

TRABAJO PRÁCTICO

FINAL UIN 2D

ALUMNO: BRUNO DE
RENZIS

PROFESOR: FEDERICO
DAVILA

MATERIA: LABORATORIO 2

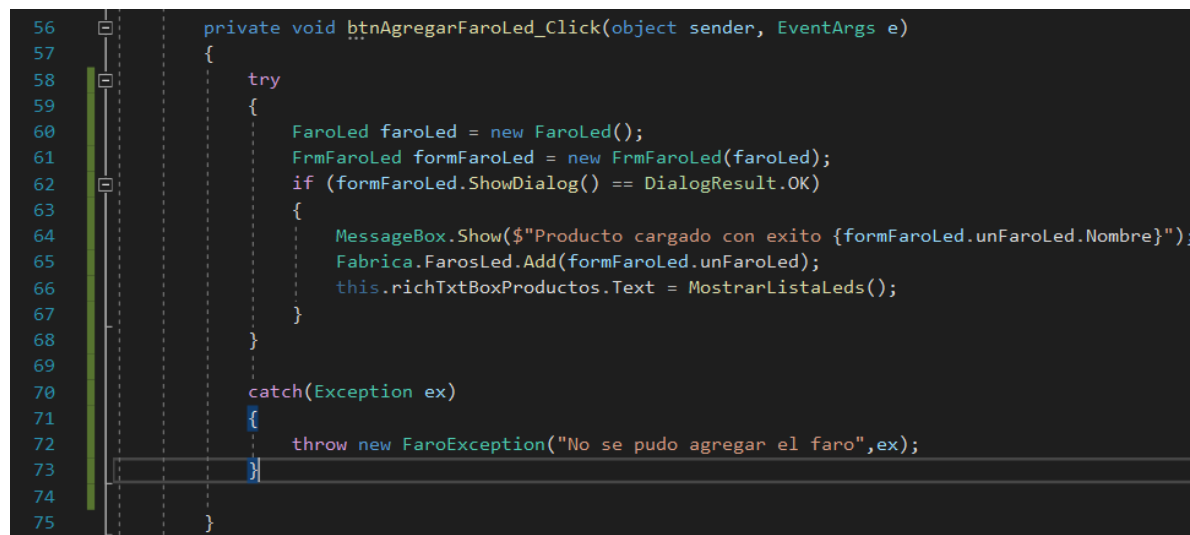
La funcionalidad del trabajo práctico es la de simular el proceso de fabricación de un faro, ya sea de led o lámpara.

El programa cuenta con la lógica para que determinando solo el tamaño del faro pueda saber cuantos materiales lo componen.

Los temas utilizados en este trabajo práctico han sido:

Clase 15: Excepciones

Clase FrmPrincipal



```
56 private void btnAgregarFaroLed_Click(object sender, EventArgs e)
57 {
58     try
59     {
60         FaroLed faroLed = new FaroLed();
61         FrmFaroLed formFaroLed = new FrmFaroLed(faroLed);
62         if (formFaroLed.ShowDialog() == DialogResult.OK)
63         {
64             MessageBox.Show($"Producto cargado con éxito {formFaroLed.unFaroLed.Nombre}");
65             Fabrica.FarosLed.Add(formFaroLed.unFaroLed);
66             this.richTextBoxProductos.Text = MostrarListaLeds();
67         }
68     }
69     catch (Exception ex)
70     {
71         throw new FaroException("No se pudo agregar el faro", ex);
72     }
73 }
74
75 }
```

Las excepciones están implementadas en el código para que al momento de estar frente a algún posible error que detenga la ejecución del programa, esta pueda informar de tal error y seguir con el flujo normal del programa, a esta posibilidad se la llama controlar esa excepción. Se las utilizará en gran parte del código, se ve en el ejemplo como se utiliza para que al momento de agregar un elemento a la lista, no haya error alguno en su carga de valores.

Clase 16: Test Unitarios

Clase AgregarFaroTest

```
namespace UnitTestFaro
{
    [TestClass]
    public class AgregarFaroTest
    {
        /// <summary>
        /// Prueba unitaria de agregar elemento a la lista
        /// </summary>
        [TestMethod]
        public void AgregarFaro()
        {
            FaroLed faroLed = new FaroLed("faroNuevo", Faro.EMedida.Mediano, 2, FaroLed.ETipoLed.Micro);
            Fabrica.FarosLed.Add(faroLed);
            Assert.IsNotNull(Fabrica.FarosLed);
        }
    }
}
```

Los test unitarios son útiles al momento de probar aisladamente alguna funcionalidad para comprobar su correcta ejecución. Se los utilizará para probar la correcta inclusión de otro elemento en la lista.

Clase 17: Tipos genéricos

Clase Xml

```
public class Xml<T> : IArchivo<T>
{
    /// <summary>
    /// Guarda un archivo en formato xml
    /// </summary>
    /// <param name="archivo"></param>
    /// <param name="datos"></param>
    /// <returns>true si se guardo correctamente, false caso contrario</returns>

    public bool Guardar(string archivo, T datos)
    {
        try
        {
            if (archivo != null)
            {
                using (XmlTextWriter writer = new XmlTextWriter(archivo, Encoding.UTF8))
                {
                    XmlSerializer ser = new XmlSerializer(typeof(T));

                    ser.Serialize(writer, datos);

                    return true;
                }
            }
        }
        catch (Exception e)
        {
        }
    }
}
```

Los tipos genéricos se implementan con la finalidad de facilitar la escritura del código ante métodos listos atributos o interfaces que permitirán una fácil sobreescritura con el desarrollo del método ya establecido. Se lo utilizará para escribir y leer las distintas listas generadas en xml.

Clase 18: Interfaces

IArchivo

```
namespace Entidades
{
    1 referencia
    interface IArchivo<T>
    {
        /// <summary>
        /// Guarda los datos de un archivo.
        /// </summary>
        /// <param name="archivo"></param>
        /// <param name="datos"></param>
        /// <returns>True si pudo guardar cambios, false si no guardó</returns>
        3 referencias
        bool Guardar(string archivo, T datos);

        /// <summary>
        /// Lee los datos de un archivo
        /// </summary>
        /// <param name="archivo"></param>
        /// <param name="datos"></param>
        /// <returns>true si se pudo leer el archivo, false si no se pudo leer el archivo.</returns>
        3 referencias
        bool Leer(string archivo, out T datos);
    }
}
```

Las interfaces se utilizan para desarrollar métodos en la clase que implemente dicha interfaz de manera obligatoria para la correcta compilación del programa. Se lo utilizará para que la clase serializadora Xml desarrolle de manera obligatoria ambos métodos y asegurarse así de la correcta carga y lectura de los archivos.

Clase 19: Archivos

```
private void FrmPrincipal_FormClosing(object sender, FormClosingEventArgs e)
{
    Fabrica.GuardarLampara(Fabrica.FarosLampara);
    Fabrica.GuardarLeds(Fabrica.FarosLed);
}
```

Los archivos se utilizarán para serializar y poder leer la cantidad de faros agregados y la cantidad de unidades de cada material que consumió cada uno en su construcción.