Alcune note sul linguaggio Java*

Demis Ballis

Dip. Matematica e Informatica, Via delle Scienze 206, 33100 Udine, Italy.
demis@dimi.uniud.it.

1 Introduzione

In questa dispensa si illustra, in modo sintetico, la sintassi dei costrutti di base del linguaggio Java, a partire dagli elementi costitutivi più piccoli (i token) fino alle classi, alle interfacce (che già conoscete) e ai package. Questa dispensa può servirvi da "manuale di riferimento" per la scrittura corretta di programmi Java, anche se non pretende assolutamente di essere una trattazione esaustiva del linguaggio. Siete invitati ad approfondire lo studio degli aspetti del linguaggio che qui sono stati omessi attraverso la consultazione della documentazione disponibile in rete o la lettura di un libro su Java (vedi `http://www.dimi.uniud.it/demis` per maggiori informazioni).

2 Costrutti fondamentali di Java

I *token* sono gli elementi costitutivi elementari del linguaggio Java, ossia i piú piccoli elementi di programma comprensibili per il compilatore. Possono essere suddivisi in cinque categorie:

1. identificatori;
2. parole chiave;
3. letterali;
4. operatori;
5. separatori.

Oltre a questi elementi un programma Java può includere commenti e spazi bianchi, che il compilatore riconosce e ignora. Il compilatore è un programma che legge il codice sorgente Java, interpreta i token e traduce il programma in un linguaggio di piú basso livello (cioè piú vicino al linguaggio macchina), detto *bytecode*. Il bytecode può essere interpretato ed eseguito dalla macchina virtuale di Java (abbreviata JVM). La JVM astrae le differenze tra i processori fisici in un unico "processore virtuale", in modo che un programma Java possa essere eseguito senza cambiamenti su qualunque calcolatore che disponga di una JVM (per questo motivo si parla di *portabilità* del codice).

* Questo documento è una versione rivista e adattata per gli studenti del Corso di Laurea in Scienze e Tecnologie Multimediali di una precedente dispensa scritta dal Dott. Nicola Vitacolonna.

2.1 Identificatori

Gli identificatori rappresentano nomi. Questi nomi possono essere assegnati a variabili, metodi, classi e cosí via per poterli identificare in modo univoco. Java distingue le lettere maiuscole da quelle minuscole (per cui `Ciao` è diverso da `ciao`). Inoltre ogni identificatore può essere composto esclusivamente da lettere, numeri e dai caratteri `'_'` e `'$'`, e deve cominciare con una lettera, con `'_'` oppure `'$'` (perciò `1studio` non è un identificatore valido). Inoltre un identificatore non può essere uguale a una parola chiave (si veda la Sezione 2.2), a un letterale booleano o a un letterale nullo (si veda la Sezione 2.3). Esempi di identificatori sono:

```
x   y1   Studente   String   VALORE_MASSIMO
```

Evitate l'uso del simbolo `'$'`.

2.2 Parole chiave

Le parole chiave hanno un uso e un significato speciale in Java. L'elenco delle parole chiave di Java è riportato in Tabella 1. Già vi siete imbattuti in alcune

```
abstract default if           private   this
boolean  do      implements   protected throw
break    double  import       public    throws
byte     else    instanceof   return    transient
case     extends int          short     try
catch    final   interface    static    void
char     finally long         strictfp  volatile
class    float   native       super     while
const    for     new          switch    continue
goto     package synchronized
```

Tabella1. Lista delle parole chiave del linguagigo Java.

di queste, altre imparerete a conoscerle durante queste esercitazioni. Si noti che `true`, `false` e `null` tecnicamente sono letterali (i.e., valori costanti), non parole chiave (si veda la prossima sezione).

2.3 I Letterali

I *letterali* si distinguono in

1. numeri interi (ad esempio, `1230`);
2. numeri decimali, detti anche numeri in virgola mobile (ad esempio, `2.34`: si noti il punto usato come separatore decimale);
3. i valori booleani `true` e `false`, per il vero e per il falso rispettivamente;

4. singoli caratteri *Unicode*, rappresentati tra apici singoli (ad esempio, `'a'`);
5. stringhe, cioè sequenze di zero o piú caratteri racchiuse tra virgolette doppie (ad esempio, `atomo`);
6. il riferimento nullo `null`.

Le stringhe in Java sono rappresentate da oggetti della classe String.

