Laboratorio di Informatica
Lezione 4

Cristian Consonni

28 ottobre 2015

Outline

Chi sono

Cristian Consonni

- **DISI - Dipartimento di Ingegneria e Scienza dell'Informazione**
- **Pagina web** del laboratorio:
  http://disi.unitn.it/~consonni/teaching
- **Email**: cristian.consonni@unitn.it
- **Ufficio**: Povo 2 - Open Space 9
  - Per domande: scrivetemi una mail
  - Ricevimento: su appuntamento via mail

Outline for section 1

Note sullo stile

- Utilizzare il camelCase/PascalCase
  - `https://en.wikipedia.org/w/index.php?title=CamelCase&oldid=686054689`
- Quando create una classe, la prima lettera è **Maiuscola**, ad es. `TestPrimo`;
- I metodi e le variabili hanno la lettera iniziale **minuscola**, ad es. `sommaInteri`, `stampaVettore`;
- le costanti vanno indicate tutte in **MAIUSCOLO**, ad es. `NMAX`, `EPSILON`;

Canali standard (I)

*≪I **canali standard** (o **standard streams**), in tutti i moderni sistemi operativi, rappresentano i dispositivi logici di input e di output che collegano un programma con l'ambiente operativo in cui esso viene eseguito (tipicamente un terminale testuale) e che sono connessi automaticamente al suo avvio.≫*

fonte: `https://it.wikipedia.org/wiki/Canali_standard`

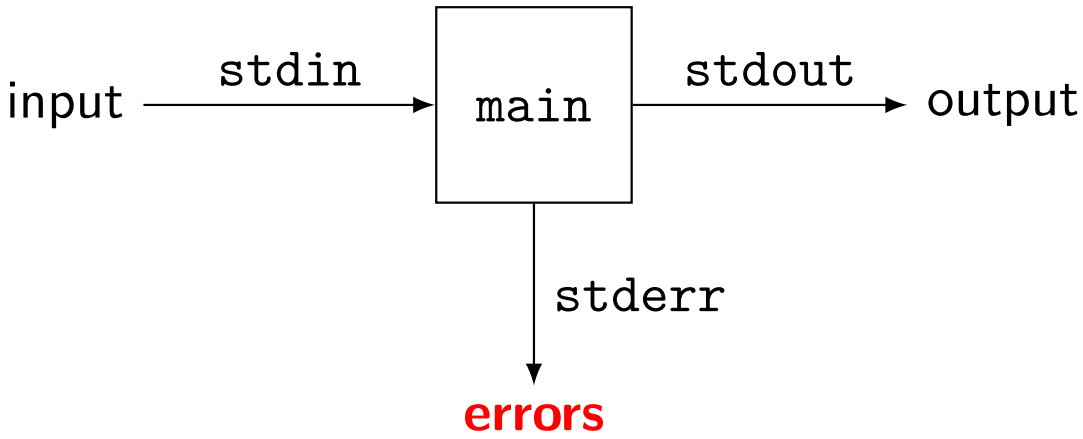
Canali standard (II)

Esistono tre canali standard predifiniti:

- **`stdin`**: *standard input*;
- **`stdout`**: *standard output*;
- **`stderr`**: *standard error*;

Canali standard (III)

I canali standard possono essere rappresentati nel modo seguente:



Canali standard (IV)

Questi canali sono tutti legati al **terminale** (o *console* o *command prompt*).

Per esempio, tramite il terminale possiamo:

- leggere un input da terminale (`stdin`);
- stamparlo a schermo (`stdout`);
- raccogliere un messaggio di errore (`stderr`);

Canali standard (IV)

Questi canali sono tutti legati al **terminale** (o *console* o *command prompt*).
Per esempio, tramite il terminale possiamo:

- leggere un input da terminale (`stdin`);
- stamparlo a schermo (`stdout`);
- raccogliere un messaggio di errore (`stderr`);

Canali standard (IV)

Questi canali sono tutti legati al **terminale** (o *console* o *command prompt*).
Per esempio, tramite il terminale possiamo:

- leggere un input da terminale (`stdin`);
- stamparlo a schermo (`stdout`);
- raccogliere un messaggio di errore (`stderr`);

Canali standard (IV)

Questi canali sono tutti legati al **terminale** (o *console* o *command prompt*).
Per esempio, tramite il terminale possiamo:

