NODEJS STANDARD LIBRARY

MODULO STREAM

- Gli stream sono uno strumento che permette di gestire in maniera efficiente operazioni come letture/scritture di file, comunicazioni via rete, o altri tipi di comunicazione.
- Ad esempio leggere un file tramite fs.readFile(), il software legge tutto il file e salva in memoria RAM il suo contenuto. Tramite uno stream invece il file viene letto un chunk alla volta senza piazzarlo tutto in memoria.
- In questo modo si ottiene efficienza di memoria, perché grosse quantità di dati non vanno tutte insieme in memoria.
- Si ottiene anche efficienza di tempo, poiché si possono iniziare a processare i dati appena disponibili le prime parti, senza aspettare che tutto sia arrivato.

- Esistono 4 tipi di stream:
  - Writable: stream verso i quali è possibile scrivere dati.
  - Readable: stream da cui i dati vengono letti
  - Duplex: stream che sono contemporaneamente Readable e Writable
  - Transform: simile al Duplex, ma l'output è una qualche trasformazione dell'input

- Importiamo il modulo stream

JS

```
const stream = require('stream');
```

TS

```
import * as stream from 'stream';

import {Readable} from 'stream';
```

Creare un Readable

```
let rStream = new Readable();

rStream.push('Hello');
```

- Esistono due modi per leggere dati da un Readable:
  - Flowing mode: i dati vengono letti più velocemente possibili e restituiti tramite eventi.
  - Paused mode: il chunk successivo deve essere esplicitamente richiesto allo stream prima di essere utilizzabile.

Leggere in flowing mode

```
import * as fs from ‘fs’;

let data = ‘’;

let readerStream = fs.createReadStream('file.txt’);
readerStream.setEncoding('UTF8');

readerStream.on('data', function(chunk) {
        data += chunk;
});

readerStream.on('end',function() {
        console.log(data);
});

readerStream.on('error', function(err) {
        console.log(err.stack);
});

console.log("Program Ended");
```

- Possiamo cambiare la dimensione del chunk di lettura tramite il parametro highWaterMark espresso in bytes.

Leggere in paused mode

```
import * as fs from ‘fs’;

let data = ‘’;
let chunk;

let readerStream = fs.createReadStream('file.txt’);
readerStream.setEncoding('UTF8');

readerStream.on(‘readable', function(chunk) {
        while ((chunk=readableStream.read()) != null) {
                data += chunk;
        }
});

readerStream.on('end',function() {
        console.log(data);
});

console.log("Program Ended");
```

Ricapitolando per i Readable

- Tutti gli stream iniziano in *paused mode* ma passano automaticamente alla *flowing mode* se viene collegata una callback all'evento *data* o se viene effettuato il piping di due stream (che vedremo più avanti).
- Si può passare dalla *flowing mode* alla *paused mode* tramite il metodo pause() o effettuando l'unpiping (che vedremo più avanti).
- E' importante ricordare che un Readable non genera dati finchè non c'è un meccanismo per consumare (o al più ignorare) i dati generati, a prescindere dal *mode.*

Creare un Writable

```
let wStream = new Writable();

wStream._write = (chunk, encoding, next) => {
        console.log(chunk.toString());
}

wStream.end();
```

- E’ necessario fornire un’implementazione del metodo _write che specifica cosa lo stream farà quando riceve un chunk. Ad esempio fs.createWriteStream() scrive il chunk su un file.

Piping

```
let wStream = new Writable();
let rStream = new Readable();

wStream._write = (chunk, encoding, next) => {
        console.log(chunk.toString());
}

rStream.pipe(wStream);

rStream.push('Hello');
rStream.push(' World!');

wStream.end();
```

- Il piping è un concetto ereditato dai sistemi Unix (è espresso con il simbolo | nei terminali linux). La sua funzione è quella di prendere l'uscita di un Readable e reindirizzarla automaticamente verso un Writable o un Duplex. In quest'ultimo caso si può fare il piping a cascata di più stream.
- In node si usa il metodo pipe().

Piping

```
stream1.pipe(stream2).pipe(stream3);
```

- Un esempio di piping multiplo.