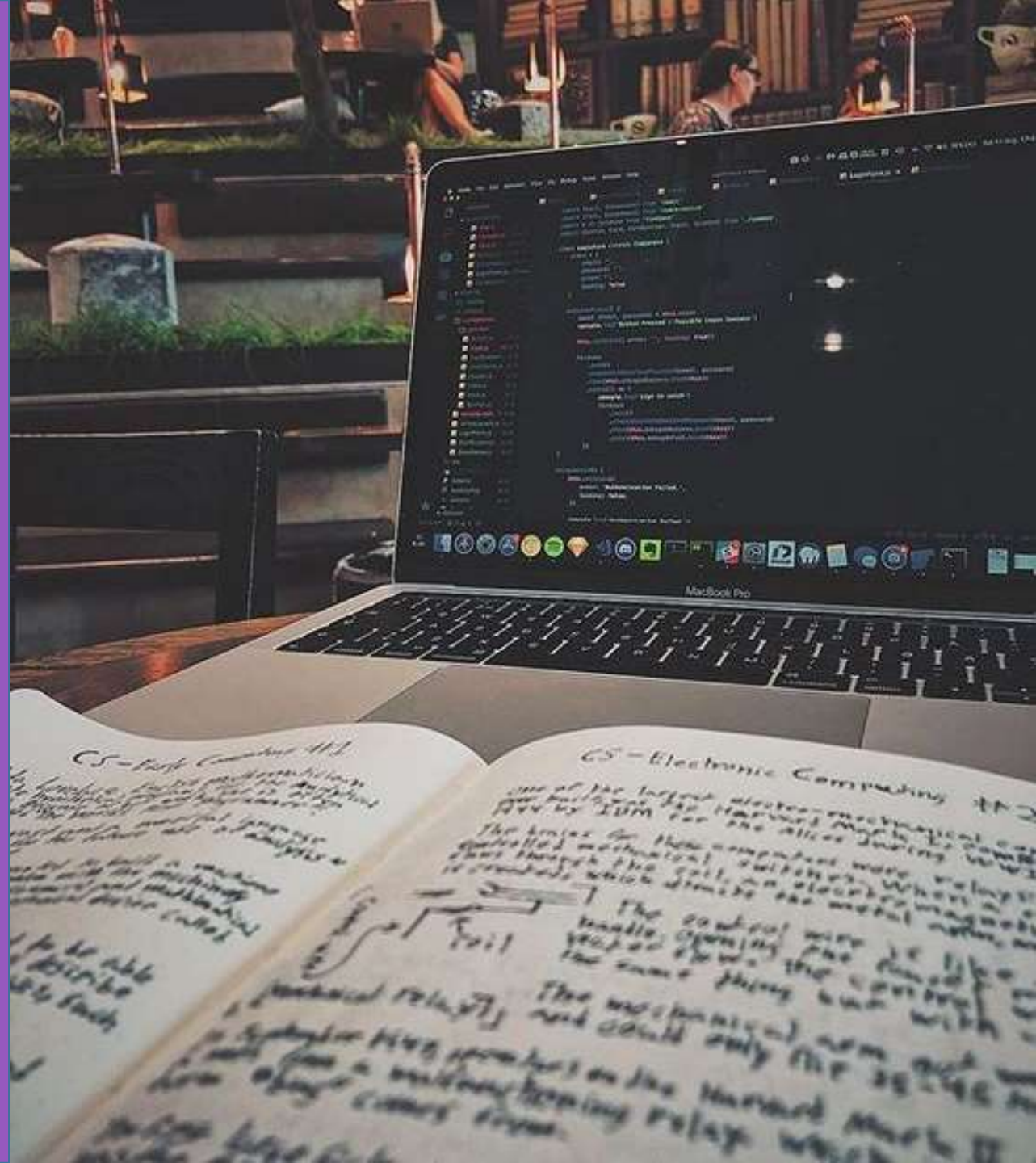
Clase Nro 8

- Archivos, directorios
- Extensiones
- Manejo de Archivos en .net core
- Librería System.IO
- Objeto directory
- Objeto File
- Helper de archivos



Directorios

En informática, un directorio es un **contenedor virtual** en el que se almacenan una agrupación de archivos informáticos y otros subdirectorios, atendiendo a su contenido, a su propósito o a cualquier criterio que decida el usuario.

Técnicamente, el directorio almacena información acerca de los archivos que contiene: como los atributos de los archivos o dónde se encuentran físicamente en el dispositivo de almacenamiento.

https://es.wikipedia.org/wiki/Directorio

Archivos

En términos computacionales es una colección de datos que tiene un nombre y se guardan en dispositivos de almacenamiento secundario: magnéticos, ópticos, electrónicos, etc.