2.4 Operatori

Gli *operatori* indicano un tipo di calcolo da eseguire su uno o piú dati, detti operandi, che possono essere letterali, variabili o risultati di valutazioni di espressioni (si vedano le sezioni succerssive). Questo è l?elenco degli operatori in Java:

```
=     >     <     !      ~     ?     :
==    <=    >=    !=    &&    ||    ++      --
+      -    *     /     &     |     ^
+=    -=    *=    /=    &=    |=    ^=
%     <<    >>    >>>
%=    <<=   >>=   >>>=
```

2.5 Separatori

I separatori sono i “segni di punteggiatura” del linguaggio. Eccone l?elenco:

```
(   )   {   }   [   ]   ;   ,   .
```

2.6 Commenti e spazi bianchi

Gli spazi bianchi (spazi, caratteri di tabulazione, caratteri di nuova riga) sono ignorati dal compilatore. Attenzione però che un identificatore non può contenere spazi, ovvero `studentestraniero` e `studente straniero` non sono la stessa cosa; in particolare il secondo esempio viene interpretato dal compilatore come *due* identificatori distinti!.

In tuttio gli altri casi, è possibile inserire spazi e andare a capo a piacere. Vi sono tuttavia regole di buona scrittura dei programmi che è bene rispettare e che vedremo piú avanti. Anche i commenti sono ignorati dal compilatore. I commenti servono a chi scrive un programma per rendere il codice maggiormente comprensibile. I commenti possono essere definiti in tre modi, come mostra la tabella 2.
Vedremo piú avanti come generare automaticamente la documentazione del codice con Javadoc.

Tipo commento	Utilizzo
`/* commento */`	Tutti i caratteri fra `/*` e `*/` sono ignorati.
`// commento`	Tutti i caratteri da `//` a fine linea sono ignorati.
`/** commento */`	Come `/* */` , ma permette di creare la documentazione *Javadoc* del codice in modo automatico.

Tabella2. Commenti in Java

2.7 Esercizi

In BlueJ, create un nuovo progetto e all'interno del progetto create una nuova classe. Aprite la classe con l'editor. Provate a dichiarare identificatori il cui nome non cominci con una lettera, con `'_'` o con `'$'`. Compilate. Provate a dichiarare identificatori il cui nome coincida con una parola chiave. Compilate nuovamente. Che cosa succede? Esaminate il codice che BlueJ genera per voi quando create una classe. Siete in grado di individuare e classificare i vari token?

3 Tipi di dato e variabili

Un tipo di dato è una coppia (S, Op) in cui S è un insieme di valori (il *supporto* del tipo) e Op è un insieme di *operatori* definiti su S. Un tipo di dato informa sul modo in cui il contenuto della memoria è interpretato. Java è un linguaggio fortemente tipato, perché ogni variabile e ogni espressione ha associato un tipo, noto a tempo di compilazione. I tipi limitano i valori che una variabile può contenere e i valori che un'espressione può produrre, limitano le operazioni eseguibili su tali valori e ne determinano il significato.

Esistono in Java due categorie di tipi di dato: i tipi *primitivi* e i tipi *riferimento*. I tipi primitivi sono:

1. il tipo booleano `boolean` (valori di verità *true* e *false*);
2. i tipi per i numeri interi `byte` (8 bit), `short` (16 bit), `int` (32 bit) e `long` (64 bit);
3. i tipi per i numeri in virgola mobile (i.e. frazionari) `float` (32 bit) e *double* (64 bit);
4. il tipo carattere `char`.

Gli operatori applicabili ai tipi primitivi sono trattati nella Sezione 5.

I tipi riferimento sono classi, interfacce e vettori (o array) (gli array sono discussi nella Sezione 6). Il supporto di un tipo riferimento è un insieme di riferimenti ad oggetti.

Una *variabile* è una locazione della memoria a cui è stato attribuito un nome, in grado di memorizzare un valore. Una variabile di tipo primitivo può contenere solo valori appartenenti al supporto del tipo. Una variabile di tipo T, dove T è una classe, può contenere un riferimento a una qualunque istanza (oggetto) di T oppure il valore `null`. Una variabile di tipo T, dove T è un'interfaccia, può contenere un riferimento a un?istanza di una qualunque classe che implementi T, oppure il valore `null`. Le variabili di tipo array sono discusse nella Sezione 6.

