7 - Programmazione procedurale: Dichiarazione e chiamata di metodi ausiliari

Programmazione e analisi di dati
Modulo A: Programmazione in Java

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa
http://pages.di.unipi.it/milazzo
milazzo@di.unipi.it

Corso di Laurea Magistrale in Informatica Umanistica
A.A. 2019/2020

Programmazione procedurale

Fino ad ora abbiamo realizzato programmi costituiti da un unico metodo (il metodo `main`)

Quando il programma da realizzare è articolato diventa conveniente

- identificare sottoproblemi che possono essere risolti individualmente
- scrivere sottoprogrammi che risolvono i sottoproblemi
- richiamare i sottoprogrammi dal programma principale (`main`)

Questo approccio prende il nome di programmazione procedurale (o astrazione funzionale)

In Java i sottoprogrammi si realizzano tramite metodi ausiliari

- Sinonimi usati in altri linguaggi di programmazione: funzioni, procedure e (sub)routines

Metodi ausiliari (1)

Un esempio: somma e prodotto dei primi 10 numeri naturali

- Scriviamo un programma `SommaProdottoDieci` che calcola, a scelta dell'utente, la somma o il prodotto dei numeri da 1 a 10

Identifichiamo i sottoproblemi:

- calcolare la somma dei numeri da 1 a 10
- calcolare il prodotto dei numeri da 1 a 10

Realizzeremo quindi i metodi ausiliari `somma10` e `prodotto10` che risolvono i due sottoproblemi

- il risultato di entrambi i metodi dovrà essere di tipo `int`

Metodi ausiliari (2)

Definizione del metodo ausiliario `somma10` (il metodo `prodotto10` è analogo)

```
private static int somma10() {
  int ris=0;

  for (int i=1; i<=10; i++)
    ris+=i;

  return ris;
}
```

- `private` specifica che si tratta di un metodo ausiliario (di aiuto a un altro metodo della stessa classe... non utilizzabile dalle altre classi)
- `static` lo capiremo più avanti
- `int` è il tipo del risultato calcolato dal metodo
- `somma10` è il nome del metodo (è un identificatore – valgono le stesse regole sintattiche delle variabili)
- `()` le capiremo tra poco
- `return` è il comando che termina l'esecuzione del metodo e fornisce il risultato (dato da un'espressione)

Metodi ausiliari (3)

```
import java.util.Scanner;

public class SommaProdottoDieci {

 // metodo principale
 public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   System.out.println("Inserisci 1 per la somma, 2 per il prodotto");
   int scelta = input.nextInt();

   if (scelta==1) {
     int risultato=somma10(); // chiamata del metodo somma10
     System.out.println(risultato);
   }
   else if (scelta==2) {
     int risultato=prodotto10(); // chiamata del metodo prodotto10
     System.out.println(risultato);
   }
   else System.out.println("ERRORE");
 }

 // metodo ausiliario che calcola la somma da 1 a 10
 private static int somma10() {
   int ris=0;
   for (int i=1; i<=10; i++)
     ris+=i;
   return ris;
 }

 ...continua...
```

Metodi ausiliari (4)

```
 ...continua...

 // metodo ausiliario che calcola il prodotto da 1 a 10
 private static int prodotto10() {
   int ris=1;
   for (int i=1; i<=10; i++)
     ris*=i;
   return ris;
 }

} // FINE CLASSE
```

La chiamata (o invocazione) di un metodo è una espressione:

- Il tipo di tale espressione è il tipo del risultato del metodo
- Il valore di tale espressione è il risultato restituito dal metodo

Avremmo potuto semplificare il programma scrivendo direttamente

```
System.out.println(somma10());
```

Parametri (1)

I metodo possono anche prevedere parametri

Esempio: metodo che calcola la somma di due numeri

```
public class SommaConMetodo {

  public static void main(String[] args) {
    int a=10, b=20;
    System.out.println(somma(a,b));
  }

  private static int somma(int x, int y) {
    return x+y;
  }
}
```

- Nella dichiarazione di `somma` tra le parentesi tonde sono indicati i parametri attesi (parametri formali) con i rispettivi tipi — `x` e `y`
- Nella chiamata di `somma` tra le parentesi tonde sono indicati i valori (parametri attuali, o argomenti) che sono passati al metodo per calcolare il risultato — `a` e `b`

Parametri (2)

I parametri formali di un metodo sono variabili locali al corpo del metodo

- Un metodo non "vede" i parametri e le variabili di un altro (nemmeno del `main`, che è un metodo come tutti gli altri)

I valori dei parametri attuali vengono copiati (assegnati) nelle variabili usate come parametri formali del metodo

- un metodo deve sempre essere chiamato con parametri attuali di tipi compatibili con quelli dei parametri formali (e nell'ordine giusto)

I nomi dei parametri formali e delle variabili di metodi diversi non interferiscono tra loro

```
public static void main(String[] args) {
  int y=10, x=20;
  int ris = ilPrimo(y,x);
  System.out.println(ris); //stampa 10
}

private static int ilPrimo(int x, int y) {
  return x;
}
```

