Input/Output in JAVA

/*

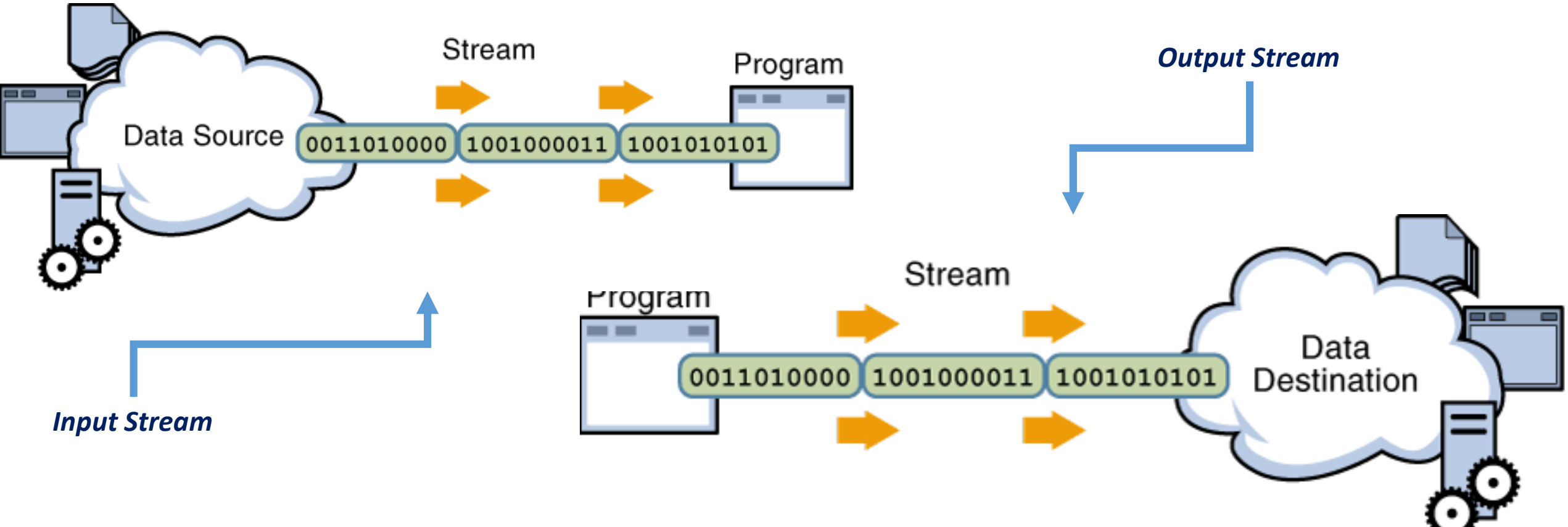
GLI STREAM IN JAVA

I file

*/

GLI STREAM IN JAVA

- In JAVA uno stream è **un canale di comunicazione (stream)**
  - **monodirezionale (o è di input o è di output);**
  - **Ad uso generale (è riferito in generale ad una sorgente e ad una destinazione);**
  - **Che trasferisce byte (o caratteri)**



GLI STREAM IN JAVA

La Sorgente e **Destinazione** possono essere di varia natura:

- **La macchina virtuale (JVM);**
- **Un file (visto dal sistema operativo sottostante);**
- **Un dispositivo di ingresso e o di uscita (tastiera, video, ecc)**
- **Un socket (connessione ad una porta di un computer in rete,**
- **Un altro flusso)**

Nel gergo JAVA i flussi vengono classificati rispetto a vari **chiavi**. Ci interessano particolarmente le seguenti:

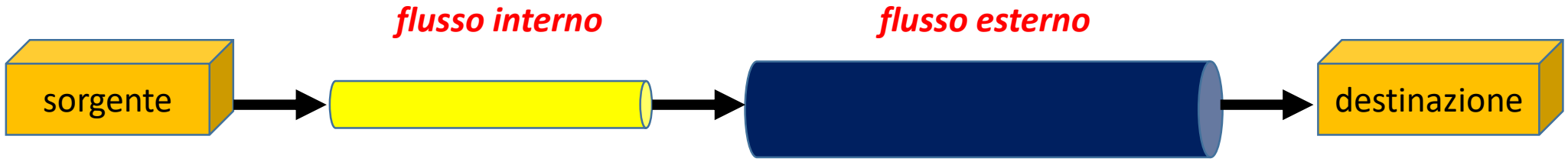
Chiave	Significato
input	Il flusso è diretto dalla sorgente alla macchina virtuale JVM;
output	Il flusso è diretta dalla macchina (JVM) virtuale alla destinazione;
stream	Il flusso è binario, sequenza di byte senza apparente significato;
reader	Flusso di carattere in input da una sorgente
writer	Flusso di carattere in output al programma
Buffered	Il flusso e' bufferizzato, cioè gestito in modo da rendere indipendente la velocita' di chi produce e di chi consuma i dati del flusso (scrittura e lettura a blocchi) la tastiera d'un esempio di flusso bufferizzato
file	Il flusso ha un file come sorgente o come destinazione
print	Flusso di uscita per dispositivi (per esempio il monitor)
object	Flusso binario costituito da sequenze di rappresentaizonni di valori di qualsiasi tipo (anche oggetti)

