Input/Output

Redirecting, scomposizione dell'input, uso di file

Lettura di una serie di numeri:

- Un altro esempio di "ciclo e mezzo". Schema:

```
boolean finito = false;
while (!finito) {
  String input = legge un dato;
  if (i dati sono finiti)
    finito = true;
  else {
    elabora il dato
}
```


Un programma che analizza una serie di valori

- Definiamo una classe **`DataSet`** i cui oggetti sono possono acquisire dati tramite un metodo **`add(`***dato***`)`** e restituire in ogni momento la media o il massimo dei dati inseriti fino a quel momento
- Altri metodi:
  - getAverage
  - getMaximum
- Vedere implementazione nel codice allegato (DataSet.java)

Inserimento dati e test

- Vediamo un programma che usa lo schema precedente per prendere una serie di dati e inserirli in un oggetto della classe DataSet
- Poi chiamiamo i metodi per la media e il massimo
- Consultare il codice allegato (InputTestLoop.java)

Inserimento dati: quando i dati sono tanti

- È molto noioso e ripetitivo inserire dati in maniera interattiva se i dati sono molti
- Sarebbe utile poter inserire tutti i dati che servono in un file di testo e poi fare in modo che il programma li prenda da lì
- Un modo molto semplice per fare questo è quello di utilizzare la lettura di dati da console e poi, in fase di esecuzione del programma, **reindirizzare l'input da un file**
- Vedi codice allegato: (InputTestLoopRedirect.java)

Chiusura dell'input

- Per poter utilizzare il programma InputTestLoopRedirect occorre lanciarlo da console (prompt dei comandi Dos o shell di Linux)

```
prompt#> java InputTestLoopRedirect
Inserisci un valore numerico. Chiudi l'input per terminare
16
Inserisci un valore numerico. Chiudi l'input per terminare
32
Inserisci un valore numerico. Chiudi l'input per terminare
^Z
Media dei dati = 24.0
Valore massimo = 32.0
```

Chiusura dell'input

- In Dos/Windows per chiudere l'input da tastiera basta premere Ctrl-z
- In Linux/Unix invece si usa Ctrl-d
- Un'alternativa è scrivere un file di testo in cui in ogni riga mettiamo un valore numerico
- Poi redirigiamo l'input del programma su questo file, invece che sulla tastiera:
- Ad esempio supponiamo di avere scritto i dati in un file "datiInputTestLoopRedirect.txt"

Redirecting dell'input

```
#> java InputTestLoopRedirect < datiInputTestLoopRedirect.txt
Inserisci un valore numerico. Chiudi l'input per terminare
Inserisci un valore numerico. Chiudi l'input per terminare
Inserisci un valore numerico. Chiudi l'input per terminare
Media dei dati = 24.0
Valore massimo = 32.0
```

- Il file di testo deve contenere esclusivamente le seguenti due righe:

```
16
32
```

- La chiusura dell'input viene segnalata quando viene incontrato il carattere di EOF (fine file)

Scomposizione di stringhe

- E se volessimo, per comodità, inserire i valori nel file di testo anche su una stessa riga?
- Ad esempio:

```
13  343.54  100.09  25
1.8  12  3
33
```

- Il metodo **`readLine()`** della classe **`BufferedReader`** legge una intera riga per volta
- Se proviamo a fare il parsing di una riga con diversi numeri separati da spazi bianchi il metodo **`parseDouble`** solleva un'eccezione

Scomposizione di stringhe

- Ci viene in aiuto la classe **`java.util.StringTokenizer`**
- Un oggetto di questa classe va costruito passandogli una certa stringa
- Dopodiché è possibile utilizzare la coppia di metodi
  - **`hasMoreToken()`**
  - **`nextToken()`**
- per prendere un token, cioè un insieme di caratteri contigui (non separati da spazi tab o newline), per volta fino all'esaurimento della stringa stessa
- Consultare il codice allegato InputTestLoopToken.java

Scomposizione

- Se si prova a chiamare **`nextToken()`** senza che ci siano token viene sollevata un'eccezione: si deve sempre controllare prima con **`hasMoreToken()`**:

```
...
StringTokenizer tokenizer = new
  StringTokenizer(input);
while (tokenizer.hasMoreTokens()) {
   String token = tokenizer.nextToken();
   double x = Double.parseDouble(token);
   data.add(x);
}
...
```


Redirecting dell'output

- Supponiamo di scrivere il file di testo di prima:
- Supponiamo di chiamarlo datiInputTestLoopToken.txt
- Lanciando il programma con il redirecting dell'input si ottengono comunque quattro stringhe in output che corrispondono alle richieste dei dati (utili solo nel caso che non si usi il redirect)