Archivos

Tipos de Archivos

Archivos de texto plano

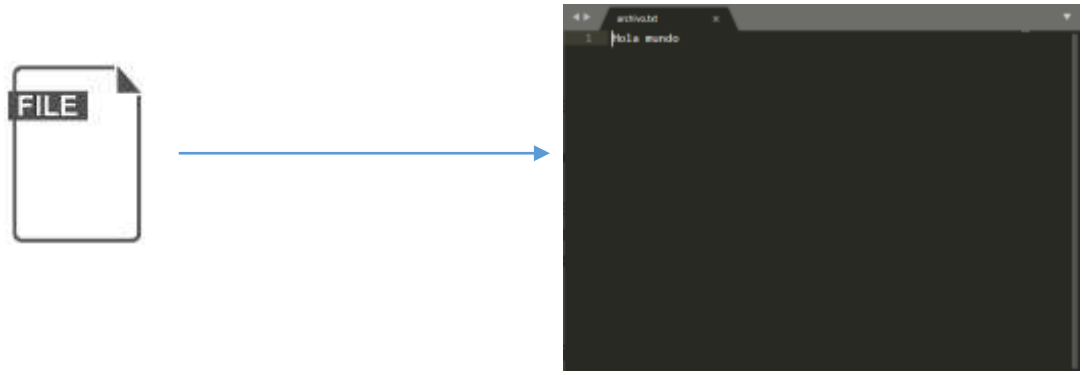
Un archivo de texto simple, texto sencillo o texto sin formato (también llamado **texto llano/plano o texto simple**; en inglés «plain text»), es un archivo informático que contiene únicamente texto formado solo por caracteres que son legibles por humanos, careciendo de cualquier tipo de formato tipográfico.1

Los documentos de texto llano son legibles por humanos -a diferencia de los archivos binarios- ,

Estos archivos están compuestos de bytes que representan caracteres ordinarios como letras, números y signos de puntuación (incluyendo espacios en blanco), también incluye algunos pocos caracteres de control como tabulaciones, saltos de línea y retornos de carro.

Extensión típica de un archivo de texto plano: .txt

Atener en cuenta en los archivos: Codificación de caracteres del archivo



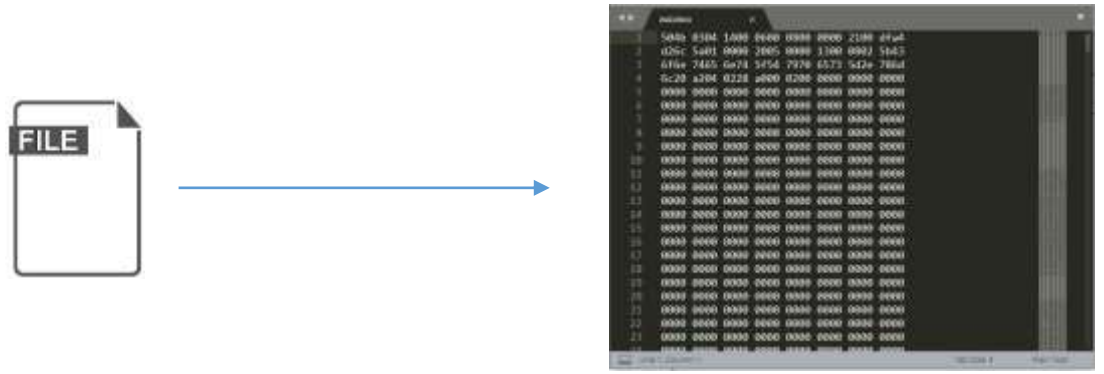
Archivos

Tipos de Archivos

Archivos binarios

Un archivo binario es un archivo informático que contiene información de cualquier tipo codificada en binario para el propósito de almacenamiento y procesamiento en ordenadores.

Ejemplo de archivo binario: los archivos informáticos que almacenan texto formateado o fotografías, así como los archivos ejecutables que contienen programas.



https://es.wikipedia.org/wiki/Archivo_binario#:~:text=Un%20archivo%20binario%20es%20un,archivos%20ejecutables%20que%20contienen%20programas.

Archivos

Formatos de Archivos

Un formato de archivo es un **estándar** que define la forma en que la información se organiza y se codifica en un **archivo informático**.

Los formatos de archivos pueden ser:

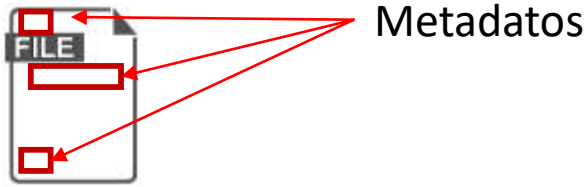
- Formatos cerrado
- Formatos abiertos

Extensión: Hace referencia a la convención que se utilizó para estructurar el archivo.

Metadatos: Estructura de datos que permite interpretar el archivo.

<Nombre_del_archivo>.jpg

Extensión



Archivos

Formatos de Archivos

Meta datos

Una segunda forma de identificar un formato de archivo es usar información relacionada con el formato almacenado dentro del propio archivo, ya sea información destinada a este fin o cadenas binarias que siempre están en ubicaciones específicas en archivos de algunos formatos.

Dado que el lugar más fácil para ubicarlos es al principio, tal área generalmente se denomina *encabezado de archivo* cuando es mayor que unos pocos bytes , o un magic number si solo tiene unos pocos bytes de longitud.