L'associazione di una variabile a un tipo ha luogo al momento della dichiarazione della variabile. Una variabile si dichiara specificandone il tipo seguito dal nome della variabile, secondo la sintassi seguente (il punto e virgola al termine della dichiarazione è obbligatorio!):

⟨tipo⟩ ⟨identificatore⟩;

Se piú variabili hanno lo stesso tipo, è possibile dichiararle insieme separandole da una virgola, secondo lo schema seguente:

⟨tipo⟩ ⟨identificatore$_1$⟩,⟨identificatore$_2$⟩,. . . ;

Esempi di dichiarazioni di variabili sono i seguenti:

```
int n; // Dichiara una variabile di tipo intero
char c; // Dichiara una variabile di tipo carattere
String s; // Dichiara una variabile di tipo riferimento
int i, j, k; // Dichiara tre variabili di tipo intero
```

La dichiarazione è obbligatoria prima di poter usare un identificatore.

3.1 Esercizi

Nella classe creata precedentemente, dichiarate e inizializzate le seguenti variabili (vi ricordo che è lecito combinare una dichiarazione con un'istruzione di assegnamento)[1]:

```
int i = 101;
long l = 20000000000000L;
float f = 12.78f;
double d = 34.999;
char c = 'a';
```

Aggiungete inoltre il seguente metodo:

```
void convertiTipo() { }
```

Considerate le seguenti istruzioni di assegnamento:

```
i = l;      d = f;
l = i;      f = d;
f = i;      i = c;
l = f;      c = i;
```

[1] Si presti attenzione all'assegnamento di un `float` o di un `long`: un numero decimale è interpretato come un numero di tipo `double`, a meno che non sia seguito dalla lettera `F` o `f`. Ad esempio: `3.2` è un letterale di tipo `double`, mentre `3.2f` è un letterale di tipo `float`. Per specificare che un intero è di tipo `long` si fa seguire il numero dalla lettera `L` o `l`.

Ripetete, per ciascuno degli assegnamenti, quanto segue. All'interno del metodo `convertiTipo()` scrivete uno degli assegnamenti. Dopo l'assegnamento, scrivete un'istruzione che stampi su schermo il contenuto della variabile a sinistra nell'assegnamento. Provate a compilare e, se la compilazione ha successo, create un'istanza della classe ed invocate convertiTipo(). Riuscite a compilare? Se sí, che cosa viene stampato?

Osservazione: il processo di conversione da un tipo di dati ad un altro si chiama *casting*. Il casting può essere *implicito* (è il caso degli assegnamenti precedenti) o *esplicito*: in tal caso il tipo in cui si vuol convertire una variabile dev'essere esplicitamente indicato tra parentesi: ad esempio, `c = (char)i`. Il casting implicito è lecito quando non comporta perdita di informazione. Sulla base di questa osservazione, siete in grado di spiegare i risultati dell'esperimento di cui sopra? Il casting esplicito causa in generale perdita di informazione e può generare errori a tempo di esecuzione, quindi, se possibile, è da evitare. Riuscite ad effettuare gli assegnamenti che vi hanno creato problemi mediante casting esplicito?

4 Blocchi e ambito

In Java, il codice sorgente è diviso in porzioni separate da parentesi graffe aperte e chiuse. Tutto quanto è compreso tra una coppia di parentesi graffe è chiamato *blocco*. I blocchi sono importanti sia da un punto di vista logico (permettendo di raggruppare le istruzioni del programma) sia dal punto di vista della sintassi, perché consentono al compilatore di distinguere le varie sezioni del codice. I blocchi possono essere annidati gli uni dentro gli altri. Esempi di blocchi sono il corpo di una classe e il corpo di un metodo (che è annidato nel corpo di una classe). I blocchi sono importanti anche perché strettamente collegati alla nozione di *ambito* di una variabile. Finora abbiamo visto che è possibile dichiarare variabili nel corpo di una classe: queste variabili rappresentano gli attributi degli oggetti di quella classe. In realtà, in Java è possibile dichiarare variabili all?interno di qualunque blocco. Ad esempio, è possibile inserire la dichiarazione di una variabile nel corpo di un metodo:

```
void unMetodo()
{
  int n; // La variabile n è dichiarata nel blocco
         // definito dal corpo del metodo
}
```

Variabili di questo tipo sono dette variabili *locali*, e non rappresentano attributi di oggetti, ma sono usate come"contenitori temporanei". Infatti, l?ambito in cui queste variabili possono essere utilizzate è limitato dal blocco in cui sono immediatamente contenute. Ad esempio, quando l'esecuzione di `unMetodo()` ha termine, la variabile `n` viene distrutta (cioè lo spazio di memoria corrispondente è deallocato) e non è piú disponibile.

