

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.

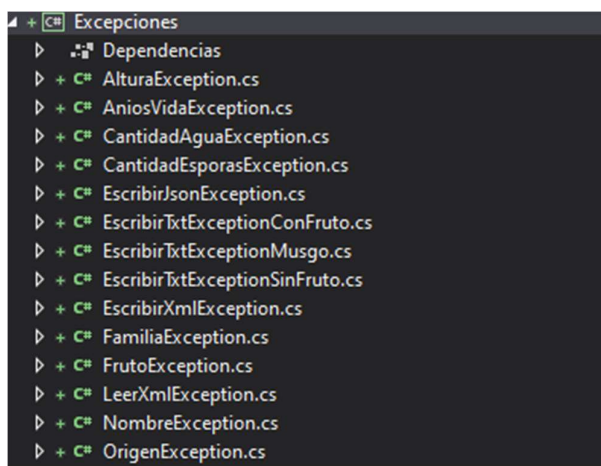
A continuación se mostrarán algunos ejemplos utilizados en el programa en función de los temas vistos en clase.

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; }
}
```

```

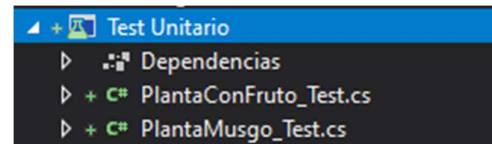
private void btnAlta_Click(object sender, EventArgs e)
{
    int.TryParse(txtCicloVida.Text, out int añosVidaParse);
    float.TryParse(txtCantidadAgua.Text, out float cantAguaParse);
    float.TryParse(txtAltura.Text, out float alturaParse);

    try
    {
        switch (cmbColorFlor.SelectedItem)
        {
            case PlantaConFruto.EColorFlor.roja:
                PlantaConFruto.AltaConFruto(new PlantaConFruto(txtNombre.Text, txtFamilia.Text, txtOrigen.Text, añosVidaParse, cantAguaParse, alturaParse));
                break;
            case PlantaConFruto.EColorFlor.rosa:
                PlantaConFruto.AltaConFruto(new PlantaConFruto(txtNombre.Text, txtFamilia.Text, txtOrigen.Text, añosVidaParse, cantAguaParse, alturaParse));
                break;
            case PlantaConFruto.EColorFlor.amarilla:
                PlantaConFruto.AltaConFruto(new PlantaConFruto(txtNombre.Text, txtFamilia.Text, txtOrigen.Text, añosVidaParse, cantAguaParse, alturaParse));
                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 hardcodedados (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.

	Id	CantidadEsporas	Nombre	Familia
▶	1	1000	Weymouthia m...	Lembophyllace...
	2	3000	Dendroligotrich...	Polytrichaceae
	3	200000	Sphagnum ma...	Sphagnaceae
	4	2060	Archidium alter...	Archidiaceae
	5	15000	Andreaea rupes...	Andreaeaceae

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;
}
```

Origen ▼

Filtrar Origen

Exportar Resultados De Origen

-SEGUNDA PARTE-

IMPORTANTE: Para cambiar la conexión al SQL ir a la clase Conexión_DB y en su constructor cambiar la conexión

```
0 referencias
static Conexion_DB()
{
    conexion = new SqlConnection(@"Server=.;\SQLEXPRESS;Database=TP4_MAGRI_LUDMILA;Trusted_Connection=True;");
    comando = new SqlCommand();
    comando.CommandType = System.Data.CommandType.Text;
    comando.Connection = conexion;
}
```

Clase 15 y 16 – Introducción a SQL, Conexión a base de datos

Se trabajó con base de datos para retornar, en forma de lista, los campos de las tablas de plantas y para realizar los ABM.

Método para traer la lista de plantas. Instrucción DML utilizada (SELECT)

```
public static List<PlantaMusgo> TraerPlantaMusgo()
{
    try
    {
        List<PlantaMusgo> auxLista = new List<PlantaMusgo>();
        comando.CommandText = "Select * from dbo.Planta_Musgo";
        if (conexion.State != System.Data.ConnectionState.Open)
        {
            conexion.Open();
        }
        reader = comando.ExecuteReader();

        while (reader.Read())
        {
            PlantaMusgo p = new PlantaMusgo(reader["Nombre"].ToString(),
                reader["Familia"].ToString(),
                reader["Origen"].ToString(),
                int.Parse(reader["Anios_De_Vida"].ToString()),
                float.Parse(reader["Cantidad_De_Agua"].ToString()));

            p.Id = int.Parse(reader["Id"].ToString());
            p.CantidadEsporas = int.Parse(reader["Cantidad_De_Esporas"].ToString());
            auxLista.Add(p);
        }
        return auxLista;
    }
    catch (Exception ex)
    {
        throw new Exception();
    }
    finally { conexion.Close(); }
}
```

Método para insertar campos en la tabla