Archivos

Formatos de Archivos

Header

Se refiere a la información suplementaria situada al principio de un bloque de información que va a ser almacenada o transmitida y que contiene información necesaria para el correcto tratamiento del bloque de información. Este bloque puede tb estar situado al final.
Información que puede incluir una cabecera:

- Formato , Dimensiones, Tamaño, Fechas importantes, tipo de compresión utilizada, Duración, etc.

Firma hexadecimal

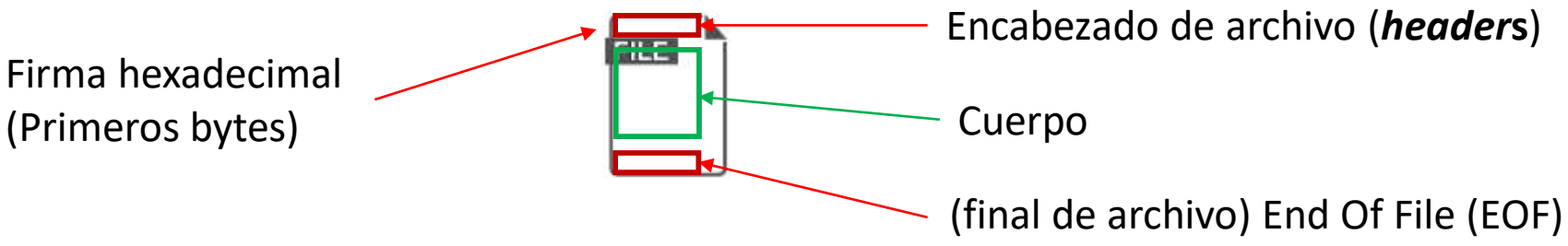
Suele ser un *Magic Number* situado al comienzo de la cabecera de una archivo.

Cuerpo

Los datos que siguen a la cabecera.

Final de archivo (End o file o EOF)

Algunos archivos destinan un “par” de bytes para indicar que se encontró el final de dicho archivo.



Archivos

Formatos de Archivos

Firma hexadecimales de algunos archivos populares

ipo de Archivo	Cabecera	En ASCII
.ZIP	50 4B 03 04	PK
.RAR	52 61 72 21	Rar!
.TAR	1F 8B 08 00	
.TGZ	1F 9D 90 70	
.DOC	D0 CF 11 E0	ÐÏ.à
.XLS	D0 CF 11 E0	
.PDF	25 50 44 46	%PDF
.WMV	30 26 B2 75	
.FLV	46 4C 56 01	FLV
.BMP	42 4D F8 A9/ 62 25 / 76 03	BM, BMp% , BMv
.GIF	47 49 46 38 39 61 / 37 61	GIF89a GIF87a
.ICO	00 00 01 00	
.JPEG	FF D8 FF E0 / FE	JFIF
.PNG	89 50 4E 47	PNG
.SFW	43 57 53 06 / 08	Cws
.MP3	49 44 33 2E /03	ID3
.EXE	4D 5A 50 00 /90 00	MZP / MZ
.DLL	4D 5A 90 00	MZ
Linux bin	7F 45 4C 46	ELF

https://www.welivesecurity.com/la-es/2015/10/01/extension-de-un-archivo-cabeceras/

Formatos de Archivos

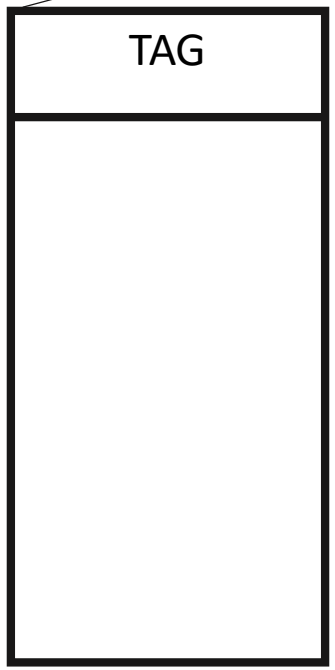
Cabecera del BMP

Primeros 56 bytes

Bytes	Información
0, 1	Tipo de fichero "BM, BMp% , BMv"
2, 3, 4, 5	Tamaño del archivo
6, 7	Reservado
8, 9	Reservado
10, 11, 12, 13	Inicio de los datos de la imagen
14, 15, 16, 17	Tamaño de la cabecera del bitmap
18, 19, 20, 21	Anchura (píxels)
22, 23, 24, 25	Altura (píxels)
26, 27	Número de planos
28, 29	Tamaño de cada punto
30, 31, 32, 33	Compresión (0=no comprimido)
34, 35, 36, 37	Tamaño de la imagen
38, 39, 40, 41	Resolución horizontal
42, 43, 44, 45	Resolución vertical
46, 47, 48, 49	Tamaño de la tabla de color
50, 51, 52, 53	Contador de colores importantes

Formatos de Archivos

Cola del MP3: Id3Tag



•Una cabecera que identifica la presencia del bloque ID3 y su versión. En concreto, dicha cabecera consta de los caracteres **TAG**.

•Título: 30 caracteres.