4.1 Esercizi

All'interno di una classe, scrivete i due seguenti metodi:

```
void metodoA()
{
  int n;
  n = 1;
  System.out.println(n);
}

void metodoB()
{
  System.out.println(n);
}
```

Compilate la classe e controllate il messaggio d?errore ricevuto. Sapete spiegarlo? Ora aggiungete all'inizio del corpo della classe la dichiarazione `int n = 200;` e ricompilate la classe. Ora la compilazione ha successo. Riuscite a capire perché? Create un'istanza della classe e invocate `metodoA()` e `metodoB()`, controllando i valori stampati. Sapete spiegare l'output?

Osservazione: nel codice che avete scritto sono presenti due dichiarazioni della variabile n, una nel corpo della classe e una nel corpo di `metodoA()`. Ciò non crea alcun conflitto, perché la dichiarazione nel metodo si trova in un blocco più interno rispetto alla dichiarazione nella classe, e "nasconde" quest'ultima all'interno del metodo. Ciò significa che la variabile usata in `metodoA()` è quella locale al metodo stesso, e il cui valore è 1. La variabile n in `metodoB()` è invece la variabile definita nel corpo della classe, e il cui valore è 200.

5 Espressioni e operatori

Un'*espressione* permette di calcolare un valore a partire da un certo numero di operandi, che possono essere variabili, letterali, valori di ritorno di metodi o altre espressioni. Ecco un esempio:

```
2*x + y
```

L?espressione precedente moltiplica (`*`) il contenuto della variabile `x` per due e somma (`+`) al risultato il contenuto della variabile `y`. Chiaramente, il tipo delle variabili deve essere compatibile con le operazioni applicate (con riferimento all?esempio precedente, `x` e `y` possono essere di tipo `int`, `float`, `double`, ma non `boolean`). Tipicamente, il risultato di un'espressione è memorizzato in una variabile, attraverso l'uso dell'operatore di assegnamento `=`, nel modo seguente:

```
risultato = 2*x + y;
```

Operatore	Descrizione
`+`	addizione
`-`	sottrazione
`*`	moltiplicazione
`/`	divisione
`%`	modulo (resto della divisione fra interi)

Tabella3. Operatori aritmetici

Gli operatori più semplici sono quelli che corrispondono alle operazioni aritmetiche, riportati in tabella 3. L'operatore di modulo restituisce il resto della divisione intera: ad esempio, `17 % 5` dà come risultato `2`. L'operatore di divisione `/` dà un risultato intero (il quoziente della divisione) se gli operandi sono interi; dà come risultato un numero in virgola mobile se gli operandi sono numeri in virgola mobile.

Altri operatori importanti sono gli operatori relazionali che confrontano due espressioni e calcolano il valore booleano corrispondente al risultato del confronto. Ad esempio, l'espressione `3 < 5` ha valore `true`, mentre l'espressione `1 > 2` ha valore `false`. Gli operatori relazionali sono riportati in tabella 4. Si noti il

Operatore	Descrizione
`<`	minore
`<=`	minore o uguale
`>`	maggiore
`>=`	maggiore o uguale
`==`	uguale
`!=`	diverso

Tabella4. Operatori relazionali

modo in cui è rappresentata l'uguaglianza (`==`). Uno degli errori piú comuni è confondere l'operatore di assegnamento `=` (che assegna un valore a una variabile) con l'operatore di uguaglianza `==` (che confronta due valori). Alle espressioni di tipo booleano si applicano gli operatori di tabella 5. Ad esempio, l'espressione

Operatore	Descrizione
&&	and logico o congiunzione
|||	or logico o disgiunzione
!	negazione

Tabella5. Operatori logici

$(3 < 4)\&\&(x > 10)$ dà come risultato `true` se `x` contiene un valore maggiore di dieci; altrimenti, dà come risultato `false`. L'espressione `!(5 < 6)` dà come

risultato `false`. Invece l'espressione `(!(5 < 6)) == false` dà come risultato `true`.