```

public static void AgregarPlantaMusgo(PlantaMusgo planta)
{
    try
    {
        comando.Parameters.Clear();
        comando.CommandText = $"INSERT INTO dbo.Planta_Musgo " +
            $"VALUES(@Nombre, @Familia, @Origen, @Anios_De_Vida, @Cantidad_De_Agua, @Cantidad_De_Esporas)";

        comando.Parameters.AddWithValue("@Nombre", planta.Nombre);
        comando.Parameters.AddWithValue("@Familia", planta.Familia);
        comando.Parameters.AddWithValue("@Origen", planta.Origen);
        comando.Parameters.AddWithValue("@Anios_De_Vida", planta.AniosVida);
        comando.Parameters.AddWithValue("@Cantidad_De_Agua", planta.CantidadAgua);
        comando.Parameters.AddWithValue("@Cantidad_De_Esporas", planta.CantidadEsporas);

        if (conexion.State != System.Data.ConnectionState.Open)
        {
            conexion.Open();
        }
        comando.ExecuteNonQuery();
    }
    catch (Exception)
    {
        throw;
    }
    finally { conexion.Close(); }
}

```

Clase 17 – Delegados y expresiones lambda

Se creó el siguiente método para poder buscar una planta por su nombre. La expresión lambda busca el nombre pasado por parámetro en la lista de plantas. Si la búsqueda coincide, retorna la lista de plantas con ese nombre. Esta expresión utiliza el delegado "Predicate".

```

public List<PlantaConFruto> BuscarPlantaPorNombre(string nombre)
{
    List<PlantaConFruto> listaNombres = new List<PlantaConFruto>();
    listaNombres = CargaDeDatos.listaPlantaConFruto.FindAll((p) => p.Nombre.Contains(nombre));
    return listaNombres;
}

```

Se crearon 3 delegados por cada clase. Los mismos fueron utilizados mediante eventos. El delegado contiene 3 manejadores asociados que luego se utilizarán en el formulario de ABM.

(Esta idea se desarrollará mejor en la Clase 19 de Eventos)

```

public delegate void MiDelegadoConFruto(List<PlantaConFruto> lista);

```

```

public delegate void MiDelegadoSinFruto(List<PlantaSinFruto> lista);

```

```

public delegate void MiDelegadoMusgo(List<PlantaMusgo> lista);

```

Clase 18 – Hilos

En cada clase se realizó la siguiente Task:

Mediante el botón “Actualizar” se ejecuta la tarea de ActualizarDatagrid en la cual, en otro hilo, irá a buscar la información que contiene las tablas en la base de datos para poder cargar el datagridview con los datos retornados.

```
private void btnActualizar_Click(object sender, EventArgs e)
{
    lblCargando.Text = "Cargando lista de Plantas";

    Task.Run(() => ActualizarDatagrid());
}
```

```
private void ActualizarDatagrid()
{
    List<PlantaConFruto> listaAux;
    listaAux = Conexion_DB.TraerPlantaConFruto();
    Thread.Sleep(1500);

    if (this.dgvConFruto.InvokeRequired)
    {
        this.dgvConFruto.BeginInvoke((MethodInvoker)delegate { ... });
    }
    else
    {
        dgvConFruto.DataSource = null;
        dgvConFruto.DataSource = CargaDeDatos.RetornarListaSinFruto();
    }
}
```

También en el menú principal se simuló una “carga” mediante la siguiente tarea:


```

private async void Cargar()
{
    await Task.Run(() =>
    {
        Thread.Sleep(5000);
        if (this.btnFiltros.InvokeRequired && this.btnPlantaConFruto.InvokeRequired &&
            this.btnPlantaSinFruto.InvokeRequired && this.btnPlantaMusgo.InvokeRequired)
        {
            this.btnFiltros.BeginInvoke((MethodInvoker)delegate ()
            {
                btnFiltros.BackColor = Color.FromArgb(181, 205, 163);
                btnFiltros.Enabled = true;
                lblMensaje.Text = "Carga completada";
                btnPlantaConFruto.BackColor = Color.FromArgb(181, 205, 163);
                btnPlantaConFruto.Enabled = true;
                btnPlantaSinFruto.BackColor = Color.FromArgb(181, 205, 163);
                btnPlantaSinFruto.Enabled = true;
                btnPlantaMusgo.BackColor = Color.FromArgb(181, 205, 163);
                btnPlantaMusgo.Enabled = true;
                this.Refresh();
                pbGif.Hide();
            });
        }
    });
}

```

Clase 19 – Eventos

Se crearon los siguientes 3 eventos en sus respectivas clases. Estos mismos tienen asociados a sus delegados nombrados anteriormente.

```
public event MiDelegadoConFruto MostrarPlantaConFruto;
```

```
public event MiDelegadoSinFruto MostrarPlantaSinFruto;
```

```
public event MiDelegadoMusgo MostrarPlantaMusgo;
```

Por medio del evento “ControlarLista” se invoca al delegado con sus manejadores asociados con la condición de que exista más de 2 o igual a 2 plantas en la lista.

