

# Unidad 4 - Laboratorio 01

## 1. Lectura y Escritura de Archivos TXT y XML.

### 1.1. Archivos TXT.

#### Objetivos

Abrir un archivo TXT, recorrerlo mostrar su contenido.  
Crear un nuevo archivo TXT y guardar datos en el mismo.

#### Duración Aproximada

20 minutos

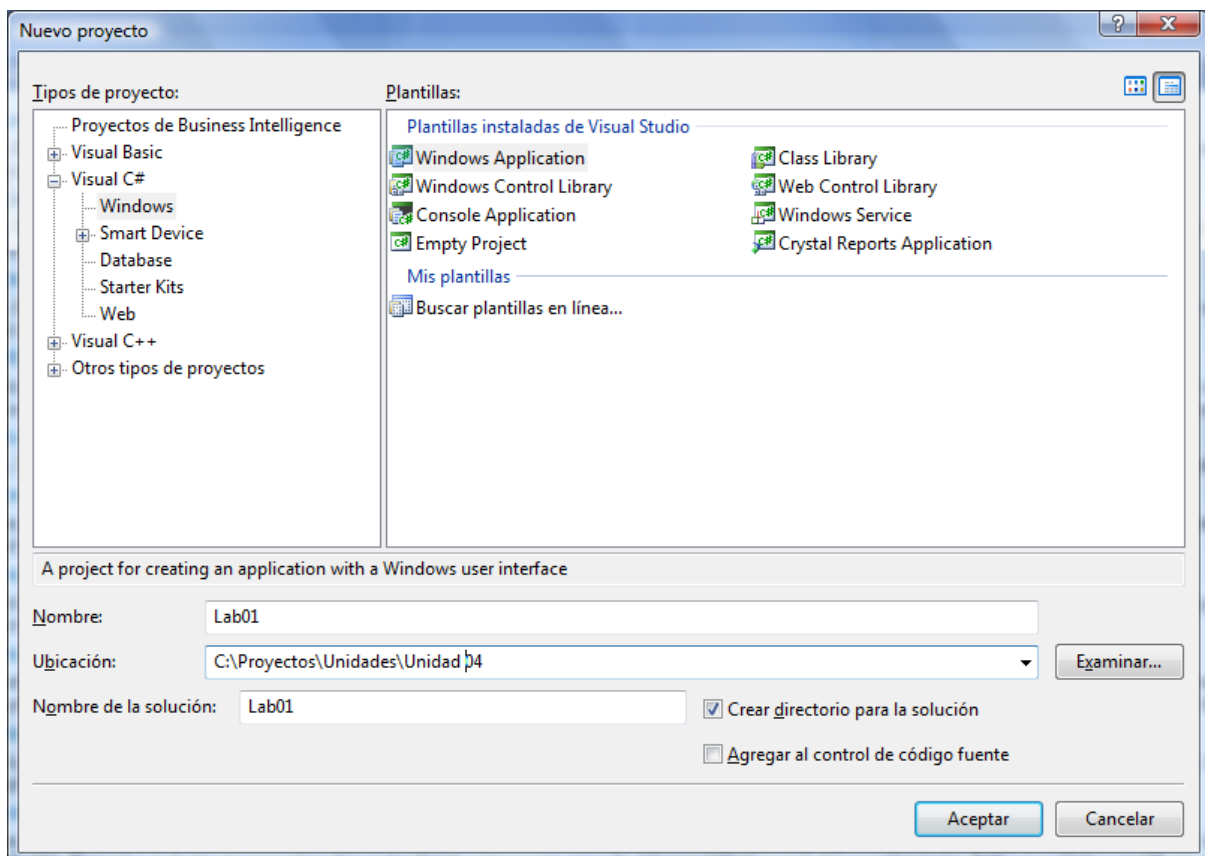
#### Pasos

- 1) Abra el Visual Studio 2005 y haga clic en Archivo -> Nuevo -> Proyecto
- 2) Elija un proyecto de la categoría C# -> Windows -> Console Application.