•Artista: 30 caracteres.

•Álbum: 30 caracteres.

•Año: 4 caracteres.

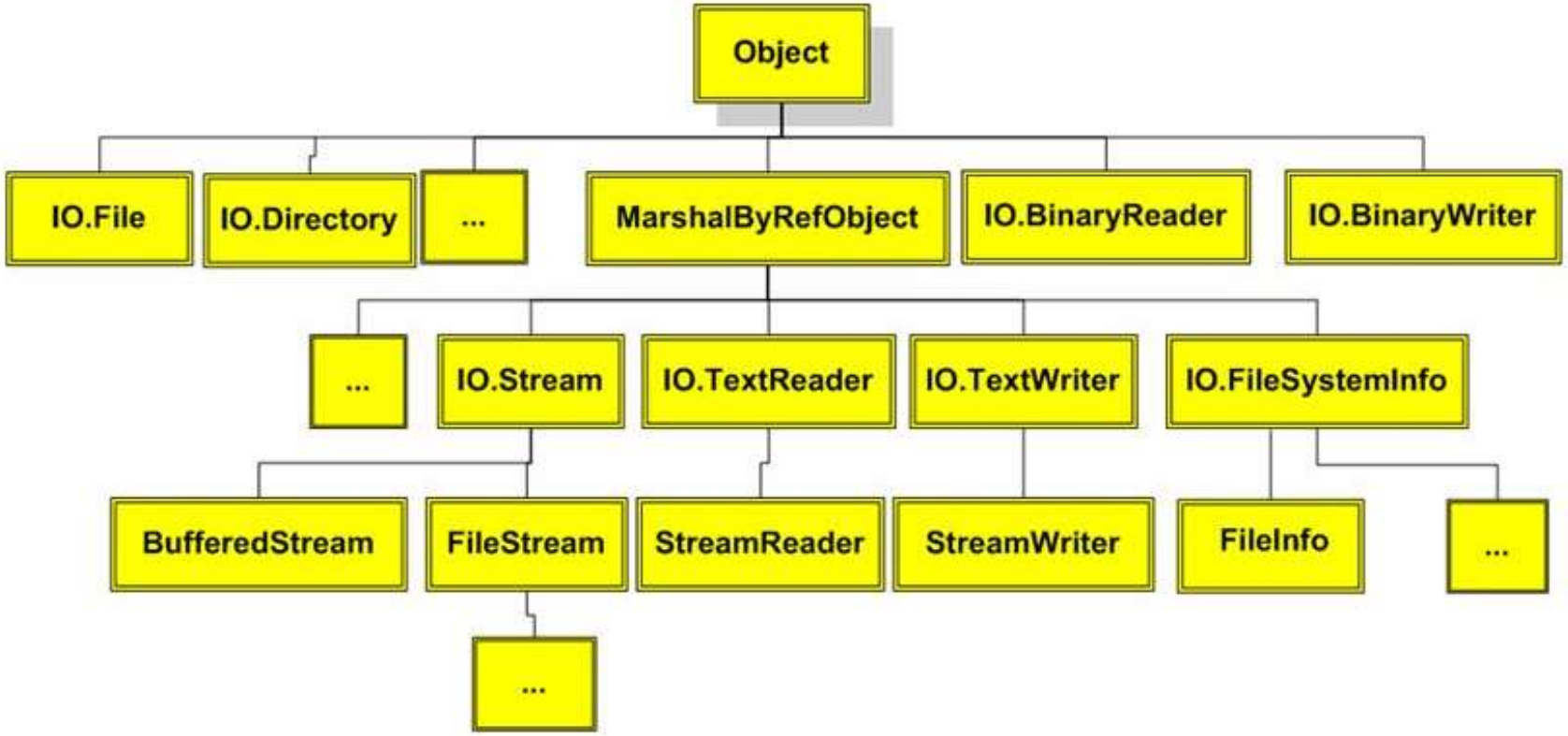
•Un comentario: 30 caracteres.

•Género (musical): un carácter.

Trabajando con Archivos y directorios en Net core

La E/S (entrada/salida) de archivos y secuencias hace referencia a la transferencia de datos con destino u origen en un medio de almacenamiento. En .NET Core, los espacios de nombres System.IO contienen tipos que permiten la lectura y escritura, tanto sincrónica como asincrónica, en archivos y flujos de datos.

Librería System.IO



https://docs.microsoft.com/en-us/dotnet/api/system.io?view=netframework-4.8

Librería System.IO

Estas son algunas clases para trabajar con archivo y directorio:

•File: proporciona métodos estáticos para crear, copiar, eliminar, mover y abrir archivos, y ayuda a crear un objeto FileStream.

•FileInfo: proporciona métodos de instancia para crear, copiar, eliminar, mover y abrir archivos, y ayuda a crear un objeto FileStream.

•Directory: proporciona métodos estáticos para crear, mover y enumerar directorios y subdirectorios.

•DirectoryInfo: proporciona métodos de instancia para crear, mover y enumerar directorios y subdirectorios.

•Path: proporciona métodos y propiedades para procesar cadenas de directorio entre plataformas.

