Corso di Laurea Ingegneria Informatica

Fondamenti di Informatica 1

Dispensa 9

Definizione di metodi

Carla Limongelli

Novembre 2007

Contenuti

- Introduzione alla definizione di metodi
  - Prototipo e firma di un metodo
  - Esecuzione di un metodo
- Metodi e variabili locali
- Parametri formali e parametri attuali
- Metodi che restituiscono un valore
- Sovraccarico di nomi
- Composizione di metodi
- Auto-referenziazione
- Visibilità delle variabili
- Programmazione strutturata (e non)
- Effetti collaterali
- Metodi di supporto
  - il modificatore private
- Legame dei parametri
  - legame dei parametri per valore
  - legame dei parametri per riferimento

Prerequisiti

Questo capitolo presuppone la conoscenza degli argomenti già trattati nelle precedenti lezioni di questo corso

Introduzione alla definizione di metodi

❑Scopo della definizione di un metodo

- implementare una operazione
- supportare la definizione di altri metodi

❑Le operazioni sono implementate dai metodi

- un **metodo** è la descrizione di una operazione
- la classe per un oggetto contiene la definizione di un metodo per ciascuna delle operazioni che l'oggetto sa eseguire

❑La definizione di un metodo è formata da

- **intestazione** – descrive il prototipo dell'operazione
- **corpo** – la sequenza di istruzioni che l'oggetto deve eseguire quando gli viene richiesto di eseguire quell'operazione

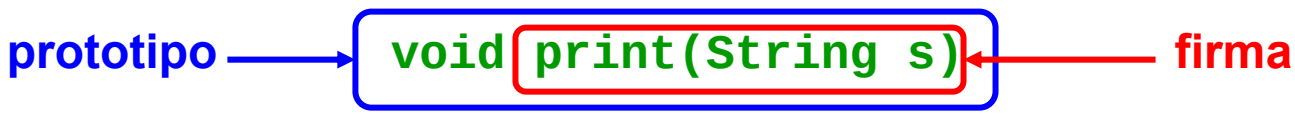
Descrizione di metodi: prototipo

- **Il prototipo di un metodo (di un oggetto) è una descrizione del formato dei messaggi con cui è possibile richiedere (a quell'oggetto) di eseguire quel metodo**
  - **nome** del metodo
  - **elenco dei parametri** del metodo
    - ogni parametro ha un nome e un tipo
  - **tipo di ritorno** del metodo
    - un metodo può restituire un valore di un certo tipo, oppure non restituire nessun valore (**void**)
  - tipo di ritorno, nome, elenco dei parametri racchiuso tra parentesi

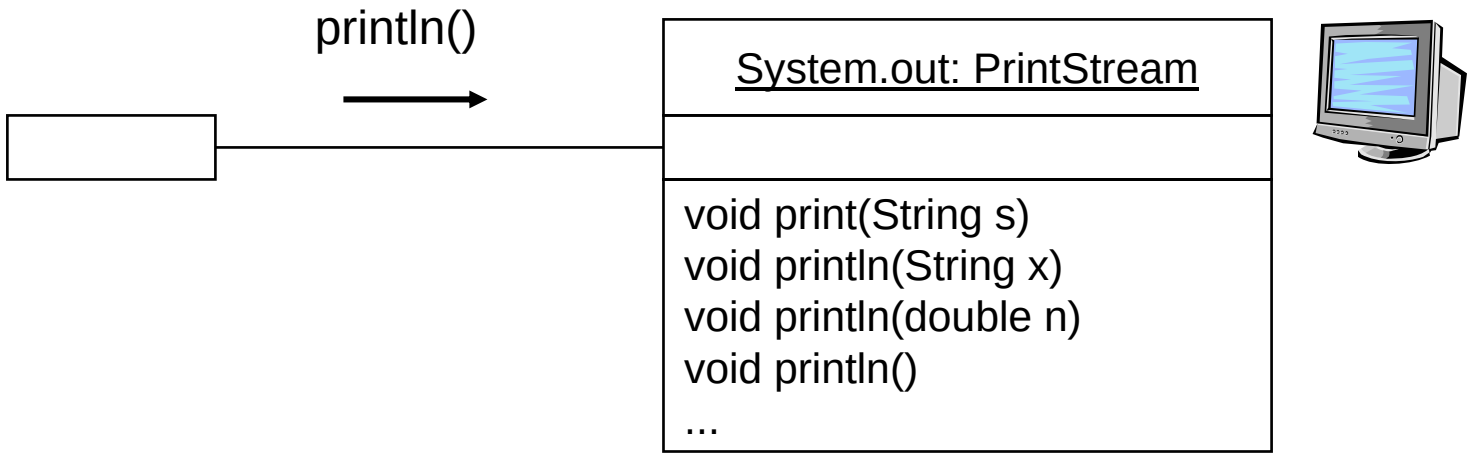
```
void println(String x)
```

Descrizione di metodi: firma

- **La firma (segnatura) di un metodo consiste del nome del metodo insieme all'elenco dei parametri del metodo**
  - **il prototipo del metodo meno tipo di ritorno del metodo**
- **I metodi sono identificati dalla firma**
  - **Nell'ambito di una classe**
    - possono esistere più metodi che hanno lo stesso nome
      - un nome usato per più metodi è detto **sovraccarico** (di significati)
    - ma non possono però esistere più metodi che hanno la stessa firma
  - **i metodi di una classe sono identificati dalla loro firma e non dal loro nome**



