,dadoDeAlta) values(@dni,@nombre,@apellido,@numeroTel,@dadoDeAlta)";
    try
    {
        using (SqlConnection connection = new SqlConnection(cadenaConexion))
        {
            connection.Open();
            SqlCommand cmd = new SqlCommand(query, connection);
            cmd.Parameters.AddWithValue("dni", cliente.Dni);
            cmd.Parameters.AddWithValue("nombre", cliente.Nombre);
            cmd.Parameters.AddWithValue("apellido", cliente.Apellido);
            cmd.Parameters.AddWithValue("numeroTel", cliente.NumeroTel);
            cmd.Parameters.AddWithValue("dadoDeAlta", cliente.DadoDeAlta);
            cmd.ExecuteNonQuery();
        }
    }
    catch (Exception)
    {
        throw new CargaInvalidaSQLException("Error al dar de alta el cliente");
    }
}
```

AltaServicio: Se da de alta un servicio y a la columna de dni_Cliente se le agrega el dni del cliente.

```
public static void AltaServicio(Servicio servicio, string dni)
{
    string query = "Insert into Servicios (precio,tipoEntrega,dni_cliente,nombre,marca,modelo,fallaUno,fallaDos,fallaTres,tipo) values(@precio,
    try
    {
        using (SqlConnection connection = new SqlConnection(cadenaConexion))
        {
            connection.Open();
            SqlCommand cmd = new SqlCommand(query, connection);
            List<String> listaFallas = CargarFallas(servicio.Producto.ListaFallas);
            cmd.Parameters.AddWithValue("precio", servicio.Precio);
            cmd.Parameters.AddWithValue("tipoEntrega", servicio.TipoEntrega);
            cmd.Parameters.AddWithValue("dni_Cliente", dni);
            cmd.Parameters.AddWithValue("nombre", servicio.Producto.GetType().Name);
            cmd.Parameters.AddWithValue("marca", servicio.Producto.Marca);
            cmd.Parameters.AddWithValue("modelo", servicio.Producto.Modelo);
            cmd.Parameters.AddWithValue("fallaUno", string.IsNullOrEmpty(listaFallas[0]) ? DBNull.Value : listaFallas[0]);
            cmd.Parameters.AddWithValue("fallaDos", string.IsNullOrEmpty(listaFallas[1]) ? DBNull.Value : listaFallas[1]);
            cmd.Parameters.AddWithValue("fallaTres", string.IsNullOrEmpty(listaFallas[2]) ? DBNull.Value : listaFallas[2]);
            if (servicio.Producto.GetType().Name == "AireAcondicionado")
                cmd.Parameters.AddWithValue("tipo", DBNull.Value);
            else if (servicio.Producto.GetType().Name == "Televisor")
                cmd.Parameters.AddWithValue("tipo", ((Televisor)servicio.Producto).Tipo);
            else if (servicio.Producto.GetType().Name == "Control")
                cmd.Parameters.AddWithValue("tipo", ((Control)servicio.Producto).Tipo);
            cmd.ExecuteNonQuery();
        }
    }
    catch (Exception)
    {
        throw new CargaInvalidaSQLException("Error al dar de alta el cliente");
    }
}
```

BajaCliente: Baja logica se pone el bit de dadoDeAlta en 0.

```
1 referencia
public static void BajaCliente(Cliente cliente)
{
    try
    {
        string query = "update Clientes set dadoDeAlta = 0 where dni = @dni";
        using (SqlConnection connection = new SqlConnection(cadenaConexion))
        {
            connection.Open();
            SqlCommand cmd = new SqlCommand(query, connection);
            cmd.Parameters.AddWithValue("dni", cliente.Dni);
            cmd.ExecuteNonQuery();
        }
    }
    catch (Exception)
    {
        throw new CargaInvalidaSQLException("Error al dar de baja el cliente");
    }
}
```

Delegados

Declaración de delegados propios.

El tipo de delegado InformacionDeCostoFinal va a retornar void y como parámetro recibe un float.

El tipo de delegado InformacionDeTiempo va a retornar void y como parámetro recibe un int.

```
public delegate void InformacionDeCostoFinal(float costoFinal);  
public delegate void InformacionDeTiempo(int tiempo);
```

Utilización de delegados

```
this.Invoke(new InformacionDeCostoFinal(CalcularCostoFinal), new object[] { precioFinal });
```

```
this.Invoke(new InformacionDeTiempo(CargandoCosto), new object[] { tiempo });
```

18,19) Programación multihilo y eventos:

Declaración de eventos con su tipo de delegado.

```
public event InformacionDeCostoFinal InformarCostoFinal;  
public event InformacionDeTiempo InformarTiempo;
```

Suscribiendo los eventos a sus métodos. Por lo tanto estos métodos se convierten en métodos manejadores.