Le classi e le interfacce di base deputate alla gestione dell I/O sono contenute nel package **java.io.**

Concatenazioni di flussi

Un concetto che può facilitare la conoscenza su come programmare la gestione dei flussi in Java e 'la ***concatenazione di flussi.***

Facendo l'ipotesi che si tratta di flusso d'ingresso al programma (***destinazione***), la destinazione effettua le operazioni facendo riferimento ad un flusso '**esterno**' che in realtà è alimentato da un flusso '**interno**' collegato a sua volta alla sorgente:



Si tratta concretamente di un unico flusso che può essere visto da due diversi punti quello interno potrebbe ad esempio, essere un flusso che gestisce solo sequenze di byte, mentre quello esterno può avere operazioni più sofisticate (come leggi una linea ignorando i caratteri di fine linea, leggi una data, leggi un oggetto, ecc). Altre volte e' utile per scrivere codice indipendente dal tipo di sorgente.

Per realizzare la concatenazione si fa tipicamente uso di una coppia di istruzioni della seguente forma:

```
FlussoInterno fint = new FlussoInterno (parametri);
FlussoEsterno fest = new FlussoEsterno (fint);
```

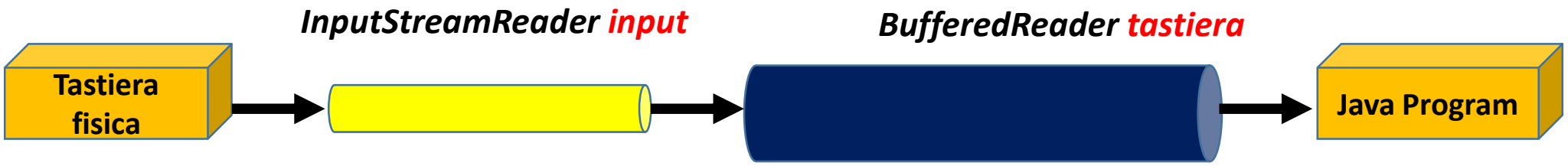
Concatenazioni di flussi: dalla tastiera al programma (lettura dalla tastiera)

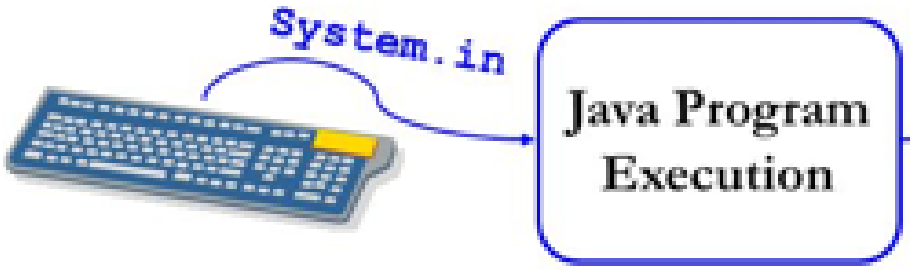
Siamo ora in grado di comprendere meglio le fasi che si inseriscono classicamente per effettuare la lettura da tastiera:

```
InputStreamReader input    = new InputStreamReader(System.in);
BufferedReader    tastiera =  new BufferedReader (input);
```

La classe *BufferedReader* dispone di un medoto ***redLine()*** che restituisce una stringa di caratteri operando in modo bufferizzato.

Lo scopo del flusso intermedio input è quello di convertire un flusso binario (byte) in un flusso di caratteri.