Messaggi, firme e metodi

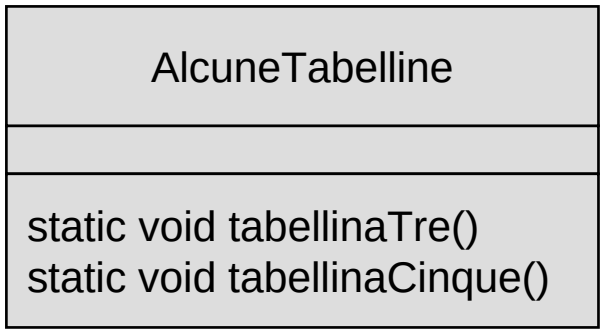


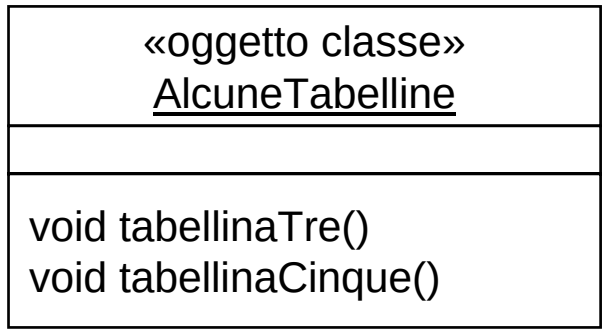
- Come fa un oggetto, quando riceve un messaggio, a selezionare il metodo che deve eseguire?
  - la selezione viene fatta sulla firma del messaggio ricevuto
  - dal messaggio risale alla firma del metodo invocato

Metodi e variabili locali

```
/* La classe AlcuneTabelline rappresenta un oggetto
 * che sa calcolare alcune tabelline. */

class AlcuneTabelline {

   public static void tabellinaTre() {
        int i;    // per iterare tra 1 e 10
        /* calcola e stampa la tabellina del tre */
        System.out.println("Tabellina del 3:");
        for (i=1; i<=10; i++)
            System.out.print(3*i + " ");
        System.out.println();
    }

   public static void tabellinaCinque() {
        int i;    // per iterare tra 1 e 10
        /* calcola e stampa la tabellina del cinque */
        System.out.println("Tabellina del 5:");
        for (i=1; i<=10; i++)
            System.out.print(5*i + " ");
        System.out.println();
    }
}
```

AlcuneTabelline





- una classe può definire più metodi
- una classe può contenere la dichiarazione di variabili

Metodi e variabili locali

```
public static void tabellinaTre() {
      int i;    // per iterare tra 1 e 10
      ...
  }

  public static void tabellinaCinque() {
      int i;    // per iterare tra 1 e 10
      ...
  }
```

- **Le variabili dichiarate in un metodo sono locali al metodo**
  - ciascun metodo può accedere solo alle proprie variabili locali
  - un metodo non può accedere alle variabili locali di un altro metodo
  - relazione di **visibilità** tra variabili e metodi

Metodi con parametri

La classe AlcuneTabelline è insoddisfacente

- sarebbe preferibile un oggetto che sa calcolare qualsiasi tabellina – in modo parametrico

```
/* calcola la tabellina del 5 */
  Tabelline.tabellina(5);
  /* calcola la tabellina del 9 */
  Tabelline.tabellina(9);
```

La classe Tabelline

```
/* La classe Tabelline rappresenta un oggetto
 * che sa calcolare tutte le tabelline. */

class Tabelline {
   public static void tabellina(int n) {
         // pre: n>=0
         int i;    // per iterare tra 1 e 10

         /* calcola e stampa la tabellina del numero n */
         System.out.println("Tabellina del " + n + ":");
         for (i=1; i<=10; i++)
             System.out.print(n*i + " ");
         System.out.println();
    }
}
```

- Il prototipo del metodo contiene la specifica del parametro. E' la dichiarazione di una variabile locale n chiamata parametro formale.