```

public void ControlarLista(List<PlantaMusgo> lista)
{
    if (CargaDeDatos.listaPlantaMusgo.Count >= 2)
    {
        MostrarPlantaMusgo.Invoke(lista);
    }
}

```

En el constructor del formulario fueron asignados los manejadores y se llamó al método que invoca al delegado

```
plantaMusgo.MostrarPlantaMusgo += MostrarColoresLista;  
plantaMusgo.MostrarPlantaMusgo += MostrarLista;  
plantaMusgo.MostrarPlantaMusgo += MostrarImagen;  
plantaMusgo.ControlarLista(listaMusgo);
```

Los manejadores asociados se encuentran en los formularios de ABM de cada clase. Estos se encargan de:

- Escribir la cantidad de plantas que contiene la lista (cargada desde base de datos)

```
private void MostrarLista(List<PlantaMusgo> lista)  
{  
    this.Refresh();  
    lblEvento.Text = "Cantidad de plantas: " + lista.Count.ToString();  
}
```

- Cambiar el color del label según la cantidad de plantas que contiene la lista

```
1 referencia  
private void MostrarColoresLista(List<PlantaMusgo> lista)  
{
```

- Cambiar la imagen asociada al número que contiene la lista

```
1 referencia  
private void MostrarImagen(List<PlantaMusgo> lista)  
{
```

La funcionalidad en la UI se muestra a continuación:

Al ingresar al ABM poseo 8 plantas

Plantas Sin Frutos (Gymnosperma)

Id

Tipo de Planta

Arbol

Nombre

Familia

Origen

Ciclo de Vida

Cantidad de agua/año

Altura promedio

Agregar Planta

GuardarEdit

Eliminar

Limpiar

Actualizar





TipoPlanta	Id	Altura	Nombre	Familia	Origen	Anios
Arbol	53	3000	Araucaria arauc...	Araucariaceae	Nativa	1000
Arbol	54	3000	Juniperus depp...	Cupressaceae	America	500
Arbusto	55	800	Taxus globosa	Taxaceae	America	10
Arbusto	56	3000	Ginkgo biloba	Ginkgoaceae	Asia	1200
Arbusto	57	4000	Pinus uncinata	Pinaceas	Europa	1000
Arbusto	58	2500	Pinus cembra	Pinaceas	Europa	1600
Arbol	59	1500	Juniperus chine...	Cupressaceae	Asia	1040
Arbol	60	1500	Juniperus chine...	Cupressaceae	Asia	1040

Editar

Guardar Cambios Json

Guardar Cambios XML

Cantidad de plantas: 8

Si elimino alguna de ellas, se mostrará de la siguiente manera:

Plantas Sin Frutos (Gymnosperma)

Id

60

Tipo de Planta

Arbol

Nombre

Familia

Origen

Ciclo de Vida

Cantidad de agua/año

Altura promedio

Agregar Planta

GuardarEdit

Eliminar

Limpiar

Planta eliminada exitosamente

Actualizar




TipoPlanta	Id	Altura	Nombre	Familia	Origen	Anios
Arbol	53	3000	Araucaria arauc...	Araucariaceae	Nativa	1000
Arbol	54	3000	Juniperus depp...	Cupressaceae	America	500
Arbusto	55	800	Taxus globosa	Taxaceae	America	10
Arbusto	56	3000	Ginkgo biloba	Ginkgoaceae	Asia	1200
Arbusto	57	4000	Pinus uncinata	Pinaceas	Europa	1000
Arbusto	58	2500	Pinus cembra	Pinaceas	Europa	1600
Arbol	59	1500	Juniperus chine...	Cupressaceae	Asia	1040

Editar

Guardar Cambios Json

Guardar Cambios XML

Cantidad de plantas: 7

Clase 20 – Métodos de Extensión

Dentro de la clase “Extensión” se encuentra el método:


```

public static string FiltrarCantidadEsporas(this List<PlantaMusgo> lista)
{
    int esporas = 0;
    int flag = 0;
    string nombre = null;
    foreach (PlantaMusgo item in lista)
    {
        if (esporas <= item.CantidadEsporas || flag == 0)
        {
            esporas = item.CantidadEsporas;
            nombre = item.Nombre;
            flag = 1;
        }
    }
    return nombre;
}

```

El mismo es una extensión del tipo "List<PlantaMusgo>" y se encargará de filtrar por la cantidad de esporas que contenga la lista. Por último retorna el nombre de la planta con más cantidad de esporas.

Dentro del FrmFiltros se utiliza esta extensión



```

private void btnEsporas_Click(object sender, EventArgs e)
{
    PlantaMusgo musgo = new PlantaMusgo();
    string nombreMusgo;
    nombreMusgo = listaMusgos.FiltrarCantidadEsporas();
    listaFiltradaMusgo = musgo.BuscarPlantaPorNombre(nombreMusgo);
    lblMensaje.Text = "La planta con mas esporas es : " + nombreMusgo;
    ActualizarDgvMusgos(listaFiltradaMusgo);
}

```