LETTURA FILE

CORSO DI GAME PROGRAMMING
1° ANNO

Docente **Davide Caio**

ACCADEMIA ITALIANA VIDEOGIOCHI

LETTURA E SCRITTURA DI FILE

Una delle operazioni più comuni che un programma si trova a dover effettuare è la lettura e la scrittura di un file di testo.

Questo permette di poter utilizzare un file di testo come un input (o output) del nostro programma, senza dover passare ad esempio dalla console.

Pensate alla configurazione di un software, sarebbe bello poter definire tutte le opzioni di configurazione in un file e poi leggerlo all'avvio del software, al posto che ogni volta chiedere da console con quali configurazioni vogliamo far partire il software.

CLASSE FILE DI SYSTEM.IO

Questa classe ti permette di creare, aprire, chiudere ed eliminare un file. Quando si parla di file di testo la classe file è ancora più utile perché ti permette di leggerne il contenuto e scriverne il contenuto in modo molto veloce.

URL DEL FILE

Per poter manipolare un file è necessario sapere il suo percorso. Attenzione che la classe string ha alcuni caratteri speciali tra cui il carattere '\'. Quando bisogna inserire l'url del file locale quindi bisogna stare attenti perché i percorsi del file system sono separati dal carattere '\'.

Doppio carattere speciale "\\MyAssets\\MyText.txt"
Oppure trattiamo la stringa come un literal puro: @"\MyAssets\MyText.txt"
→ this

LETTURA E SCRITTURA ONE SHOT

Se vogliamo effettuare una **lettura** di tutto il file in una volta sola, in modo sincrono, allora la classe File ci mette a disposizione il metodo:

- string[] ReadAllLines (string path) → Apre il file, lo legge tutte le righe, le ritorna come una array di stringhe e poi lo richiude.
- string[] ReadLines (string path) → Legge tutte le righe di un file e le ritorna come una collezione.

La differenza principale è che puoi iniziare a enumerare la collezione in ReadLines prima che sia ritornata in toto. Mentre ReadAllLines ritorna un array, quindi prima devi attendere che sia ritornato. Se il file da leggere è molto grande allora ReadLines è più efficiente.

Se invece vogliamo **scrivere** un array (o una IEnumerator) di stringhe allora possiamo utilizzare il metodo:

- void WriteAllLines (string path, IEnumerable<string> contents) → Crea un nuovo file (quindi quello vecchio sarà sovrascritto se già esiste) scrive tutte le stringhe nella IEnumerable e poi chiude il file. Esiste anche la versione che accetta un array.

ESERCIZIO 1

Leggere il testo di un file che si trova in un percorso salvato in una variabile del main, stampa a console il suo contenuto e sovrascrivilo aggiungendo una linea con scritto “Ciaone, ho aggiunto una linea :)”.

ESERCIZIO 2

Creare una classe Persona con Nome, Cognome, Età (intero).
Creare un file di testo formattato in questo modo:
Davide,Caio,30
Mario,Rossi,55
Carlo,Verdi,70

Creare una lista di Persone leggendo i dati nel file di testo, aggiungere alla lista di Persone una nuova persona Pippo,Franco,80 e aggiungerla al file.

PROBLEMA

I metodi che abbiamo visto adesso **leggono o scrivono tutto il file** in una volta e in modo sincrono (il programma non continua l'esecuzione fino a quando non ha finito di leggere o scrivere in toto il file).

Questo può portare a un enorme rallentamento dell'esecuzione, quando magari quello che si vuole fare è solamente leggere le prime N righe, oppure quando si vuole leggere una riga alla volta, fare qualcosa e poi continuare la lettura.

Quello che vorremmo fare noi è avere uno stream di un file. (flusso di byte, o testo).

SOLUZIONE PER FILE DI TESTO IN SOLA LETTURA

è possibile utilizzare il metodo

- File.OpenText (string path) → ritorna uno StreamReader che è una classe che legge caratteri da uno streaming di byte IN FORMATO UTF-8.

```
static void Main (string[] args) {
        string path = @"MyAssets\MyText.txt";

        // Open the stream and read it back.
        StreamReader sr = File.OpenText (path);
        string s = "";
        while ((s = sr.ReadLine ()) != null) {
        Console.WriteLine (s);
        }
        Console.ReadLine ();
}
```

ESERCIZIO 3

Rifare l’esempio 2 senza utilizzare File.ReadAllLines ma utilizzando lo StreamReader.