Concatenazioni di flussi: dalla sorgente al programma (LETTURA FILE riga per riga)

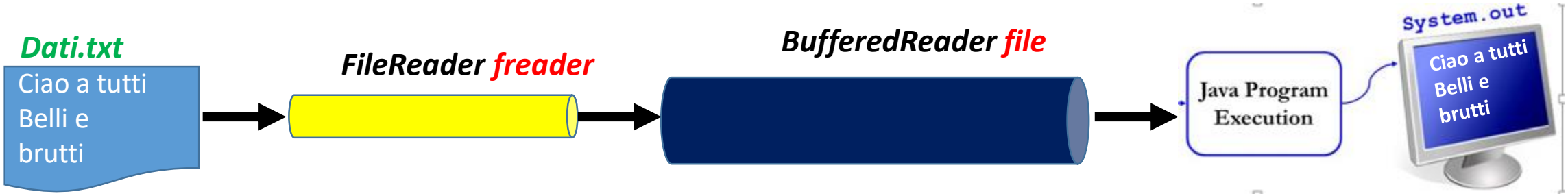
Di seguito le operazioni per leggere da un file del testo (stringhe, riga per riga);

```
try{

        // apre il file in lettura leggendo una riga alla volta

        FileReader     freader = new FileReader ("Dati.txt");
        BufferedReader file    =  new BufferedReader(freader);
        String s;
        s = readeLine(); // legge la prima riga facendo puntare alla seconda il puntatore
        while (s!=null){
                System.out.println (s); // stampa sul monitor quanto letto;
                // legge la riga corrente dal file memorizzandola sulla variabile s;
                s = file.readLine();
        } //while
        fin.close();
}catch (IOException e){
        System.out.println ("Errore nella lettura dal file: \n");
        e.printStackTracer();
        System.exit(1);
} //catch
```

Il metodo *readLine()* restituisce **null** se la riga è vuota



LETTURA DA FILE

(riga per riga)

```
FileReader f;
f = new FileReader("prova.txt");
String s;

BufferedReader b;
b = new BufferedReader(f);

/*
//legge la prima riga ed il puntatore di lettura passa alla seconda
s = b.readLine();
System.out.println(s);

//legge la seconda riga e cosi via
s = b.readLine();
System.out.println(s);
*/

//legge la tutte le righe in una passata
while(true){
        s = b.readLine();
        if(s == null) break;
        System.out.println(s);
} //cuntinuera' a leggere le righe sino a ritornare null
```

Concatenazioni di flussi: dalla sorgente al programma (LETTURA FILE un carattere alla volta)

Di seguito le operazioni per leggere da un file del testo (un carattere alla volta);

Il metodo ***read()*** restituisce **interi** che devono essere convertiti in caratteri Prima di essere visualizzati

```
try{

        // apre il file in lettura leggendo un carattere alla volta

        FileReader freader = new FileReader ("Dati.txt");

        int a;
        a = freader.read(); // legge il primo carattere
        while (a!= -1){
                System.out.println ((char) a); // stampa sul monitor quanto letto;
                // legge la riga corrente dal file memorizzandola sulla variabile a;
                a = freader.read();
        } //while
        freader.close();
}catch (IOException e){
        System.out.println ("Errore nella lettura dal file: \n");
        e.printStackTracer();
        System.exit(1);
} //catch
```

Concatenazioni di flussi: dal programma alla destinazione (SCRITTURA FILE riga per riga)

Di seguito le operazioni per scrivere in un file del testo (stringhe);

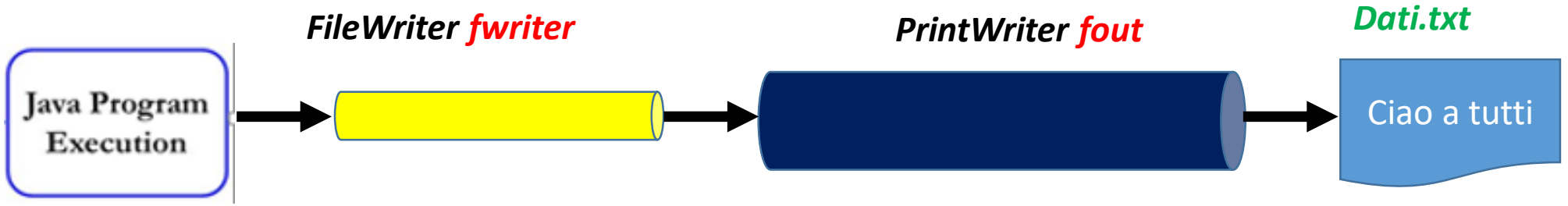
```
try{

        // crea un file nuovo, cancellando eventualmente un omonimo gia' presente

         FileWriter  fwriter = new FileWriter ("Dati.txt");
         PrintWriter fout    =  new PrintWriter(fwriter);

}catch (IOException e){
         System.out.println ("Problemi nella creazione del file: \n");
         e.printStackTracer();
         System.exit(1);
}

// scrivo riga per riga sul file mediante il metodo fout.println("Ciao a tutti");
fout.println("Ciao a tutti");
fout.close;
```



Input/Output in JAVA

/*

CARICARE I DATI DI INPUT PRENDENDOLI DA UN FILE E MEMORIZZARLI IN UN ARRAY

*Se il numero di righe di un file è noto a priori, leggere tutte le righe del file e caricarli in un *array. Questo si può realizzare con un ciclo, come nel programma seguente. Lo stesso sistema si *può usare se vanno letti i primi n elementi del file: si fa un ciclo con dieci iterazioni, e gli * *elementi successivi verranno ignorati.

