Lab. Programmazione (CdL Informatica) & Informatica (CdL Matematica) a.a. 2022-23

Monica Nesi

Università degli Studi dell'Aquila

8 Novembre 2022

Programmazione strutturata e modulare

Finora abbiamo considerato semplici programmi Java che svolgono un'operazione (e.g. massimo di due numeri, moltiplicazione, calcolo del fattoriale, etc.).

Tuttavia, durante la computazione effettuata da un programma, un'operazione potrebbe essere applicata più volte e su argomenti di volta in volta diversi.

In tali casi non si deve duplicare la porzione di codice che svolge l'operazione richiesta.

Una buona tecnica di programmazione consiste nello *strutturare* la computazione in maniera *modulare* individuando quelle operazioni (o *funzionalità*) che possono essere definite in modo più *astratto* per operare su argomenti diversi.

Programmazione strutturata e modulare: metodi

Un *metodo* in Java definisce un'operazione ad alto livello (*sottoprogramma*) che consente di manipolare dati e oggetti.

Ricordiamo:
Ogni metodo è contenuto in una classe.
Una classe può contenere la definizione di vari metodi.
Un programma in Java è un insieme di dichiarazioni di classi.

I metodi contengono dichiarazioni ed istruzioni.
Un metodo può richiedere degli *argomenti su cui operare* e
i) *restituisce un valore* che è il risultato della sua computazione
oppure
ii) *non restituisce alcun valore*, ma semplicemente effettua delle modifiche sullo stato della macchina (e.g., il metodo modifica una qualche caratteristica di un oggetto o stampa una sequenza di caratteri).

Esempio: moltiplicazione di due numeri

Abbiamo già scritto un semplice programma Java che calcola il prodotto di due numeri naturali tramite l'algoritmo basato sulla somma (vedi `MultProg`).

Poiché durante una computazione può essere necessario applicare più volte l'operazione del prodotto, è opportuno definire un metodo che, dati due numeri naturali, calcola e restituisce il risultato del loro prodotto.

Il metodo `main` può limitarsi a prendere in input i numeri da moltiplicare (forniti dall'utente), invocare l'applicazione del metodo che implementa l'operazione di prodotto e stampare il risultato.

Programma, classi e metodi

D'ora in poi, i nostri programmi saranno costituiti da più classi ed organizzati come segue.

Si definisce *una classe per file* (con la corrispondenza tra i nomi già ricordata).

I vari metodi sono definiti in una (o più classi) con il metodo `main` in una classe *separata* di *test* (o *utilizzo*).

Per l'esempio della moltiplicazione, definiamo:
- nel file `Mult.java` la classe `Mult` contenente un metodo `mult` che calcola il prodotto di due numeri naturali;
- nel file `MultTest.java` la classe `MultTest` contenente il solo metodo `main`, che invoca il metodo `mult` e gestisce l'input/output (interazione con l'esterno).

La classe `MultTest`

Consideriamo prima la classe di test con il metodo `main`:

```
public class MultTest {
  public static void main (String[] args) {
    int x = Integer.parseInt(args[0]);
    int y = Integer.parseInt(args[1]);
    System.out.println(Mult.mult(x,y));
  }
}
```

Nota: nel seguito le classi verranno definite *pubbliche* tramite la parola riservata `public`.

N.B. In un file può essere definita al più una classe pubblica ed il suo nome deve corrispondere al nome del file.

La classe `MultTest`: commenti

Il metodo `main`
- prende in input due valori (stesse prime righe del corpo del `main` nel programma `MultProg`),
- poi chiama (invoca) il metodo `mult` della classe `Mult` su tali valori,
- quindi stampa il risultato della chiamata.

Il metodo `mult` è quindi chiamato (o invocato) dal metodo `main`, che agisce da *metodo chiamante*.

In Java ogni metodo (tranne il metodo iniziale `main`) può essere chiamato solo all'interno di un altro metodo.

Nota: poiché viene usato il tipo base `int` per i numeri da moltiplicare, in questo programma assumiamo di avere in input soltanto numeri naturali.

La classe Mult

```
public class Mult {
  public static int mult (int x, int y) {
    int z = 0;
    while (y > 0) {
      z = z+x;
      y = y-1;
    }
    return z;
  }
}
```

Il metodo `mult` definisce l'operazione di moltiplicazione di due numeri naturali tramite la somma.

Il metodo `mult`: intestazione

L'intestazione del metodo `mult` è

```
public static int mult (int x, int y)
```

e fornisce le seguenti informazioni. Il metodo `mult`

- è pubblico (può essere invocato da qualsiasi punto del programma);
- è statico (si dice anche che è un *metodo di classe*), in quanto dipende solo dalla classe in cui è definito;
- restituisce un valore di tipo intero (denotato dal tipo `int` dopo la parola riservata `static`);
- si chiama `mult` (nome o identificatore del metodo);
- si aspetta due valori interi in ingresso, denotati da `(int x, int y)`, detti *parametri formali* del metodo.

