Programmazione I

A.A. 2002-03

Funzioni

(Lezione XX , Parte I)

Passaggio dei parametri

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

Input delle funzioni : parametri

- Una funzione può prendere in ingresso zero o più input, detti parametri.
  - Se non si prende alcun input occorre comunque indicarlo con delle parentesi vuote "( )"
    - sia nell'intestazione della funzione,
    - sia nella chiamata della stessa.
  - Se invece sono presenti dei parametri occorre dichiarare il tipo per ognuno di essi.
    - I parametri presenti sono delle vere e proprie variabili *locali.*

Parametri formali ed attuali

- I parametri elencati
  - nell'*intestazione* di una funzione si dicono ***parametri formali***;
  - nella *chiamata* di una funzione si dicono ***parametri attuali***.
- La corrispondenza tra i tipi dei parametri *formali* ed *attuali* è necessaria !!!
  - Il compilatore fa un controllo sul tipo dei parametri che la funzione attende e quelli che passiamo nella chiamata.
    - Se tale corrispondenza non è perfetta il compilatore tenta in automatico le conversioni di tipo già viste in precedenza.
      - Se il compilatore non riesce si genera un errore di compilazione!

Il passaggio dei parametri

- Molti linguaggi di programmazione prevedono l'impiego delle seguenti modalità di passaggio dei parametri:
  - per *valore* (o copia),
  - per *riferimento* (o indirizzo).
- In generale, è il programmatore a scegliere la modalità con cui passare ogni singolo parametro.
  - In Java il passaggio dei parametri avviene solamente per *valore*.
    - In realtà, il compilatore Java utilizza anche, in maniera del tutto trasparente al programmatore, il passaggio per *riferimento*.

Passaggio per valore di tipi di dato primitivi

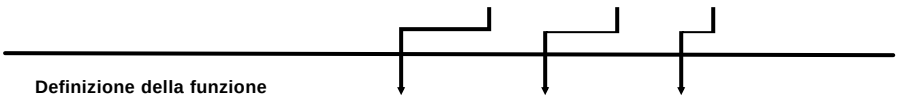
- Cosa accade in Java quando si passa un tipo di dato *primitivo* ?
  - Il parametro è passato per *valore* !
    - I parametri *attuali* all'atto della chiamata sono **copiati** nei parametri *formali.*
      - L'esecuzione della funzione avviene utilizzando solo una *copia* del parametro attuale e dunque loriginale non può essere alterato in alcuna maniera.

Passaggio per valore di tipi di dato primitivi (cont.)

- In Java, ad ogni chiamata di una funzione, i parametri *attuali* all'atto della chiamata sono **copiati** nei parametri *formali.*

Chiamata della funzione

```
int M = min_max (saldo, totale, 'M');
```

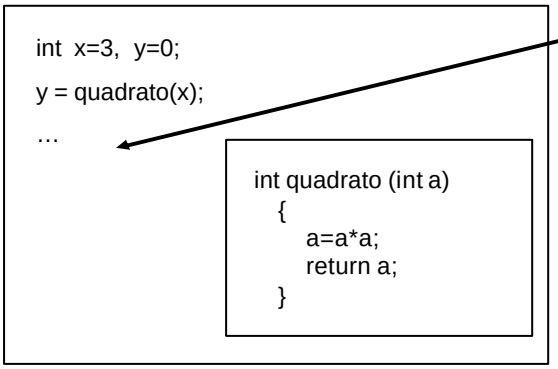


Definizione della funzione

```
int min_max (float x, float y, char op)
    {
      if (op=='m')  return  (int) Math.min(x,y);
              else  return  (int) Math.max(x,y);
    }
```

Es. 04

Passaggio per valore di tipi di dato primitivi (cont.)



Quanto valgono qui le variabili x e y?

Si ha

x=3 e y=9

La funzione "quadrato" ha utilizzato solo una COPIA della variabile x senza alterare l'originale.

Es. 05

Es. 06

Passaggio per valore di tipi di dato non primitivi

- Cosa accade in Java quando si passa un tipo di dato *non primitivo* (oggetto) ?
  - Il parametro è passato sempre per *valore* !
    - Questa volta il parametro è però solamente un *riferimento* all'oggetto. La *copia* che viene effettuata è del riferimento e non dell'oggetto vero e proprio.
      - Durante l'esecuzione della funzione, attraverso la *copia* del riferimento all'oggetto si potrà accedere, e dunque alterare, i dati contenuti nell'oggetto originale (si pensi agli array !).

Passaggio per valore di tipi di dato non primitivi (cont.)

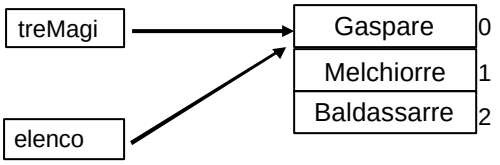
```
String [ ] treMagi = { "Gasparre",
                       "Melchiorre",
                       "Baldassarre" };

testaCoda (treMagi);

System.out.println ( treMagi[0] );    //cosa stamperà?
```

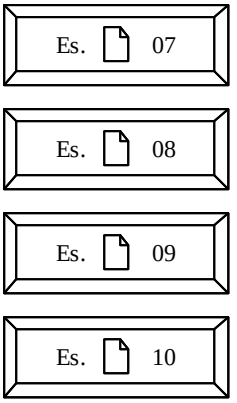
Baldassarre

```
void testaCoda  (String [ ] elenco)
   {
     String  temp = elenco[0];
     elenco[0] = elenco[elenco.length-1];
     elenco[elenco.length-1] = temp;
   }
```

Perché?

