TRABAJO PRÁCTICO FINAL LABORATORIO II

La funcionalidad del programa se basa en filtrar plantas con diferentes características. Las plantas a ser listadas consisten en su división morfológica (Angiosperma, Gymnosperma y Briófitas). Para una mayor comprensión del programa (y para no caer en tecnicismos) se las ha nombrado de la siguiente manera: Planta Con Fruto, Planta Sin Fruto, Planta Musgo respectivamente.

Se puede buscar plantas por su Origen, Color de Flor, Tipo de Fruto, Altura Máxima y el Musgo con mayor cantidad de esporas. Además, se pueden agregar, modificar y eliminar cualquiera de los tres tipos de plantas.

Los reportes realizados en función al filtro seleccionado se pueden exportar en formato .xml.

Los AMB pueden ser exportados en formato .xml y .json según el usuario lo decida.

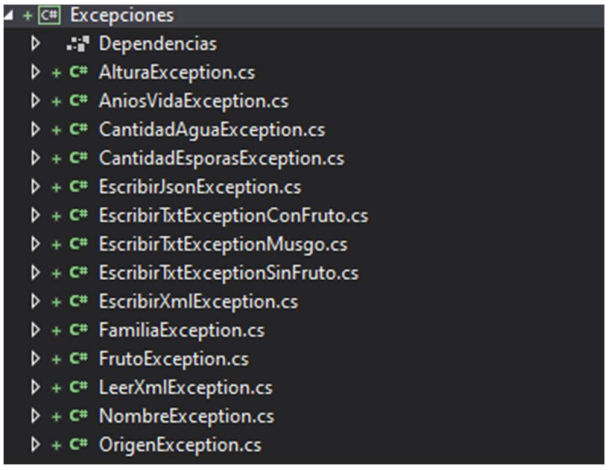
A continuación se mostrarán algunos ejemplos utilizados en el programa en función de los temas vistos en clase.

Temas utilizados

Clase 10 – Excepciones

Se crearon excepciones personalizadas para distintos métodos.

Se las utilizó en las validaciones de las propiedades como también para dar de alta o modificar una planta. Si el usuario ingresó algún campo incorrectamente, se le notificará mediante un texto (label) en el programa.



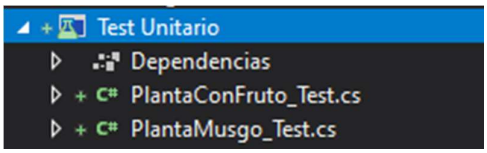

```
public string Nombre
{
    set
    {
        string aux = value;
        if (Validar.SoloLetras(aux))
            this.nombre = aux;
        else
            throw new NombreException();
    }
    get { return this.nombre; }
}
```

```
private void btnAlta_Click(object sender, EventArgs e)
{
    int.TryParse(txtCicloVida.Text, out int añosVidaParse);
    float.TryParse(txtCantidadAgua.Text, out float cantAguaParse);
    float.TryParse(txtAltura.Text, out float alturaParse);

    try
    {
        switch (cmbColorFlor.SelectedItem)
        {
            case PlantaConFruto.EColorFlor.roja:
                PlantaConFruto.AltaConFruto(new PlantaConFruto(txtNombre.Text, txtFamilia.Text, txtOrigen.Text, añosVidaParse, cantA
                break;
            case PlantaConFruto.EColorFlor.rosa:
                PlantaConFruto.AltaConFruto(new PlantaConFruto(txtNombre.Text, txtFamilia.Text, txtOrigen.Text, añosVidaParse, cantA
                break;
            case PlantaConFruto.EColorFlor.amarilla:
                PlantaConFruto.AltaConFruto(new PlantaConFruto(txtNombre.Text, txtFamilia.Text, txtOrigen.Text, añosVidaParse, cantA
                break;
        }
        ActualizarDgv();
    }
    catch (NombreException)
    {
        NombreException name = new NombreException();
        lblCatchNombre.Text = name.Message.ToString();
    }
```

Clase 11 – Pruebas Unitarias

Se testearon métodos de dos clases: PlantaConFruto y PlantaMusgo.