Redirecting dell'output

```
#> java InputTestLoopToken < datiInputTestLoopToken.txt
Inserisci un valore numerico. Chiudi l'input per terminare
Inserisci un valore numerico. Chiudi l'input per terminare
Inserisci un valore numerico. Chiudi l'input per terminare
Inserisci un valore numerico. Chiudi l'input per terminare
Media dei dati = 66.42875000000001
Valore massimo = 343.54
```

- Tutto questo output può essere anch'esso reindirizzato su un file di testo, ad esempio output.txt:

```
#> java InputTestLoopToken < datiInputTestLoopToken.txt > output.txt
```


Lettura e scrittura di file

- Naturalmente possiamo anche inserire esplicitamente nei nostri programmi la lettura di input da un file e la scrittura di output su un file
- Il tutto senza utilizzare il redirecting dello standard input o output
- Vediamo come è stato modellato il concetto di stream in Java e le varie classi per leggere/scrivere stream

Stream

- Uno stream è un flusso di entità
- Uno stream è di input se le entità fluiscono dall'esterno verso la nostra applicazione
- Uno stream è di output se le entità fluiscono dalla nostra applicazione verso l'esterno
- Generalmente le entità che scorrono in uno stream possono essere viste in due modi:
  - caratteri
  - byte

Stream

- Gli stream servono per memorizzare dati o per leggere dati precedentemente memorizzati
- Se i dati sono in formato testo allora sono rappresentati da caratteri, altrimenti da byte
- '1' '2' '3' '4' '5' rappresenta il numero 12345 in un **file di testo**
- 0 0 48 57 è una sequenza di 4 byte che rappresenta il numero 12345 (12345 = 48 * 256 + 57) in un **file binario**

Le classi da usare in java.io

- Gli oggetti delle classi **`FileReader`** e **`FileWriter`** rappresentano **file di testo** in input o in output
- Gli oggetti delle classi **`FileInputStream`** e **`FileOutputString`** rappresentano **file binari** in input o in output
- Alla creazione di uno qualsiasi di questi oggetti va passata al cotruttore una stringa contentente il path+nome del file da cercare

Apertura di file

```
FileReader reader = new
  FileReader("input.txt");
```

- Apre un file di testo in lettura

```
FileWriter writer = new
  FileWriter("output.txt");
```

- Apre un file di testo in scrittura

```
FileInputStream inputStream = new
  FileInputStream("input.dat");
```

- Apre un file binario in lettura

```
FileOutputStream outputStream = new
  FileOutputStream("output.dat");
```

- Apre un file binario in scrittura

Lettura

- Sia **`FileInputStream`** che **`FileReader`** hanno un metodo **`read()`** che serve per leggere un byte o un carattere (rispettivamente) alla volta
- In ogni caso entrambi i metodi restituiscono un int:
  - se il valore restituito è –1 allora è stata raggiunta la fine del file
  - Se il valore restituito è non negativo allora si può fare il casting a **`byte`** o a **`char`** (rispettivamente) per ottenere il valore letto

Lettura

- File di testo:

```
int next = reader.read();
char c;
if (next != -1)
  c = (char) next; // c è il carattere letto
else fine file
```

- File binario:

```
int next = inputStream.read();
byte b;
if (next != -1)
  b = (byte) next; // b è il byte letto
else fine file
```


Scrittura

- File di testo

```
int char = ...;
writer.write(c);
```

- File binario:

```
byte b = ...;
outputStream.write(b);
```


Chiusura

- Ogni file aperto in qualsiasi modalità va chiuso quando il programma ha finito di operare su di esso:

```
riferimentoAlFile.close();
```


Agevolazioni per i file di testo

- Leggere o scrivere un carattere per volta nei file di testo può risultare scomodo
- Possiamo incapsularli in oggetti più sofisticati che realizzano un interfaccia a linee
- È quello che facciamo sempre, ad esempio, quando leggiamo linee di testo dallo standard input
- La classe da usare per i file di testo in lettura la conosciamo già: è **`BufferedReader`**

Agevolazioni per i file di testo

```
FileReader file = new
  FileReader("input.txt");
BufferedReader in = new
  BufferedReader(file);
String inputLine = in.readLine();
```

- Già lo conosciamo: otteniamo una linea di testo con il metodo **`readLine()`**

Agevolazioni per i file di testo

```
FileWriter file = new
  FileWriter("output.txt");
PrintWriter out = new PrintWriter(file);
out.println(29.25);
out.println(new Rectangle(5,10,20,30));
out.println("Hello, World!");
```

- La classe **`PrintWriter`** è molto simile alla classe **`PrintStream`** che già conosciamo (è la classe a cui appartiene **`System.out`**)
- Il metodo **`println`** si usa nel modo che conosciamo