- leggere un input da terminale (`stdin`);
- stamparlo a schermo (`stdout`);
- raccogliere un messaggio di errore (`stderr`);

Canali standard (IV)

Questi canali sono tutti legati al **terminale** (o *console* o *command prompt*).
Per esempio, tramite il terminale possiamo:

- leggere un input da terminale (`stdin`);
- stamparlo a schermo (`stdout`);
- raccogliere un messaggio di errore (`stderr`);

Canali standard (V)

In sistemi Unix (Linux e Mac OS X) è possibile separare (redirezionare) i diversi streams lanciando il comando con una sintassi speciale:

```
java LetturaFile <in 1>out 2>err
```

- `<` redireziona lo `stdin`;
- `1>` redireziona lo `stdout`;
- `2>` redireziona lo `stderr`;

Canali standard (V)

In sistemi Unix (Linux e Mac OS X) è possibile separare (redirezionare) i diversi streams lanciando il comando con una sintassi speciale:

```
java LetturaFile <in 1>out 2>err
```

- < redireziona lo `stdin`;
- 1> redireziona lo `stdout`;
- 2> redireziona lo `stderr`;

Canali standard (V)

In sistemi Unix (Linux e Mac OS X) è possibile separare (redirezionare) i diversi streams lanciando il comando con una sintassi speciale:

```
java LetturaFile <in 1>out 2>err
```

- < redireziona lo `stdin`;
- 1> redireziona lo `stdout`;
- 2> redireziona lo `stderr`;

Canali standard (V)

In sistemi Unix (Linux e Mac OS X) è possibile separare (redirezionare) i diversi streams lanciando il comando con una sintassi speciale:

```
java LetturaFile <in 1>out 2>err
```

- < redireziona lo `stdin`;
- 1> redireziona lo `stdout`;
- 2> redireziona lo `stderr`;

Outline for section 2

Lettura input da tastiera (I)

Abbiamo già visto che per stampare dei valori a schermo si usa:

- `System.out`

questo permette di interagire con lo `stdout`.

Analogamente:

- `System.in`

permette di interagire con lo `stdin`.

Lettura input da tastiera (II)

Nella libreria `java.util` di Java esiste l'oggetto **`Scanner`** che ci permette di utilizzare l'input della console:

```
import java.util.Scanner;

public class LetturaFile {

  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    ...

    // Al termine della lettura chiudere il canale di input
    input.close();
  }
}
```

Lettura input da tastiera (III)

Per potere utilizzare **`Scanner`** dobbiamo *importare* la libreria **`java.util`** aggiungendo la riga:

```
import java.util.Scanner;
```

all'inizio del file.

Lo statement import è il modo in cui si possono *importare* librerie in un programma Java.

Una libreria è un insieme di funzioni e/o strutture dati predefinite e predisposte per essere riutilizzate facilmente all'interno di svariati programmi.

Modalità di lettura da `stdin` (I)

Dopo aver dichiarato l'oggetto di tipo `Scanner`

```
Scanner input = new Scanner(System.in);
```

si possono usare i seguenti **metodi**:

- `input.nextInt()` $\Rightarrow$ lettura di un int;
- `input.nextDouble()` $\Rightarrow$ lettura di un double;
- `input.next()` $\Rightarrow$ lettura di una stringa;
- `input.nextLine()` $\Rightarrow$ legge l'intera linea fino al carattere `newline` (Invio);

Modalità di lettura da `stdin` (II)

Se viene inserito un valore di un tipo non corrispondendente viene causato un **errore** ovvero viene *lanciata un'**eccezione***.

Se per esempio uso `input.nextInt()`:

```
Inserisci un numero intero: 3.2

Exception in thread "main" java.util.InputMismatchException
  at java.util.Scanner.throwFor(Scanner.java:864)
  at java.util.Scanner.next(Scanner.java:1485)
  at java.util.Scanner.nextInt(Scanner.java:2117)
  at java.util.Scanner.nextInt(Scanner.java:2076)
  at LetturaIntero.main(LetturaIntero.java:11)
```

Modalità di lettura da `stdin` (II)

Se viene inserito un valore di un tipo non corrispondendente viene causato un **errore** ovvero viene *lanciata un'**eccezione***.