Se testeó el método del UltimoId para comprobar si retornaba algo que no era null.

Se testeó el método EscribirTxt para comprobar si se lanzaba la excepción esperada.

```
[TestClass]
0 referencias
public class PlantaConFruto_Test
{
    [TestMethod]
    0 referencias
    public void TestUltimoId_01()
    {

        int idEsperado;

        idEsperado = PlantaConFruto.UltimoId();

        Assert.IsNotNull(idEsperado);
    }


    [ExpectedException (typeof(EscribirTxtException))]
    [TestMethod]
    0 referencias
    public void TestEscribirTxt_01()
    {
        List<PlantaConFruto> listaTest = new List<PlantaConFruto>();
        listaTest = null;
        string archivo = "TestArchivo";

        PlantaConFruto.EscribirTxt(listaTest, archivo);

        Assert.IsTrue(true);

    }
```

En la otra clase, PlantaMusgo, se testeó el método de FiltrarCantidadEsporas para verificar si retornaba un nombre.

```
[TestClass]
0 referencias
public class PlantaMusgo_Test
{
    [TestMethod]
    0 referencias
    public void FiltrarCantidadEsporas_01()
    {
        List<PlantaMusgo> lista = new List<PlantaMusgo>();
        PlantaMusgo plantaMusgo = new PlantaMusgo("Dendroligotrichum dendroides", "Polytrichaceae ", "America", 500, 4000, 3000);

        lista.Add(plantaMusgo);

        plantaMusgo.Nombre = PlantaMusgo.FiltrarCantidadEsporas(lista);

        Assert.IsNotNull(plantaMusgo.Nombre);
        Assert.IsTrue(plantaMusgo.Nombre.ToString() != string.Empty);

    }
}
```

Clase 12 - Tipos Genéricos

La clase Archivos decidí hacerla genérica ya que empleo métodos genéricos que pueden ser utilizados por mis 3 clases.

Por ejemplo, el método EscribirXml recibe una lista de datos genérica para crear un archivo dentro de la ruta que le indique. Este método, al estar en una clase genérica, puede escribir listas con distinto tipo de dato.

```
/// <summary>
/// Escribe datos genericos en un archivo .xml
/// </summary>
/// <param name="listaDatos">lista de datos a escribir</param>
/// <param name="nombreArchivo">nombre del archivo a guardar</param>
/// <returns>bool</returns>
7 referencias
public static bool EscribirXml(T listaDatos, string nombreArchivo)
{
    string archivo = ruta + $"{nombreArchivo}" + ".xml";

    if(listaDatos != null)
    {
        if (!Directory.Exists(ruta))
        {
            Directory.CreateDirectory(ruta);
        }
        using (XmlTextWriter sw = new XmlTextWriter(archivo, System.Text.Encoding.UTF8))
        {
            XmlSerializer xlmSer = new XmlSerializer(typeof(T));
            xlmSer.Serialize(sw, listaDatos);
        }
        return true;
    }
    else
    {
        throw new EscribirXmlException();
    }
}
```

Realicé una clase llamada FiltrosGenericos donde se emplea el método FiltrarPorOrigen para que pueda ser utilizado por mis 3 clases ya que comparten el mismo atributo “Origen”. Este método buscará en la lista (según el tipo de dato que se esté haciendo referencia) el origen seleccionado por el usuario dentro del formulario.

```
/// <summary>
/// Filtra el atributo "Origen" que tiene en comun con las 3 clases.
/// Se aplica Tipos Genericos
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="lista">lista generica a filtrar</param>
/// <param name="origen">origen seleccionado</param>
/// <returns>lista generica</returns>
3 referencias
public static List<T> FiltrarPorOrigen<T>(List<T> lista, string origen) where T : Planta
{
    List<T> listaFiltrada = new List<T>();
    foreach (T item in lista)
    {
        if (item.Origen == origen)
        {
            listaFiltrada.Add(item);
        }
    }
    return listaFiltrada;
}
```