Ricerca di un file nelle cartelle

- Può essere utile, quando si vuole aprire un file in input, presentare all'utente la classica finestra di selezione di un file che gli permette di navigare tra le sue cartelle
- Ci viene in aiuto la classe **`javax.swing.JFileChooser`**

Ricerca di un file nelle cartelle

```
JFileChooser chooser = new JFileChooser();
FileReader in = null;
if (chooser.showOpenDialog(null) ==
            JFileChooser.APPROVE_OPTION) {
  File selectedFile =
                chooser.getSelectedFile();
  in = new FileReader(selectedFile);
}
```

- Vedere il codice allegato MyTextCopy.java

Esercizio

- Modificare il programma MyTextCopy in modo che possa copiare file binari qualsiasi

Esercizio

- Scrivere un programma che prenda in input un certo numero n di numeri, li memorizzi in un array, poi ordini l'array e stampi l'array ordinato in output
- Suggerimento: usare due cicli for annidati ed effettuare uno scambio fra gli elementi a[i] e a[j] se a[i] > a[j] (Bubble Sort)

Esercizio

- Leggere da input una serie di numeri x_i e memorizzarli in un array
- Calcolare la media μ dei valori
- Cercare il primo elemento dell'array minore o uguale alla media
- Calcolare la deviazione standard (attenti agli indici: nella formula partono da 1!)

$$\sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n-1}}$$

Esercizio

- Scrivere una classe **`FactorGenerator`** che serva a calcolare tutti i fattori di un certo numero dato
- Ad esempio per il numero 150 deve dare i fattori: 2 3 5 5
- Mettere a disposizione i metodi **`nextFactor()`** e **`hasMoreFactor()`**

Esercizio

- Scrivere una classe **`PrimeGenerator`** che permetta di calcolare tutti i numeri primi minori di un certo numero dato
- Ad esempio per 20 deve stampare: 2 3 5 7 11 13 17 19
- Mettere a disposizione il metodo **`nextPrime()`**

Esercizio

- Determinare il valore di e elevato a un certo numero "reale" x utilizzando la seguente serie di potenze e fermando il calcolo quando l'addendo attuale della somma diventa minore di una certa soglia fissata:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Esercizio

- Un array bidimensionale si dichiara nel seguente modo

```
int a[][] = new int[3][3];
```

- Rappresenta una tabella 3 per 3 di interi
- Il generico elemento in riga i e colonna j si indica con **`a[i][j]`**
- Le righe e le colonne vengono numerate a partire da 0
- Continua....

Esercizio cont'd

- Scrivere un programma che utilizzi un array bidimensionale 3 per 3 come una scacchiera per il gioco del tris
- Il programma deve permettere a due utenti di specificare le proprie mosse alternativamente
- Il programma deve controllare se un utente ha fatto tris e proclamarlo vincitore
- Il programma non deve accettare mosse errate
- Continua...

Esercizio cont'd

- Scrivere una classe **`Scacchiera`** che gestisca la scacchiera con metodi per eseguire le mosse corrette, visualizzare la scacchiera corrente e per controllare se ci sono vincitori
- Scrivere poi una classe **`Tris`** che utilizzi la **`Scacchiera`** per far giocare i due giocatori

Esercizio

- Il gioco del Nim
- Ci sono n biglie in un mucchio
- Due giocatori prelevano a turno biglie dal mucchio
- Ad ogni mossa un giocatore deve prendere almeno una biglia e non può prenderne più della metà di quelle che sono attualmente nel mucchio
- Perde chi è costretto a prendere l'ultima biglia
- Continua...

Esercizio cont'd

- Scrivere un programma che giochi contro l'utente al gioco del Nim
- Utilizzare la classe **`java.util.Random`** (vedere API) per generare numeri pseudocasuali
- Il programma deve seguire i seguenti passi:

1. Genera un numero casuale tra 10 e 100 che indica il numero iniziale di biglie
2. Genera un numero casuale tra 0 e 1 per decidere chi dei due giocatori inizia

Continua...

Esercizio cont'd

3. Genera un numero casuale tra 0 e 1 per decidere se giocare in maniera stupida o utilizzare la strategia vincente

- Se gioca in maniera stupida ad ogni sua mossa il programma prenderà un numero casuale di biglie fra 1 e il massimo di biglie prelevabili
- La strategia vincente consiste in questo: ad ogni mossa il computer deve prelevare un numero di biglie tali che quelle che restano siano una potenza di due meno 1 (cioè 3, 6, 15, 31 o 63)
- Continua...

Esercizio cont'd

- È sempre una mossa valida tranne quando il numero di biglie disponibili è proprio una potenza di due meno 1
- Se è questo il caso allora il computer toglie un numero casuale di biglie purché ammissibile
- Se il computer inizia la partita e decide di usare la strategia vincente risulterà imbattibile
- Negli altri casi si può vincere