I parametri formali di un metodo

I *parametri formali* di un metodo sono specificati mediante una lista di *coppie* (separate da una virgola) della forma

$$<Tipo> <ParametroIde>$$

dove $<Tipo>$ è l'insieme dei tipi del linguaggio e $<ParametroIde>$ è l'insieme degli identificatori di parametri. Quindi,

$$T_1\, x_1, \ldots, T_n\, x_n$$

denota n parametri formali $x_i \in <ParametroIde>$ di tipo $T_i \in <Tipo>$ per $i = 1, \ldots, n$.

N.B. Un metodo può non avere parametri, ma le parentesi vanno sempre scritte (sia in fase in definizione che in fase di chiamata).

I parametri formali possono essere visti come il *punto di ingresso* nel metodo. Esiste *un solo punto di ingresso* nel metodo.

Il metodo `mult`: corpo

```
public static int mult (int x, int y) {
  int z = 0;
  while (y > 0) {
    z = z+x;
    y = y-1;
  }
  return z;
}
```

Il corpo del metodo è costituito dalla dichiarazione della variabile intera `z` (variabile *locale* al metodo), seguita da un'istruzione di ciclo che codifica l'algoritmo della moltiplicazione di due numeri naturali tramite la somma.

Il corpo del metodo termina con l'istruzione

```
return z;
```

L'istruzione return

L'istruzione `return` consente di *uscire* dal metodo completando la sua esecuzione, eventualmente restituendo il risultato calcolato, e *restituisce il controllo* al metodo chiamante.

Nel nostro caso il comando `return` è seguito da un'espressione, in particolare la variabile `z`, secondo la seguente sintassi:

se il metodo restituisce un valore di tipo T (i.e., il tipo che appare prima del nome del metodo nella sua intestazione), allora l'istruzione di uscita è data dalla parola riservata `return` seguita da un'espressione `E`, il cui tipo è uguale a (o compatibile con) T, e terminata da un `;`.

La semantica dell'istruzione `return E;` consiste nel valutare `E`, ottenendo il suo valore v_E, e nel terminare l'esecuzione del metodo restituendo il controllo al metodo chiamante *passandogli* il valore v_E come *risultato della chiamata del metodo*.

Chiamata di metodo

Nel metodo `main`, dopo aver memorizzato i valori in input nelle variabili `x` ed `y` locali al `main`), si ha la chiamata del metodo `mult` (come argomento del metodo `println`) come segue:

```
System.out.println(Mult.mult(x,y));
```

Il metodo `mult` è definito `static`, quindi la sua invocazione si effettua premettendo il nome della classe in cui è definito (seguito da un punto) al nome del metodo. La chiamata di metodo

```
Mult.mult(x,y)
```

è sintatticamente un'espressione di tipo `int` (il tipo del valore restituito da `mult`) ottenuto applicando il metodo `mult` della classe `Mult` sugli argomenti, detti *parametri attuali*, dati dalle espressioni `x` ed `y` (che sono, in particolare, degli identificatori di variabili).

Passaggio dei parametri

Nell'intestazione del metodo `mult` abbiamo i *parametri formali* `x` ed `y`, entrambi di tipo `int`. Nella chiamata del metodo `mult` abbiamo i *parametri attuali* `x` ed `y`, entrambi di tipo `int`.

Come si *associano* i parametri attuali ai parametri formali?

La *modalità di passaggio dei parametri* in Java è il *passaggio per valore* (in inglese *call-by-value*):
quando `mult` viene invocato, vengono prima valutati i suoi parametri attuali (i.e., le variabili `x` ed `y` locali al `main`) ed i valori ottenuti vengono associati ai suoi parametri formali, rispettando l'ordine dei parametri.

Il corpo del metodo `mult` viene eseguito tenendo conto di tale legame tra parametri formali e parametri attuali.

N.B. Non confondere i parametri formali `x` ed `y` del metodo `mult` con le variabili locali `x` ed `y` del metodo chiamante `main`.

Esecuzione del programma

Il programma può essere quindi compilato tramite

```
javac Mult.java
javac MultTest.java
```

o semplicemente (se i due file sono nella stessa cartella) tramite

```
javac MultTest.java
```

ovvero compilando solo il file in cui si trova il `main` (il compilatore provvede a compilare anche le altre classi da cui dipende quella con il `main` e genera anche i loro file .class).

Quindi vengono generati i file `Mult.class` e `MultTest.class` e si può eseguire il programma mandando in esecuzione il file .class in cui si trova il `main`. Ad esempio:

```
java MultTest 4 8
32
```

Esecuzione del programma (cont.)

