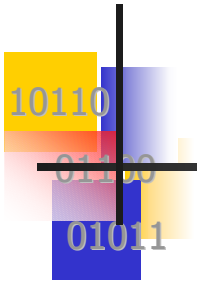


Concetto di Funzione e Procedura
METODI in Java

Metodi e Sottoprogrammi

- Mentre in Java tramite le **classi** e gli **oggetti** è possibile definire nuovi tipi di dati, tramite i **metodi** è possibile definire nuove operazioni sulla base delle operazioni predefinite (**istruzioni**).

- I metodi in Java realizzano **sottoprogrammi**: programmi che sono definiti per scopi particolari ma che non possono "vivere" da soli.

- Essi hanno
  - un **tipo**,
  - un **nome**,
  - dei **parametri di input** dove sono contenuti i dati di ingresso,
  - sono composti da un **insieme di istruzioni**, e
  - dei **parametri di output** dove sono contenuti i risultati.

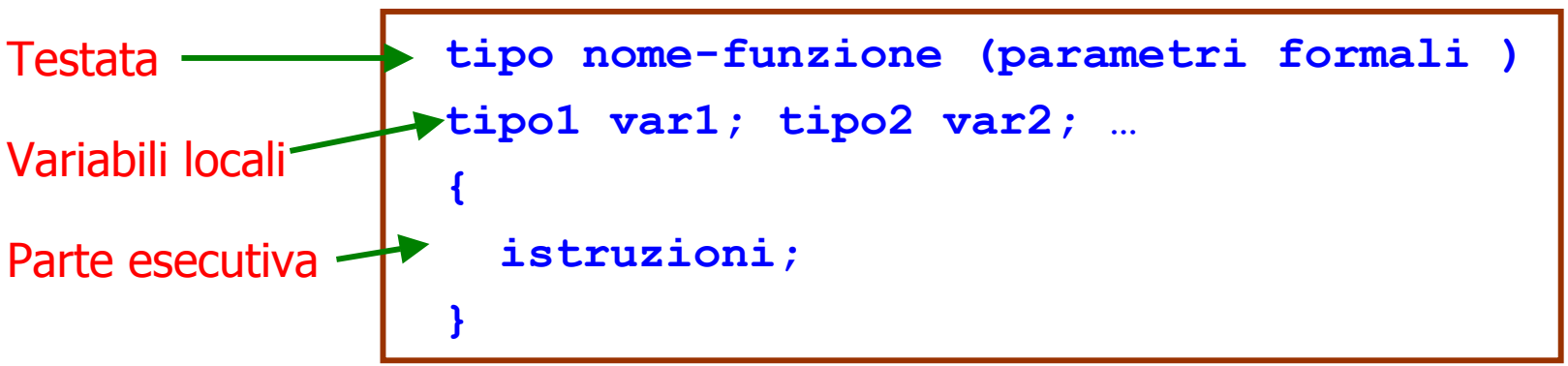
Funzioni e Procedure

- Nei linguaggi imperativi di alto livello ci sono due tipi di sottoprogrammi:
  - **Funzioni** e
  - **Procedure**
- Nei linguaggi object-oriented (come Java) le funzioni e le procedure sono dette **metodi**.
- La programmazione tramite funzioni e procedure ha notevoli benefici:
  - codice non ripetuto,
  - modularità,
  - leggibilità,
  - "debuggabilità".

Funzioni : definizione

- Le funzioni nei linguaggi imperativi hanno punti in comune con le funzioni matematiche, ma non sono esattamente la stessa cosa (vedi gli effetti collaterali).

- **Definizione delle funzioni**



- I parametri rappresentano il dominio della funzione, mentre il risultato (ritornato tramite l'istruzione `return espr`) ne rappresenta il co-dominio.

Procedure : definizione

- Le procedure sono simili alle funzioni, ma non hanno il compito di calcolare un risultato e ritornarlo come un valore. Lo scopo di una procedura non è quindi di produrre un valore, ma di modificare lo stato, cioè il contenuto della memoria di un programma.

