



#### TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

### INTRODUCCIÓN

Las operaciones de archivo son algo a lo que todo programador debe enfrentarse en un momento u otro. La clase **System.IO** contiene métodos para leer y escribir en y desde archivos. Esta clase simplifica la E/S de archivos y su manipulación y brinda un gran control de acceso a archivos. En esta práctica aprenderá a leer y escribir datos en y desde archivos.

#### Como acceder a archivos

Para acceder al contenido de un archivo trabajaremos con objetos de alguna clase derivada de **Stream**. Ésta se encuentra definida en la Clase **System.IO** una clase abstracta, es decir, no se pueden crear objetos a partir de ella, se utiliza como base de las clases **FileStream**, **MemoryStream**, **BufferedStream**, **CryptoStream** y **NetworkStream**, cada una de ellas especializada en almacenar o recuperar el flujo de información de un determinado medio, como puede ser el disco, la memoria o una conexión de red.

Para poder crear un objeto que nos permita la conexión con un archivo utilizamos la clase derivada **FileStream** a la cual deberemos enviarle como parámetro la ruta completa seguida del nombre del archivo.

Para poder leer y escribir datos en el flujo tenemos dos parejas de métodos `ReadByte( )` y `Read( )`, para lectura, y `WriteByte( )` y `Write( )` para escritura. Los primeros leen o escriben un solo byte mientras que los segundos leen o escriben una secuencia de bytes. `WriteByte( )` necesita como único parámetro el byte a escribir, mientras que `ReadByte( )` no toma parámetros y devuelve el dato leído como un `int`.

Los métodos `Read( )` y `Write( )` toman ambos tres parámetros: un arreglo de tipo `byte` en el que va a almacenarse o donde se encuentra la secuencia de bytes, un entero indicando el elemento de ese arreglo a partir del que se almacenará y otro especificando el número de bytes que hay que leer o escribir. En los dos casos la posición, entregada como segundo parámetro, hace referencia al punto del arreglo donde van a alojarse los bytes leídos o se encuentran los bytes que se van a escribir, no a la posición relativa en el flujo.

Cuando se abre un flujo de datos se crea un puntero o marcador que generalmente, se sitúa al inicio de ese flujo. Cada vez que se efectúa una operación de lectura o escritura el marcador se desplaza hacia delante, indicando la posición en la que se efectuará la próxima operación.

La propiedad **Length** contiene el número de bytes de información existentes en el flujo. Conociendo ese dato, podemos usar la Propiedad **Position** tanto para saber cuál es la posición actual como para modificarla.

Los métodos `Write( )` y `Read( )`, así como `WriteByte( )` y `ReadByte( )`, heredados de la clase `Stream` son de bajo nivel, ya que sólo nos permiten trabajar con bytes individuales o secuencias de bytes.

En el ámbito `System.IO` encontramos algunas clases más especializadas, como **StreamReader** y **StreamWriter**, que facilitan la lectura y escritura de datos que se ajustan a los tipos de datos que ya conocemos y estamos habituados a usar en variables y propiedades.

Para escribir datos, utilizando un **StreamReader**, empleamos los métodos `Write( )` y `WriteLine( )` que pueden tomar parámetros de distintos tipos: caracteres, números enteros, decimales, cadenas, etc. El método `WriteLine( )` introduce además un salto de línea al final de los datos.

La clase **StreamReader** facilita varios métodos de lectura: `Read( )`, con dos versiones distintas, `ReadLine( )` y `ReadToEnd`. El primero lee un solo carácter o un número máximo de caracteres, el



#### TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

segundo lee una línea completa y la devuelve como una cadena mientras que el tercero lee todo el contenido del archivo y lo devuelve como una cadena.

#### Apertura de Archivos

Para leer y almacenar información en un archivo en disco utilizaremos habitualmente la clase **FileStream**. Usualmente indicaremos el nombre del archivo que se va a abrir, incluyendo la ruta completa, seguido de un valor de la enumeración **FileMode** indicando el modo de apertura y uno de los valores de **FileAccess** comunicando la utilización que vamos a hacer del archivo.

#### Valores de la enumeración FileMode

FileMode	Cómo se abre el archivo
Open	Sólo si existe, en caso contrario lanza la excepción <a href="#">FileNotFoundException</a> .
CreateNew	Se crea y si existe lanza una excepción <a href="#">IOException</a>
Create	Se crea y si existe lo sobrescribe
OpenOrCreate	Si existe lo abre en caso contrario lo crea.
Truncate	Asumiendo que existe y truncando su contenido tras la apertura
Append	Se crea si no existe, en caso contrario lo abre y se posiciona al final para seguir añadiendo datos

#### Valores de la Enumeración FileAccess

FileAccess	Operación que podrá efectuarse en el archivo
Read	Lectura
Write	Escritura
ReadWrite	Lectura y escritura

#### Flujo de Datos Primitivo

Muchas aplicaciones requieren de que los datos sean almacenados de acuerdo al tipo de información que están almacenando para ello el espacio de nombres System.IO proporciona las clases **BinaryReader** y **BinaryWriter**, las cuales permiten leer y escribir tipos de datos primitivos en un archivo.



#### TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

### Formato de Rutas y nombres de archivo

Forma 1: `string ruta="archivo.txt";`

Se guarda en la carpeta de proyecto, `proyecto>bin>debug`

Forma 2: `string ruta="../..../prueba.txt";`

Se guarda tres carpetas atrás de donde está el archivo de proyecto ejecutable, ".exe" (`bin>debug`)

Forma 3: `string ruta="c:/usuarios/misdoc/tecnicas/prueba.txt";`

Se guarda en la ruta absoluta indicada. Si la ruta no es correcta se genera una excepción.

### StreamWriter y StreamReader

Permiten manipular la información de archivos en formato texto.

#### Para escribir en archivos:

- `StreamWriter SW = new StreamWriter(ruta);`

`//Si el archivo no existe lo crea y si existe lo sobre escribe.`

- `StreamWriter SW = new StreamWriter(ruta, true);`

`//Si el archivo no existe lo crea y si existe comienza a escribir hasta el final del contenido del archivo.`

- `SW.Write("texto");` //Escribe y conserva el cursor en la misma línea.
- `SW.WriteLine(variable);` //Escribe el valor de la variable y luego pasa el cursor a la siguiente línea.
- `SW.Close();` //Se debe cerrar el archivo.

#### Para leer de archivos

- `StreamReader SR = new StreamReader(Ruta);`
- `int c = SR.Read();` //Retorna un solo carácter en el equivalente numérico al ASCII.
- `string s = SR.ReadLine();` //Lee una línea completa
- `string s= SR.ReadToEnd` //Lee todo el contenido del archivo y lo devuelve como cadena.

```
while(!SR.EndOfStream) //Ciclo para recorrer carácter por carácter o línea por línea hasta el final
{
    //Del archivo.
}
```



#### TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

#### BinaryWriter y BinaryReader

##### BinaryWriter

Permite escribir y leer de archivos en modo binario (almacena y recupera las variables con el mismo tipo de dato que tenían en el código).

//Si el archivo no existe lo crea y si existe se va al final del archivo y comienza ahí la escritura.

- FileStream FS= new FileStream(Ruta, FileMode.Append, FileAccess.Write);

//Si el archivo no existe lo crea y si existe lo sobre escribe.

- FileStream FS= new FileStream(Ruta, FileMode.Create, FileAccess.Write);
- BinaryWriter BW= new BinaryWriter(FS);
- BW.Write(variable); //Almacena la variable con el mismo formato que tenía en el programa.
- BW.Close();
- FS.Close();

##### BinaryReader

Permite leer del archivo, la lectura debe extraer el dato en el mismo formato que tenía al ser almacenada.

- FileStream FS= new FileStream(Ruta, FileMode.Open, FileAccess.Read);
- BinaryReader BR= new BinaryReader(FS);
- int a = BR.ReadInt32();
- string b = BR.ReadString();
- float c = BR.ReadSingle();
- char d = BR.ReadChar();

//Para recorrer secuencialmente el archivo

```
While(FS.Position < FS.Length)
```

```
{  
}
```

//Lee en forma iterativa mientras no llegue al final del archivo.

```
BR.Close(); //cerrar el filtro
```

```
FS.Close(); //Cerrar el archivo
```



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

**Ejercicios de la Práctica**

Mediante estos ejercicios el alumno entenderá a manejar las clases asociadas al manejo de archivos.

Realiza cada uno de los siguientes ejercicios en un método distinto. Crea un menú para acceder a ellos.

**Números**

1. Dado un listado de números enteros guardados en un archivo, generar otro archivo que almacene los números que sean primos y la sumatoria de los mismos.

Ej.

Arch1	Arch2
1	1 1
2	2 3
5	5 15
9	11 66
11	
15	

2. Dados dos archivos de números enteros de 5 dígitos ordenados en forma creciente, generar un archivo que contenga todos los números, ordenados de igual forma. No hay números repetidos.

**Cadenas**

3. Dado un archivo con una lista de nombres NO ordenados alfabéticamente, generar otro archivo que contenga los nombres ordenados de manera alfabética y convertidos a Mayúsculas.
4. Dado un archivo de números enteros entre 0 y 255, guardar en otro archivo los caracteres correspondientes según el código ASCII.
5. Leer un archivo de caracteres que representa un texto formado por oraciones terminadas en punto. Copiarlo en otro archivo eliminando los espacios en blanco.

**Arreglos**

6. Solicitar dos vectores de igual dimensión por teclado y almacenar en dos archivos, generar un tercer archivo que almacene el producto vectorial.