Clase 13 - Interfaces

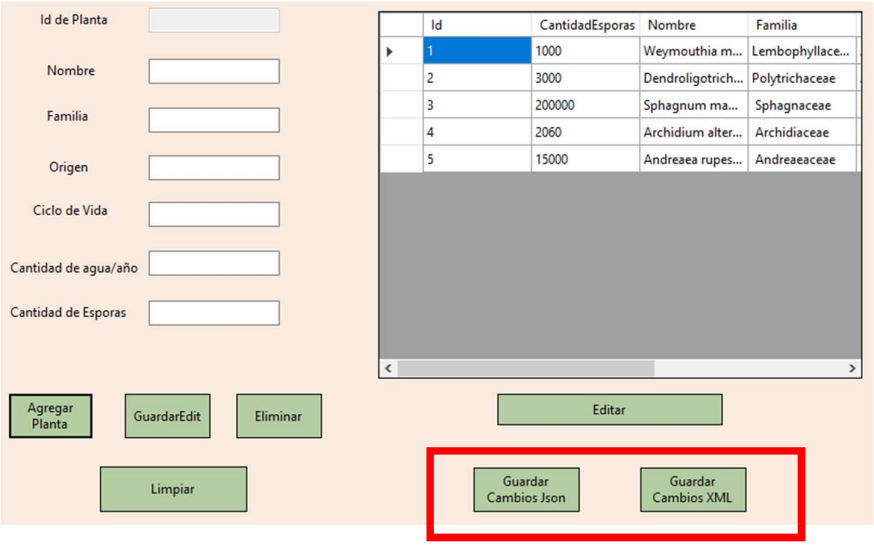
En la clase IAltura definí un método para que pueda ser implementado en dos de mis clases (PlantaConFruto y PlantaSinFruto) ya que las mismas comparten un mismo atributo que es la Altura. Este atributo no podía ser colocado en mi clase padre Planta ya que la clase PlantaMusgo no llevaba ese atributo. Por este motivo, se decidió realizar una interfaz, para que dos clases puedan implementarla y llevar el mismo método.

```
/// <summary>
/// Filtra la planta con la altura maxima. Metodo implentando la interfaz.
/// </summary>
/// <returns>retorna el nombre de la planta que encontro</returns>
2 referencias
public string AlturaMaxima()
{
    float altura = 0;
    string nombre = null;
    int flag = 0;
    PlantaConFruto plantita = new PlantaConFruto();
    foreach (PlantaConFruto item in CargaDeDatos.listaPlantaConFruto)
    {
        if (altura <= item.altura || flag == 0)
        {
            altura = item.altura;
            nombre = item.Nombre;
            flag = 1;
        }
    }
    return nombre;
}
```

Clase 14 – Archivos

El programa comienza cargando las listas con datos hardcodeados (si es que no posee un archivo en su computadora) o bien, leyendo un archivo si coincide con el nombre pasado por parámetro del método.

Dentro del ABM el usuario puede guardar la lista que modificó en formato .json o .xml.



Para los reportes, el usuario puede generar un archivo .txt en donde estarán escritos todos los datos que se filtraron de acuerdo a la selección.

```
/// <summary>
/// Escribe en .txt los datos pasados por parametro
/// </summary>
/// <param name="datos">datos a escribir en el archivo</param>
/// <param name="nombreArchivo">nombre que tendra el archivo</param>
/// <returns>retorna true si salio bien la ejecucion</returns>
5 referencias
public static bool EscribirTxt(List<PlantaConFruto> datos, string nombreArchivo)
{
    string ruta = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
    ruta += @"\Archivos\";
    string archivo = ruta + $"{nombreArchivo}" + ".txt";

    if (!Directory.Exists(ruta))
    {
        Directory.CreateDirectory(ruta);
    }
    using (StreamWriter escribir = new StreamWriter(archivo))
    {
        if(datos != null)
        {
            foreach (PlantaConFruto item in datos)
            {
                escribir.WriteLine(item.ToString());
                escribir.WriteLine("\n*****************************************");
            }
        }
        else
        {
            throw new EscribirTxtException();
        }
    }
    return true;

}
```