Passaggio per valore di tipi di dato non primitivi (cont.)





Alla funzione è stato passato il riferimento "treMagi" all'array, ossia l'indirizzo in RAM dove iniziano le locazioni contigue dell'array.

In seguito al passaggio per valore, la variabile "elenco" conterrà lo stesso *indirizzo* d'inizio dei dati dell'array.

Un po' più complesso ma…

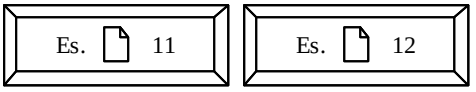
- … molto più efficiente!
  - Nel passaggio per valore occorre riservare lo *spazio* per i parametri da copiare ed è necessario del *tempo* per effettuare la copia vera e propria.
    - Se i parametri sono di tipo primitivo la copia non richiede tempo o spazio eccessivi.
    - In caso di array o di oggetti complessi invece il tempo e lo spazio richiesti per la copia potrebbero essere spropositati.
      - Se si desidera che l'array di partenza non venga alterato ?
        - È un caso raro, tuttavia occorrerebbe ricopiare l'array in un nuovo array e quindi passare il "nome" della copia.

Argomenti dalla linea di comando

- L'intestazione della funzione **`main`** indica la presenza di un parametro *"args"* di tipo array di String.
  - I valori del parametro provengono dalla riga di comando e vengono passati quando viene chiamato l'interprete:

```
> java Progr Uno Due Tre
```

  - All'atto dell'invocazione l'interprete passa al metodo **`main`** un array di tre oggetti String contenente le stringhe inserite nella linea di comando.
    - Le tre stringhe passate come parametro sono referenziabili con args[0], args[1], args[2].



Regole di "buona" programmazione

- Ogni funzione deve limitarsi ad eseguire un compito singolo e ben definito.
  - Il nome della funzione deve esprimere in modo chiaro ed inequivocabile tale compito.
    - Anche i nomi dei parametri devono essere significativi.
- Una funzione dovrebbe essere relativamente piccola (non più di una pagina).
  - Le funzioni brevi favoriscono il *riutilizzo del software* e sono più facili da testare, modificare e comprendere rispetto a quelli di grandi dimensioni.

Decomposizione di funzioni

- Una funzione potenzialmente complessa dovrebbe venire *decomposta* in diverse sotto-funzioni più piccole.
  - Una funzione che richiede un gran numero di parametri probabilmente deve essere decomposta.
- Una funzione può invocare al suo interno altre funzioni.
  - Una funzione può contenere la chiamata a se stessa (*ricorsione*) !
  - Le funzioni non possono essere annidate, cioè una funzione non può contenere la definizione di un'altra funzione !

Fine

Programmazione I

A.A. 2002-03

Funzioni

(Lezione XX , Parte II)

Overloading delle funzioni

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

Overloading delle funzioni

Se provassi a inserire in una classe entrambe le funzioni:

```
double doppio (int x) {return 2*x;}
double doppio (int x) {return x+x;}
```

Errore : non si può definire due volte la stessa funzione!

PERO'...

```
double doppio (int x) {return 2*x;}

double doppio (double x) {return x+x;}
```

non darebbe nessun problema: una funzione può avere "forme" diverse! Questo si chiama overloading, e si spiega ricordando che il tipo dei parametri è parte integrante della "identità" della funzione.

Segnatura delle funzioni

- L'***overloading di funzioni*** avviene quando si usa lo stesso nome per indicare funzioni diverse.
- La ***segnatura*** (o firma) di ciascuna funzione convolta dal processo di *overloading* deve essere **unica.**
  - La *segnatura* include il numero, tipo e l'ordine dei parametri.
  - Il tipo restituito dalla funzione non fa parte della *segnatura* ma del *prototipo.*
- Il compilatore deve essere in grado di determinare quale versione della funzione sta per essere invocata analizzando i parametri presenti nella chiamata.

Esempio

versione 1

```
float prova (int x)
{
   return x + .375;
}
```

versione 2

```
float prova (int x, float y)
{
   return x*y;
}
```

invocazione

```
result = prova (25, 4.32)
```


Overloading di "print"

- Il metodo `println` è sovraccaricato:

```
println (String s)
println (int i)
println (double d)
```
etc.

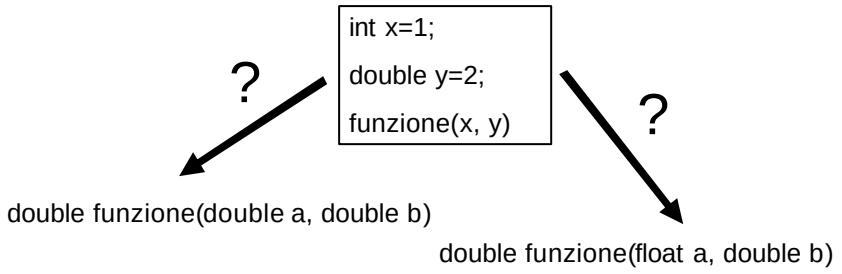
- E si possono usare le diverse versioni del metodo `println`:

```
System.out.println ("Il totale è:");
System.out.println (totale);
```

Risoluzione dell'overloading

- Il compilatore segue delle regole precise per la determinazione della funzione della famiglia di *overloading*.
- In particolare:
  - verifica se esiste un accoppiamento esatto, ed in caso contrario
  - tenta le conversioni automatiche di tipo.

Problema!



Il sistema preferirà l'accoppiamento che rende la traduzione più semplice…per lui!
Quindi la definizione con il float…

Meglio evitare queste costruzioni ambigue!!!

Es. 13

Fine