Parametri formali e parametri attuali

❑L'intestazione di un metodo riporta l'elenco dei parametri formali del metodo

- i parametri formali di un metodo sono variabili locali del metodo

```
public static void tabellina(int n) { ... }
```

❑L'invocazione di un metodo riporta i parametri attuali dell'invocazione del metodo

- i parametri attuali dell'invocazione di un metodo sono espressioni dello stesso tipo dei parametri formali corrispondenti

```
Tabelline.tabellina(5);
```

Esecuzione di un metodo con parametri

```
public static void tabellina(int n) { ... }
```

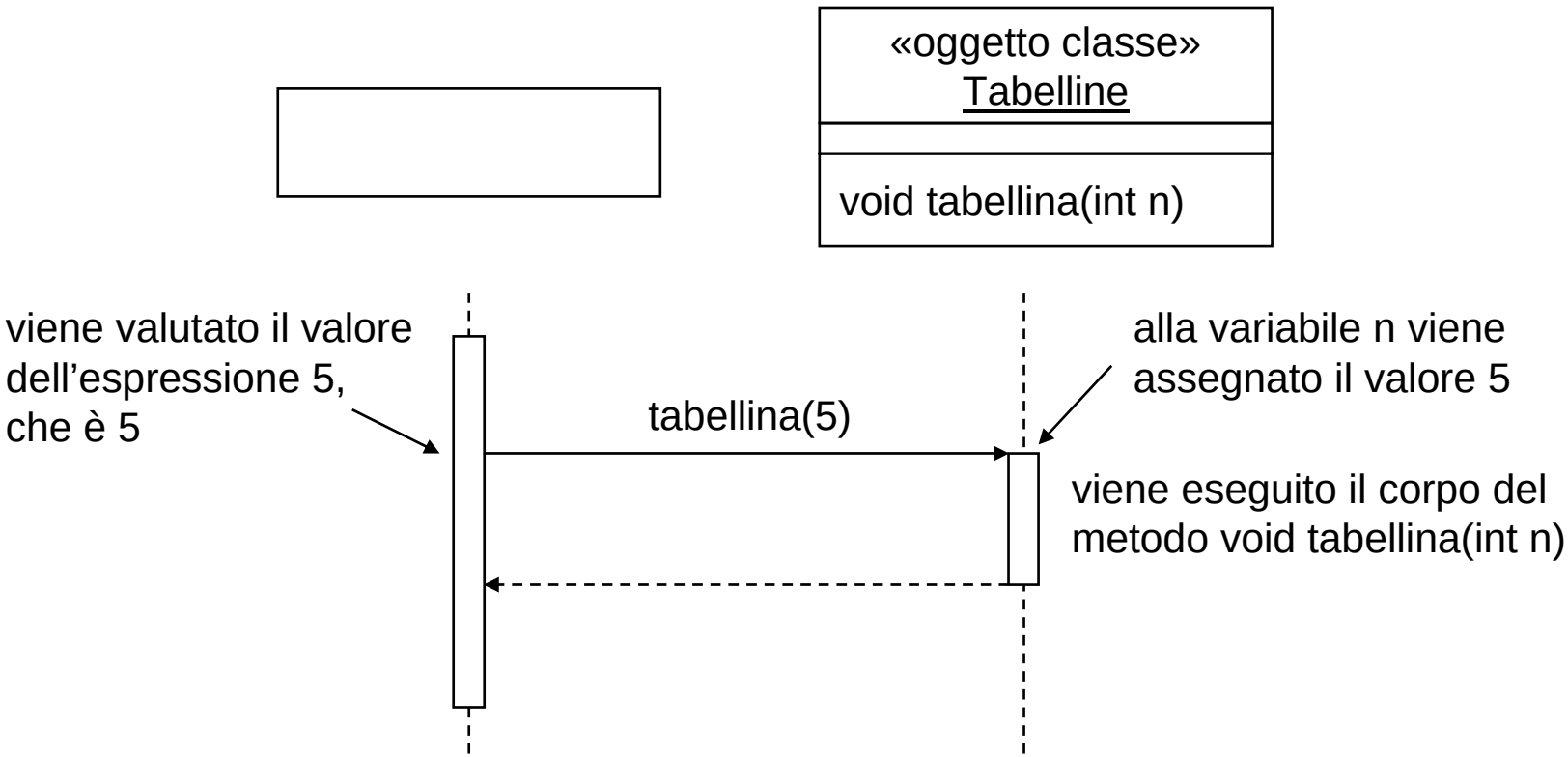
definizione del metodo

```
Tabelline.tabellina(5);
```

invocazione del metodo

- **Definizione di un metodo**
  - Deve contenere il tipo di ritorno del metodo e l'elenco dei parametri formali con il loro tipo
- **Invocazione di un metodo**
  - deve contenere tanti parametri attuali quanti sono i parametri formali del metodo
- **Esecuzione del metodo**
  - valuta i parametri attuali dell'invocazione
  - assegna a ciascuna variabile parametro formale, come valore iniziale, il valore del parametro attuale corrispondente
  - esegue le istruzioni del corpo del metodo