Il comando `return`

L'espressione che segue ogni `return` di un metodo deve avere un tipo compatibile con il tipo del metodo

```
private static int prova() {
  return 3.5; // errore!
}
```

Il comando `return` può essere usato più volte in un metodo

```
if (x<0)
  return 0;
else
  return x;
```

Il compilatore controlla che il comando `return` sia sempre raggiungibile

```
private static int prova(int x) {
  if (x>0) return 3; // return non sempre raggiungibile
                     // (condizionato dalla guardia dell'if)
}

private static int prova() {
  if (true) return 3; // ATTENZIONE: anche qui da' errore!!
}
```

Il tipo void (1)

Quando un metodo non prevede un risultato si usa il tipo void

```
private static void stampaMessaggio() {
    System.out.println("Messaggio di prova");
}
```

L'invocazione di un metodo di tipo void avviene senza assegnamento!

```
...
stampaMessaggio();
...
```

Il tipo `void` (2)

In questi casi il comando `return` è solitamente omesso, ma può essere usato (privo di espressione successiva) per interrompere l'esecuzione del metodo:

```
private static void stampaAsterischi(int n) {
  if (n<=0) {
    System.out.println("valore errato");
    return; // corretto, ma usare con cautela!
  }
  for(int i=0;i<n;i++) System.out.print("*"); // niente else!
}
```

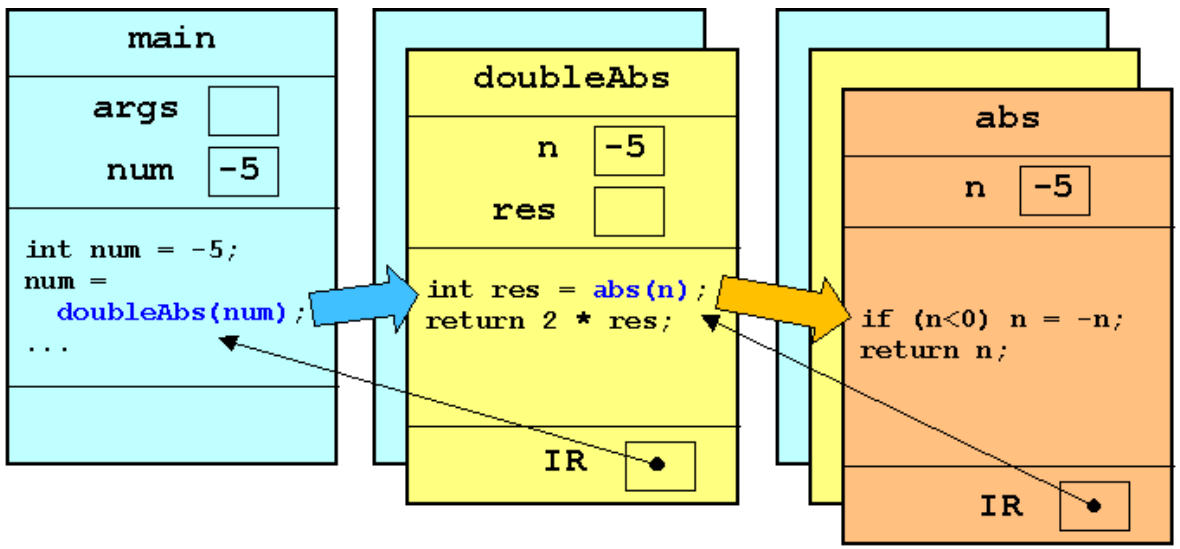
Questo modo di usare `return` va usato con cautela (solo se effettivamente semplifica il programma senza confondere il lettore).

- E' decisamente meglio evitare di usare `return` all'interno di cicli

Che cosa succede quando si invoca un metodo? (1)

```
public class ChiamataMetodi {
    public static void main(String[] args){
        int num = -5;
        num = doubleAbs(num);
        System.out.println(num); // stampa 10
    }
    private static int doubleAbs(int n){
        int res = abs(n);
        return 2 * res;
    }
    private static int abs(int n){
        if (n < 0) n = -n;
        return n;
    }
}
```

In memoria:
La pila
dei record
di attivazione



Che cosa succede quando si invoca un metodo? (2)

Quando si incontra l'invocazione di un metodo, l'esecuzione del metodo corrente viene sospesa fino al completamento del metodo invocato.

Per eseguire il metodo invocato, viene allocato in memoria un record di attivazione (o frame). Questo contiene fra l'altro:

- una variabile per ogni parametro formale del metodo invocato, inizializzata con il corrispondente parametro attuale;
- le variabili locali del metodo invocato;
- l'indirizzo di ritorno (IR), cioè il punto del metodo chiamante cui bisogna cedere il controllo (e restituire il risultato) alla fine dell'esecuzione del metodo invocato.

