Uso dei file attraverso lo Stream (flusso)

Uno Stream rappresenta un flusso sequenziale di dati sia in lettura che in scrittura.
Un flusso può riguardare

1. un file
2. la memoria
3. la rete
4. PipeStream
5. Isolated Storage

Spiegazione	
1. un file	per leggere e scrivere un file
2. la memoria	per leggere e scrivere da e verso la memoria
3. la rete	per la trasmissione o ricezione di dati dalla rete (vedi per es. la lettura di una pagina web)
4. PipeStream	per leggere e scrivere dati attraverso un canale di comunicazione tra processi
5. Isolated Storage	le applicazioni .NET può sfruttare una propria area riservata di memorizzazione dei dati in modo che le altre applicazioni non possano averne accesso

In queste lezioni, ci occuperemo solamente della gestione di file.

Le classi che ci interessano in questo contesto sono:

1. **StreamReader** – che legge un file di testo o da altro flusso
2. **StreamWriter** – che scrive in un file di testo o in altro flusso

La classe **StreamReader** ha 11 costruttori:

StreamReader(Stream)	Inizializza una nuova istanza della classe StreamReader per il flusso specificato.
StreamReader(Stream, Boolean)	Inizializza una nuova istanza della classe StreamReader per il flusso specificato, con l'opzione specificata per il rilevamento dei byte order mark (*BOM).
StreamReader(Stream, Encoding)	Inizializza una nuova istanza della classe StreamReader per il flusso specificato, con la codifica dei caratteri specificata.
StreamReader(Stream, Encoding, Boolean)	Inizializza una nuova istanza della classe StreamReader per il flusso specificato, con la codifica dei caratteri e l'opzione per il rilevamento dei byte order mark specificati.
StreamReader(Stream, Encoding, Boolean, Int32)	Inizializza una nuova istanza della classe StreamReader per il flusso specificato, con la codifica dei caratteri, l'opzione per il rilevamento dei byte order mark e le dimensioni del buffer specificati.
StreamReader(Stream, Encoding, Boolean, Int32, Boolean)	Inizializza una nuova istanza della classe StreamReader per il flusso specificato in base alla codifica caratteri, all'opzione per il rilevamento dei byte order mark e alle dimensioni del buffer specificati. Facoltativamente mantiene aperto il flusso.

StreamReader(String)	Inizializza una nuova istanza della classe StreamReader per il nome file specificato.
StreamReader(String, Boolean)	Inizializza una nuova istanza della classe StreamReader per il nome file specificato, con l'opzione specificata per il rilevamento dei byte order mark.
StreamReader(String, Encoding)	Inizializza una nuova istanza della classe StreamReader per il nome file specificato, con la codifica dei caratteri specificata.
StreamReader(String, Encoding, Boolean)	Inizializza una nuova istanza della classe StreamReader per il nome file specificato, con la codifica caratteri e l'opzione per il rilevamento dei byte order mark specificati.
StreamReader(String, Encoding, Boolean, Int32)	Inizializza una nuova istanza della classe StreamReader per il nome file specificato, con la codifica caratteri, l'opzione per il rilevamento dei byte order mark e le dimensioni del buffer specificati.

* BOM = è una piccola sequenza di byte che viene posizionata all'inizio di un flusso di dati di puro testo, tipicamente un file, per indicarne il tipo di codifica Unicode. Nel caso di un file o di un'altra sequenza di dati di testo e non binaria, il BOM permette di identificare subito se il testo è in formato Unicode e, in caso affermativo, il tipo esatto di codifica. Ciò è utile quando non si conosce a priori la codifica utilizzata; se invece in una particolare situazione questa è sempre nota i byte del BOM possono risultare inutili o addirittura dannosi. A seconda delle applicazioni l'uso del BOM può essere obbligatorio, opzionale, oppure potrebbe non essere supportato e causare errori. Un semplice programma come il Blocco note di Windows è in grado di riconoscere la codifica dei file di testo aperti in base al BOM e di mostrarli correttamente nascondendo all'utente i byte iniziali che compongono il BOM stesso.

Per leggere un file di testo, potremmo usare questo semplice codice, con una applicazione a console, che utilizza il costruttore a cui viene passato il percorso (path) del nostro file:

```
class Program
    {
        static void Main(string[] args)
        {
            string riga;
            StreamReader sr = new StreamReader(@"F:\Documents\Preghiamo.txt");
            while((riga=sr.ReadLine())!=null)
             {
                Console.WriteLine(riga);
             }
            sr.Close();
            Console.ReadKey();
        }
    }
```

La condizione del while,

```
((riga=sr.ReadLine())!=null)
```

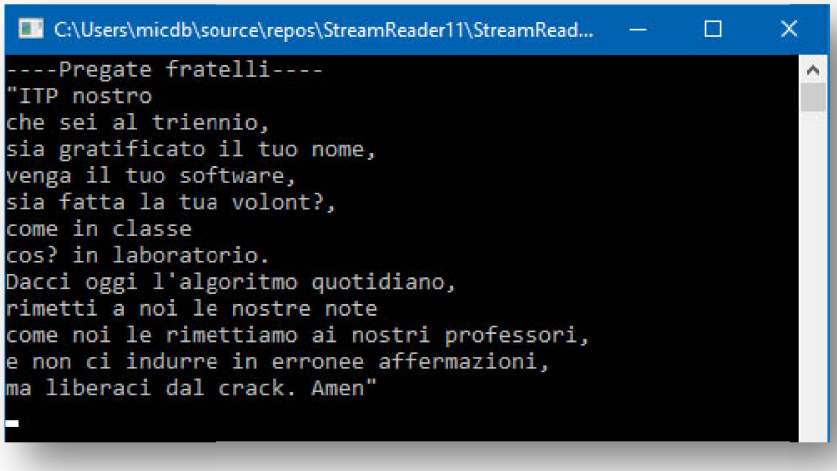
controlla se si è arrivati alla fine del file e non ci sono più righe da leggere.

L'istruzione

```
Console.WriteLine(riga);
```

visualizza la riga letta e va a capo.

In figura una prova di esecuzione:



In quest'altro esempio, invece, leggeremo il file carattere per carattere:

```
class Program
    {
      static void Main(string[] args)
        {
            int car;
            StreamReader sr = new StreamReader(@"F:\Documents\Preghiamo.txt");
            while ((car=sr.Read())!=-1)
               {
                  Console.Write((char)car);
               }
            sr.Close();
            Console.ReadKey();
        }
    }
```

La condizione del while ora è cambiata,

```
((car=sr.Read ())!=-1)
```

controlla se non ci sono più caratteri da leggere, mentre l'istruzione

```
Console.Write((char)car);
```

visualizza il carattere letto convertendo il codice numerico nel carattere corrispondente. Il risultato non cambia.

(continua….)