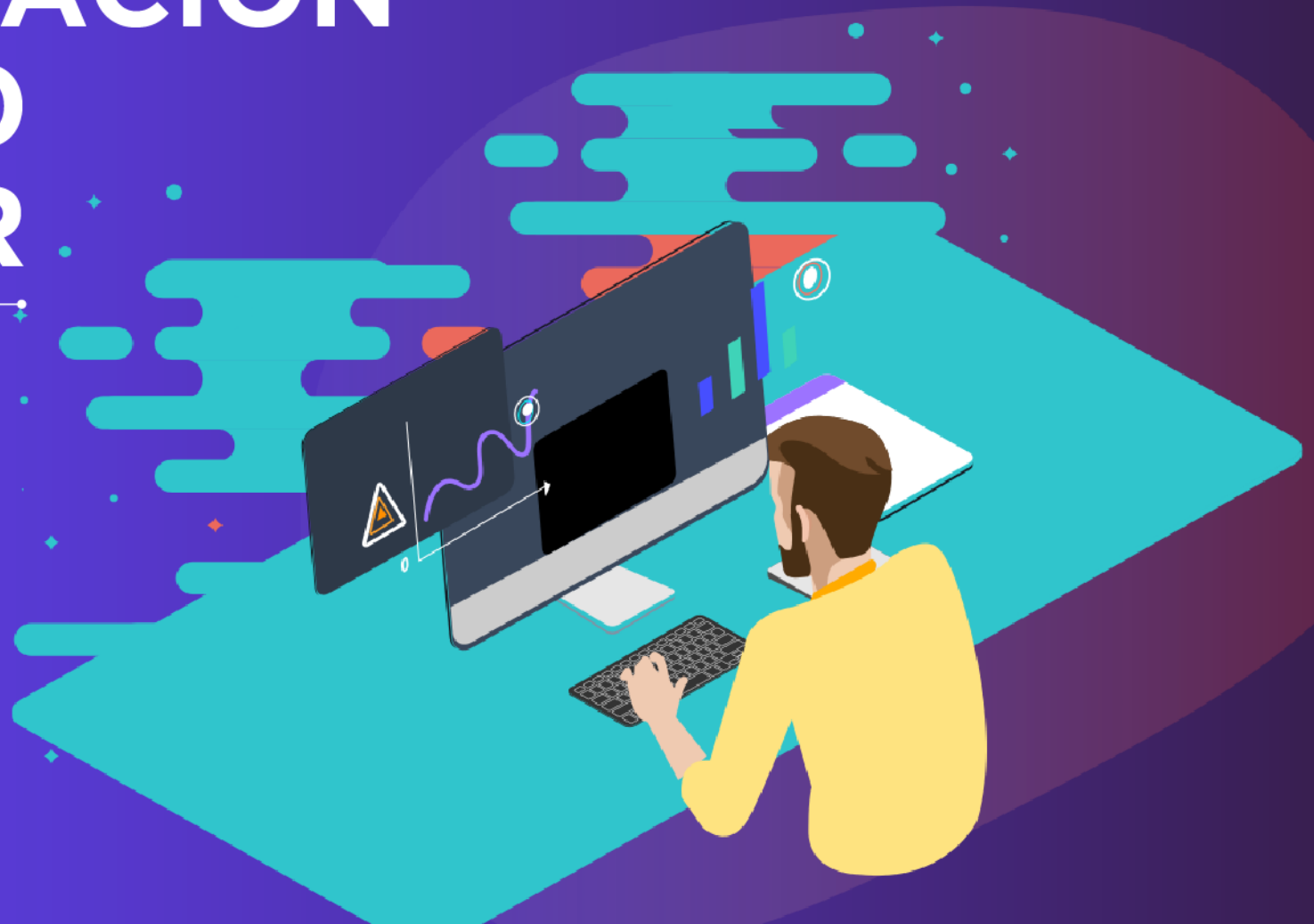
PROGRAMA

.NET ESPECIALIZACIÓN: NET

ESPECIALIZACIÓN ASP.NET 5.0 DEVELOPER



CURSO VIRTUAL

Fundamentos de Programación

Colecciones, Programación Concurrente y archivos

Instructor: Erick Aróstegui
earostegui@galaxy.edu.pe

GALAXY TRAINING

TEMAS

01	Arreglos(1,2 y n dimensiones)
02	List, ArrayList, SortedList, Dictionary y HashTable
03	Stack y Queue, Tuple y ValueTuple
04	Programación concurrente e Hilos(Thread)
05	Lectura y escritura de archivos(Stream I/O, File y FileInfo)