Objeto Directory

System.IO.Directory

GetLogicalDrives	Obtiene los discos lógicos de un sistema
GetDirectories	Obtiene todos los directorios dentro de un directorio
GetDirectoryRoot	Obtiene la ruta del archivo
GetFiles	Todos los archivos dentro un directorio
Exists	Devuelve true / false dependiendo si el directorio existe

Objeto File

System.IO.File

File.Exists	Método utilizado para chequear la existencia de una archivo.
File.ReadAlllines	Método utilizado para leer todas las lineas y dejarlas en un Arreglo de string.
File.ReadAllText	Método utilizado para leer todas las lineas de un arhivo y devuelve un string
File.Copy	Método para copiar archivos de una ubicación a otra.
File.Delete	Método para eliminar Archivos.

Ejercicio

- Comprobar si un directorio “c:\PruebaDeDirectorio” existe, en caso de no existir, cree uno nuevo
- Crear un archivo nuevo en caso de no existir dentro del directorio
- ¿Que es un Archivo CSV? ¿Cómo está estructurado?
- Trabajar con archivo CSV
- ¿Como lo harías? PENSAR SOLUCIONES, no programarla
- Lea un archivo CSV, donde figure *nombre, apellido y edad de una persona*

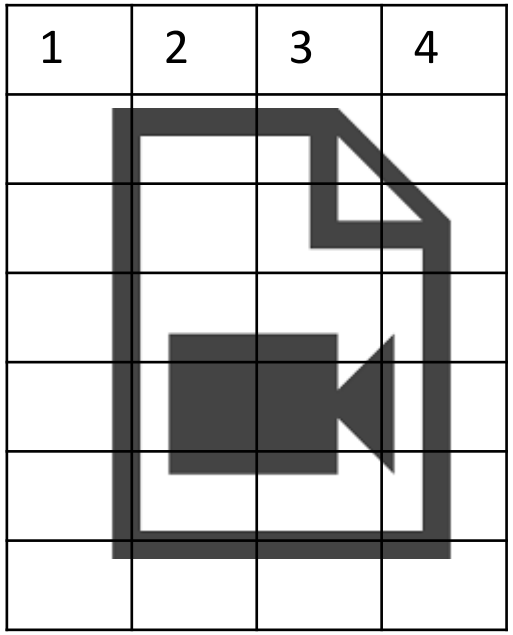
Stream

También se conoce como como **flujo de datos**.

La idea detrás del stream existe desde hace mucho tiempo. Es utilizada cuando un archivo tiene un gran tamaño y se necesita cargar en memoria para poder utilizarlo o los datos que se desea transferir de un extremo a otro son mayores al *ancho de banda/memoria/recursos* disponibles.

En estos casos la transferencia de una fuente a un receptor se realiza fragmentando un archivo por partes y enviado esas partes de un extremo a otro de forma secuencial.