A) Completar los siguientes datos:

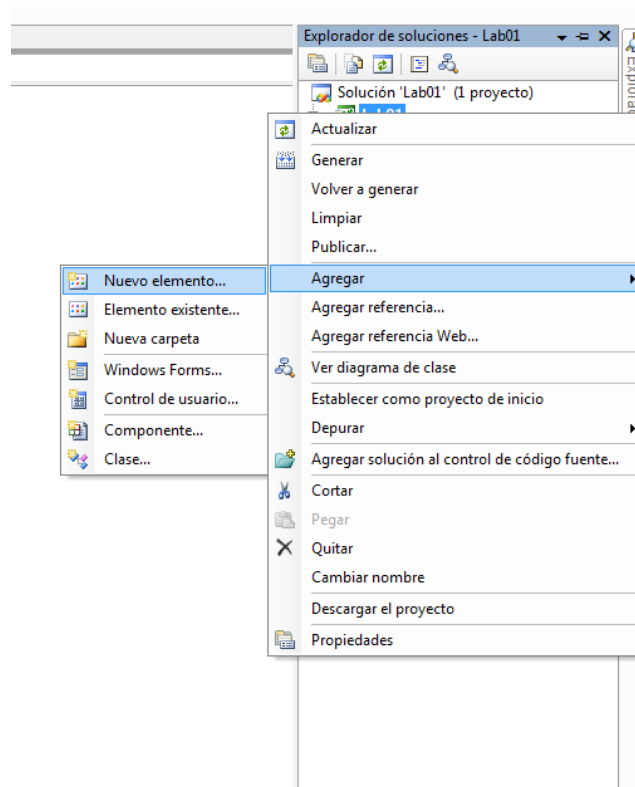
- Nombre del Proyecto (**Name**): *Lab01*
- Ubicación de la solución (**Location**):

*C:\Proyectos\Unidades\Unidad 04*



- 3) Presione Ok.

4) hacemos clic con el botón derecho sobre el proyecto Lab01. Allí elegimos Agregar -> Nuevo Elemento



5) Elegimos Text File y lo llamamos agenda.txt

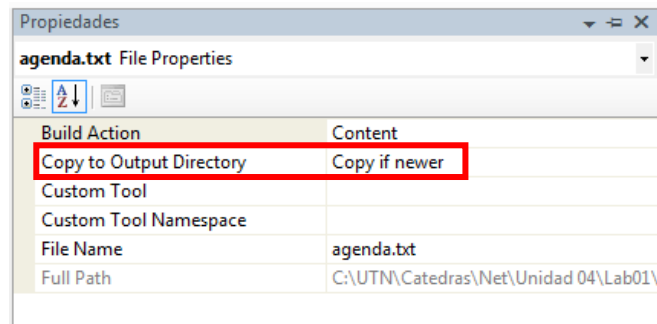
6) En agenda.txt pegamos:

```
Juana;De Blanco;jdblanco@gmail.com;411-1111  
Jose;Gonzales;johny_smithy@gmail.com;422-2222  
Rodrigo;Rodriguez;rodrirodri@gmail.com;433-3333
```

**Precaución:** dejar una línea en blanco al final, sin caracteres ni espacios en blanco sólo el ENTER.

Este será nuestro archivo inicial.

7) Hacemos clic sobre agenda y vamos a la pestaña de propiedades. Cambiamos la propiedad Copy to Output Directory a Copy if newer. Para que al compilar si el archivo agenda.txt no existe el mismo se genere.



8) En el archivo Program.cs agregamos la directiva `using System.IO;` para tener acceso directo a las clases de entrada salida que utilizaremos a continuación.