Si noti l'uso delle parentesi per specificare l?ordine in cui eseguire la valutazione delle espressioni. Alcune parentesi possono di solito essere omesse, tenendo conto che le espressioni sono valutate da sinistra a destra e che tra gli operatori è definita una relazione di precedenza (ad esempio, `*` ha la precedenza su `+`). Se non si è sicuri della relazione di precedenza tra due operatori è opportuno usare le parentesi.

Terminiamo questa rassegna (incompleta!) degli operatori introducendo due operatori unari molto frequenti: `++` e `--`. Questi operatori sono usati, con notazione prefissa o postfissa, per incrementare o decrementare di uno il contenuto di una variabile. Ad esempio, se la variabile `x` contiene il valore `1`, l?espressione `x++` (o `++x`) fa sí che il contenuto della variabile `x` diventi `2`. Gli usi prefisso o postfisso di questi operatori *non* sono equivalenti. Si considerino, ad esempio, i due seguenti assegnamenti:

```
y = x++;  e   y = ++x;
```

La prima istruzione assegna alla variabiley il valore di `x` e poi incrementa il contenuto di `x` di uno. Il secondo assegnamento prima incrementa il contenuto di `x` di uno e poi assegna il nuovo valore di `x` a `y`.

5.1 Esercizi

1. Scrivete un metodo che sommi due orari del tipo `hh:mm` e stampi su schermo l'ora risultante. In altre parole, completate il seguente metodo inserendo le espressioni mancanti (suggerimento: si usino l'operatore di quoziente fra interi `/` e l'operatore di modulo `%`).

   ```
   void sommaOre(int hh1, int mm1, int hh2, int mm2)
   {
     System.out.println("Ore: " + ...);
     System.out.println("Minuti: " + ...);
   }
   ```

2. Scrivete un metodo `xor()` che riceva in input due valori booleani e restituisca l'xor (or esclusivo) dei due valori. L'operatore `xor` produce il valore `true` se e solo se uno dei due operandi è `true` e l'altro è `false`.

6 Vettori

Un *vettore*, o *array*, è un oggetto che fornisce lo spazio di memoria per un elenco di elementi dello stesso tipo. Il tipo di ciascuno degli elementi dell'array è detto tipo *componente*. Se il tipo componente è T, allora il tipo array si denota con $T[\,]$. Ad esempio,

```
int[] x; // Dichiara un array di interi
String[] s; // Dichiara un array di stringhe
char[] vocali; // Dichiara un array di caratteri
```

Il tipo di un array è un tipo riferimento. Un riferimento a un array si ottiene mediante l'operatore *new*. Quando si crea un array con un'istruzione `new` è necessario specificarne la dimensione, cioè il numero massimo di elementi che l'array potrà contenere. La sintassi è la seguente:

```
new tipo[dimensione]
```

Ad esempio:

```
x = new int[10]; // Crea un array di dieci interi
s = new String[3]; // Crea un array di tre stringhe
```

Un altro modo per creare un array consiste nel fornire esplicitamente l'elenco degli elementi, fra parentesi graffe e separati da virgole, al momento della dichiarazione, come nell'esempio seguente:

```
char[] vocali = {'a', 'e', 'i', 'o', 'u'};
int[] x = {2, 3, 5, 7, 11, 13};
```

Le istruzioni precedenti creano rispettivamente un array di cinque elementi, assegnando un valore (in questo caso, un carattere) a ciascun elemento, e un array di sei elementi, a ciascuno dei quali è assegnato un numero primo. Ciascuna componente di un array è una variabile. Queste variabili sono indicizzate usando il nome dell'array e un numero intero non negativo. Se un array ha n componenti, le componenti sono referenziate usando indici che vanno da 0 a $n-1$. Ad esempio, `vocali[0]` è una variabile che contiene il valore `'a'`, mentre `vocali[4]` contiene il valore `'u'`. Array multidimensionali possono essere creati usando più coppie di parentesi quadre al momento della dichiarazione. Ad esempio:

```
int[][] matrice; // Dichiara un array di array di interi
```

dichiara un vettore bidimensionale, cioè una matrice[2], di numeri interi. La creazione dell?oggetto avviene con un?istruzione del tipo

```
matrice = new int[3][2]; // Crea una matrice 3x2 di interi
```

dopodiché le singole componenti restano individuate da una coppia di indici: ad esempio, `matrice[0][0]` è la variabile corrispondente al primo elemento della matrice.