- Esempi di procedure sono la stampa di un insieme di dati, l'inserimento di un elemento in una struttura dati, etc.
- **Definizione delle procedure**

Testata

Variabili locali

Parte esecutiva

```
void nome-procedura(parametri formali)
tipo1 var1; tipo2 var2; …
{
  istruzioni;
}
```

- **Non c'è un risultato da ritornare** (non si usa l'istruzione `return espr`).

Funzioni : esecuzione

Chiamata di funzioni

- Una volta dichiarate, le funzioni si possono utilizzare nei programmi tramite la cosiddetta chiamata di funzione.
- Ad esempio, se si è dichiarata la funzione `int max(int a, b, c)`
- Si può chiamare nelle istruzioni
  - `y = max(1, 2, 3);`
  - `y = max(x, y, z);`
  - `y = max(10, x, 45);`
- Ad `y` sarà assegnato il valore argomento della `return`.
- I parametri della chiamata, se esistono, sono detti **parametri attuali** e devono essere dello stesso tipo dei parametri formali.

Procedure : esecuzione

Chiamata di procedure

- Una volta dichiarate, le procedure si possono utilizzare nei programmi tramite la cosiddetta **chiamata di procedura**.

- Le procedure lavorano su variabili globali e modificano il loro contenuto, mentre le funzioni lavorano su variabili locali e ritornano un risultato.

- Ad esempio, se si è dichiarata la procedura

```
void stampa(char a, b)
```

- Si può chiamare/eseguire tramite le istruzioni

```
stampa('A','B');
stampa(x, y);
```

- **Non c'è assegnamento** perchè le procedura non ritorna un valore !

Metodi : definizione

- Il concetto di funzione e procedura nei linguaggi orientati agli oggetti è stato realizzato ed esteso tramite i **metodi**.

- **Definizione di un metodo**

Intestazione

Variabili locali

Parte esecutiva o corpo

```
spec tipo nome (parametri formali)
tipo1 var1; tipo2 var1; …
{
   istruzioni;
   return espressione;
}
```

- I parametri rappresentano il dominio del metodo, mentre il risultato (ritornato tramite l'istruzione `return`) ne rappresenta il co-dominio.

Metodi : intestazione

L'intestazione contiene

- Uno specificatore di accesso **`public`**, **`private`**

  specifica quali altri metodi possono chiamare/eseguire il metodo

- Il tipo di dati restituito **`int`**, **`float`**, **...**, **`void`**

  indica il tipo di dato del valore che calcola il metodo (deve essere uguale alla espressione indicata nell'istruzione **`return`**). Senza espressione il tipo è **`void`**.

- Il nome del metodo **`max`**, **`stampa`**, **`dividi`**, **...**

  è l'identificatore da usare quando si chiama/esegue il metodo.

- Un elenco dei parametri formali

  rappresentano l'input del metodo e si indicano con tipo e nome separati da virgole:

```
public float dividi(float a, int b);
```

Metodi : overloading

- La ***firma*** o ***segnatura*** indica l'insieme del nome e dei parametri di un metodo, essa deve essere unica.
- In Java possono esistere **due metodi con lo stesso nome** a patto che abbiano parametri formali di numero e/o tipo differente.
- Ad esempio:

```
void stampa (char x)
void stampa (int x)
void stampa (int x, int y)
```

- In questo caso di parla di **overloading** o **sovraccarico**.
- Il tipo restituito non contribuisce alla firma del metodo. Ad esempio `int somma(int x, int y)` e `float somma(int x, int y)` hanno la stessa firma è quindi non sono ammissibili.
- In questo caso occorre usare un nome diverso o parametri di tipo diverso.