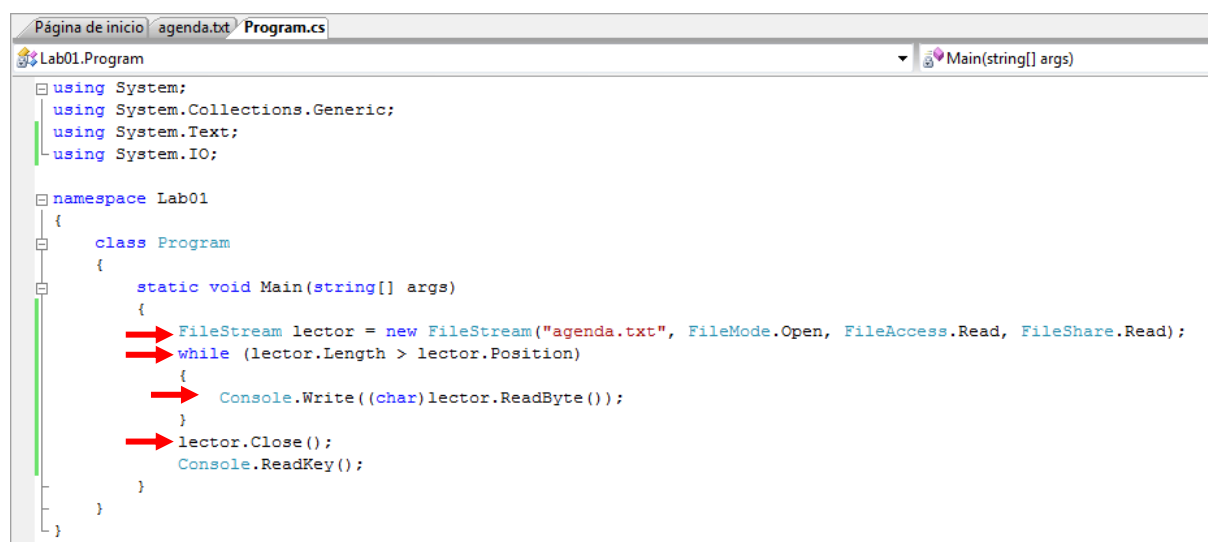
Para acceder y escribir en el archivo utilizaremos clases que se encuentran agrupadas en *System.IO* que es el namespace que agrupa las clases que se relacionan con las entradas y salidas de datos.

La entrada/salida (E/S) en C# está organizada alrededor del concepto de *stream*, o canal. Un canal puede corresponder a un fichero en disco, pero también a una cadena o canales de comunicaciones entre procesos. En este laboratorio lo aplicaremos a escritura y lectura de ficheros en disco.

9) Dentro del método `main` declararemos un objeto de tipo `FileStream` llamado `lector`. En el constructor del `FileStream` especificamos el nombre del archivo, el modo de apertura (sólo lectura), el tipo de acceso (en este caso para leer) y cómo les permitimos a las demás aplicaciones mientras utilizamos el archivo (en este caso les permitimos leerlo mientras lo utilizamos pero no modificarlo)

10) Una vez que terminamos de leer el archivo cerramos el stream.

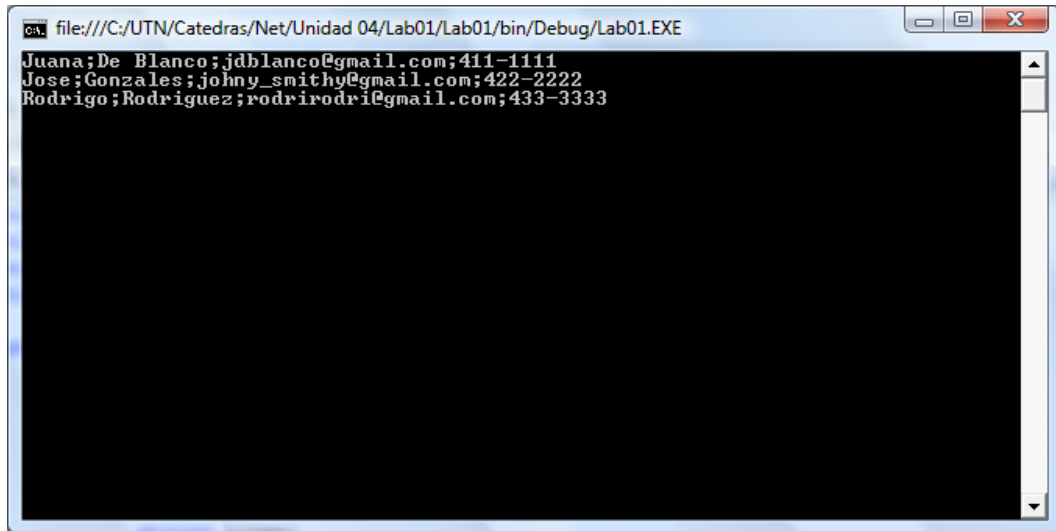
11) Utilizaremos entonces el método `ReadByte` para leer el archivo y mostrarlo. Y al finalizar escribimos `Console.ReadKey();` para que la consola no se cierre hasta que oprimamos una tecla.