Arreglos

Se puede almacenar varias variables del mismo tipo en una estructura de datos de matriz. Se declara la matriz especificando el tipo de sus elementos

Si desea que la matriz almacene elementos de cualquier tipo, puede especificar object como su tipo.

```
static void Main(string[] args)
{
    //declarar una matriz unidimensional de 5 enteros
    int[] array1 = new int[5];

    //declarar y establecer valores de elementos de matriz
    int[] array2 = new int[] { 1, 3, 5, 7, 9 };

    //sistaxis alternativa
    int[] array3 = { 1, 2, 3, 4, 5, 6 };

    //declarar una matriz bidimensiona
    int[,] multiDimensionalArray2 = { {1,2,3 }, {4,5,6} };

    //declarar una matriz irregular
    int[][] jaggedArray = new int[6][];

    //establecer los valores de la primera matriz en la estructura de matriz irregular
    jaggedArray[0] = new int[4] { 1, 2, 3, 4 };
```

Descripción general de la matriz

- Una matriz puede ser unidimensional, multidimensional o irregular
- El número de dimensiones y la longitud de cada dimensión se establecen cuando se crea la instancia de matriz.
- Una matriz dentada es una matriz de matrices y, por lo tanto, sus elementos son tipos de referencia y se inicializan en null
- Los elementos de matriz pueden ser de cualquier tipo, incluido un tipo de matriz.

Matrices unidimensionales

Un matriz unidimensional es un tipo de datos estructurado que está formado por una colección finita y ordenada de datos del mismo tipo.

```
string[] stringArray = new string[6];
```

Puede inicializar los elementos de una matriz cuando declara la matriz. El especificador de longitud no es necesario porque se infiere por el número de elementos en la lista de inicialización.

```
int[] array1 = new int[] { 1, 3, 5, 7, 9 };
```

Matrices unidimensionales

El siguiente código muestra una declaración de una matriz de cadenas donde cada elemento de la matriz se inicializa con el nombre de un día:

```
string[] weekDays = new string[] {"lun","mar","mier","jue","vier" };
```

Puede evitar la new expresión y el tipo de matriz cuando inicializa una matriz tras la declaración, como se muestra en el siguiente código.

```
int[] array2 = { 1, 3, 5, 7, 9};
string[] weekDays2= { "lun", "mar", "mier", "jue", "vier" };
```

Matrices Multidimensionales

Puede declarar una variable de matriz sin crearla, pero debe usar el new operador cuando asigne una nueva matriz a esta variable.

```
int[] array3;
array3=new int[] { 1, 3, 5, 7, 9 }; //OK
//array3 ={ 1, 3, 5, 7, 9 }; //ERROR
```

Este también es un tipo de dato estructurado, que está compuesto por n dimensiones.

```
int[,,] array4 = new int[4, 2, 3];
```

Matrices Multidimensionales

Se puede inicializar la matriz sin especificar el rango.

```
int[,] array4 = { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

Si elige declarar una variable de matriz sin inicialización, debe usar el new operador para asignar una matriz a la variable.

```
int[,] array5;
array5 = new int[,] { {1,2 },{3, 4},{5, 6},{7, 8} };
//array5={ {1,2 },{3, 4},{5, 6},{7, 8} }; //ERROR
```

List

Representa una lista fuertemente tipada de objetos a los que se puede acceder por índice. Proporciona métodos para buscar, ordenar y manipular listas.

```
static void Main(string[] args)
{
    List <int> numberList=new List<int>();
     numberList.Add(5);
     numberList.Add(10);
     numberList.Add(15);
     numberList.Add(20);
     numberList.Add(25);
     numberList.Add(30);
     numberList.Add(35);

    List<string> departamentos = new List<string>();
     departamentos.Add("Lima");
     departamentos.Add("Tumbes");
     departamentos.Add("Arequipa");
     departamentos.Add("Cuzco");
     departamentos.Add("Cajamarca");
     departamentos.Add("Apurimac");
}
```

Crear un List

Es una colección genérica, por lo que es necesario especificar un parámetro de tipo para el tipo de datos que puede almacenar.