[2] Dati due numeri naturali m e n, una *matrice* $m \times n$ con valori in un insieme D è una funzione $t : 0, \ldots, m-1 \times 0, \ldots, n-1 \to D$, rappresentata solitamente come una tabella di elementi di D. Ad esempio, questa è una matrice 3×2 di interi:

$$\begin{pmatrix} -1 & 0 \\ 10 & -2 \\ -2 & 22 \end{pmatrix}$$

Example 1. Si scriva una classe `VotiEsame` che contenga un attributo `voti` di tipo vettore di interi e si implementino i seguenti metodi:

```
public VotiEsame(int n)     //costruttore che permette di allocare
                            //n posizioni per il vettore voti
public void generavoti()    //riempie il vettore voti con degli
                            //interi (pseudo)casuali compresi tra
                            //15 e 30
public void stampavoti()    //stampa il vettore voti
public int max()            //ritorna l'elemento massimo del vettore
                            //voti
public int min()            //ritorna l'elemento minimo del vettore
                            //voti
public double media()       //ritorna la media aritmetica degli
                            //elementi del vettore voti
public boolean trova(int x) //ricerca l'elemento x nel vettore voti
```

Soluzione.

```
import java.util.*;

public class VotiEsame
{
    // variabile di tipo vettore di interi
    private int voti[];

    public VotiEsame(int n)
    {
        // inizializza un vettore di lunghezza n
        voti = new int[n];
    }

    // genera un vettore contenente interi pseudo-casuali compresi tra 15 e 31
    public void generavoti()
    {
      int i=0;
      Random r=new Random();
      for (i=0;i<voti.length;i++)
         voti[i]=15+r.nextInt(16);
      System.out.println("Vettore generato!");
    }

    // stampa vettore
    public void stampavoti()
    {
      int i=0;
      for (i=0;i<voti.length;i++)
```

```
     System.out.println("Elemento in posizione "+i+": "+voti[i]);
}

//calcola il valore massimo del vettore di interi voti
public int max(){
    int i,max;
    max=voti[0];
    for(i=1;i<voti.length;i++)
       if (voti[i]>max)
            max=voti[i];
    return max;
}

//calcolare il valore minimo del vettore di interi voti
public int min(){
    int i,min;
    min=voti[0];
    for(i=1;i<voti.length;i++)
       if (voti[i]<min)
            min=voti[i];
    return min;
}

//calcola la media del vettore di interi voti
public double media(){
    int i;
    double a,m;
    a=0;
    for(i=1;i<voti.length;i++)
       a=a+voti[i];
    m=(a/voti.length);
    return m;
}

//cerca un voto x nel  vettore di interi voti
public boolean trova(int x)
{
  int i=0;
  boolean trovato=false;
  while (i<voti.length && !trovato)
  {
       if (voti[i]==x)
            trovato=true;
       i++;
  }
```

```
      return trovato;
    }
}
```

6.1 Esercizi

Scrivete un metodo (senza parametri) in cui vengano creati due array capaci di contenere ciascuno tre numeri decimali. Inizializzate i due array con valori arbitrari. Il metodo deve restituire il valore del prodotto scalare dei due array. Si ricorda che il prodotto scalare di due vettori $u = (u_1, \ldots, u_n)$ e $v = (v_1, \ldots, v_n)$ è dato da $\Sigma_{i=1}^{n} u[i] * v[i]$.

7 Strutture di controllo

Quando un metodo viene invocato, le istruzioni che compongono il corpo del metodo sono eseguite una dopo l'altra sequenzialmente. Esistono due modi per modificare questo flusso di esecuzione: le *diramazioni* e i *cicli*. Le diramazioni permettono di selezionare la parte di un programma da eseguire sulla base di una certa condizione. I cicli permettono di ripetere una parte di programma molte volte.

7.1 Istruzioni condizionali

Le diramazioni sono implementate attraverso istruzioni condizionali. La piú comune istruzione condizionale è l'istruzione `if-else`, la cui sintassi è:

```
if ( condizione )
  { istruzione1 }
else
  { istruzione2 }
```

La condizione dev'essere un'espressione il cui risultato è di tipo booleano. Se la condizione è valutata `true`, viene eseguito il blocco `istruzioni1`, altrimenti viene eseguito il blocco `istruzioni2`. Il ramo `else` può essere assente: in tal caso, l'istruzione condizionale assume la forma:

```
if (condizione)
    { istruzioni }
```

il cui significato è: esegui `istruzioni` solo se `condizione` è `true`. Le parentesi graffe possono essere omesse quando il corpo dell'istruzione `if`, o dell'istruzione `if-else`, è costituito da un'unica istruzione. Come esempio, si consideri il seguente frammento di codice:

```
if (sonoStanco)
   devoDormire = true;
else
   devoDormire = false;
```