Aquí estamos leyendo byte por byte cada caracter del archivo y mostrándolo. Cuando abrimos el archivo la posición del puntero que nos indica en que parte del archivo (`lector.Position`) indica estamos está al comienzo del mismo (posición 0). Cada vez que se ejecuta el método `ReadByte()` la posición avanza un byte. Repetimos esto hasta que llegamos al final del archivo (`lector.Length`).

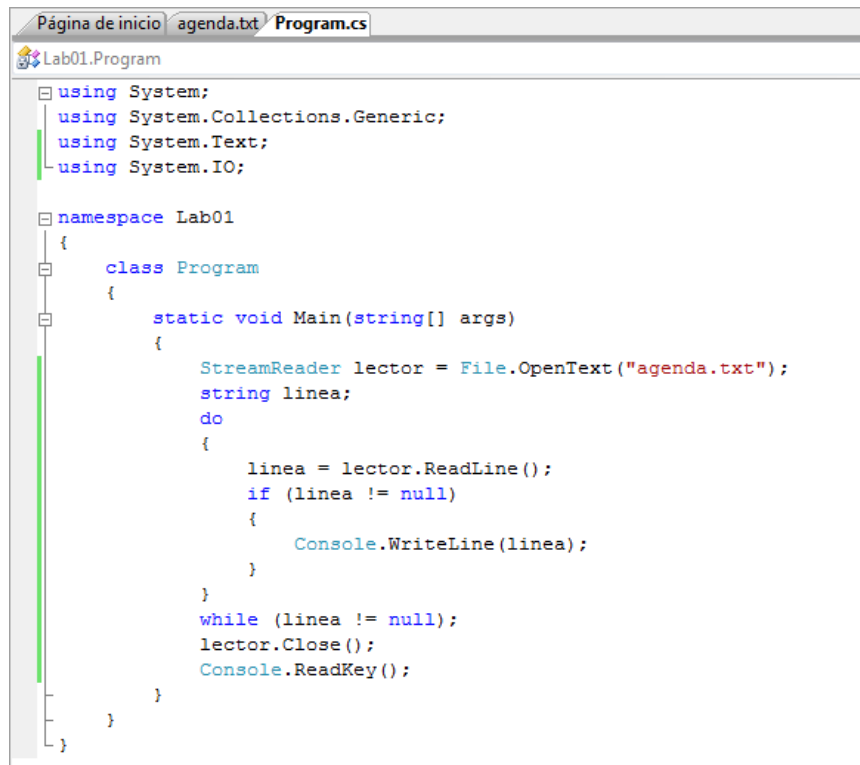
Finalmente cerramos el archivo.

- 12) Presionamos F5 y ejecutamos la aplicación. Entonces deberemos observar el siguiente resultado:



```
file:///C:/UTN/Catedras/Net/Unidad 04/Lab01/Lab01/bin/Debug/Lab01.EXE
Juana;De Blanco;jdblanc@gmail.com;411-1111
Jose;Gonzales;johny_smithy@gmail.com;422-2222
Rodrigo;Rodriguez;rodrirodri@gmail.com;433-3333
```

- 13) A continuación cambiaremos la clase utilizada para leer archivos por una más eficiente para leer archivos de texto, que además, nos permite leer línea a línea en lugar de por caracteres. Utilizaremos el `StreamReader` y para leer los datos utilizaremos el método `ReadLine()`



```
Página de inicio agenda.txt Program.cs
Lab01.Program
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace Lab01
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamReader lector = File.OpenText("agenda.txt");
            string linea;
            do
            {
                linea = lector.ReadLine();
                if (linea != null)
                {
                    Console.WriteLine(linea);
                }
            } while (linea != null);
            lector.Close();
            Console.ReadKey();
        }
    }
}
```

14) Si ejecutamos la aplicación deberíamos obtener el mismo resultado que antes.