Esecuzione di un metodo con parametri



Parametri formali e attuali

❑I parametri sono usati

- per rendere parametrica una operazione
  - i parametri formali rappresentano informazioni che bisogna specificare per richiedere l'esecuzione dell'operazione
- per rendere specifico un messaggio inviato a un oggetto
  - i parametri attuali specificano valori che devono essere utilizzati nell'esecuzione di una operazione

❑i parametri formali di un metodo sono relativi alla definizione del metodo

- **i parametri formali sono variabili**

❑i parametri attuali sono relativi alle invocazioni del metodo

- **i parametri attuali sono espressioni**

Metodi che restituiscono un valore

- **Molte operazioni prevedono la restituzione di un valore**

```
int risultato;
 /* calcola il fattoriale di 5 */
 risultato = Fattoriale.fattoriale(5);    // 120

 /* calcola e visualizza il fattoriale di 4 */
 System.out.println( Fattoriale.fattoriale(4) );   //24
```

La classe Fattoriale

```
/* Classe per il calcolo del fattoriale. */

class Fattoriale {
    /* Calcola e restituisce il fattoriale di n. */

   public static int fattoriale(int n) {
         // pre: n>=0
         int f;    // il fattoriale di n
         int i;    // per iterare tra 1 e n

       /* calcola il fattoriale di n */
         f=1;
         for (i=1; i<=n; i++)
             f = f*i;
         return f;
    }
}
```

Metodi che restituiscono un valore

```
public static int fattoriale(int n) {
      // pre: n>=0
      int f;    // il fattoriale di n
      int i;    // per iterare tra 1 e n

      /* calcola il fattoriale di n */
      f = 1;
      for (i=1; i<=n; i++)
          f = f*i;
      return f;
  }
```

- il tipo di ritorno del metodo deve essere diverso da void
- il metodo deve terminare con una istruzione return

Istruzione return

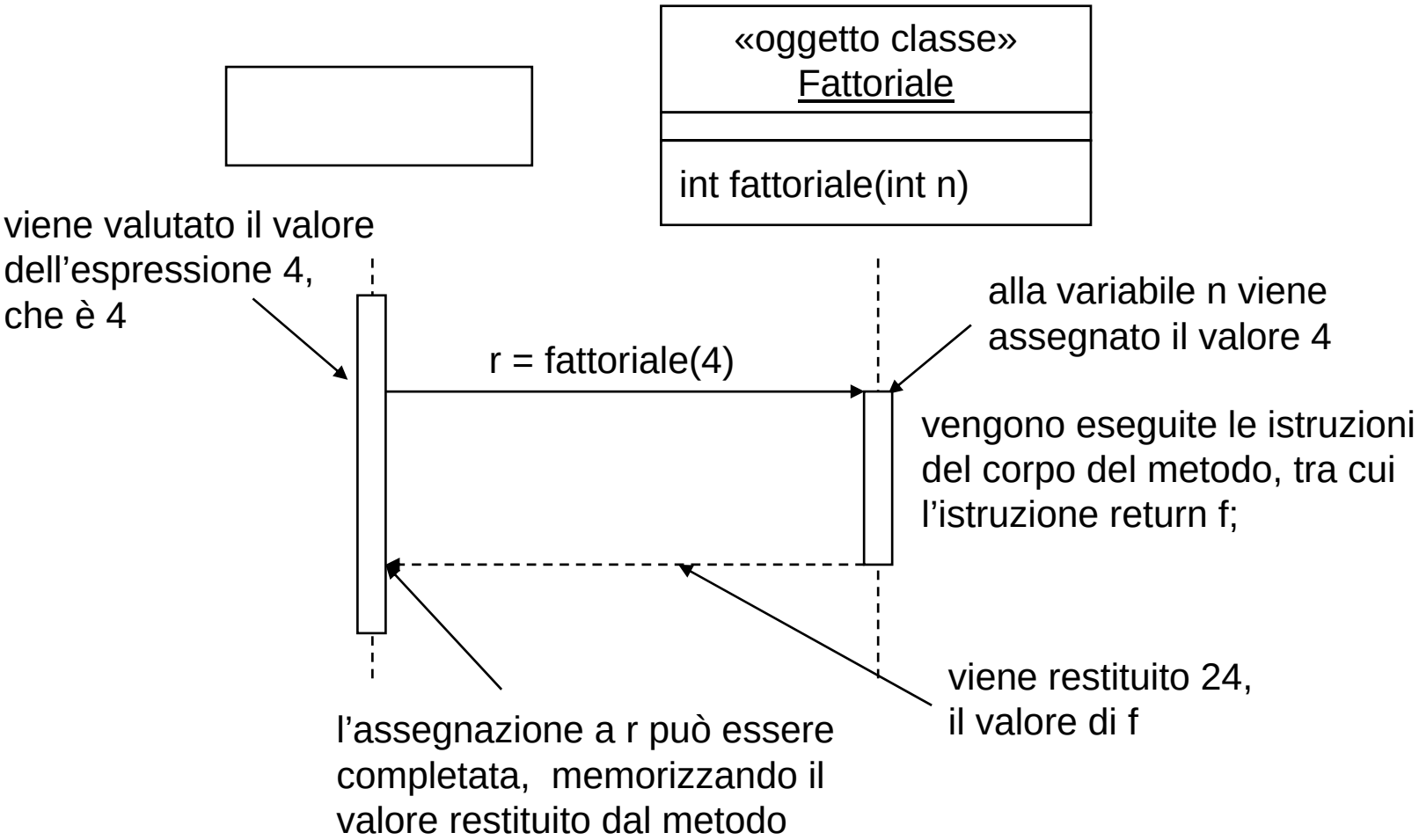
```
public static int fattoriale(int n) {
        // pre: n>=0
        int f;    // il fattoriale di n
        int i;    // per iterare tra 1 e n
         f = 1;
        for (i=1; i<=n; i++)
            f = f*i;
        return f;
    }
```