**ANEXO A.**

**Ejemplos**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ReadByteWriteByte
{
    class Program
    {
        static void Main(string[] args)
        {
            byte dato = 5;
            byte[] datos = {1,2,3,4,5}, datosleidos=new byte[2];
            int dato1, dato2;
            //Para escribir
            FileStream FS = new FileStream("prueba.txt", FileMode.Create, FileAccess.Write); //abriendo un
archivo
            FS.WriteByte(dato);
            FS.Write(datos, 0, 5);
            FS.Close(); //cerrando un archivo

            //Para leer
            FS = new FileStream("prueba.txt", FileMode.Open, FileAccess.Read);
            dato1 = FS.ReadByte();
            dato2 = FS.Read(datosleidos, 0, 2);
            FS.Close();
            Console.WriteLine(dato1);
            Console.WriteLine(dato2);

            foreach(byte B in datosleidos)
                Console.Write(B + " ");
            Console.WriteLine();
        }
    }
}
```



#### TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace practica11
{
    class Program
    {
        FileStream FS;
        static void Main(string[] args)
        {
            Program objeto_archivo = new Program();

            string ruta_archivo = "prueba.txt";
            objeto_archivo.abrir(ruta_archivo);
            objeto_archivo.cerrar();

            ruta_archivo = "../prueba.txt";
            objeto_archivo.abrir(ruta_archivo);
            objeto_archivo.cerrar();

            ruta_archivo = "c:/prueba.txt";
            objeto_archivo.abrir(ruta_archivo);
            objeto_archivo.cerrar();
        }
        public void abrir(string ruta)
        {
            FS = new FileStream(ruta, FileMode.Create, FileAccess.ReadWrite);
            Console.WriteLine("Archivo creado y abierto");
        }
        public void cerrar()
        {
            FS.Close();
            Console.WriteLine("Archivo cerrado");
        }
    }
}
```



#### TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace practica11
{
    class Program
    {
        StreamReader SR;
        StreamWriter SW;
        static void Main(string[] args)
        {
            Program objeto_archivo = new Program();

            string ruta_archivo = "prueba.txt";
            objeto_archivo.escribir(ruta_archivo);
            Console.WriteLine("*****Leyendo linea*****");
            Console.WriteLine(objeto_archivo.leer_linea(ruta_archivo));
            Console.WriteLine("*****Leyendo caracter*****");
            Console.WriteLine((char)objeto_archivo.leer_caracter(ruta_archivo));
            Console.WriteLine("*****Leyendo texto hasta el final del archivo*****");
            Console.WriteLine(objeto_archivo.leer_todo(ruta_archivo));
        }

        public void escribir(string ruta)
        {
            SW = new StreamWriter(ruta);
            SW.WriteLine("Tecnicas de programacion");
            SW.WriteLine("Manejo de archivos");
            SW.Write(" Practca 11");
            SW.Close();
        }

        public string leer_linea(string ruta)
        {
            SR = new StreamReader(ruta);
            string cadena_leida;
            cadena_leida = SR.ReadLine();
            return cadena_leida;
        }

        public int leer_caracter(string ruta)
        {
            SR = new StreamReader(ruta);
            int cadena_leida;
            cadena_leida = SR.Read();
            return cadena_leida;
        }

        public string leer_todo(string ruta)
        {
            SR = new StreamReader(ruta);
            string cadena_leida;
            cadena_leida = SR.ReadToEnd();
            return cadena_leida;
        }
    }
}
```



#### TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace clase_binary
{
    class Program
    {
        FileStream FS;
        BinaryWriter BW;
        BinaryReader BR;
        static void Main(string[] args)
        {
            Program myFile = new Program();
            myFile.escribe();
            myFile.lee();
        }
        public void escribe()
        {
            int a=5,b=8;
            string c="hola";
            float d=3.4F;
            char e='G';

            FS = new FileStream("prueba.dat", FileMode.Append, FileAccess.Write);
            BW = new BinaryWriter(FS);
            BW.Write(a);
            BW.Write(c);
            BW.Write(d);
            BW.Write(b);
            BW.Write(e);
            BW.Close();
            FS.Close();
        }
        public void lee()
        {
            int A, B;
            string C;
            float D;
            char E;
            FS = new FileStream("prueba.dat", FileMode.Open, FileAccess.Read);
            BR = new BinaryReader(FS);
            A = BR.ReadInt32();
            C = BR.ReadString();
            D = BR.ReadSingle();
            B = BR.ReadInt32();
            E = BR.ReadChar();
            Console.WriteLine("Valores enteros {0}, {1}", A, B);
            Console.WriteLine("Valor cadena {0}", C);
            Console.WriteLine("Valor flotante {0}", D);
            Console.WriteLine("Valor caracter {0}", E);
            BR.Close();
            FS.Close();
        }
    }
}
```



### ANEXO B. XML Primera entrega (Información básica de sus variables y clases)

Con la etiqueta <summary> podemos hacer una descripción de la información de nuestras clases y variables que podrá ser visualizada al utilizarla en otra clase o proyecto.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication1
{
    /// <summary>
    /// Clase especial <c><b>Numeros</b></c> para manejar numeros
    /// </summary>
    class Numeros
    {
        /// <summary>
        /// Miembro público de la clase <c><b>Numeros</b></c>
        /// Representa la coordenada x.
        /// </summary>

        public int x;
    }
}
```