15) Ahora modificaremos el programa para mostrar la información de forma más presentable. Para lo cual utilizaremos el método Split

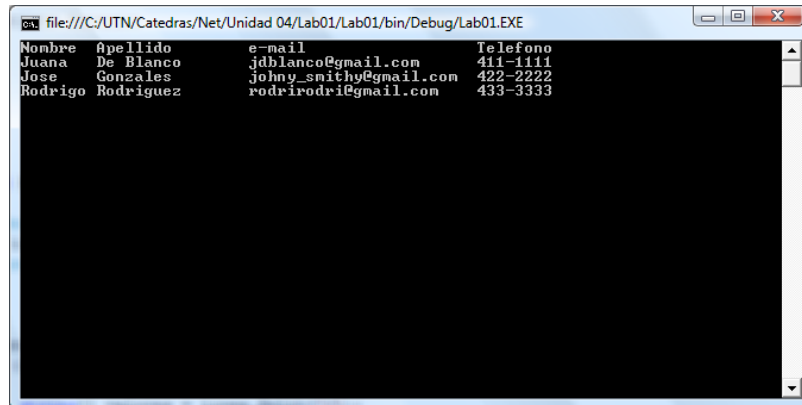


```
Página de inicio agenda.txt Program.cs
Lab01.Program
Main(string[] args)
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace Lab01
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamReader lector = File.OpenText("agenda.txt");
            string linea;
            Console.WriteLine("Nombre\tApellido\tE-mail\t\tTelefono");
            do
            {
                linea = lector.ReadLine();
                if (linea != null)
                {
                    string[] valores = linea.Split(';');
                    Console.WriteLine("{0}\t{1}\t{2}\t{3}", valores[0], valores[1], valores[2], valores[3]);
                }
            } while (linea != null);
            lector.Close();
            Console.ReadKey();
        }
    }
}
```

El método Split permite dividir un string en un array de varios strings divididos por un carácter en particular. En este caso el ';'.

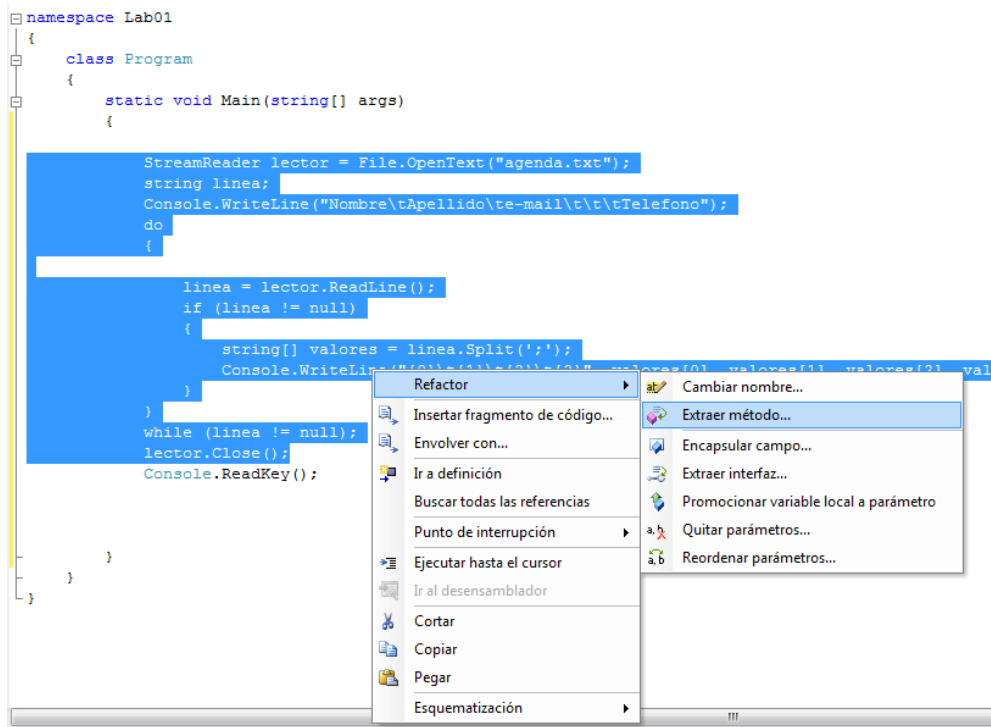
16) Ejecutamos F5 y el resultado obtenido será:



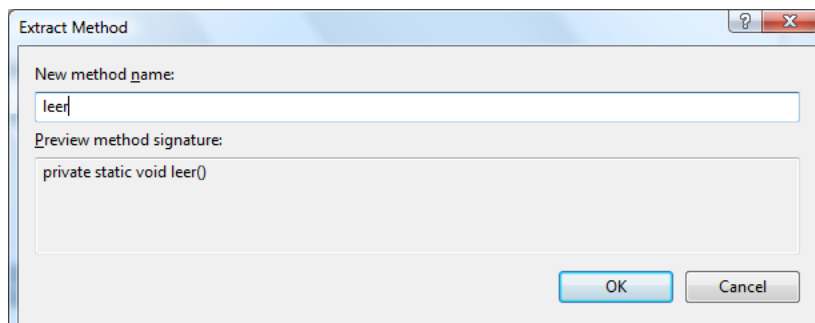
Nombre	Apellido	e-mail	Telefono
Juana	De Blanco	jdblanc@gmail.com	411-1111
Jose	Gonzales	johny_smithy@gmail.com	422-2222
Rodrigo	Rodriguez	rodrirodri@gmail.com	433-3333

Ahora modificaremos el programa para que además de mostrarnos el contenido nos permita agregar nuevos registros. Pero antes reorganizaremos el código para que sea más fácil de entender.

17) Seleccionamos todo el código concerniente a la lectura del archivo y hacemos clic con botón derecho. Elegimos Refactor → Extraer método.



18) Luego aparecerá una ventana para que ingresemos el nombre del nuevo método que llamaremos Leer.



- 19) Presionamos OK y Visual Studio generará un nuevo método formado con todo el código que estaba seleccionado y reemplazará dicho código por una llamada al nuevo método.

```
{
} class Program
{
    static void Main(string[] args)
    {
        → leer();
        Console.ReadKey();

    }
}

private static void leer()
{
    StreamReader lector = File.OpenText("agenda.txt");
    string linea;
    Console.WriteLine("Nombre\tApellido\tE-mail\t\tTelefono");
    do
    {
        linea = lector.ReadLine();
        if (linea != null)
        {
            string[] valores = linea.Split(';');
            Console.WriteLine("{0}\t{1}\t{2}\t{3}", valores[0], valores[1], valores[2], valores[3]);
        }
    } while (linea != null);
    lector.Close();
}
```

- 20) Luego creamos un nuevo método escribir que agregará texto al final del archivo. El mismo utiliza la clase StreamWriter para escribir línea por línea

```

private static void escribir()
{
    → StreamWriter escritor = File.AppendText("agenda.txt");
    Console.WriteLine("Ingrese nuevos contacto");
    string rta = "S";
    while (rta == "S")
    {
        Console.Write("Ingrese nombre:");
        string nombre = Console.ReadLine();
        Console.WriteLine();
        Console.Write("Ingrese apellido:");
        string apellido = Console.ReadLine();
        Console.WriteLine();
        Console.Write("Ingrese e-mail:");
        string email = Console.ReadLine();
        Console.WriteLine();
        Console.Write("Ingrese telefono:");
        string telefono = Console.ReadLine();
        Console.WriteLine();
        Console.WriteLine();

        → escritor.WriteLine(nombre + ";" + apellido + ";" + email + ";" + telefono);

        Console.Write("¿Desea ingresar otro contacto? (S/N)");
        rta = Console.ReadLine();
    }
    → escritor.Close();
}

```

En este caso abrimos el archivo con File.AppendText que crea un Stream (canal) optimizado para leer agregar texto al final del archivo. Luego escribimos los datos en el archivo utilizando escritor.WriteLine. Una vez que terminamos de escribir todos los datos en el archivo lo cerramos.