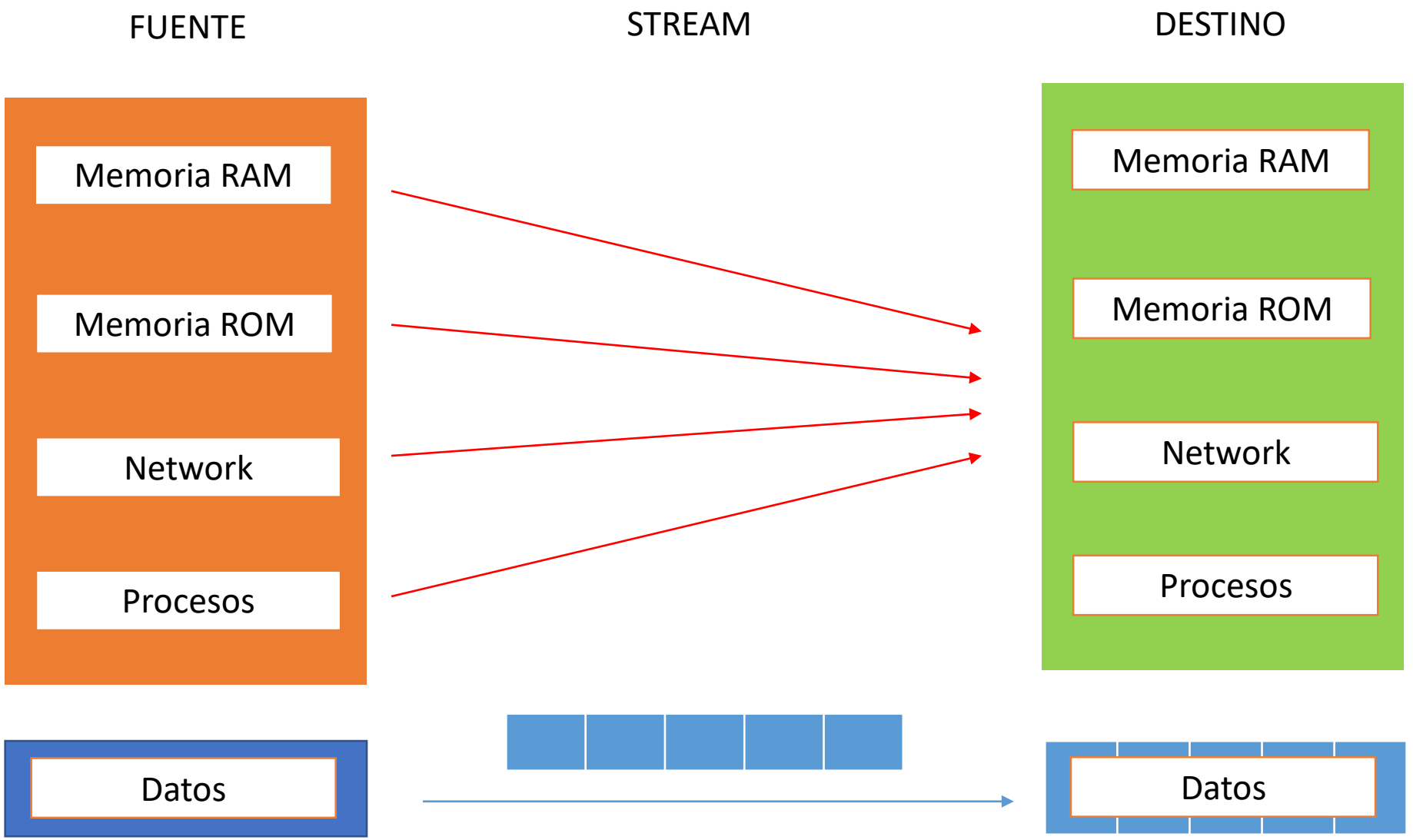
Fuente: Video.MP4



Destino: Fracciones de Video.MP4

1	2	3	4

Stream


FUENTE
STREAM
DESTINO
Memoria RAM
Memoria ROM
Network
Procesos
Memoria RAM
Memoria ROM
Network
Procesos
Datos
Datos

Stream

Tipos de Flujos de datos: texto o bytes

Flujo de texto: secuencia de caracteres (datos de texto). Según el SO, cada línea puede terminar con un carácter de nueva línea (normalmente la secuencia “\r\n”). Se usan para datos textuales.

Flujo binario: secuencia de bytes (el contenido de un programa ejecutable). Se usan para datos no textuales, en donde el contenido exacto de los datos se mantiene sin importar la apariencia

Un **stream** puede transferir datos pueden ser en dos posibles direcciones:

- Si los datos son transferidos desde una fuente externa al programa, entonces se habla de “leer desde el stream”.
- Si los datos son transferidos desde el programa a alguna fuente externa, entonces se habla de “escribir al stream”.

Stream

Resumiendo

- Los streams se utilizan normalmente cuando se leen datos de archivos grandes o fragmentos de archivos.
- Al utilizar streams, los datos de archivos grandes se dividen en pequeños fragmentos y se envían al flujo.
- Estos fragmentos de datos se pueden leer desde la aplicación.
- La ventaja de dividirlo en partes pequeñas se debe al impacto en el rendimiento de leer un archivo grande de una sola vez.
- Si tuviera que leer los datos de, por ejemplo, un archivo de 1000 MB a la vez, la aplicación podría colgarse y volverse inestable.
- El mejor enfoque es utilizar las secuencias para dividir el archivo en partes manejables.