- Il metodo int fattoriale(int n), al termine dell'esecuzione, deve restituire il valore memorizzato nella variabile f
- l'istruzione return è l'istruzione che permette a un metodo di restituire un valore
- la parola return è seguita da una espressione
  - deve essere del tipo di ritorno del metodo
  - il cui valore deve essere restituito dall'esecuzione del metodo

Istruzione return

- **L'istruzione return permette a un metodo di restituire il controllo a chi lo ha invocato e, contestualmente, di restituirgli un valore**
  - **quando in un metodo viene eseguita una istruzione return:**
    - l'espressione che segue **return** viene valutata e il valore calcolato viene restituito a chi ha invocato il metodo
    - il metodo termina (ultima istruzione)
  - **se un metodo ha tipo di ritorno diverso da void, l'uso dell'istruzione return è obbligatorio**
  - **se un metodo ha tipo di ritorno T, l'espressione che segue return deve essere di tipo T**
- **una esecuzione di un metodo può restituire al più un solo valore**

Esecuzione di un metodo che restituisce un valore



Sulla composizione dell'invocazione di metodi

- Come gli operatori aritmetici possono essere composti per scrivere espressioni complesse, lo stesso avviene per espressioni che coinvolgono l'invocazione di metodi
  - ci sono due modalità per comporre invocazioni di metodi
    - **composizione** dell'invocazione di metodi
    - **cascata** di invocazione di metodi

Composizione dell'invocazione di metodi . . .

- **La composizione dell'invocazione di metodi consiste nell'usare il risultato prodotto dall'invocazione di un metodo come argomento in un'altra invocazione di metodo**
  - ad esempio, se si vogliono concatenare tre stringhe

```
/* concatena s, t e u */
   v = s.concat( t.concat(u) );
```

  - per visualizzare la radice quadrata di x

```
/* calcola e visualizza la radice quadrata
di  x */
   System.out.println( Math.sqrt(x) );
```

. . . Composizione dell'invocazione di metodi

- **Vale la seguente regola riguardo l'esecuzione di istruzioni che comprendono l'invocazione di metodi**
  - **in una invocazione di un metodo**
    - **vengono prima valutati il destinatario del metodo e tutti gli argomenti del metodo**
    - **viene quindi attivato il metodo**

Cascata di invocazione di metodi

- La cascata di invocazione di metodi consiste nell'usare il risultato prodotto dall'invocazione di un metodo come oggetto da usare come destinatario di un'altra invocazione di metodo
  - ad esempio, se si vogliono concatenare tre stringhe

```
/* concatena s, t e u */
   v = s.concat(t).concat(u);
```

Sovraccarico di nomi

❑Una classe può definire più metodi che hanno lo stesso nome

- questo è possibile purché ciascun metodo abbia firma differente

```
class Maggiore {
    /* Calcola il maggiore tra i due numeri a e b. */
    public static int maggiore(int a, int b) {
        ...
    }
    /* Calcola il maggiore tra i tre numeri a, b e c. */
    public static int maggiore(int a, int b, int c) {
        ...
    }
}
```

- **il nome maggiore è sovraccarico (overloaded)**
- **questa scelta è accettabile perché i due metodi hanno sostanzialmente la stessa semantica**

La classe Maggiore

```
class Maggiore {
    /* Calcola il maggiore tra i due numeri a e b. */
    public static int maggiore(int a, int b) {
        int max;    // il maggiore tra a e b
        if (a>b)
            max = a;
        else
            max = b;
        return max;
    }

    /* Calcola il maggiore tra i tre numeri a, b e c. */
    public static int maggiore(int a, int b, int c) {
        int maxAB;     // il maggiore tra a e b
        int maxABC;    // il maggiore tra a, b e c
        maxAB = Maggiore.maggiore(a,b);
        maxABC = Maggiore.maggiore(maxAB,c);
        return maxABC;
    }
}
```