```
static void Main(string[] args)
{
    List<int> numeros = new List<int>();
    numeros.Add(1);   //agregar elementos usando el método agregar
    numeros.Add(3);
    numeros.Add(5);
    numeros.Add(7);

    List<string> ciudades = new List<string>();
    ciudades.Add("Lima");
    ciudades.Add("Villa el savador");
    ciudades.Add("Rimac");
    ciudades.Add("Callao");
    ciudades.Add(null);//se permiten nulos para la lista de tipos de referencias
    //agregar elementos usando la sintaxis del inicializador de colecciones
    var ciudad = new List<string>()
    {
        "Lima",
    "Villa el savador",
    "Rimac",
    "Callao"
    };
```

ArrayList

Es una colección no genérica de objetos cuyo tamaño aumenta de forma dinámica. Es lo mismo que Array excepto que su tamaño aumenta dinámicamente. También se puede usar para agregar datos desconocidos en los que no conoce los tipos y el tamaño de los datos.

```
ArrayList arrayList = new ArrayList();
arrayList.Add("hola1");
arrayList.Add("hola2");
arrayList.Add("hola3");
arrayList.Add("hola4");
arrayList.Add("hola5");
Console.WriteLine("Tamaño: " + arrayList.Count);
Console.WriteLine("Capacidad: " + arrayList.Capacity);
/*CREAR SORTED LIST
```

Crear un ArrayList

La ArrayList clase incluida en el System.Collections espacio de nombres. Crea un objeto ArrayList usando la new palabra clave

```
{
    ArrayList lista0 = new ArrayList();
    //o
    var lista1 = new ArrayList();// RECOMENDADO
```

SortedList

Representa una colección de pares clave / valor que están ordenados por claves y son accesibles por clave y por índice.

```
public class SortedList : ICloneable, System.Collections.IDictionary
```

Crear SortedList

La clase SortedList se define en el espacio de nombres System.Collections. Asegúrese de importar el espacio de nombres en su código antes de usarlo.

```
SortedList<int, string> sortedlist = new SortedList<int, string>();
```


Agregar SortedList

El método SortedList.Add () agrega elementos a la colección. El siguiente fragmento de código agrega varios elementos a la colección.

```
SortedList<int, string> sortedlistl1 = new SortedList<int, string>();
sortedlistl1.Add(1, "uno");
sortedlistl1.Add(2, "dos");
sortedlistl1.Add(3, "tres");
sortedlistl1.Add(4, "cuatro");
sortedlistl1.Add(5, "cinco");
SortedList<int, string> sortedlistl2 = new SortedList<int, string>();
sortedlistl2.Add(1, "uno");
sortedlistl2.Add(2, "dos");
sortedlistl2.Add(3, "tres");
sortedlistl2.Add(4, "cuatro");
sortedlistl2.Add(5, "cinco");
SortedList<double, int> sortedlistl3 = new SortedList<double, int>();
sortedlistl3.Add(1.0 ,100);
sortedlistl3.Add(2.5 ,250);
sortedlistl3.Add(3.9, 390);
sortedlistl3.Add(3.5 ,350);
sortedlistl3.Add(4.3 ,430);
```

Dictionary

Es una colección genérica que pares tiendas clave-valor en ningún orden en particular.

Características:

- Almacena pares clave-valor.
- Implementa la Interfaz IDictionary
- Las claves deben ser únicas y no pueden ser nulas.
- Los valores pueden ser nulos o duplicados.
- Los elementos se almacenan como objetos KeyValuePair

Crear Dictionary

Puede crear el Diccionario objeto pasando el tipo de claves y valores a almacenar.

El siguiente ejemplo muestra cómo crear un diccionario y agregar pares clave-valor.

```
Dictionary<string, int> usuarios = new Dictionary<string, int>()
{
  { "John Doe", 42 },
  { "Jane Doe", 38 },
  { "Joe Doe", 12 },
  { "Jenna Doe", 12 }
};
foreach (KeyValuePair<string, int> user in usuarios)
{
    Console.WriteLine(user.Key + " es " + user.Value + " años ");
}
```

Hashtable

Representa una colección de pares clave - valor que se organizan en función del código hash de la clave.

```
0 referencias
static void Main(string[] args)
{
    Dictionary<int, string> dict = new Dictionary<int, string>();
dict.Add(1,"uno");
dict.Add(2,"dos");
dict.Add(3,"tres");
    Hashtable ht = new Hashtable(dict);
```

Crear Hashtable

El siguiente ejemplo demuestra la creación de una tabla hash y la adición de elementos.

