Corso sul linguaggio Java

Modulo JAVA6

A1 – I file testo

Prerequisiti

- Programmazione base in Java
- Utilizzo di classi e oggetti
- Modello produttore consumatore
- Operazioni logiche su struttura file

Introduzione

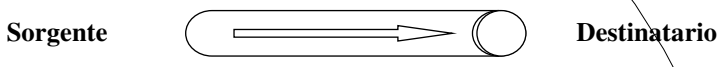
La necessità di memorizzare grandi quantità di dati in modo permanente su memoria secondaria (dischi) viene soddisfatta facendo in modo che le applicazioni Java siano in grado di leggere e scrivere dati su disco in aree dette **file**.

Tutti i moderni linguaggi di programmazione, in particolare i linguaggi *object-oriented*, trattano i file in modo astratto con il nome di **flusso** (*stream*).

In questa Unità vediamo come attraverso il concetto di *stream* un'applicazione sia in grado di usare la memoria di massa per registrare dati di qualunque tipo.

Gli *stream*

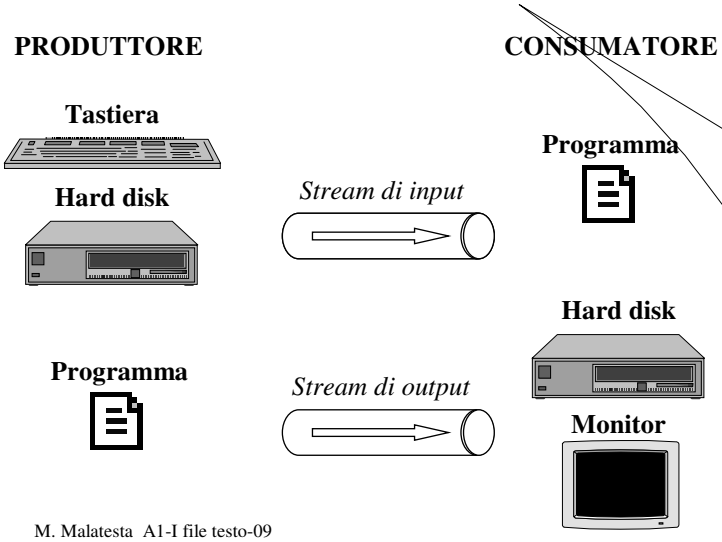
Uno *stream* può essere visto come un qualsiasi canale di comunicazione tra un **sorgente** (**processo produttore**) ed un **destinatario** (**processo consumatore**).



Possiamo avere:

- **stream di input**: trasferiscono dati da un sorgente al programma
- **stream di output**: trasferiscono dati dal programma ad un destinatario

Esempi di *stream*



I file

Nelle considerazioni che faremo, in un qualunque *stream* consideriamo come *sorgente* e come *destinatario* un **file**, ossia una sequenza di dati.

In questo modo una sequenza di dati su disco, la tastiera, il monitor, il modem o uno scanner possono essere visti come **file**.

In particolare consideriamo i **file sequenziali**, ossia quei file in cui per accedere ad una data informazione è necessario scandire tutte le precedenti.

L'informazione viene letta o scritta serialmente, con modalità **FIFO**

Tipi di file

In Java possiamo considerare:

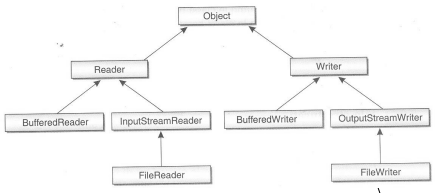
- **file testo**, formati da caratteri (**UNICODE, 16 bit**) e stringhe e leggibili mediante un qualunque editor di testi (in questa Unità)
- **file binari**, formati da sequenza di byte (Unità successive)
- **file di tipi primitivi** (Unità successive)
- **file di oggetti** (Unità successive)

A ciascuno di questi tipi corrispondono opportune classi che vedremo con i relativi metodi.

Stream testo

Le classi orientate al carattere sono **Reader** (per gli *stream* di input) e **Writer** (per gli *stream* di output) che sono **classi astratte**, per cui, normalmente usiamo le loro sottoclassi indicate in figura.

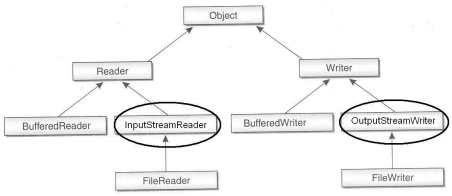
- **Reader**: contiene una parziale implementazione e le API (metodi e attributi) per realizzare stream che leggono caratteri
- **Writer**: contiene una parziale implementazione e le API (metodi e attributi) per realizzare stream che scrivono caratteri



Le sottoclassi di **Reader** e **Writer** implementano *stream* specializzati

Stream testo standard

- **InputStreamReader** per lo *stream standard* input (*tastiera*)
- **OutputStreamWriter** per lo *stream standard* output (*video*)



Metodi:

- **int read()** legge un carattere come intero dallo *stream* (-1 se il file è finito)
- **void write** (**char** c) stampa a video il carattere c
- **void close()** chiude lo *stream*

Stream testo standard

Questa applicazione legge da input una serie di caratteri. Quando lo *stream* di input termina (**Ctrl-Z**) la serie viene stampata (provare da linea di comando)