21) Entonces agregamos en el programa principal la llamada el método Escribir() para agregar los nuevos contactos y volvemos a agregar una llamada al método leer para ver el resultado de nuestro trabajo. Siempre con un Console.ReadKey(); para que la consola nos deje observar el resultado antes de cerrarse.

22) Presionar F5.

**Aclaración:** Si dentro del Visual Studio revisan el archivo agenda.txt desde el explorador de soluciones verán que no ha sido modificado. Esto se debe a que no es este el archivo que hemos modificado. En el paso 7) Seteamos la propiedad Copy to Output Directory del archivo. Esto significa que una copia del archivo se escribe en el directorio donde se compila el ejecutable este es el archivo que hemos estado utilizando.

El mismo se encuentra dentro del directorio de la solución, dentro del directorio del proyecto dentro de la carpeta \bin\Debug\

Si crearon el proyecto en la carpeta que se indicó en el paso 2 el archivo estará en la carpeta

C:\Proyectos\Unidades\Unidad 04\Lab01\Lab01\bin\Debug\



El archivo se llama agenda.txt igual que en la solución pero este contendrá los nuevos contactos que agregamos.

## 1.1. Archivos XML.

### Objetivos

Crear un archivo XML y guardar los datos en el mismo. Abrir el archivo y acceder a los datos del mismo

### Duración Aproximada

15 minutos

### Pasos

- 1) Las clases que sirven para manipular archivos XML están en el namespace System.XML por lo que agregamos al comienzo del archivo la directiva using System.XML;
- 2) Creamos un método EscribirXML. En este método guardaremos en un archivo los datos del archivo agenda.txt con formato XML.

Este método es similar al método leer pero en lugar de mostrar los datos en la consola lo escribiremos en un archivo llamado agendaxml.xml

```
private static void escribirXML()
{
    1 ➡ XmlTextWriter escritorXML = new XmlTextWriter("agendaxml.xml", null);
    2 ➡ escritorXML.Formatting = Formatting.Indented; //esto no es necesario pero hará más fácil para nosotros leer el contenido del archivo
    3 ➡ escritorXML.WriteStartDocument(true);
    4 ➡ escritorXML.WriteStartElement("DocumentElement");
    //este elemento no es necesario lo agregaremos para compatibilidad con los xml generados en el siguiente laboratorio
    StreamReader lector = File.OpenText("agenda.txt");
    string linea;
    do
    {
        linea = lector.ReadLine();
        if (linea != null)
        {
            string[] valores = linea.Split(';');
            escritorXML.WriteStartElement("contactos");
            escritorXML.WriteStartElement("nombre");
            escritorXML.WriteValue(valores[0]);
            escritorXML.WriteEndElement(); //cerramos el tag de nombre
            escritorXML.WriteStartElement("apellido");
            escritorXML.WriteValue(valores[1]);
            escritorXML.WriteEndElement(); //cerramos el tag de apellido
            escritorXML.WriteStartElement("email");
            escritorXML.WriteValue(valores[2]);
            escritorXML.WriteEndElement(); //cerramos el tag de email
            escritorXML.WriteStartElement("telefono");
            escritorXML.WriteValue(valores[3]);
            escritorXML.WriteEndElement(); //cerramos el tag de telefono
            escritorXML.WriteEndElement(); //cerramos el tag de contactos
        }
    } while (linea != null);
    4 ➡ escritorXML.WriteEndElement(); //cerramos el tag de DocumentElement
    4 ➡ escritorXML.WriteEndDocument();
    4 ➡ escritorXML.Close();

    lector.Close();
}
```

Las flechas rojas y el rectángulo rojo indican las sentencias específicas para crear el archivo XML.

1: Crea el objeto EscritorXML del tipo XmlTextWriter que nos permitirá generar el archivo y escribirle los datos.

2: WriteStartDocument es el método que se encarga de escribir el encabezado del documento donde hay información de la estructura del xml, la versión y la codificación

3: Son las sentencias para guardar los nodos de XML que es donde se almacena la información.

4: Es la sentencia que se utiliza para indicar en que es el final del documento XML.