Se per esempio uso **`input.nextInt()`**:

```
Inserisci un numero intero: 3.2

Exception in thread "main" java.util.InputMismatchException
  at java.util.Scanner.throwFor(Scanner.java:864)
  at java.util.Scanner.next(Scanner.java:1485)
  at java.util.Scanner.nextInt(Scanner.java:2117)
  at java.util.Scanner.nextInt(Scanner.java:2076)
  at LetturaIntero.main(LetturaIntero.java:11)
```

Modalità di lettura da stdin (III)

Esistono dei metodi per controllare il tipo di dato inserito:

- `input.hasNextInt()` $\Rightarrow$ controlla che venga letto un int;
- `input.hasNextDouble()` $\Rightarrow$ controlla che venga letto un double;
- etc.

Queste funzioni ritornano true se il prossimo valore è del tipo indicato:

```
print("Inserisci la tua altezza (in cm): ");
while (!input.hasNextInt()) {
  println("Il numero inserito non è un intero valido.");
  print("Inserisci la tua altezza (in cm): ");
  input.next();
}
```

Outline for section 3

Scrittura su stdout (I)

Abbiamo già visto dei comandi/funzioni per scrivere sullo standard output ("a schermo"):

- `System.out.print( stringa )`
- `System.out.prinlnf( stringa )`
- `System.out.printf( formato, argomenti, ...)`

Scrittura su `stdout` (II)

Alcune cose utili:

- Con printf il formato "%nd", dove *n* è un numero (intero), stampa spazi aggiuntivi in modo tale che il numero occupi sempre *n* spazi (si veda l'esercizio `MatriceTrasposta`);
- Con printf il formato "%0nd", dove *n* è un numero (intero), stampa degli zeri aggiuntivi in modo tale che il numero occupi sempre *n* spazi, ad esempio "%03d" stampa il numero 7 come 007;
- altre *sequenze di escape* particolari sono `\t` per inserire una tabulazione (un numero di spazi variabile che allinea l'output lungo una data colonna)

Outline for section 4

Esercizi (parte I) (I)

Scrivere un programma in cui:

- si inserire un numero intero $N \leq 10$ da console;
- Inizializzare un array di interi di lunghezza N con i valori presi in input da console (ciclare finché non si ottengono N valori del tipo desiderato);
- Dopo aver letto tutti i valori stampare progressivamente i valori inseriti;

Esercizi (parte I) (II)

Per esempio, se l'array inserito è il seguente $\{2, 5, 7, 4\}$. Il programma deve ciclare sull'array stampando:

```
2
2, 5
2, 5, 7
2, 5, 7, 4
```

- Suggerimento: sono due for annidati (uno all'interno dell'altro) utilizzate `System.out.print` per stampare i valori e quando uscite dal for interno stampate una riga vuota

Outline for section 5

Lettura da file (I)

Il caso della lettura da file è simile a quello della lettura da `stdin`:

- viene creato un **oggetto** `File` che rappresenta il file nel *filesystem*.
- si usa uno `Scanner` come nel caso della lettura da *standard input*

Lettura da file (II)

```
import java.io.File;
import java.util.Scanner;

public class LetturaFile {
  public static void main(String[] args) {

    File file = new File("esempio.txt");
    Scanner scan = new Scanner(file);

    ...
```

Si noti l'utilizzo delle due librerie:

- `java.io.File`
- `java.util.Scanner`

Lettura da file (II)

```
import java.io.File;
import java.util.Scanner;

public class LetturaFile {
  public static void main(String[] args) {

    File file = new File("esempio.txt");
    Scanner scan = new Scanner(file);

    ...
```

Si noti l'utilizzo delle due librerie:

- `java.io.File`
- `java.util.Scanner`

Percorsi (I)

Prima abbiamo indicato "esempio.txt" come nome del file, in generale possiamo indicare un **percorso** (o **path**).

I percorsi si specificano in modo diverso per sistemi operativi diversi:

- Unix (Linux, Mac OS X)
  `/utente/unitn/informatica/lab/prova.txt`
- Windows
  `C:\utente\unitn\informatica\lab\prova.txt`

Il nome del file va sempre inserito **completo di estensione**.