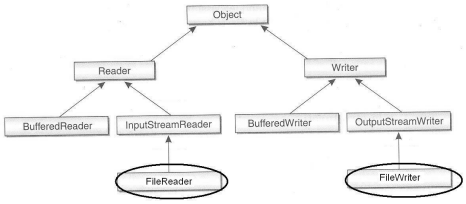
```
import java.io.*;
class streamStandard
{  InputStreamReader stdin; // attributi
   OutputStreamWriter stdout;
   public streamStandard()   // costruttore
   {       stdin = new InputStreamReader(System.in);
           stdout = new OutputStreamWriter(System.out);  }
   public static void main(String args[]) throws Exception
   {       int c;
           streamStandard ss = new streamStandard();
           while ((c=ss.stdin.read())!='*')
           {          ss.stdout.write((char)c);
                      ss.stdout.flush();   // altrimenti  stampa solo alla fine
           }
           ss.stdin.close();    ss.stdout.close();
   }
}
```

Standard input

Standard output

Stream testo su disco

- **FileReader** per *stream* di input su disco
- **FileWriter** per *stream* di output su disco



Metodi:

- **int read()** legge un carattere come intero dallo *stream* (-1 se *stream* finito)
- **void write (char** c**)** scrive il carattere c nello *stream*
- **void close ()** chiude lo *stream*

Stream testo su disco

Questa applicazione legge uno ad uno i caratteri da uno *stream* su disco e li copia in uno *stream* di output sempre su disco.

```
import java.io.*;
class copiaFile
{  FileReader fin;          // attributi
   FileWriter fout;
   public copiaFile()       throws IOException, FileNotFoundException
   {       fin = new FileReader("Testoin.txt");          // costruttore
           fout = new FileWriter("Testoout.txt");        }
   public static void main(String args[]) throws IOException,
                                                FileNotFoundException
   {       int c;
           copiaFile cf = new copiaFile();
           while((c=cf.fin.read())!=-1)
           {          cf.fout.write((char)c);      }
           cf.fin.close();      cf.fout.close();
   }
}
```

Stream testo su disco

1. Inserire sempre il controllo delle eccezioni, in particolare **IOException** e **FileNotFoundException**
2. Volendo evitare il blocco **try-catch** si può scrivere
   public static void main(String args[]) throws IOException, FileNotFoundException
   {… }
3. **FileWriter** fout = **new FileWriter**("Testoout.txt“, **true**);
   In questo caso i nuovi dati vengono aggiunti a quelli esistenti

Stream testo su disco

ATTIVITA': scrivere una applicazione *leggiScriviTesto.java* che acquisisca da standard input una serie di caratteri fino all'immissione di un '*' e la scriva in un file testo *testo.txt*.

ATTIVITA': scrivere una applicazione che acquisisca da standard input una serie di numeri interi terminata da uno 0, la memorizzi in un file testo *numeri.dat* e la restituisca successivamente in stampa a video.

Stream testo bufferizzati

Per migliorare l'efficienza si usa la **bufferizzazione**, ossia la presenza di un'area di memoria (*buffer*) interposta fra sorgente e destinatario.

La bufferizzazione richiede classi di input e di output apposite che vedremo fra breve.

Quando si usa la bufferizzazione, occorre **concatenare** gli oggetti dello *stream* aperto con quelli del buffer da utilizzare.

Un esempio, che abbiamo spesso utilizzato è:

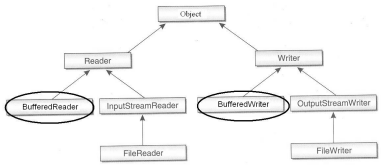
InputStreamReader In = **new InputStreamReader** (System.in);
BufferedReader tastiera = **new BufferedReader** (In);

Il primo oggetto rappresenta lo *stream* usato come input; il secondo oggetto, concatenato con il primo è quello che possiede i metodi effettivi per leggere (ad es. **readLine**()).

Stream testo bufferizzati

Le classi di bufferizzazione per i file testo sono:.

- **BufferedReader** per *streaam bufferizzati* di input
- **BufferedWriter** per *stream bufferizzati* di output



Metodi:

- **String readLine()** legge una stringa (**null** se lo *stream* è finito)
- **void write (String** s**)** scrive la stringa s nello *stream* di output
- **void newLine()** inserisce un "a capo" nello *stream* di output
- **void close** **()** chiude lo *stream*

Stream testo bufferizzati

ATTIVITA': scrivere un'applicazione che legga una ad una le righe di uno *stream* su disco e le copi in uno *stream* di output su disco (usare la bufferizzazione)

```
import java.io.*;
class copiaTesto
{  public static void main(String args[]) IOException, FileNotFoundException
   {  String s;
      FileReader fin = new FileReader("Testoin.txt");
      BufferedReader bufin = new BufferedReader (fin);
      FileWriter fout = new FileWriter ("Testoout.txt", false);
      BufferedWriter bufout = new BufferedWriter (fout);
      while((s=bufin.readLine())!=null)
      {   bufout.write(s);
          bufout.newLine();
      }
      bufin.close();
      bufout.close();
   }
}
```

Sostituiscono blocco **try-catch**

Il controllo di fine *stream* richiede **null**

Argomenti

- Gli *stream*
- Esempi di *stream*
- I file
- Tipi di file
- *Stream* testo
- *Stream* testo standard
- *Stream* testo su disco
- *Stream* testo bufferizzati

Altre fonti di informazione

- P.Gallo, F.Salerno – Java, ed. Minerva Italica
- M. Bigatti – Il linguaggio Java, ed. Hoepli