El XML resultante es: (excepto por los números de línea)

```
1.  <?xml version="1.0" standalone="yes"?>
2.  <DocumentElement>
3.    <contactos>
4.      <nombre>Juana</nombre>
5.      <apellido>De Blanco</apellido>
6.      <email>jdblanc@gmail.com</email>
7.      <telefono>411-1111</telefono>
8.    </contactos>
9.    <contactos>
10.     <nombre>Jose</nombre>
11.     <apellido>Gonzales</apellido>
12.     <email>johny_smithy@gmail.com</email>
13.     <telefono>422-2222</telefono>
14.   </contactos>
15.   <contactos>
16.     <nombre>Rodrigo</nombre>
17.     <apellido>Rodriguez</apellido>
18.     <email>rodrirodri@gmail.com</email>
19.     <telefono>433-3333</telefono>
20.   </contactos>
21. </DocumentElement>
```

El archivo `agendaxml.xml` queda almacenado en el mismo directorio de `agenda.txt` y para comprender como se genera el archivo, la tabla a continuación detalla cual sentencia es responsable de generar cada una de las líneas del xml

Línea	Sentencia que la crea
1	<code>escritorXML.WriteStartDocument(true);</code>
2	<code>escritorXML.WriteStartElement("DocumentElement");</code>
3, 9 y 15	<code>escritorXML.WriteStartElement("contactos");</code>
4, 10 y 16	<code>escritorXML.WriteStartElement("nombre");</code> <code>escritorXML.WriteValue(valores[0]);</code> <code>escritorXML.WriteEndElement(); //cerramos el tag de nombre</code>
5, 11 y 17	<code>escritorXML.WriteStartElement("apellido");</code> <code>escritorXML.WriteValue(valores[1]);</code> <code>escritorXML.WriteEndElement(); //cerramos el tag de apellido</code>
6, 12 y 18	<code>escritorXML.WriteStartElement("email");</code> <code>escritorXML.WriteValue(valores[2]);</code> <code>escritorXML.WriteEndElement(); //cerramos el tag de email</code>
7, 13 y 19	<code>escritorXML.WriteStartElement("telefono");</code> <code>escritorXML.WriteValue(valores[3]);</code> <code>escritorXML.WriteEndElement(); //cerramos el tag de telefono</code>
8, 14 y 20	<code>escritorXML.WriteEndElement(); //cerramos el tag de contactos</code>
21	<code>escritorXML.WriteEndElement(); //cerramos el tag de DocumentElement</code>

- 3) Luego creamos el método LeerXML que nos permitirá visualizar el archivo XML que acabamos de crear.

```
private static void leerXML()
{
    XmlTextReader lectorXML = new XmlTextReader("agendaxml.xml");

    string tagAnterior = "";
    while (lectorXML.Read())
    {
        if (lectorXML.NodeType == XmlNodeType.Element)
        {
            tagAnterior = lectorXML.Name;
        }
        else if (lectorXML.NodeType == XmlNodeType.Text)
        {
            Console.WriteLine(tagAnterior+": "+lectorXML.Value);
        }
    }
    lectorXML.Close();
}
```

Para leer utilizamos el objeto LectorXML que es de la clase XmlTextReader.

Luego para leer cada uno de los nodos XML que contiene el archivo utilizamos el método Read. Una vez que hemos leído el nodo podemos acceder a los elementos del último nodo leído a través de los métodos del XmlTextReader: NodeType, Name y Text.

Los nodos como contactos, DocumentElement, nombre, apellido, email y teléfono son nodos de tipo Elemento mientras que los valores son nodos de tipo Text.

- 4) Finalmente modificamos nuestro programa principal (main) de la siguiente forma:

```
static void Main(string[] args)
{
    Console.WriteLine("Presione una tecla para generar el archivo agendaxml.xml con los datos de agenda.txt");
    Console.ReadKey();
    escribirXML();
    Console.WriteLine("Archivo agendaxml.xml generado correctamente\n\nPresione una tecla para ver su contenido");
    Console.ReadKey();
    Console.WriteLine();
    leerXML();
    Console.ReadKey();
}
```

- 5) Presionamos F5.