*/

LETTURA DA FILE

Il numero di righe Del file è noto a priori, legge tutte le righe del file e li carica in un array

```
import java.io.*;

public class CaricaArrayDaFile  {
	public static void main(String[] args) throws IOException
	{
		int v[]=new int[10];
		String str;

		FileReader f = new FileReader("numeri.txt");
		BufferedReader tastiera = new BufferedReader(f);

		// lettura da file e caricamento dell'array
		for(int i=0; i<10; i++)
		{
			//leggiDafile e memeorizza in una var temporanea
			str = tastiera.readLine();
			if(str == null) break; //e'successo un errore in lettura
			v[i]=Integer.valueOf(str).intValue();
		} //for

		// stampa dell'array
		for(int i=0; i<10; i++) { System.out.println(v[i]); } //for
	} //main
} //class
```

Input/Output in JAVA

/*

CARICARE I DATI DI INPUT PRENDENDOLI DA UN FILE E MEMORIZZARLI IN UN ARRAY

*Un metodo alternativo consiste nell'inserire il numero di elementi del vettore come prima riga
*del file. In questo caso, la prima riga del file deve ovviamente contenere il numero di elementi
*seguenti.

*/

LETTURA DA FILE

Il numero di righe Del file è scritto nella prima riga del file

```
import java.io.*;
public class CaricaArrayDaFile{
	public static void main(String[] args) throws IOException{
		int vett[]; // viene creata la variabile ma non il vettore
		String str;
		int DIM; //dimensione dell'array

		FileReader f = new FileReader("numeri.txt");
		BufferedReader tastiera = new BufferedReader(f);

		// lettura dimensione e creazione dell'array
		str = tastiera.readLine();
		DIM = Integer.valueOf(str).intValue();
		vett = new int[DIM];

		// lettura da file e caricamento dell'array
		for(int i=0; i<DIM; i++){
			//leggiDafile e memeorizza in una var temporanea
			str = tastiera.readLine();
			if(str == null) break; //e'successo un errore in lettura
			vett[i] = Integer.valueOf(str).intValue();
		} //for
		// stampa dell'array
		for(int i=0; i<10; i++) {System.out.println(v[i]);} //for
	} //main
} //class
```

Input/Output in JAVA

/*

SCRITTURA DEI DATI IN UN FILE

* Si può scrivere su file usando due modalità:

* 1 carattere per carattere (mediante la classe FileWriter) ;

* 2 per stringhe (mediante la classe BufferedWriter) ;

*/

SCRITTURA IN UN FILE (scrittura di un carattere alla volta)

```
import java.io.*;

public class ScritturaFile
{
	public static void main(String[] args) throws IOException
	{
		FileWriter fileDaScrivereAcarattere;

		//puntatore a file
		fileDaScrivereAcarattere = new FileWriter("dati.txt");

		//scrivo carattere
		fileDaScrivereAcarattere.write('m');
		//scrivo due righe vuote
		fileDaScrivereAcarattere.write('\n');
		fileDaScrivereAcarattere.write('\n');

	} //main
} //class
```

SCRITTURA IN UN FILE

(scrittura di una stringa alla volta)

```
import java.io.*;

public class ScritturaFile
{
	public static void main(String[] args) throws IOException
	{
		FileWriter fileDaScrivereAcarattere;
		BufferedWriter fileDaScrivereAStringhe;

		//puntatore a file
		fileDaScrivereAcarattere = new FileWriter("scrittura.txt");
		fileDaScrivereAStringhe = new BufferedWriter (fileDaScrivereAcarattere);


		//scrivo una stringa
		fileDaScrivereAStringhe.write("abcd\nefghi");
		fileDaScrivereAStringhe.write("123");

		//scrivo due righe vuote
		fileDaScrivereAcarattere.write('\n');

		//Dopo aver scritto tutto bisogna fare flush.
		fileDaScrivereAStringhe.flush();
	} //main
} //class
```