```
private void btnCalcularCosto_Click(object sender, EventArgs e)  
{  
    gestionCostos.InformarCostoFinal += CalcularCostoFinal;  
    gestionCostos.InformarTiempo += CargandoCosto;  
}
```

En el método IniciarCalculo vamos a invocar a los dos eventos.

```
public void IniciarCalculo(List<Servicio> lista)  
{  
    int tiempo = 4000;  
    float precio;  
    while (tiempo > 0)  
    {  
        if (this.InformarTiempo is not null)  
        {  
            this.InformarTiempo.Invoke(tiempo);  
        }  
        Thread.Sleep(tiempo);  
        tiempo -= 1000;  
    }  
    if (this.InformarCostoFinal is not null)  
    {  
        precio = CalcularCosto(lista);  
        this.InformarCostoFinal.Invoke(precio);  
    }  
}
```

En el botón calcular costo lanzó una nueva tarea en otro hilo. Esta tarea a realizar es la del método IniciarCalculo mencionada anteriormente.


```
private void btnCalcularCosto_Click(object sender, EventArgs e)
{
    gestionCostos.InformarCostoFinal += CalcularCostoFinal;
    gestionCostos.InformarTiempo += CargandoCosto;
    List<Servicio> listaAux = gestionServicios.ListaClientes[indexCliente].ListaServicios;
    btnCalcularCosto.Enabled = false;
    Task tarea = Task.Run(() => gestionCostos.IniciarCalculo(listaAux));
}
```

Desde otro hilo no se puede modificar un control de WinForm. Por lo tanto se pregunta si se quiere modificar el lblFinal si la prop InvokeRequired da true se hace un callback y luego entra al else modificando el control.

```
public void CalcularCostoFinal(float precioFinal)
{
    if (this.InvokeRequired)
    {
        this.Invoke(new InformacionDeCostoFinal(CalcularCostoFinal), new object[] { precioFinal });
    }
    else
    {
        lblCostoFinal.Text = $"Costo Final de todos los Servicios ${precioFinal}";
        lblCostoFinal.BackColor = Color.Green;
    }
}
```

```
2 referencias
public void CargandoCosto(int tiempo)
{
    if (this.InvokeRequired)
    {
        this.Invoke(new InformacionDeTiempo(CargandoCosto), new object[] { tiempo });
    }
    else
    {
        lblCostoFinal.Text = $"Calculando costos... {tiempo / 1000}";
        lblCostoFinal.BackColor = Color.Red;
    }
}
```

20) Método de extensión

El siguiente método Verifica que el string solo contenga letras o espacios. Esto último mencionado es para que el apellido pueda ser por ejemplo: Del Potro.

```
public static bool VerificarContieneSoloLetras(this String dato)
{
    foreach (char letra in dato)
    {
        if (!Char.IsLetter(letra) && letra.ToString() != " ")
            return false;
    }
    return true;
}
```

Utilización

```
else if (!apellido.VerificarContieneSoloLetras())
{
    throw new ApellidoInvalidoException("El apellido tiene que contener solo letras.");
}
return true;
}
```

```

else if (!nombre.VerificarContieneSoloLetras())
{
    throw new NombreInvalidoException("El nombre tiene que contener solo letras.");
}
return true;
}

```

El método ContarNumeros cuenta los números de un string extendiendo el tipo de dato String

```

1 referencia
public static int ContarNumeros(this String dato)
{
    int contador = 0;
    foreach (char item in dato)
    {
        if (!Char.IsLetter(item))
            contador++;
    }
    return contador;
}

```

Utilización

```

else if (numeroTel.ContarNumeros() != 10)
{
    throw new NumeroInvalidoException("Los numeros telefonicos tienen 10 digitos!");
}
return true;

```

Algunos Test unitarios del método de extensión VerificarSoloContieneLetras

```

[TestMethod]
0 referencias
public void VerificarContieneSoloLetras_CuandoRecibeDosPalabrasSeparadaPorEspacio_DeberiaRetornarTrue()
{
    //Arrange
    string texto = "Franco Messina";
    bool expected = true;
    bool actual;

    //Act
    actual = texto.VerificarContieneSoloLetras();

    //Assert
    Assert.AreEqual(expected, actual);
}

[TestMethod]
0 referencias
public void VerificarContieneSoloLetras_CuandoRecibeUnNumeroDeberiaRetornarFalse()
{
    //Arrange
    string texto = "Lucas1";
    bool expected = false;
    bool actual;

    //Act
    actual = texto.VerificarContieneSoloLetras();

    //Assert
    Assert.AreEqual(expected, actual);
}

[TestMethod]

```

TIPS:

Utilizar la carpeta archivo en escritorio como el TP anterior.
