Tipo File

Per memorizzare un dato su un supporto magnetico come un hard disk o un nastro, o più in generale su un'unità di memoria di massa viene utilizzata un tipo di dato chiamato file.

Un **file** può essere considerato come una ***sequenza di byte***.

> Un **file** è una struttura dati permanente che contiene un blocco logicamente contiguo di informazioni che risiede su disco.

Si ricordi che il sistema di I/O di Java come del C++ (che si basa sull'uso di *flussi* cioè sequenze di byte che viaggiano da un'origine ad una destinazione lungo un percorso di comunicazione) ha un'interfaccia indipendente dall'hardware, ossia è analogo accedere a terminali, unità a disco, a nastro eccetera: ogni dispositivo pur diverso dagli altri, viene associato ad un dispositivo logico chiamato ***stream*** e le *operazioni sui file* si riferiscono non solo ai file su disco, ma anche alle periferiche (anche i messaggi di posta elettronica in partenza ed in arrivo, i caratteri battuti sulla tastiera, l'output sul video del terminale, i dati che passano da un programma all'altro possono essere utilizzati dai programmi come file, in quanto non sono altro che sequenze di byte).

L'I/O in Java è definito in termini di ***stream*** (***flussi***). Gli stream sono un'astrazione di alto livello per rappresentare la connessione a un canale di comunicazione. Il **canale di comunicazione** può essere costituito fra entità molto diverse, le più importanti delle quali sono:

- un file;
- una connessione di rete (ad esempio TCP/IP);
- un buffer in memoria.

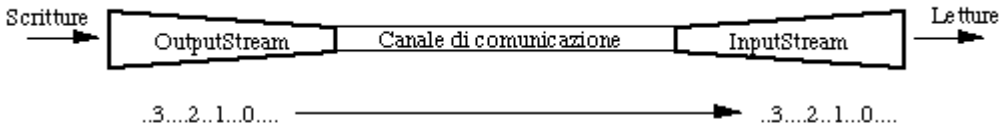

Figura 1-1: Stream di input e output

Tutto ciò che viene scritto sul canale tramite l'*OutputStream* viene letto dall'altra parte dal corrispondente *InputStream*.

Grazie all'astrazione rappresentata dagli stream, le operazioni di I/O dirette a (o provenienti da) uno qualunque degli oggetti di cui sopra sono realizzate con la stessa interfaccia.

Gli stream hanno diverse proprietà:

- sono ***FIFO***: ciò che viene scritto da un `OutputStream` viene letto nello stesso ordine dal corrispondente `InputStream`;
- sono ***ad accesso sequenziale***: non è fornito alcun supporto per l'accesso casuale
- sono ***read-only oppure write-only***: uno stream consente di leggere (`InputStream`) o scrivere (`OutputStream`) ma non entrambe le cose. Se ambedue le funzioni sono richieste, ci vogliono 2 distinti stream: questo è un caso tipico delle connessioni di rete, tant'è che da una connessione (`Socket`) si ottengono due stream, uno in scrittura e uno in lettura;
- sono ***bloccanti***: la lettura blocca il programma che l'ha richiesta finché i dati non sono disponibili. Analogamente, la scrittura blocca il richiedente finché non è completata;
- quasi tutti i loro metodi possono ***generare eccezioni***.

I file posso essere di due tipi:

1. FILE di TESTO: è una sequenza di *caratteri*, che durante il trasferimento può subire anche delle conversioni a seconda delle necessità dell'ambiente di destinazione. È stampabile.

2. FILE BINARI: è una sequenza di *byte* avente una *corrispondenza uno a uno* con la sequenza ricevuta dal/inviata al dispositivo esterno. Non è stampabile.

I *file* costituiscono le strutture di supporto per flussi di dati, quali sono considerate a livello di file system, e vengono gestite – in Java – a livello globale dalla classe **File**. Le operazioni specifiche sul contenuto dei file dipendono dall'organizzazione e dal tipo di accesso per essi previsti.

Il linguaggio Java prevede per i file le due tipologie di **accesso sequenziale** e **diretto**, distinguendo inoltre tra file **formattati** e **non formattati**.

La classe **File** definita nel **package java.io** dà la possibilità di uniformare gli accessi in funzione dell'host su cui si opera, in modo da risolvere alcune delle difficoltà di trasporto del codice da una macchina ad un'altra.

La classe File **non agisce sui flussi**, descrive le proprietà del file stesso, è la classe fondamentale per accedere alle informazioni relative ai file ed alle directory.

> La classe **File** definisce oggetti file che vengono associati ai file fisici del file system e un insieme di metodi per creare, cancellare e gestire file.

I principali **metodi** della classe File sono preposti alla creazione di file o directory, all'acquisizione dei loro pathname o delle directory di appartenenza, alla definizione dei privilegi di lettura e scrittura, alla loro ridenominazione o rimozione dal file system.

