Beauty Life
CENTRO DE ESTÉTICA

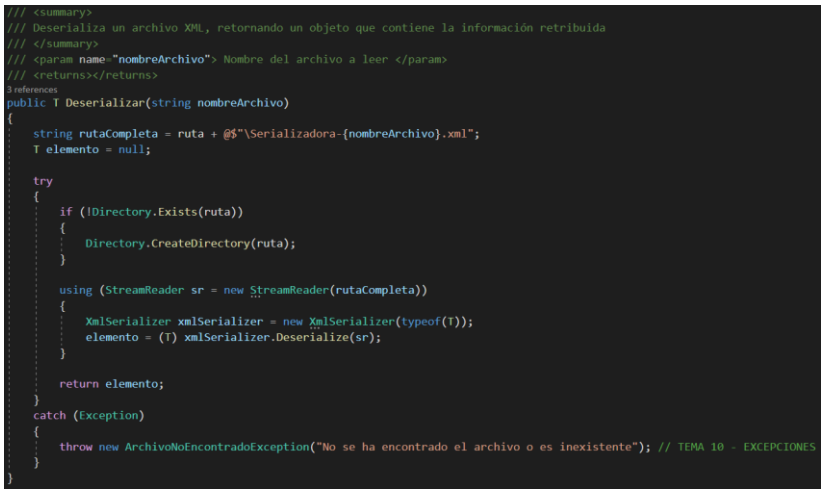
Contexto:

Se desarrolló una aplicación para un centro de estética, donde se darán de alta clientes, con un turno reservado, se permitirá manejar estos clientes y modificarlos según sea necesario. También se podrán agregar productos y registrar entradas y salidas de los mismos.

Temas incluidos:

Tema 10 – Excepciones:

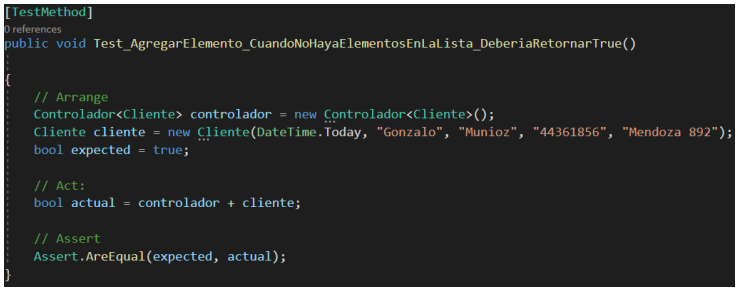
- Se implementó una excepción llamada ArchivoNoEncontradoException, la cual se lanzará si el archivo que se intenta deserializar existe o no coincide la ruta.



```
/// <summary>
/// Deserializa un archivo XML, retornando un objeto que contiene la información retribuida
/// </summary>
/// <param name="nombreArchivo"> Nombre del archivo a leer </param>
/// <returns></returns>
3 references
public T Deserializar(string nombreArchivo)
{
    string rutaCompleta = ruta + @$"\Serializadora-{nombreArchivo}.xml";
    T elemento = null;

    try
    {
        if (!Directory.Exists(ruta))
        {
            Directory.CreateDirectory(ruta);
        }

        using (StreamReader sr = new StreamReader(rutaCompleta))
        {
            XmlSerializer xmlSerializer = new XmlSerializer(typeof(T));
            elemento = (T) xmlSerializer.Deserialize(sr);
        }

        return elemento;
    }
    catch (Exception)
    {
        throw new ArchivoNoEncontradoException("No se ha encontrado el archivo o es inexistente"); // TEMA 10 - EXCEPCIONES
    }
}
```

Tema 11 – Pruebas unitarias:

- Se implementaron 6 tests unitarios, probando 2 funcionalidades.



```
[TestMethod]
0 references
public void Test_AgregarElemento_CuandoNoHayaElementosEnLaLista_DeberiaRetornarTrue()

{
    // Arrange
    Controlador<Cliente> controlador = new Controlador<Cliente>();
    Cliente cliente = new Cliente(DateTime.Today, "Gonzalo", "Munioz", "44361856", "Mendoza 892");
    bool expected = true;

    // Act:
    bool actual = controlador + cliente;

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

Tema 12 – Tipos genéricos:

- Se implementó generics utilizando una clase controladora, que será la que se hará cargo de administrar nuestro listado (puede ser de cualquier objeto), y demás funciones. Además, se implementó generics para desarrollar una clase serializadora capaz de serializar cualquier objeto.