Eseguendo il programma con i valori in input 4 e 8

```
java MultTest 4 8
32
```

si ha che le variabili `x` ed `y` locali al `main` sono inizializzate con 4 e 8 rispettivamente.

Al momento della chiamata `Mult.mult(x,y)`, tali valori sono passati al metodo `mult` ed associati ai parametri formali `x` ed `y` di `mult`.

Il codice del metodo `mult` viene eseguito tenendo conto di tali legami.

Il risultato calcolato e restituito dalla chiamata `Mult.mult(x,y)` viene infine stampato tramite l'esecuzione del metodo `println`.

Altri metodi per funzioni matematiche

Definiamo una classe `Arith` che contiene metodi per le operazioni di prodotto ed esponenziazione.

```
public class Arith {
  public static int mult (int x, int y) {
    int z = 0;
    while (y > 0) {
      z = z+x;
      y = y-1;
    }
    return z;
  }

// la definizione continua ...
```

Altri metodi per funzioni matematiche (cont.)

```
// secondo metodo della classe Arith

  public static int exp (int x, int y) {
    int z = 1;
    while (y > 0) {
      z = mult(x,z);
      y = y-1;
    }
    return z;
  }
}
```

Poiché il metodo `exp` invoca un metodo definito nella sua stessa classe, è possibile evitare di specificare la classe nella chiamata del metodo `mult`.

Una classe di test per Arith

```
public class ArithTest {
  public static void main (String[] args) {
    int x = Integer.parseInt(args[0]);
    int y = Integer.parseInt(args[1]);
    System.out.println(Arith.exp(x,y));
  }
}
```

Compilazione ed esecuzione:

```
javac ArithTest.java
java ArithTest 2 10
1024
```

Ancora sull'istruzione return

Se un metodo non restituisce alcun risultato, ovvero è un metodo nella cui intestazione compare la parola riservata `void` prima del nome del metodo, allora la sintassi dell'istruzione di uscita è data solamente dalla parola riservata `return` seguita da `;`.

La semantica dell'istruzione `return;` consiste nell'uscire dal metodo completando la sua esecuzione e nel restituire il controllo al metodo chiamante.

Se l'uscita da un metodo di tipo `void` coincide con la fine del corpo del metodo, allora non è necessario mettere `return;`.

N.B. Sintatticamente la chiamata di un metodo di tipo `void` è un comando (o istruzione).

Possono esistere *più punti di uscita da un metodo*, ovvero più di una istruzione `return` (una per ogni flusso di esecuzione).

Metodo di tipo void: esempio

Scrivere un metodo in Java che, dati in ingresso due numeri naturali m, n tali che $m < n$, stampa tutti i numeri pari compresi tra m ed n, ovvero tutti i numeri x tali che x è pari e $m \leq x \leq n$.

```
public static void stampaPari(int m, int n) {
   int x;
   if (m%2 == 0)
      x = m;
   else
      x = m+1;
   while (x <= n) {
      System.out.println(x);
      x = x+2;
   }
}
```

Una classe di test

Assumiamo che il metodo `stampaPari` sia definito nella classe `StampaPari`.

Un semplice programma di prova è il seguente:

```
public class StampaPariTest {
  public static void main (String[] args) {
    int m = Integer.parseInt(args[0]);
    int n = Integer.parseInt(args[1]);
    StampaPari.stampaPari(m,n);
  }
}
```

Il tipo del metodo `stampaPari` è `void`, quindi la chiamata del metodo `StampaPari.stampaPari(m,n);` è un'istruzione.

Un metodo per il fattoriale

Scrivere un metodo che calcola il fattoriale di un numero naturale `n`: $0! = 1$ ed $n! = n \times (n-1) \times \ldots \times 2 \times 1$ per $n \geq 1$.
Se al metodo viene passato un numero intero negativo, il metodo restituisce `-1`.

```
public static int fattoriale (int n) {
   if (n<0) return -1;
   int f = 1;
   for (int i=1; i<=n; i++)
      f = f*i;
   return f;
}
```

Definire una semplice classe di test.

Un metodo per i numeri di Fibonacci

Scrivere un metodo che calcola l'n-esimo numero di Fibonacci: $fib(0) = 1$, $fib(1) = 1$, $fib(n) = fib(n-1) + fib(n-2)$ per $n \geq 2$.
Se al metodo viene passato un numero intero negativo, il metodo restituisce -1.

```
public static int fibonacci (int n) {
  if (n<0) return -1;
  int fib = 1, x = 0, temp;
  for (int i=1; i<=n; i++) {
    temp = fib+x;
    x = fib;
    fib = temp;
  }
  return fib;
}
```

Definire una semplice classe di test.