File

```
File (String path)
File (File dir, String name)

String getName()
String getPath()
boolean exists()

boolean isFile()
boolean isDirectory()
String[] list()

boolean mkdir()
boolean renameTo (File dest)
boolean delete()
```

- I **costruttori** della classe sono preposti alla creazione di oggetti di tipo File, afferenti al file system del calcolatore, il cui *pathname* può essere espresso con modalità distinte. Nella *prima forma* il parametro del costruttore definisce il pathname completo del file. Nella *seconda forma*, il costruttore richiede l'identificatore name del file e la directory dir di collocazione: qualora sia null, viene assunta quale directory di collocazione la directory corrente.

- I metodi **getName()** e **getPath()** forniscono, rispettivamente, l'identificatore del File a cui è inoltrato il messaggio oppure il pathname completo dello stesso.
- Il metodo **exists()** controlla l'esistenza o meno del file a cui è inoltrato il messaggio. Similarmente, i metodi **isFile()** e **isDirectory()** stabiliscono se l'oggetto destinatario del messaggio è un file oppure una directory.
- Il metodo **list()** è attivato in risposta a messaggi inoltrati a directory. Fornisce una lista dei file in essa presenti.
- Il metodo **mkdir()** assegna la struttura di directory all'oggetto File a cui è inoltrato il messaggio. In modo similare il metodo **renameTo()** ridenomina il file destinatario del messaggio, così che sia identificato dal pathname specificato dal parametro dest.
- Il metodo **delete()** rimuove il file destinatario: qualora si tratti di una directory, questa *deve* essere vuota.

Esiste anche il metodo **length()** che restituisce la lunghezza del file. Non sono definiti metodi per la lettura e scrittura su file.

N.B: La classe File non contiene operazioni di lettura e scrittura su file!

Un oggetto di tipo File è una rappresentazione astratta di un "documento", viene visualizzato per ottenere o manipolare le informazioni associate ad un file sul disco, quali i permessi, l'ora, la data e il percorso della directory e per gestire le gerarchie delle sottodirectory. Infatti, una directory viene trattata come File, con un'ulteriore proprietà: un elenco di nomi di file che possono essere esaminati con un metodo.

Gli oggetti della classe File rappresentano dunque file o directory afferenti al file system del calcolatore da gestire quali *entità atomiche*. Ognuno di essi è individuato da un *pathname*, assoluto oppure relativo alla directory di lavoro corrente, che è strutturato secondo le convenzioni del sistema.

*È importanta ricordare i problemi di **portabilità** del codice: per quanto riguarda i separatori nel percorso Java accetta sia lo standard Unix/Internet con la barra (/) sia la convenzione Windows/DOS di barra rovesciata (\) con l'accortezza di dover utilizzare la sua sequenza di escape (\\) in una stringa. Ad esempio per creare un oggetto file che referenzi la radice di un disco si può fare indistintamente nei due seguenti modi*

```
File f1=new File("/");
File f2=new File("\\");
```

Path

- Unix: /bin**:**/usr/bin**:**/usr/local/bin
- Windows: c:\bin; c:\java\bin

Percorso di un file

- Unix: **/**usr**/**java**/**bin**/**javac
- Windows: c:****java****bin

Nella classe File sono definite anche **costanti** statiche

```
import java.io.*;

public class Prova {                                    // uso costanti della classe File

        public static void main(String args[]){
                System.out.println(
                File.separator + " - " +
                File.separatorChar + " - " +
                File.pathSeparator + " - " +
                File.pathSeparatorChar);
        }
}               //  Output (su Windows): \ - \ - ; - ;
```

Costruttori

public File(String path)

Esempio:
File f = new File("C:\prova.txt");
File cartella = new File("C:\cartella");

public File(String path, String name)

Esempio:
File f = new File("C:\", "prova.txt");

public File(File dir, String name)

Esempio:
File f = new File(cartella, "prova.txt");

File e Stream

Distinguiamo tra due classi di funzionalità:

- Funzionalità per *accedere ad informazioni* relative ai file ed alle directory.
  Esempio: Dimensione dei file, elenco dei file contenuti in una directory,...
  Ed abbiamo visto che la classe fondamentale per accedere alle informazioni relative ai file ed alle directory è la classe **File**

- Funzionalità per *leggere* e *scrivere* dati

Si distingue tra:

- classi per l'accesso *sequenziale* ai dati (di utilizzo generale, non limitate ai file)
- classi per l'accesso *diretto* (random) a dati contenuti su file

Il meccanismo di lettura e scrittura sequenziale dei dati è basato sul concetto di ***stream***

Leggere o scrivere su un file creando un oggetto *stream*

Gli ***stream*** possono essere suddivisi per…

tipo di dato:

- stream di caratteri (16 bit)
- stream di byte (8 bit)

funzionalità:

- **data sink** stream: durante il transito i dati non sono processati.
  Stream di tipo **FileWriter** (per scrivere un singolo carattere alla volta all'interno del file) o di tipo **FileReader** (per leggere un singolo carattere alla volta da file)

- **processing** stream: i dati vengono processati (es bufferizzazione)
  Stream di tipo BufferedWriter (per scrivere intere stringhe) o di tipo **`BufferedReader`** (per leggere intere stringhe) riducendo gli accessi ed aumentando l'efficienza.