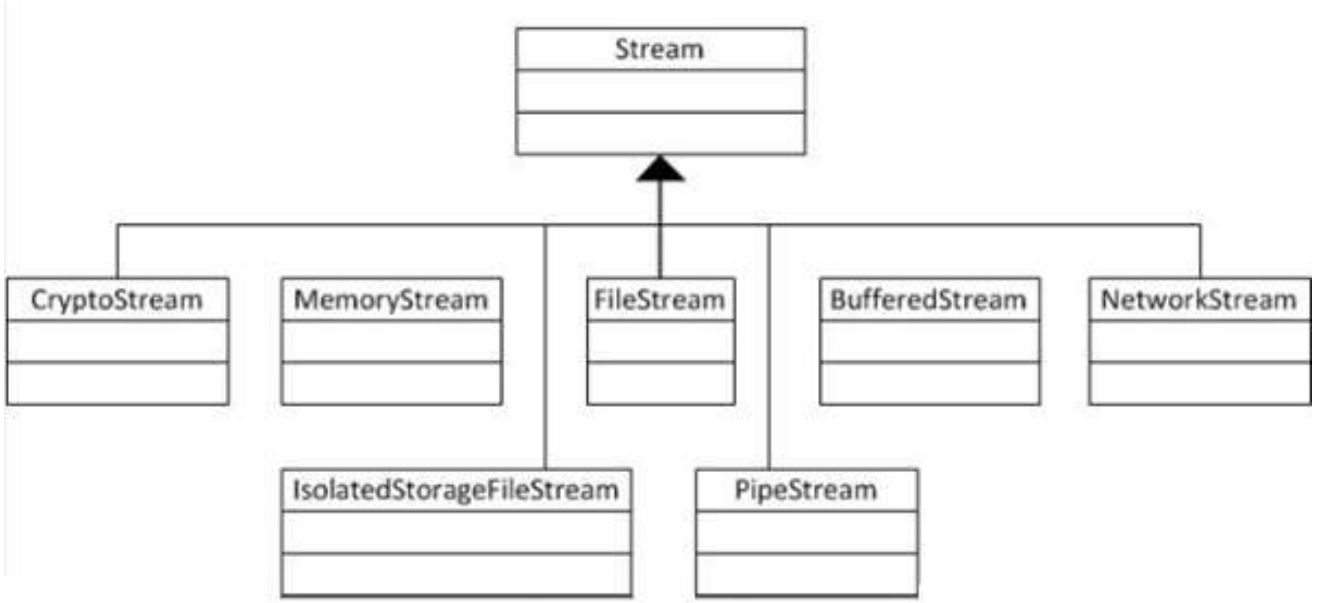
Usar Stream en .net core

Para poder leer un archivo, se asocia este a un Flujo (llamado secuencia) que es el elemento que permite leer los datos del archivo. Esta es una secuencia de bytes que se transfiere desde el programa al archivo o viceversa.

En el ambiente .NET se puede encontrar muchas clases que representan este concepto y que trabaja con archivos o con datos de memoria (como se muestra en la figura de abajo).

Clase Stream

Stream: *System.IO.Stream* es una clase abstracta que proporciona métodos estándar para transferir bytes (lectura, escritura, etc.) a la fuente.



https://docs.microsoft.com/en-us/dotnet/api/system.io.stream?view=netframework-4.8

Clase Stream

- La clase base abstracta Stream admite lectura y escritura de bytes.

- Todas las clases que representan flujos heredan de la clase Stream.

- La clase Stream y sus clases derivadas proporcionan una vista común de las fuentes de datos y los
  repositorios, y aíslan al programador de los detalles específicos del sistema operativo y los dispositivos subyacentes.

  Las transmisiones involucran tres operaciones fundamentales:

  - **Lectura**: transferencia de datos de un flujo a una estructura de datos, como una matriz de bytes.
  - **Escritura**: transferencia de datos a un flujo desde un origen de datos.
  - **Búsqueda**: consulta y modificación de la posición actual dentro de un flujo.

  Nota: dependiendo de la clase que se utilice, sea admiten todas o sólo alguna de las operaciones anteriores.

Clase Stream

Dependiendo de la **fuente de stream** de donde consumamos datos:

FileStream lee o escribe bytes desde / a un archivo físico, ya sea un .txt, .exe, .jpg o cualquier otro archivo.

MemoryStream: MemoryStream lee o escribe bytes que se almacenan en la memoria.

BufferedStream: BufferedStream Esta clase se utiliza para leer y para escribir a otro stream.
El uso de streams para la lectura y escritura de archivo es directa pero lenta.
Por esta razón la clase BufferedStream existe y es más eficiente.
Para operaciones de archivo es posible utilizar FileStream, donde el buffering está ya incluido.

NetworkStream: NetworkStream lee o escribe bytes desde un socket de red. (conjunto IP + Puerto)

PipeStream: PipeStream lee o escribe bytes de diferentes procesos.

CryptoStream: CryptoStream es para vincular flujos de datos a transformaciones criptográficas.

Clase FileStream

BufferedStream

Las clases más relacionadas con la escritura y lectura de archivos (File Input/Output o File I/O) son:

- **FileStream**, cuyo propósito es lectura y escritura de datos binarios (no de texto legible), a cualquier archivo de tipo binario, aunque se puede utilizar para acceder a cualquier tipo de archivo, inclusive los de texto.

- **StreamReader y StreamWriter**, las cuales están diseñadas para lectura y escritura de archivos de texto. Estas clases se asumen como de un nivel más alto que FileStream

Clase FileStream

CanRead	Obtiene un valor que indica si una secuencia admite operaciones de lectura.
CanSeek	Obtiene un valor que indica si una secuencia admite búsquedas.
CanTimeout	Obtiene un valor que determina si se puede agotar el tiempo de espera de la secuencia actual. (Inherited from Stream)
CanWrite	Obtiene un valor que indica si una secuencia admite operaciones de escritura.
Capacity	Obtiene la longitud de la secuencia (tamaño) o la cantidad total de memoria asignada a una secuencia (capacidad).
Length	Obtiene la longitud de los datos de una secuencia.
Position	Obtiene o establece la posición actual en una secuencia.