Auto-referenziazione

```
/* Calcola il maggiore tra i tre numeri a, b e c. */
  public static int maggiore(int a, int b, int c) {
      int maxAB;     // il maggiore tra a e b
      int maxABC;    // il maggiore tra a, b e c

      maxAB = Maggiore.maggiore(a,b);
      maxABC = Maggiore.maggiore(maxAB,c);
      return maxABC;
  }
```

- **L'oggetto Maggiore, per calcolare il maggiore tra tre numeri, chiede a se stesso di calcolare il maggiore tra due numeri**
  - si parla di auto-referenziazione
  - è possibile e comune

Auto-referenziazione

In caso di auto-referenziazione

- nell'invocazione del metodo è possibile omettere l'oggetto a cui viene richiesta l'esecuzione del metodo
  - **è implicitamente l'oggetto stesso**

```
/* Calcola il maggiore tra i tre numeri a, b e c. */
  public static int maggiore(int a, int b, int c) {
      int maxAB;     // il maggiore tra a e b
      int maxABC;    // il maggiore tra a, b e c

      maxAB = maggiore(a,b);
      maxABC = maggiore(maxAB,c);
      return maxABC;
  }
```

equivalente a **Maggiore.maggiore(a,b)**

... Auto-referenziazione

- **L'auto-referenziazione consente di scrivere espressioni più compatte ma ancora leggibili**
  - **in questo caso possono usare anche la composizione dell'invocazione di metodi**

```
/* Calcola il maggiore tra tre numeri. */
  public static int maggiore(int a, int b, int c) {
      return maggiore( maggiore(a,b), c );
  }
```

Visibilità delle variabili

- **Una variabile locale a un metodo è visibile (ovvero, può essere acceduta) solo nell'ambito di una specifica attivazione di quel metodo**
  - il seguente frammento di codice - che ha lo scopo di visualizzare il valore della variabile a - non è corretto

```
public static void alfa() {
      int a;
      a = 5;
      beta();                        // invoca un altro metodo
  }

  public static void beta() {
      System.out.println(a);     // NO, a non è visibile
  }
```

  - il metodo beta dovrebbe avere un parametro

Visibilità delle variabili

Un altro esempio di codice non corretto

```
public static void gamma() {
     doppio(5);                  // invoca un altro metodo
     System.out.println(d);      // NO, d non è visibile
  }

  public static void doppio(int n) {
     int d;    // il doppio di n
     d = n*2;
  }
```

- il metodo doppio dovrebbe restituire un valore

Scambio di dati tra metodi

- **Come può un metodo scambiare dati con un altro metodo?**
  - un metodo non può scambiare dati con un altro metodo mediante le variabili locali
  - un metodo può fornire dati a un altro metodo sotto forma di parametri
  - un metodo può restituire un singolo dato al metodo che l'ha invocato
  - altre modalità di scambio di dati tra metodi saranno studiati nel contesto della definizione di classi per istanziare oggetti
- **Se i parametri sono tipi primitivi il metodo deve restituire un valore oppure eseguire delle stampe**
- **Se i parametri sono oggetti essi possono essere modificati perché è solo il riferimento che viene passato per valore.**

Esercizio

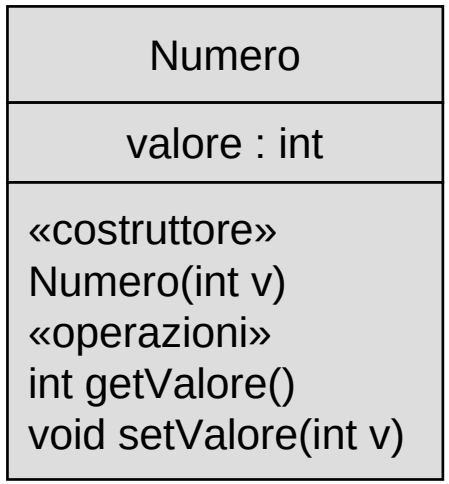
- **Quale risultato viene stampato dal seguente programma?**

```
class EsVisibilita1{
  public static void gamma() {
    int d=doppio(5);
    System.out.println(d);
  }
  public static int doppio(int n) {
    int d;
    return d = n*2;
  }
  public static void main(String[] args){
        int d=0;
        gamma();
        System.out.println("d= "+d);
    }
}
```

Effetti collaterali

- **Un effetto collaterale provocato dall'esecuzione di un metodo (o costruttore) produce la modifica di una variabile che non è locale al metodo (o costruttore)**

Esempio di effetti collaterali . . .