Metodi : Chiamata o Invocazione

- Nell'invocazione di un metodo in Java i parametri attuali devono essere compatibili con i parametri formali per numero, ordine e tipo.

```
public static int power (int b, int n)
{
   int i, p;
   p=1;
   for (i=1; i<=n; ++i)
        p=p*b;
   return p;
}
```

parametri formali

variabili locali

valore del metodo

```
public void static main (…)
{
   int potenza, esp, base;
   ……
   potenza=power(base, esp);
   ……
}
```

parametri attuali

Metodi : visibilità delle variabili

- Ogni dichiarazione di un dato ha una sua **visibilità** o **scope**. La visibilità ci indica l'ambiente dove il nome dichiarato potrà essere usato.
- **I parametri formali di un metodo possono essere usati solo nell'ambito del metodo** e
- Anche le variabili locali al metodo possono essere usate **solo nel metodo stesso**.
- **Non si possono avere variabili con lo stesso nome all'interno di un metodo**. Tuttavia si possono avere variabili con lo stesso nome all'interno di metodi diversi.
- Si noti che **il nome del parametro formale non deve essere necessariamente uguale a quello del parametro attuale.**

Metodi : return

- Per ritornare i valori si usa l'istruzione **`return`** seguita da una espressione.

```
return ris;
```

- La **`return`** deve tornare un valore dello stesso tipo o di un tipo compatibile con quello del metodo.

- Se l'espressione non c'è come argomento della **`return`** questa serve solo a far terminare un metodo.

```
return;
```

- Se non viene eseguita una **`return`** o viene eseguita senza argomenti, nel metodo chiamante non bisogna eseguire un assegnamento ma solo l'invocazione.

Metodi : passaggio dei parametri

Passaggio dei parametri nell'invocazione di un metodo

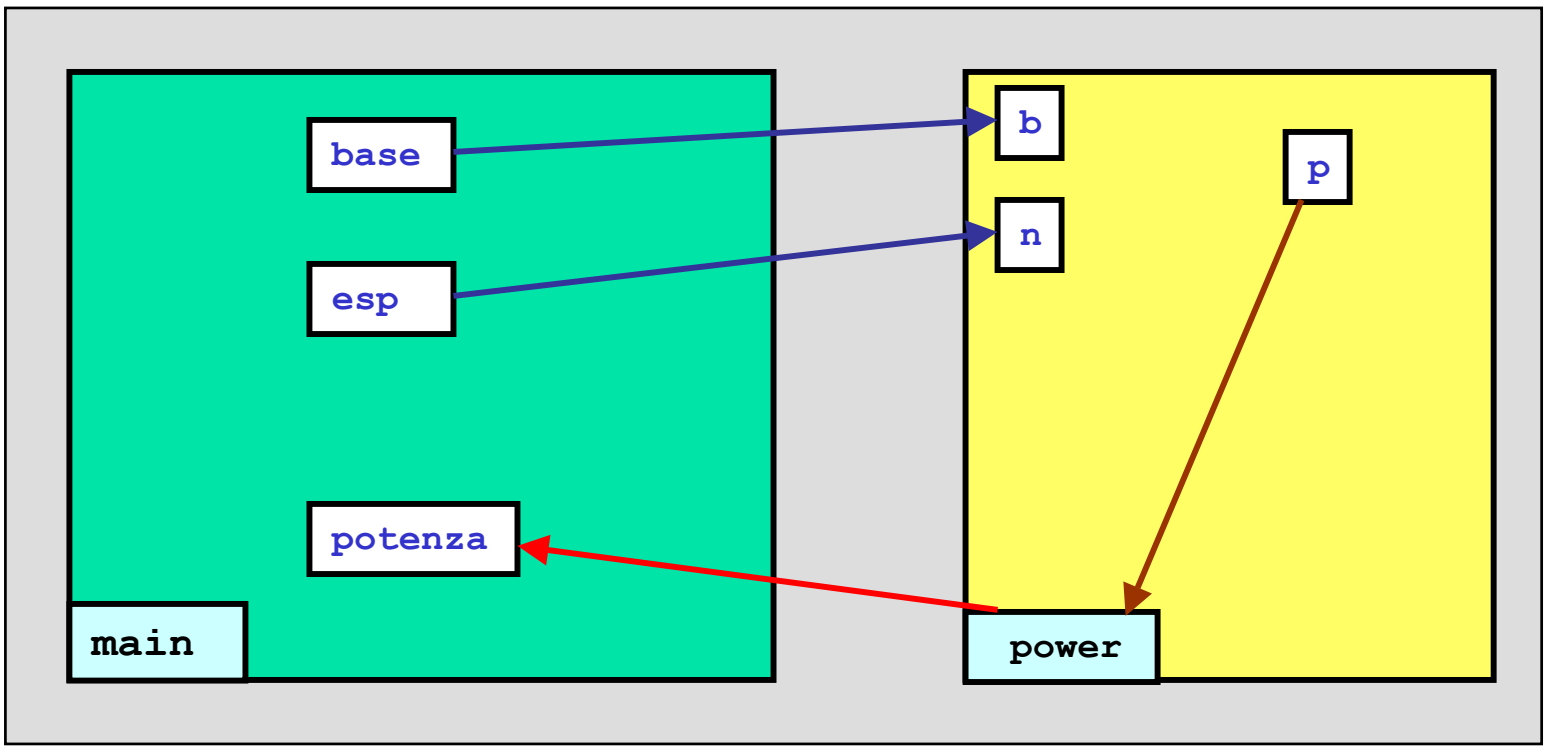
- **Passaggio per valore**
- Ogni metodo ha il proprio ambiente locale di esecuzione.
- L'informazione tra l'ambiente del **main** e l'ambiente del metodo **passa copiando i valori passati nelle celle di memoria dei parametri** e il ritorno del risultato assegnando il valore dell'espressione in una variabile del main.

```
potenza = power(base, esp);
```