FILESTREAM CLASS

La classe FileStream di System.IO gestisce uno stream di un file (generico non per forza di testo), supportando sia lettura e scrittura sincrona che asincrona.
Come si ottiene un FileStream?

- File.Create (string path) → crea o sovrascrive un file al path specificato e ritorna un FileStream con accesso in lettura/scrittura per poterlo elaborare.
- File.Open (string path, FileMode mode, FileAccess access) → fornisce un file stream del file al path desiderato, aprendolo nella modalità mode (solitamente *FileMode.OpenOrCreate*) e con gli accessi access (*Read, Write, Read/Write*).

PROBLEMA
Qui noi andiamo ad aprire uno stream di byte, ma noi vogliamo leggere la stringa che rappresenta il testo del file.

CONVERTIRE ARRAY DI BYTE

Ripasso endianness

- big-endian : prima byte più significativi
- little-endian: prima byte meno significativi.

La classe BitConverter ti permette di convertire un qualsiasi tipo predefinito in un array di byte e viceversa.

BitConverter.IsLittleEndian ritorna se l'architettura del sistema immagazzina i dati in little endian. Nel caso bisogna ricordarsi che l'array di byte deve essere invertito in lettura, e anche in scrittura.

- int → byte[] : GetBytes(int) byte[] → int : ToInt32
- bool → byte[] : GetBytes(bool) byte[] → bool : ToBoolean
- char → byte[] : GetBytes (char) byte[] → char : ToChar

E per le stringhe? ToString ()?

ESERCIZIO 4

Effettuare delle prove di conversione da tipo predefinito a array di byte e viceversa.

Testare cosa succede quando utilizzi il ToString () sulla classe BitConverter.

CONVERTIRE ARRAY BYTE - STRINGHE

Per le stringhe la conversione diventa un po' più complessa in quanto esistono molti tipi di Encoding.
In C# abbiamo a disposizione la classe astratta Encoding da dove derivano tutte le classi che rappresentano una determinata codifica del testo.

- ASCIIEncoding → decodifica caratteri Unicode come un 7-bit ASCII carattere singolo
- UTF7Encoding → decodifica caratteri Unicode usando la codifica UTF7
- UTF8Encoding → decodifica caratteri Unicode usando la codifica UTF8
- UnicodeEncoding → decodifica caratteri Unicode usando la codifica UTF-16
- UTF32Encoding → decodifica caratteri Unicode usando la codifica UTF-32

CONVERTIRE ARRAY BYTE - STRINGHE

In ognuna di quelle classi troverete:

- GetBytes (string) → ritorna un array di bytes che rappresenta la stringa in una determinata codifica.
- GetString (byte[]) → ritorna una stringa decodificando l'array di byte in quella determinata codifica

FILESTREAM CLASS READ/WRITE

Ora possiamo leggere e scrivere degli stream di byte.

- int Read (byte[] array, int offset, int count);
  - array: array che conterrà i byte letti dallo stream tra gli indici offset e offset+count - 1, mentre gli altri saranno rimasti gli stessi di prima
  - offset : l'indice dell'array di byte da cui inizierà la sovrascrittura dei byte da quelli letti dalla sorgente.
  - count: il numero massimo di byte da leggere
  - ritorna il numero effettivo di byte letti

- Write (byte[] array, int offset, int count)
  - array: il buffer contenente i dati da scrivere
  - offset: l'indice dell'array di byte dal quale iniziare a copiare i bytes dello stream.
  - count: il numero massimo di byte da scrivere

ESERCIZIO 5

Scrivere l’esempio 2 utilizzando File.Open e decodificando l’array di byte nello stream.

DOCUMENTAZIONE MICROSOFT

La gestione di file è molto complessa e la classe File mette a disposizione moltissimi tipo di metodi per ogni evenienza. Anche le classi FileStream e Encoding mettono a disposizione tantissimi altri metodi.

è consigliata una profonda lettura della documentazione ms di queste classi e quelle collegate:

- File https://docs.microsoft.com/en-us/dotnet/api/system.io.file?view=netframework-4.8
- FileStream https://docs.microsoft.com/en-us/dotnet/api/system.io.filestream?view=netframework-4.8
- Encoding https://docs.microsoft.com/en-us/dotnet/api/system.text.encoding?view=netframework-4.8