```
// Creamos e insertamos datos
Hashtable miDiccio = new Hashtable();

miDiccio.Clear();

miDiccio.Add("byte", "8 bits");
miDiccio.Add("pc", "personal computer");
miDiccio.Add("PC", "ordenador personal");
miDiccio.Add("kilobyte", "1024 bytes");
miDiccio.Add("bit", "");

miDiccio.Remove("PC");

// Mostramos algún dato
Console.WriteLine("Cantidad de palabras en el diccionario: {0}",
miDiccio.Count);
if (miDiccio.ContainsKey("bit"))
{

    Console.WriteLine("El significado de bit es: {0}",
    miDiccio["bit"]);
}
else
{
    Console.WriteLine("No existe esa palabra!");
}

Console.ReadKey();
```

Stack y Queue

Stack y Queue son dos colecciones muy similares entre si ya que solo varia la forma en que guardan y extraen los elementos que contienen. En ciertas cosas estas dos colecciones se parece a un ArrayList

```
{
    Stack stack = new Stack();
    stack.Push(2);
    stack.Push(3);

    Queue queue = new Queue();
    queue.Enqueue(4);
    queue.Enqueue(5);
```

Stack

Es una colección en la que todo nuevo elemento se ingresa al final de la misma, y únicamente es posible extraer el ultimo elemento de la colección. Es conocido como una colección LIFO ya que siempre el ultimo elemento ingresado a la colección, será el primero en salir

```
static void Main(string[] args)
{
    Stack stack = new Stack();
    stack.Push(2);
    stack.Push(3);
    stack.Push(4);
    Console.WriteLine("Elementos en el stack :" + stack.Count);
    Console.WriteLine("Ultimo elemento (Peek) :" + stack.Peek());
    Console.WriteLine("Ultimo elemento (Pop) :" + stack.Pop());
```

Queue

Queue, tiene el comportamiento contrario al Stack. Todo nuevo elemento se agrega al principio de la colección y solo se puede extraer el ultimo elemento. Por esta razón, la cola se conoce como una colección FIFO ya que el primer elemento que ingresa a la cola es el primer elemento que sale

```
Queue queue = new Queue();
queue.Enqueue(4);
queue.Enqueue(5);
queue.Enqueue(6);
Console.WriteLine("Elementos en el queue : " + queue.Count);
Console.WriteLine("Ultimos elementos (Peek) : " + queue.Peek());
Console.WriteLine("Ultimos elementos (Dequeue): " + queue.Dequeue());
Console.WriteLine("Ultimo elementos (Peek)  : " + queue.Peek());
```

Tuple

La clase genérica Tuple de .NET, es una clase que pertenece al nombre de espacios System. Como clase genérica, podemos crear una tupla del tipo de datos que nos parezca oportuno.

```
var subtupla=new Tuple<int,string>(11,"ejemplo1");
var tupla = Tuple.Create<int, string, Tuple<int, string>>(1, "ejemplo1", subtupla);
Console.WriteLine(tupla.Item1);
Console.WriteLine(tupla.Item2);
Console.WriteLine(tupla.Item3.Item1);
Console.WriteLine(tupla.Item3.Item2);
```

En realidad, los constructores de Tuple nos permite crear tuplas de hasta 8 tipos de datos. Aunque nadie nos impide que usemos otro objeto Tuple dentro de otro Tuple

ValueTuple

Es una estructura de datos que tiene un número y una secuencia de elementos específicos. Posee métodos estáticos que nos permite crear tuplas por valor.

```
var user = (1, " Rohan", "Guntur",90.5);
//OR
ValueTuple<int, string, string, double> usuario1 = (1, " Rohan", "Guntur", 90.5);
//OR
(int, string, string, double) usuario2 = (1, " Rohan", "Guntur", 90.5);
```

Thread(Hilos)

son un mecanismo mediante el cual podemos dividir una aplicación en diferentes partes que se pueden ejecutar de forma paralela, existiendo mecanismos por los que pueden compartir información.

Este parámetro se indica mediante la utilización de un delegado, que es el mecanismo que, entre otras cosas, se utiliza en .NET para utilizar punteros a funciones de forma segura.

La creación de cada hilo se realiza mediante las líneas Thread th1= new Thread(new ThreadStar (msg.Mostrar1)); Esto indica que se crea una instancia de la clase Thread, con nombre th1, a partir de un delegado de la clase ThreadStart, que apunta al método Mostrar1 del objeto msn creado anteriormente.