- Quando viene effettuata la chiamata, il controllo passa alle istruzioni del metodo per ritornare al `main` dopo l'esecuzione dell'operazione `return`, che se non viene eseguita non si ha ritorno di valori.

Metodi : passaggio dei parametri per valore

Esempio della chiamata del metodo `potenza=power(base, esp);`



Metodi : passaggio dei parametri per riferimento

Passaggio dei parametri per riferimento

- Passando i parametri per valore **non si ha la modifica dei loro valori al termine della chiamata** del metodo.
- Questo avviene perché **viene fatta una copia dei parametri** attuali nell'ambiente del metodo chiamato.
- Per ottenere l'effetto della **modifica al termine della chiamata**, i linguaggi di alto livello usano il meccanismo di passaggio dei parametri per **riferimento** o indirizzo.
- In questo caso, invece di fare una copia dei parametri nell'ambiente locale, **si passano gli indirizzi delle celle di memoria** dell'ambiente globale che contengono i parametri.

Metodi : passaggio dei parametri per riferimento

- Nel metodo l'accesso avviene tramite l'indirizzo e quindi **le modifiche avverranno nelle locazioni originali**.

Parametri attuali **Parametri formali**

Ind(X) = 2056 X=75	→	a =75	**Passaggio per valore**
Ind(Y) = 3010 Y=14	→	b=*3010*	**Passaggio per riferimento**

- In questo secondo caso non opero su una copia ma sulla cella originale. Un assegnamento nel metodo cambia il valore nel main.

Passaggio dei parametri in Java

- In Java il **passaggio dei parametri** avviene **solo per valore**. Quindi nessuna modifica nel metodo sui parametri formali cambierà i valori dei parametri attuali passati come argomenti.
- Il **passaggio per riferimento** si ha implicitamente quando si hanno parametri formali di tipo **array** o **oggetti**.
- In questo caso viene **passata la referenza** e quindi si passa implicitamente l'indirizzo di memoria dell'array o dell'oggetto.
- Questo implica che **le eventuali modifiche verranno effettuate nell'area di memoria globale che contiene l'array o l'oggetto**.

Passaggio dei parametri in Java

- Nel seguente esempio

```
public class parametri2
{
   public static void main(String args[])
   {
    int x = 1;
    double d[] = new double[2];

    d[0] = 25;
    metodo1(d, x);
    System.out.println("Il valore di d[0] = " + d[0]);
    System.out.println("Il valore di x = " + x);
   }

  static void metodo1(double s[], int i)
  {
    s[0] = 100;
    i = 18;
  }
}
```

- Il valore stampato nel **`main`** sarà pari a 100. Perché l'assegnamento effettuato nel metodo viene eseguito nella stessa locazione di memoria originale di **`d`**. Mentre il valore di **`x`** non cambia e sarà stampato il valore 1.