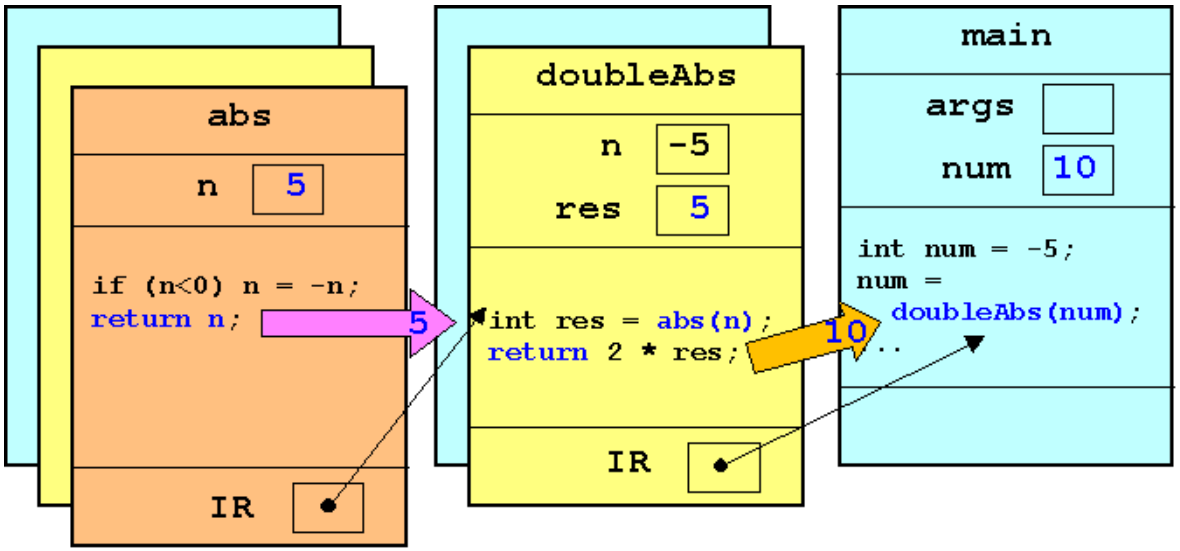
In una sequenza di chiamate di metodi (uno dentro l'altro) l'ultimo metodo chiamato è il primo a terminare l'esecuzione.

- I record di attivazione formano quindi una pila (o stack).

Che cosa succede quando si invoca un metodo? (3)

Quando un metodo termina l'esecuzione:

- il controllo passa all'istruzione del metodo chiamante riferita dall'indirizzo di ritorno, eventualmente con il passaggio del risultato
- Il frame del metodo corrente viene disallocato, mentre il metodo chiamante riprende l'esecuzione



Esempi d'uso: Orario (1)

Ricordate il programma Orario?

```
import java.util.Scanner;

public class Orario4 {

  public static void main(String[] args) {

  Scanner input = new Scanner(System.in);

  int ore,minuti;

  System.out.println("Inserire ore e minuti:");
  ore = input.nextInt();
  minuti = input.nextInt();

  boolean ore_ok = (ore>=0) && (ore<24);
  boolean minuti_ok = (minuti>=0) && (minuti<60);

  if ( ore_ok && minuti_ok )
    System.out.println("E' un orario");
  else
    System.out.println("Non e' un orario");
  }
}
```

Esempi d'uso: Orario (2)

Facciamo fare il controllo a un metodo ausiliario

```
import java.util.Scanner;

public class OrarioMetodo {

  public static void main(String[] args) {

    Scanner input = new Scanner(System.in);

    int ore,minuti;
    System.out.println("Inserire ore e minuti:");
    ore = input.nextInt();
    minuti = input.nextInt();

    if ( orarioOk(ore, minuti) ) // chiama il metodo ausiliario
      System.out.println("E' un orario");
    else
      System.out.println("Non e' un orario");
  }

  // restituisce true se l'orario e' valido, false altrimenti
  private static boolean orarioOk(int o, int m) {

    boolean ore_ok = (o>=0) && (o<24);
    boolean minuti_ok = (m>=0) && (m<60);

    return ore_ok && minuti_ok;
  }
}
```

Esempi d'uso: `ProdottoPotenza` (1)

Un metodo ausiliario può essere invocato in punti diversi del programma, e anche più volte di seguito (ad esempio, in un ciclo)

Esempi d'uso: ProdottoPotenza (2)

```
import java.util.Scanner;

public class ProdottoPotenza {

  public static void main(String[] args) {

    Scanner input = new Scanner(System.in);

    System.out.println("Inserire due numeri interi:");
    int x = input.nextInt();
    int y = input.nextInt();
    System.out.println("Inserire 1 per prodotto, 2 per potenza:");
    int scelta = input.nextInt();

    if (scelta==1)
      System.out.println(prodotto(x,y)); //chiama prodotto qui...
    else if (scelta==2) {
      int ris=1;
      for (int i=0; i<y; i++) ris=prodotto(ris,x); //...e anche qui!
      System.out.println(ris);
    }
    else System.out.println("ERRORE");
  }

  // calcola (banalmente) il prodotto di due numeri
  private static int prodotto(int x, int y) {
    return x*y;
  }
}
```