```
46 references
public class Controlador<T> where T : class // TEMA 12 - TIPOS GENÉRICOS
{
    List<T> listaDeElementos;

    11 references
    public List<T> ListaDeElementos
    {
        get { return listaDeElementos; }
        set { listaDeElementos = value; }
    }

    8 references
    public int CantidadDeElementos
    {
        get { return listaDeElementos.Count; }
    }

    5 references
    public Controlador()
    {
        this.listaDeElementos = new List<T>();
    }
```

Tema 13 – Interfaces:

- Se implementó interfaces, utilizando una clase genérica serializadora, en la que se define la firma para el método Serializar y Deserializar, y en las clases SerializadoraXML y SerializadoraJSON se le da la implementación.

```
2 references
public interface ISerializadora<T> // TEMA 13 - INTERFACES
{
    /// <summary>
    /// Serializa y guarda en un archivo cualquier objeto
    /// </summary>
    /// <param name="elemento"> Elemento a serializar </param>
    /// <param name="nombreArchivo"> Nombre del archivo a escribir </param>
    6 references
    public void Serializar(T elemento, string nombreArchivo);

    /// <summary>
    /// Deserializa un archivo y almacena la información retribuida en un objeto del mismo tipo
    /// </summary>
    /// <param name="nombreArchivo"> Nombre del archivo a leer </param>
    /// <returns></returns>
    6 references
    public T Deserializar(string nombreArchivo);
}
```


Tema 14 – Archivos:

- Se implementó archivos para manipular los archivos de datos XML y JSON

Tema 15 – Serialización:

- Se implementó serialización para serializar y deserializar archivos XML y JSON de nuestro objeto, para así poder cargarlos o sobreescribirlos a conveniencia.

Tema 16 y 17 – Base de datos (SQL):

- Se implementó la conexión a la base de datos, y la capacidad de realizar modificaciones, guardado y cargado de datos a la misma.

```
9 references
public static class ClienteDBManager
{
    static string cadenaConexion;
    static SqlConnection conexion;
    static SqlCommand comando;

    0 references
    static ClienteDBManager()
    {
        cadenaConexion = @"Data Source=.;Initial Catalog=BEAUTY_LIFE_DB;Integrated Security = True";
        conexion = new SqlConnection(cadenaConexion);
        comando = new SqlCommand();
        comando.Connection = conexion;
        comando.CommandType = System.Data.CommandType.Text;
    }
```

Tema 18 – Delegados y expresiones lambda:

- Se implementó delegados para manipular los eventos, y las expresiones lambda fueron implementadas a la hora de manipular nuevos hilos.

```
public delegate void DelegadoNotificarCardaDeDatos(); // TEMA 18 - DELEGADOS
```

```
Task tarea = Task.Run(() => frmGuardarLeerDatos.ShowDialog()); // TEMA 19 - PROGRAMACIÓN MULTI-HILO, TEMA 18 - EXPRESIONES LAMBDA
```

Tema 19 – Programación multihilo y concurrencia:

- Se implemento la programación multihilo para permitir abrir distintos menús a la vez, y poder interactuar con ellos.

```
Task tarea = new Task(() => frmModificarCliente.ShowDialog()); // TEMA 19 - PROGRAMACIÓN MULTI-HILO, TEMA 18 - EXPRESIONES LAMBDA
```

Tema 20 – Eventos:

- Se implementó eventos para informar cuando la carga de datos ya ha sido completada.

```
public event DelegadoNotificarCardaDeDatos DatosCargados; // TEMA 20 - EVENTOS
```

```
1 reference
private void frmGuardarLeerDatos_Load(object sender, EventArgs e)
{
    DatosCargados += CerrarVentanas;
}
```

```
1 reference
private void CerrarVentanas()
{
    MessageBox.Show("Los datos han sido cargados correctamente", "¡Éxito!");
    this.Close();
}
```

Tema 21 – Métodos de extensión:

- Se implementó métodos de extensión para extender la clase string y poder facilitar la validación del campo DNI.

```
0 references
public static class StringExtension // TEMA 21 - MÉTODOS DE EXTENSIÓN
{
    /// <summary>
    /// Recorre un string validando que cada caracter sea de tipo numérico, y tenga una longitud de 8 caracteres
    /// </summary>
    /// <param name="dni"> Cadena a validar </param>
    /// <returns></returns>
    5 references
    public static bool ValidarDNI(this string dni)
    {
        if(dni is null)
        {
            return false;
        }

        if (dni.Length == 8)
        {
            foreach (char caracter in dni)
            {
                if (!char.IsDigit(caracter))
                {
                    return false;
                }
            }

            return true;
        }

        return false;
    }
}
```