Reindirizzamento di input e output

Calcolare la somma di numeri

```
import java.util.Scanner;

public class Sum
{
    public static void main(String[] args)
    {
        Scanner console = new Scanner(System.in);
        double sum = 0;
        while (console.hasNextDouble()){
            sum = sum + console.nextDouble();
        }
        System.out.println("Somma: " + sum);
        console.close();
    }
}
```

Reindirizzamento di input e output

- Usando il programma **Sum** (Sum.java) si inseriscono dei numeri da tastiera, che al termine non vengono memorizzati
  - ***per sommare una serie di numeri, bisogna digitarli tutti, ma non ne rimane traccia! Se si fa un errore…***
- Alternativa interessante e utile:
  il programma legge i numeri da un file
  - questo si può fare con il **reindirizzamento dell'input standard**

Reindirizzamento di input e output

- Il reindirizzamento dell'input standard, sia nei sistemi Unix sia nei sistemi Microsoft Windows, si indica con il carattere < seguito dal **nome del file da cui ricevere l'input**

```
java Sum < numeri.txt
```

- Si dice che il file **numeri.txt** viene ***collegato*** all'input standard
- Il programma non ha bisogno di alcuna istruzione particolare, semplicemente **System.in** non sarà più collegato alla tastiera ma al file specificato
  - La tastiera "non funziona più"… non si possono introdurre dati *anche* dalla tastiera

Reindirizzamento di input e output

- A volte è comodo anche il reindirizzamento dell'output
  - ad esempio, quando il programma produce molte righe di output, che altrimenti scorrono velocemente sullo schermo senza poter essere lette

```
java Sum > output.txt
```

    - Se il file output.txt non esiste viene creato, se esiste viene sovrascritto
    - Per non sovrascrivere il file di output bisogna utilizzare:

```
java Sum >> output.txt
```

- I due reindirizzamenti possono anche essere combinati

```
java Sum < numeri.txt > output.txt
```

Canalizzazioni ("pipes")

Canalizzazioni (pipes) un esempio

- Supponiamo di aver scritto la classe Split che scrive ciascuna parola ricevuta in ingresso su una riga di output separata

```
java Split < testo.txt
```

- Una elaborazione molto comune consiste nell'ordinare poi tali parole questa elaborazione è talmente comune che quasi tutti i sistemi operativi hanno un programma sort
- in alternativa, possiamo scrivere una classe Sort

Canalizzazioni (pipes)

- Per ottenere le parole di **testo.txt** una per riga e ordinate, abbiamo bisogno di un ***file temporaneo***

```
java Split < testo.txt > temp.txt
sort < temp.txt > testoOrdinato.txt
```

- Il file temporaneo **temp.txt** ***serve soltanto per memorizzare il risultato intermedio***, prodotto dal primo programma e utilizzato dal secondo
- Questa situazione è talmente comune che quasi tutti i sistemi operativi offrono un'alternativa

Canalizzazioni (pipes)

- Anziché utilizzare un file temporaneo per memorizzare l'output prodotto da un programma che deve servire da input per un altro programma, si usa una canalizzazione ("pipe")

```
java Split < testo.txt | java Sort > testoOrdinato.txt
```

- La canalizzazione può anche prolungarsi… ad esempio, possiamo eliminare eventuali parole ripetute da testoOrdinato.txt

```
java Split < testo.txt | java Sort | java Unique > out.txt
```

Formattazione di numeri

Formattazione di numeri

- Non sempre il formato standard per stampare numeri corrisponde ai nostri desideri

```
double total = 3.50;
final double TAX_RATE = 8.5; // aliquota
d'imposta in percentuale
double tax = total * TAX_RATE / 100;
System.out.println("Total: " + total);
System.out.println("Tax: " + tax);
```

- Ci piacerebbe di piu' visualizzare i numeri
  - Con due cifre decimali
  - Incolonnati

Total: 3.5
Tax: 0.2975

Total: 3.50
Tax: 0.30

Formattazione di numeri

- Java fornisce il metodo **printf**
  - Il primo parametro esplicito di **printf** è una **stringa di formato** che contiene dei caratteri da stampare e degli **specificatori di formato**
    - Ogni specificatore di formato comincia con il carattere %
  - I parametri successivi sono i **valori da visualizzare** secondo i formati specificati

```
System.out.printf("Total:%5.2f", total);
```

- Produce: `Total: 3.50`

spazio

Formattazione di numeri

- **%5.2f** è lo specificatore di formato: numero **in virgola mobile** (**%f**) formato da **5 caratteri** (compreso il punto!) con **due cifre dopo la virgola**
- Questo formato viene applicato alla variabile total, che è il secondo parametro del metodo

Tipi di formato e modificatori di formato

Codice	Tipo	Esempio
d	Intero decimale	123
x	Intero esadecimale	7B
o	Intero ottale	173
f	Virgola mobile	12.30
e	Virgola mobile esponenziale	1.23e+1
g	Virgola mobile generico (notazione esponenziale per i numeri molto grandi o molto piccoli)	12.3
s	Stringa	Tax:
n	Fine riga indipendente dalla piattaforma	

Codice	Significato	Esempio
-	Allinea a sinistra	1.23 seguito da spazi
0	Mostra gli zeri iniziali	001..23
+	Mostra il segno più per numeri positivi	+1.23
(	Racchiude tra parentesi i numeri negativi	(1.23)
,	Mostra il separatore di migliaia	12,300
^	Usa lettere maiuscole	12.3E+1