```
public class Dictionary
{
    1 referencia
    public   void mostrar1()
    {
        for(int i=0; i<10; i++)
        {
            Console.WriteLine(" Escrbiendo desde ==>1");
                Thread.Sleep(100);
        }
     1 referencia
    }public void mostrar2()
    {
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine(" Escrbiendo desde ==>2");
            Thread.Sleep(100);
        }
    }
}
0 referencias
public class ejemplo
{
    0 referencias
    public static void Main()
    {
        Dictionary msn = new Dictionary();
        Thread th1 = new Thread(new ThreadStart(msn.mostrar1));
        Thread th2 = new Thread(new ThreadStart(msn.mostrar2));
        th1.Start();
        th2.Start();

        th1.Join();
        th2.Join();
    }
}
```

Una vez creados los dos hilos hay que activarlos, para lo que se llama al método Start de cada uno de ellos. Tras este punto cada hilo se ejecuta en paralelo entre si, y con el programa principal, por lo que utilizamos el método Join de ambos hilos para esperar a que terminen los hilos antes de finalizar el programa.

Leer un archivo de texto

El siguiente código usa la StreamReader clase para abrir, leer y cerrar el archivo de texto. Puede pasar la ruta de acceso de un archivo de texto al StreamReader constructor para abrir el archivo automáticamente.

El ReadLine es un método que lee cada línea de texto e incrementa el puntero de archivo a la siguiente línea al leer. Cuando el RedLine método alcanza el final del archivo, devuelve una referencia nula.

```
static void Main(string[] args)
{
    string line;
    try
    {
        //pasar el archivo y el nombre del archivo al constructor streamreader
        StreamReader sr = new StreamReader("c:\\Ejemplo.txt");
        // leer la primera línea de texto
        line = sr.ReadLine();
        //continúe leyendo hasta llegar al final del archivo
        while (line != null)
        {
            // escribe la ventana de la consola de mentiras
            Console.WriteLine(line);
            //lee la siguiente línea
            line = sr.ReadLine();
        }
        //cerrar el archivo
        sr.Close();
        Console.ReadLine();
    }
    catch(Exception e)
    {
        Console.WriteLine("Ejecutando: " + e.Message);
    }
    finally
    {
        Console.WriteLine("ejecutando finalmente bloquear: ");
    }
```

Escribir un archivo de texto

El siguiente código usa la StreamWriter clase para abrir, escribir y cerrar el archivo de texto. De forma similar a la StreamReader clase, puede pasar la ruta de acceso de un archivo de texto al StreamWriter constructor para abrir el archivo automáticamente.

```
static void Main(string[] args)
{
    try
    {
        //pasar la ruta del archivo y el nombre del archivo en el constructor de Streamwriter
        StreamWriter sw = new StreamWriter("c:\\Text.txt");
        // escribir una línea de texto
        sw.WriteLine("Hola mundo!!");
        //escribe una segunda línea de texto
        sw.WriteLine(" de la clase Streamwriter");
        //cerrar las filas
        sw.Close();
    }catch(Exception e)
    {
        Console.WriteLine("Exception: " + e.Message);
    }
    finally
    {
        Console.WriteLine("ejecutando finalmente bloquear");
    }
```

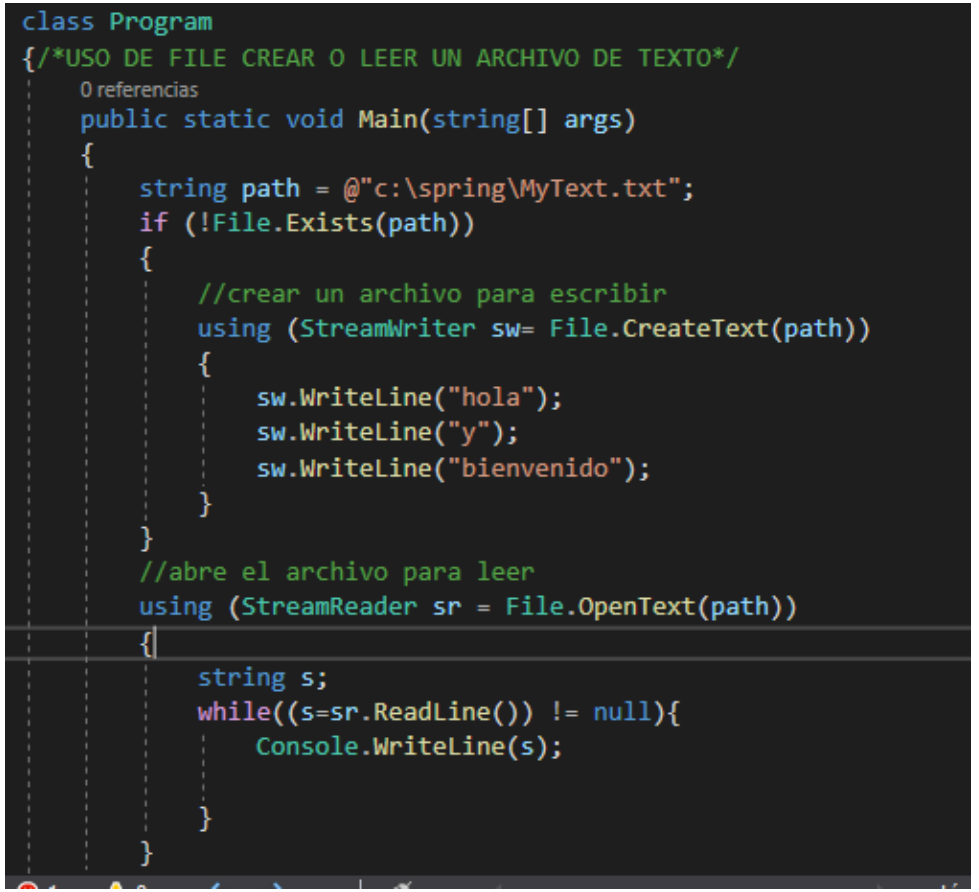
File

Proporciona métodos estáticos para la creación, copia, eliminación, movimiento y apertura de un solo archivo, y ayuda en la creación de objetos FileStream .