Percorsi (II)

Inoltre un percorso può essere:

- relativo, se si riferisce alla cartella corrente.
  - i path relatvi si riferiscono alla cartella corrente (indicata con `.`, `..` alla cartella genitore di quella corrente).
  - in Eclipse la cartella corrente è la **cartella principale del progetto** all'interno del *workspace*.
  - "esempio.txt" è un path relativo alla cartella corrente.
- assoluto, se si riferisce alla radice del filesystem (`/` su sistemi Unix, `C:\` su Windows)
  - `/utente/unitn/info/prova.txt`
  - `C:\utente\unitn\info\prova.txt`

Lettura da file (III)

```
import java.io.File;
import java.util.Scanner;

public class LetturaFile {
  public static void main(String[] args) {

    File file = new File("esempio.txt");
    Scanner scan = new Scanner(file);

    ...
```

Dopo aver scritto la dichiarazione dello `Scanner` dovremmo avere ottenuto:

- un warning relativo al fatto che non chiudiamo lo scanner.
- un errore relativo al fatto che una **eccezione non è gestita** (*unhandled*).

Lettura da file (III)

```
Scanner scan = new Scanner(file);
```

Possiamo risolvere il problema in due modi (vedere anche la funzionalità *quickfix* attivata con il comando `Ctrl + 1`):

- **lanciare** (**throw**) un'eccezione;
- **gestire** l'eccezione con il costrutto **`try - catch`**;

Questo avviene perché la lettura da file può causare errori come per esempio un *file non esistente* oppure *non accessibile*.

Lanciare un'eccezione

Se decidiamo di lanciare un'eccezione allora la dichiarazione del `main` viene modificata come segue:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class LetturaFile {

  ... main(String[] args) throws FileNotFoundException {

    File file = new File("esempio.txt");
    Scanner scan = new Scanner(file);
    ...
```

Questa sintassi segnala che la funzione `main` può terminare a causa dell'errore `FileNotFoundException`.

Gestire un'eccezione con try - catch

La gestione dell'eccezione con try - catch avviene nel modo seguente:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class LetturaFile {

  public static void main(String[] args) {

    File file = new File("esempio.txt");
    Scanner scan;
    try {
      scan = new Scanner(file);
    } catch (FileNotFoundException e) {
      e.printStackTrace();
      return;
    }
```

Gestire un'eccezione con `try - catch` (II)

Le istruzioni nel blocco `try` vengono provate, se non causano errori l'esecuzione del programma prosegue normalmente, **in caso di errore** viene eseguito il blocco `catch`:

```
try {
  scan = new Scanner(file);
} catch (FileNotFoundException e) {
  // Queste istruzioni vengono eseguite solo se il
  // blocco try ha restituito un errore del tipo
  // FileNotFoundException
  e.printStackTrace();
  return;
}
```

Lo scopo del blocco `try - catch` è quello di compiere le operazioni necessarie in conseguenza dell'errore.

Outline for section 6

Scrittura file (I)

In modo analogo a quanto avviene per la lettura da file, per la scrittura:

1. dobbiamo creare un oggetto che rappresenti il file su cui vogliamo scrivere, esso rappresenta il file nel *filesystem*;
2. utilizziamo la classe di Java `FileWriter` , questa classe si preoccupa di preparare il file per la scrittura gestendo, per esempio, l'**encoding** del file.
3. creiamo un "buffer" `BufferedWriter` che effettivamente conterrà il testo da scrivere sul file.
4. per scrivere usiamo il metodo `write()` del buffer
5. Quando abbiamo terminato, chiudiamo il buffer.

Scrittura file (I)

In modo analogo a quanto avviene per la lettura da file, per la scrittura:

1. dobbiamo creare un oggetto che rappresenti il file su cui vogliamo scrivere, esso rappresenta il file nel *filesystem*;
2. utilizziamo la classe di Java `FileWriter` , questa classe si preoccupa di preparare il file per la scrittura gestendo, per esempio, l'**encoding** del file.
3. creiamo un "buffer" `BufferedWriter` che effettivamente conterrà il testo da scrivere sul file.
4. per scrivere usiamo il metodo `write()` del buffer
5. Quando abbiamo terminato, chiudiamo il buffer.