- un oggetto **Numero** rappresenta un numero intero, con il suo valore
  - **il valore di un oggetto Numero viene memorizzato in una variabile d'istanza associata all'oggetto**
- **num = new Numero(3)** crea un nuovo oggetto **num** di tipo **Numero** che rappresenta il numero **3**
- **num.getValore()** restituisce il valore rappresentato dall'oggetto **num**
- **num.setValore(8)** modifica il valore rappresentato dall'oggetto **num**

. . . Esempio di effetti collaterali

```
Numero a;
  a = new Numero(3);
  System.out.println( a.getValore() );    // 3
```

- l'esecuzione del costruttore modifica il valore della variabile d'istanza dell'oggetto Numero
  - è un effetto collaterale perchè a non è una variabile locale del costruttore

```
a.setValore(4);
  System.out.println( a.getValore() );    // 4
```

- l'esecuzione del metodo setValore modifica il valore della variabile d'istanza dell'oggetto Numero
  - è un effetto collaterale perchè a non è una variabile locale del costruttore
- l'esecuzione del metodo getValore non ha effetti collaterali

Altri effetti collaterali

- **Consideriamo il caso in cui un oggetto è referenziato da più variabili**

```
Numero a,b;
 a = new Numero(4);
 b = a;    // b referenzia lo stesso oggetto
           // referenziato da a
 System.out.println( b.getValore() );    // 4
```

- si considerino anche le seguenti istruzioni

```
a.setValore(5);
 System.out.println( a.getValore() );    // 5
 System.out.println( b.getValore() );    // 5
```

- la modifica dell'oggetto referenziato da **a** ha modificato anche l'oggetto referenziato da **b**
  - **in realtà si tratta dello stesso oggetto**
- gli effetti collaterali più spiacevoli sono quelli che modificano lo stato di oggetti accessibili da più variabili

Effetti Collaterali

- **Possiamo suddividere gli effetti collaterali in tre categorie:**
  - **modifica dello stato dell'oggetto a cui viene inviato un messaggio per invocare un metodo o un costruttore**
  - **modifica dello stato di un oggetto passato come parametro attuale a un metodo**
  - **modifica dello stato di un oggetto che non è stato passato come parametro ne è quello a cui è stato inviato il messaggio (da evitare)**
- **Gli effetti collaterali vanno compresi e usati opportunamente**
- **La programmazione orientata ad oggetti non sarebbe possibile senza effetti collaterali**

Metodi di supporto…

- **Nella realizzazione di un metodo M può essere utile definire altri metodi per semplificare la scrittura di M**
  - **si tratta di metodi di supporto**
    - **non implementano operazioni**
    - **sono di supporto alla definizione di altri metodi**
- **Voglio definire una operazione (metodo) per verificare se un numero intero positivo N è perfetto**
  - N è perfetto se la somma dei suoi divisori (N escluso) è uguale a N
    - **6 = 1+2+3, 10 ≠ 1+2+5**
  - è utile un metodo di supporto per calcolare la somma dei divisori di un numero intero positivo N

…Metodi di supporto…

```
/* Verifica se il numero n è perfetto. */
  public static boolean perfetto(int n) {
      // pre: n>0
      return n==sommaDivisori(n);
  }

  /* Calcola la somma dei divisori interi di n,
   * n escluso. */
  private static int sommaDivisori(int n) {
      // pre: n>0
      int sd;    // somma dei divisori di n
      int d;     // per iterare da 1 a n-1

      sd = 0;
      for (d=1; d<n; d++)
          if (n%d==0)
              sd = sd+d;
      return sd;
  }
```

Il modificatore private

```
private static int sommaDivisori(int n) { ... }
```

- **per il metodo sommaDivisori è stato usato il modificatore private**
  - **sommaDivisori** è un **metodo privato**
- **i metodi privati non definiscono operazioni**
  - **possono essere invocati solo da istruzioni scritte nella stessa classe**
  - non possono essere invocati da istruzioni scritte al di fuori della classe
- **la privatezza dei metodi è definita nell'ambito di una classe**

Legame dei parametri

- Il legame dei parametri è l'associazione che avviene tra parametri attuali e parametri formali durante l'attivazione di un metodo

- Nei linguaggi di programmazione esistono due modalità principali di legame dei parametri
  - nel legame dei parametri per valore ciascuna variabile parametro formale prende come valore iniziale il valore del parametro attuale nella posizione corrispondente
  - nel legame dei parametri per riferimento ciascuna variabile parametro formale è usata come riferimento alla variabile usata come parametro attuale nella posizione corrispondente
- Nel linguaggio Java esiste solo la modalità di legame per valore

Legame dei parametri per valore

- **Nel legame dei parametri per valore**
  - **i parametri attuali sono espressioni**
  - **di cui interessa il valore al momento dell'invocazione del metodo**

Legame per valore e parametri di tipo primitivo

- Quali valori vengono stampati dalla seguente applicazione?

```
class LegamePerValore1 {
    public static void alfa(int m) {
        System.out.println(m);    // sicuramente 5
        m = m+1;
        System.out.println(m);    // sicuramente 6
    }
    public static void main(String[] args) {
        int k = 5;
        System.out.println(k);    // sicuramente 5
        alfa(k);
        System.out.println(k);    // 5 oppure 6?
    }
}
```

- Il parametro attuale nell'invocazione `alfa(k)` è un espressione di cui ci interessa solo il valore assunto al momento dell'invocazione; il parametro viene valutato e il suo valore viene usato come valore iniziale del parametro formale.