Clase FileStream

Read	Obtiene una secuencia de bytes si el stream admite operaciones de lectura.
Seek	Establece una posición dentro de una secuencia.

https://docs.microsoft.com/en-us/dotnet/api/system.io.filestream?view=netframework-4.8

Clase FileStream

Ejemplo de uso

Utiliza el **enum** FileMode permite especificar como va a ser el tipo de apertura de archivo que usaremos

```
public enum FileMode
{
        CreateNew,
        Create,
        Open,
        OpenOrCreate,
        Truncate,
        Append
};
```

Contructor, permite generar la instancia de esta clase para trabajar con un stream desde un archivo. Para ello requiere:

```
FileStream MiStream = new FileStream(NombreDelArchivoCopia, FileMode.Open);
```

Clase FileStream

Ejemplo de uso

```
byte[] buffer = new byte[128];
```

Generar la instancia de esta clase nos permitirá trabajar con un Strem desde un archivo.
Con el método Open de la Clase estática File

```
FileStream MiArchivo = new File.Open(NombreDelArchivo, FileMode.Open);
```

También se puede usar el constructor de Filestream

```
FileStream MiArchivo2 = new FileStream(NombreDelArchivoCopia, FileMode.Open);
```

Lee una cantidad de bytes (en el ejemplo, el objeto llamado fileS, lee 128 [bytes]) al hacerlo el puntero Position se mueve hasta el próximo byte. Método: Read(buffer, desde , hasta);

```
MiArchivo.Read(Buffer, 0, 128);
```

Si queremos movernos en cualquier posición dentro de un archivo podemos utilizar

```
MiArchivo.Seek(PosicionDentroDelArchivo, Desde_donde_se_mueve);
```

Cierre de la secuencia y libera todos los recursos tomados por MiArchivo

```
MiArchivo.Close();
```

Clase FileStream

Ejemplo de uso

Transformando bytes texto

Los datos recuperados por FileStream están en formato binario, si queremos utilizarlos como texto debemos aplicar una conversión. Para esto, existe el método **GetString (buffer, inicio, fin)** que permitirá convertir de un arreglo (buffer) de bytes a un string donde podremos tratarlo com texto.

System.Text.Encoding.Default.**GetString**(fuente de bytes, Posición Original, posición final);

Clase Helper para trabajar con Stream

Clases auxiliares para el manejo sencillo de stream .

StreamReader: StreamReader es una clase auxiliar para leer caracteres de un flujo al convertir bytes en caracteres usando un valor codificado. Puede utilizarse para leer cadenas (caracteres) de diferentes transmisiones como FileStream, MemoryStream, etc.

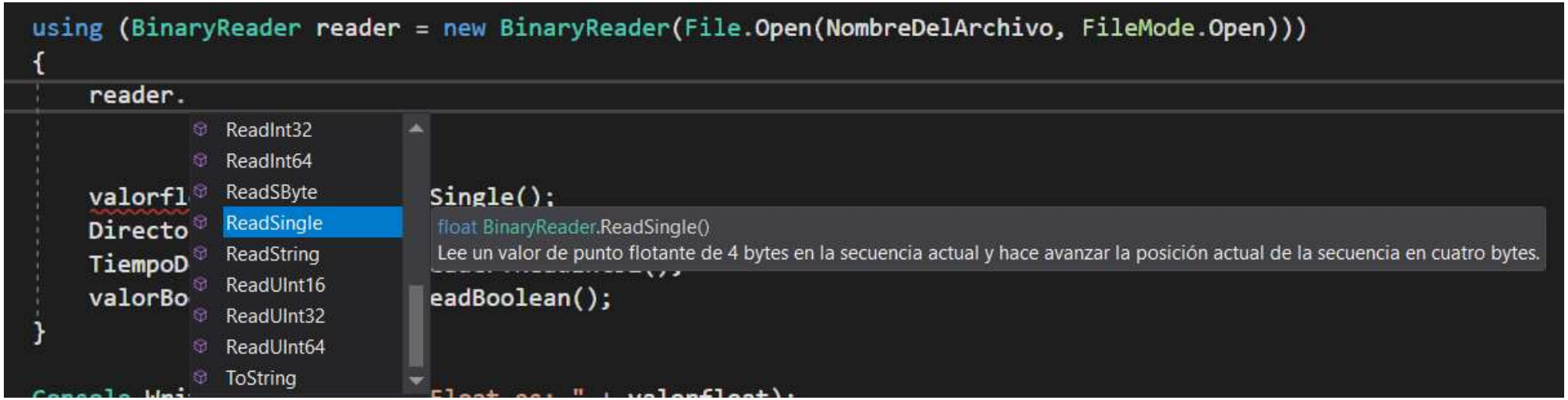
StreamWriter: StreamWriter es una clase auxiliar para escribir una cadena en un Stream mediante la conversión de caracteres en bytes. Se puede usar para escribir cadenas en diferentes transmisiones como FileStream, MemoryStream, etc.

BinaryReader: BinaryReader es una clase auxiliar para leer tipos de datos primitivos de bytes.

BinaryWriter: BinaryWriter escribe tipos primitivos en binario.

Clase Helper para trabajar con Stream

Las BinaryReader y streamReader ofrecen una colección de métodos para leer archivos binarios.
Son muy útiles cuando se conocen exactamente como está organizado un archivo.



Ejercicio

- Utilizando la clase FileStream Leer la etiqueta de un archivo MP3 según las especificaciones del archivo.
- Utilice este procedimiento y busque en una carpeta todos los archivos con extensión mp3, contabilícelos, y muestre los datos asociados a los archivo que se encuentran dentro del directorio

- Opcional: Cambie algún campo del archivo MP3 y guárdelo en el archivo