```
public static class File
```

File

El siguiente ejemplo demuestra cómo usar la clase File para verificar si un archivo existe y, según el resultado, cree un nuevo archivo y escriba en él, o abra el archivo existente y lea de él.



```
class Program
{/*USO DE FILE CREAR O LEER UN ARCHIVO DE TEXTO*/
    0 referencias
    public static void Main(string[] args)
    {
        string path = @"c:\spring\MyText.txt";
        if (!File.Exists(path))
        {
            //crear un archivo para escribir
            using (StreamWriter sw= File.CreateText(path))
            {
                sw.WriteLine("hola");
                sw.WriteLine("y");
                sw.WriteLine("bienvenido");
            }
        }
        //abre el archivo para leer
        using (StreamReader sr = File.OpenText(path))
        {
            string s;
            while((s=sr.ReadLine()) != null){
                Console.WriteLine(s);

            }
        }
```

FileInfo

Proporciona propiedades y métodos de instancia para la creación, copia, eliminación, movimiento y apertura de archivos, y ayuda en la creación de objetos FileStream, esta clase no puede heredarse

```
public sealed class FileInfo : System.IO.FileSystemInfo
{
```


FileInfo

Cuando se recuperan las propiedades por primera vez, FileInfo llama al método Refresh y almacena en caché la información sobre el archivo. En llamadas posteriores, debe llamar a Refresh para obtener la última copia de la información.

```
public static void Main()
{
    string path = Path.GetTempFileName();
    var fil = new FileInfo(path);

    //crear un archivo para escribir
    using (StreamWriter sw = fil.CreateText())
    {
        sw.WriteLine("hola");
        sw.WriteLine("y");
        sw.WriteLine("bienvenido");
    }

    //abre el archivo para leer
    using (StreamReader sr = File.OpenText(path))
    {
        string s;
        while((s=sr.ReadLine()) != null){
            Console.WriteLine(s);

        }
}
    try
    {
        string path2 = Path.GetTempFileName();
        var fi2 = new FileInfo(path2);

        //asegurarse de que el objetivo no exista
        fi2.Delete();

        //COPIAR EL ARCHIVO
        fil.CopyTo(path2);
        Console.WriteLine($"{path} fue copiado a {path2} :");

        //eliminar el archivo recién creado
        fi2.Delete();
        Console.WriteLine($"{path2} Fue borrado con éxito :");

    }catch(Exception e)
    {
        Console.WriteLine($"el procedimiento fallo:  {e.ToString() }");
    }
     }
```

PREPÁRATE
PARA SER EL
MEJOR

+ ENTREMIENTO
EXPERIENCIA

BIENVENIDOS.

GRACIAS
POR TU PARTICIPACIÓN

Por favor, bríndanos tus comentarios
y sugerencias para mejorar nuestros servicios.

GALAXY
TRAINING


GALAXY
TRAINING