Legame per valore e parametri di tipo primitivo

- Quali valori vengono stampati dalla seguente applicazione?

```
class LegamePerValore2 {
    public static void alfa(int k) {
        System.out.println(k);    // sicuramente 5
        k = k+1;
        System.out.println(k);    // sicuramente 6
    }
    public static void main(String[] args) {
        int k = 5;
        System.out.println(k);    // sicuramente 5
        alfa(k);
        System.out.println(k);    // 5 oppure 6?
    }
}
```

- Le due variabili k sono due variabili distinte anche se hanno lo stesso nome perchè sono locali a due metodi diversi

Legame per valore e parametri di tipo riferimento

- Quali valori vengono stampati dalla seguente applicazione?

```
class LegamePerValore3 {
    public static void alfa(Numero m) {
        System.out.println(m.getValore());  // sicuramente 5
        m = new Numero(6);
        System.out.println(m.getValore());  // sicuramente 6
    }
    public static void main(String[] args) {
        Numero k = new Numero(5);
        System.out.println(k.getValore());  // sicuramente 5
        alfa(k);
        System.out.println(k.getValore());  // 5 oppure 6?
    }
}
```

- Con l'istruzione `m=new Numero(6)` si fa referenziare **m** ad un altro oggetto. **k** referenzia ancora l'oggetto creato per primo che non è mai stato modificato.

Legame per valore e parametri di tipo riferimento

- Quali valori vengono stampati dalla seguente applicazione?

```
class LegamePerValore4 {
    public static void alfa(Numero m) {
        System.out.println(m.getValore());  // sicuramente 5
        /* modifica lo stato dell'oggetto
         * passato come parametro */
        m.setValore(6);
        System.out.println(m.getValore());  // sicuramente 6
    }
    public static void main(String[] args) {
        Numero k = new Numero(5);
        System.out.println(k.getValore());  // sicuramente 5
        alfa(k);
        System.out.println(k.getValore());  // 5 oppure 6?
    }
}
```

- Il metodo alfa ha l'effetto collaterale di modificare lo stato dell'oggetto a cui la variabile m fa riferimento.

Legame per valore e parametri di tipo riferimento

- Quali valori vengono stampati dalla seguente applicazione?

```
class LegamePerValore5 {
    public static void alfa(Quadrato q) {
        System.out.println(q.getLato());  // sicuramente 5
        q = new Quadrato(6);
        System.out.println(q.getLato());  // sicuramente 6
    }
    public static void main(String[] args) {
        Quadrato k = new Quadrato(5);
        System.out.println(k.getLato());  // sicuramente 5
        alfa(k);
        System.out.println(k.getLato());  // 5 oppure 6?
    }
}
```

Legame per valore e parametri di tipo riferimento

- Quali valori vengono stampati dalla seguente applicazione?

```
class LegamePerValore6 {
    public static void alfa(Quadrato q) {
        System.out.println(q.getLato());  // sicuramente 5
        q.setLato(6);
        System.out.println(q.getLato());  // sicuramente 6
    }
    public static void main(String[] args) {
        Quadrato k = new Quadrato(5);
        System.out.println(k.getLato());  // sicuramente 5
        alfa(k);
        System.out.println(k.getLato());  // 5 oppure 6?
    }
}
```

Considerazioni sul legame dei parametri per valore

- **Nel legame dei parametri per valore**
  - **i parametri attuali sono delle espressioni**
    - **è possibile usare come parametro attuale anche espressioni formate semplicemente dal nome di una variabile**
  - **se come parametro attuale viene utilizzata una espressione formata semplicemente dal nome di una variabile, il valore di questa variabile non può essere modificato dal metodo invocato**
  - **se il parametro attuale è un riferimento a un oggetto, il metodo invocato può modificare lo stato dell'oggetto che gli è stato passato come parametro**
    - **lo stato di un oggetto passato come parametro può essere modificato**
    - **invece, il riferimento a un oggetto passato come parametro non può essere modificato**

Cosa abbiamo visto finora

- Definizione di nuovi metodi
- Parametri dei metodi
- Parametri formali e parametri attuali
- Passaggio di parametri
- Effetti collaterali
- Metodi di supporto
- Il legame per valore con i parametri di tipo primitivo e di tipo riferimento

Riferimenti al libro di testo

- Per lo studio di questi argomenti si fa riferimento al libro di testo, e in particolare al capitolo 15