Programmazione I

A.A. 2002-03

Funzioni

(Lezione XIX , Parte I)

Categorie di memorizzazione

Prof. Giovanni Gallo
Dr. Gianluca Cincotti
Dipartimento di Matematica e Informatica
Università di Catania
e-mail : {gallo, cincotti}@dmi.unict.it

Categorie di memorizzazione

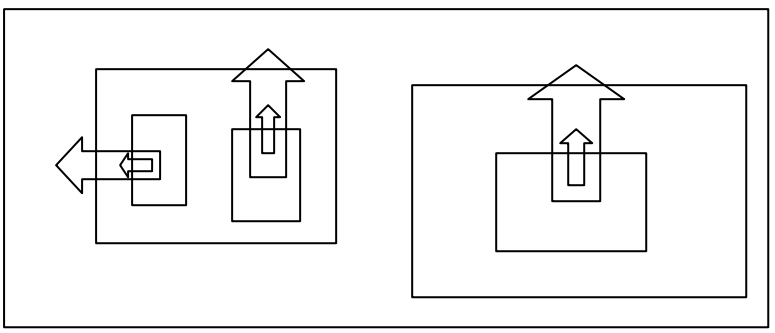
- Caratteristiche delle variabili:
  - *Visibilità* (o "scope")
    - È la porzione di programma in cui la variabile può essere referenziata.
  - *Ciclo di vita* (o durata)
    - È il periodo durante il quale la variabile esiste in memoria.

Variabili locali

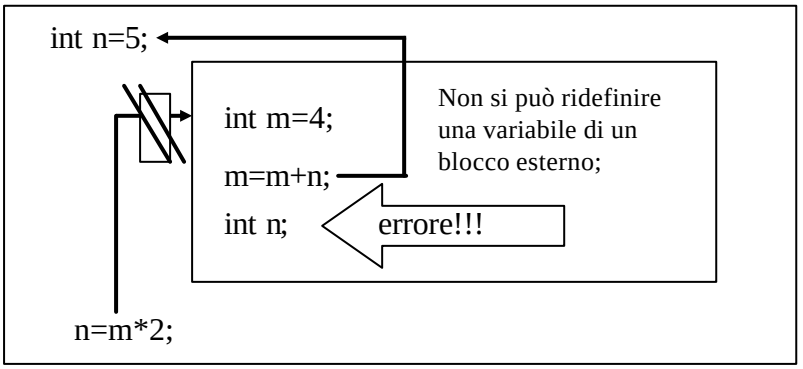
- In Java un qualunque *blocco* può contenere dichiarazioni di variabili al suo interno, tali variabili si dicono *locali* al blocco.
- Una variabile locale ad un blocco
  - è *visibile*:
    - nel blocco in cui è definita, e
    - in ogni blocco contenuto nel blocco in cui la variabile è definita.
  - ha il seguente *ciclo di vita*:
    - viene creata ogni volta che si entra nel blocco e viene distrutta quando si esce da tale blocco.

Annidamento di blocchi

- In pratica ogni blocco è una "scatola" semitrasparente:
  - da dentro vedo le variabili che sono fuori, ma non posso vedere le variabili dentro i sottoblocchi.



Esempio



Il blocco interno "vede" n;
Il blocco esterno NON vede m;

Es. 01

Fine

Programmazione I

A.A. 2002-03

Funzioni

(Lezione XIX , Parte II)

Dichiarazione di funzioni

Prof. Giovanni Gallo
Dr. Gianluca Cincotti
Dipartimento di Matematica e Informatica
Università di Catania
e-mail : {gallo, cincotti}@dmi.unict.it

Il concetto di funzione

- Una ***funzione*** è un *"blocco di codice"* che prende un input e fornisce un output.
  - Nell'ambito dei linguaggi di programmazione la definizione di *funzione* è assimilabile a quella di:
    - funzione matematica, ovvero,
    - di "black box" (scatola nera).

Funzioni, procedure e metodi

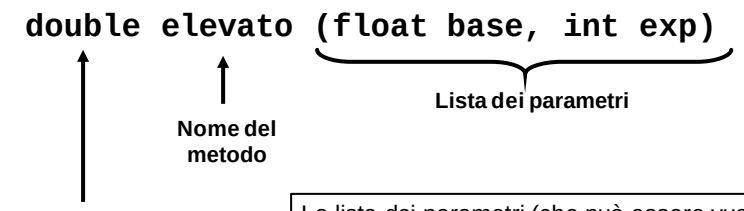
- Una *procedura* è una particolare funzione che non restituisce alcun valore.
  - Il concetto di procedura e/o funzione è alla base del paradigma della programmazione *imperativa* (o procedurale).
    - FORTRAN, Pascal, C
  - Nel paradigma della programmazione *orientata agli oggetti (OOP)* le procedure e/o funzioni prendono il nome di ***metodi***.
    - SmallTalk, C++, Java

Dichiarazione di funzione

```
TipoRestituito  NomeFunzione  ( Par1, Par2, Par3, …)
        {
            Corpo della funzione;
        }
```

Dichiarazione di funzione (cont.)

- La dichiarazione di una funzione inizia con l'***intestazione:***



La lista dei parametri (che può essere vuota) specifica il tipo e il nome di ciascun parametro

I parametri nell'intestazione di una funzione si dicono parametri formali.

Dichiarazione di funzione (cont.)