Scrittura file (I)

In modo analogo a quanto avviene per la lettura da file, per la scrittura:

1. dobbiamo creare un oggetto che rappresenti il file su cui vogliamo scrivere, esso rappresenta il file nel *filesystem*;
2. utilizziamo la classe di Java `FileWriter` , questa classe si preoccupa di preparare il file per la scrittura gestendo, per esempio, l'**encoding** del file.
3. creiamo un "buffer" `BufferedWriter` che effettivamente conterrà il testo da scrivere sul file.
4. per scrivere usiamo il metodo `write()` del buffer
5. Quando abbiamo terminato, chiudiamo il buffer.

Scrittura file (I)

In modo analogo a quanto avviene per la lettura da file, per la scrittura:

1. dobbiamo creare un oggetto che rappresenti il file su cui vogliamo scrivere, esso rappresenta il file nel *filesystem*;
2. utilizziamo la classe di Java `FileWriter` , questa classe si preoccupa di preparare il file per la scrittura gestendo, per esempio, l'**encoding** del file.
3. creiamo un "buffer" `BufferedWriter` che effettivamente conterrà il testo da scrivere sul file.
4. per scrivere usiamo il metodo `write()` del buffer
5. Quando abbiamo terminato, chiudiamo il buffer.

Scrittura file (I)

In modo analogo a quanto avviene per la lettura da file, per la scrittura:

1. dobbiamo creare un oggetto che rappresenti il file su cui vogliamo scrivere, esso rappresenta il file nel *filesystem*;
2. utilizziamo la classe di Java `FileWriter` , questa classe si preoccupa di preparare il file per la scrittura gestendo, per esempio, l'**encoding** del file.
3. creiamo un "buffer" `BufferedWriter` che effettivamente conterrà il testo da scrivere sul file.
4. per scrivere usiamo il metodo `write()` del buffer
5. Quando abbiamo terminato, chiudiamo il buffer.

Scrittura file (II)

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

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

    String text = "TestoEsempio";

    File file = new File("esempio.txt");
    FileWriter fw = new FileWriter(file);
    BufferedWriter bw = new BufferedWriter(fw);

    bw.write(text);
    bw.flush();
    bw.close();
    ...
```

Scrittura file (III)

Ogni qual volta aggiungiamo una delle classi necessarie alla scrittura su File Java segnala un errore che può essere risolto aggiungengo gli import *statement* necessari.

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
```

Scrittura file (V)

Anche quando aggiungiamo il `FileWriter` dobbiamo preparci a gestire l'eccezione, in questo caso segnaliamo il fatto che il main può generare una `IOException`.

```
public class ScritturaFile {

  public static void main(String[] args) throws IOException {

    String text = "TestoEsempio";

    File file = new File("esempio.txt");
    FileWriter fw = new FileWriter(file);
    BufferedWriter bw = new BufferedWriter(fw);
```

Scrittura file (VI)

```
public class ScritturaFile {

  public static void main(String[] args) throws IOException {

    String text = "TestoEsempio";

    File file = new File("esempio.txt");
    FileWriter fw = new FileWriter(file);
    BufferedWriter bw = new BufferedWriter(fw);

    bw.write(text);
    // Forza lo svuotamento del buffer e la scrittura su
    // file
    bw.flush();
    bw.close();
  }
}
```

Outline for section 7

Esercizi (I)

- Scrivere un programma che apra il file "esercizio01.txt" (scaricatelo da qui `http://bit.ly/esercizi-labinfo` e inseritelo nella cartella principale del progetto)
- Il file contiene una serie di array di interi, la prima riga indica il numero di elementi per riga, mentre le righe successive sono effettivamente gli array. Stampate a video il massimo di ciascuna riga e alla fine il valore massimo totale.

Esempio di input:

```
8
1  2  9  5  7  12  2  3
4  9  1  5  8  14  5  6
0  6  11 5  9  2   6  0
```

Esercizi (II)

- Scrivere un programma che apra il file "divina.txt" (scaricatelo da http://bit.ly/esercizi-labinfo e inseritelo nella cartella principale del progetto);
- Il file contiene il primo capitolo della divina commedia. Leggete linea per linea, contando il numero di "a", "e", "i", "o", "u";
- Una volta letto tutto il file, create un nuovo file chiamato "risultati.txt" in cui scriverete il numero di "a", "e", "i", "o", "u" presenti;

Suggerimento:

- utilizzate 5 variabili diverse o un array da 5 elementi, come preferite) scorrete la stringa e controllate carattere per carattere con il metodo

```
nomeStringa.charAt(indice)
```