- L'intestazione di una funzione è seguita dal ***corpo***:

```
double elevato (float base, int exp)
   {
     // exp deve essere >= 0 !
     double risultato = 1.0;

     for (; exp>0; exp--)  risultato *=base;
     return risultato;
   }
```

Chiamata di funzione

```
{
    …
    NomeFunzione  ( Par1, Par2, Par3, …);
    …
}
```


Un mistero svelato …

- Quante volte ci siamo chiesti cosa vuol dire:

  public static void main (String [ ] args)

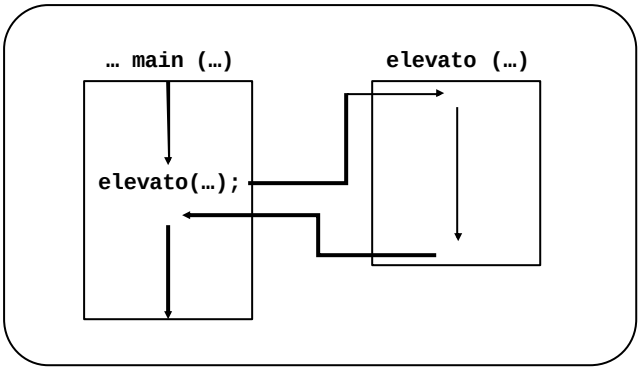
  { … }

- Si tratta dell'intestazione di una *funzione* di nome "main" che presenta un parametro di tipo array di String, chiamato args.
  - Utile per passare informazioni e input alla classe quando la si chiama dal prompt di linea.

Flusso di controllo per le funzioni

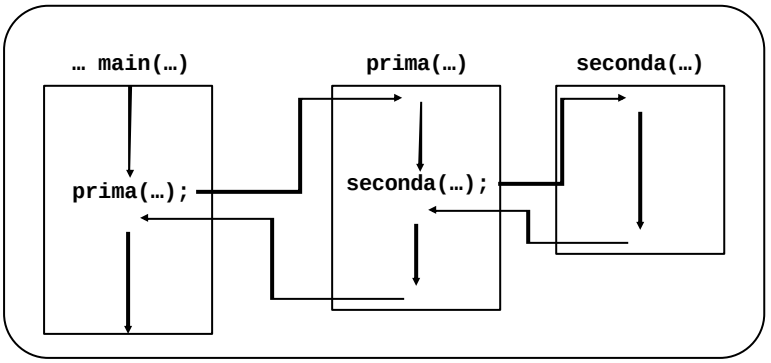
- La *dichiarazione* di una funzione specifica il codice che verrà eseguito quando la funzione verrà invocata.
  - L'esecuzione di una funzione può comprendere la restituzione di un valore.
- Quando una funzione viene invocata il *flusso di controllo* passa a tale funzione ed il codice relativo viene eseguito.
  - Quindi, il flusso ritorna al punto da cui è partita la chiamata e continua da lì.

Flusso di controllo per le funzioni (cont.)



- L'esecuzione del main :
  - si interrompe al momento della chiamata di una funzione;
    - si esegue la funzione invocata;
  - e prosegue dalla prima istruzione seguente la chiamata.

Flusso di controllo per le funzioni (cont.)



- Una funzione chiamata può a sua volta chiamarne un'altra!

Es. 02

Dati locali alle funzioni

- È possibile dichiarare variabili *locali* ad una funzione.
  - Visibilità :
    - Una variabile locale è visibile solo all'interno del corpo della funzione dove essa è dichiarata.
  - Ciclo di vita:
    - Una variabile locale viene creata ogni volta che la funzione viene invocata e viene distrutta quando la funzione ha finito di essere eseguita (*variabili automatiche*).
- I parametri sono da considerarsi variabili *locali*.

Esempio

```
double elevato (float base, int exp)
   {
     // exp deve essere >= 0 !
     double risultato = 1.0;

     for (; exp>0; exp--)  risultato *=base;
     return risultato;
   }
```

L'espressione che produce il valore che viene riportato deve essere consistente con il tipo di valore restituito dal metodo.

`risultato` è una variabile *locale*.
Viene creata ogni volta che il metodo viene invocato e viene distrutta quando il metodo ha finito di essere eseguito.

L'output delle funzioni : l'istruzione "return"

- Il tipo restituito da una funzione indica il tipo del valore che la funzione restituisce al chiamante.
  - Una funzione che non restituisce alcun valore è di tipo **`void`**.
- L'istruzione **`return`** all'interno del corpo di una funzione specifica il valore che verrà restituito dalla stessa.
  - L'espressione specificata nel return deve essere conforme alla specifica di tipo dichiarata nell'intestazione.

Funzioni che restituiscono un valore

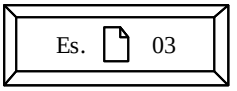
- Se una funzione restituisce *un* valore:
  - occorre specificare nell'intestazione di che "tipo" è il valore restituito, e
  - nel corpo della funzione deve essere presente *almeno* un'istruzione **return** che specifica qual è il valore da restituire.
    - essa può essere presente ovunque all'interno del corpo.

```
double Math.sqrt (double x)
      {   …
          return unValore;
      }
```


Funzioni che non restituiscono valori

- Se una funzione *non* restituisce alcun valore:
  - occorre specificare nell'intestazione il tipo "void", e
  - nel corpo della funzione non è necessaria alcuna istruzione **return**.
    - nel qual caso, l'esecuzione della funzione termina alla fine del corpo.

```
void  visualizza (int x)
      {   …
          System.out.print (2*x);
      }
```



Fine