Se `sonoStanco` ha valore `true`, allora la variabile `devoDormire` viene impostata su `true`. Altrimenti, a `devoDormire` è assegnato il valore `false`. È possibile unire piú diramazioni `if-else` per esprimere piú possibilità condizionali, con la sintassi seguente:

```
if (condizione1)
    { istruzioni1 }
else if (condizione2)
    { istruzioni2 }
 [...]
else
    { istruzionin }
```

Spesso esiste un modo piú conveniente per esprimere istruzioni condizionali con molte alternative, basato sull'uso dell'istruzione `switch`. La sintassi è la seguente:

```
switch (espressione)
{
  case letterale1: istruzioni1
  case letterale2: istruzioni2
  [...]
  case letteralen-1: istruzionin-1
  default: istruzionin
}
```

La valutazione dell'espressione deve produrre un risultato di tipo `byte`, `short`, `int` o `char`. Il tipo di ciascun letterale dev'essere compatibile con il tipo di espressione. L'istruzione `switch` è eseguita in questo modo: prima di tutto viene valutata l'espressione. Il risultato viene poi confrontato con il letterale di ciascun `case`, nell'ordine in cui tali valori compaiono. Se il letterale dell'i-esimo case coincide col valore dell'espressione, allora tutte le istruzioni `istruzionii` sono eseguite. Se il valore dell'espressione non coincide con il valore di alcun `case`, allora vengono eseguite le istruzioni dopo la clausola `default`.

Si presti attenzione al fatto che, se si vuole eseguire solo il blocco di istruzioni `istruzionii` corrispondente al letterale `i` uguale a `espressione`, bisogna terminare tale blocco di istruzioni con la parola chiave `break`. Vediamo un esempio (si assume che la variabile `x` sia stata precedentemente dichiarata):

```
x = 1;
switch (x)
{
  case 0: System.out.print("Zero ");break;
  case 1: System.out.print("Uno ");break;
  default: System.out.print("Default");
}
```

Questo frammento di codice farà stampare la stringa `Uno`, perché il valore di `x` è 1 e quindi corrisponde al valore del secondo `case`. Si consideri invece il codice seguente:

```
x = 5;
switch (x)
{
  case 0: System.out.print("Zero "); break;
  case 1: System.out.print("Uno "); break;
  default: System.out.print("Default");
}
```

Questo codice stamperà la stringa `Default`.

Example 2. Si scriva una classe di nome `Cirano` per la generazione di *lettere d'amore* casuali. Nella classe si definisca un metodo che sia in grado di stampare casualmente una delle lettere componenti la stringa `AMORE`.
Soluzione.

```
import java.util.*;

public class Cirano
{
    private char x;

    /* ritorna un numero intero (pseudo)casuale tra 0 e 4 */
    public int random04()
    {
      Random r;
      int k;
      r= new Random();
      k= Math.abs((r.nextInt())%5);
      return k;
    }

    /* genera una lettera d'amore e la assegna all'attributo x */
    public void genera()
    {
     int k;
     k=random04();
     if (k==0)
        x='A';
     else if  (k==1)
        x='M';
     else if  (k==2)
        x='O';
     else if  (k==3)
        x='R';
     else
       x='E';
    }
```

```
    /* stampa il valore dell'attributo x */
    public void stampa()
    {
     System.out.println("La lettera d'amore generata e': "+x);
    }
}
```

7.2 Esercizi

1. Scrivete un metodo per il calcolo delle soluzioni di un'equazione di secondo grado. Si ricorda che, data un'equazione $ax^2 + bx + c = 0$, le soluzioni si calcolano con la formula

$$x_{1,2} = -b \pm \frac{\sqrt{b^2 - 4ac}}{2a}.$$

   Il metodo deve ricevere tre parametri che corrispondono ai tre coefficienti a, b e c. Deve stampare un messaggio d'errore se $a = 0$. Deve inoltre determinare, in base al valore del discriminante $b^2 - 4ac$, se esistono soluzioni reali e se queste coincidono, e stampare i risultati sullo schermo.
   Nota: usate il metodo `Math.sqrt()` per calcolare la radice quadrata di un'espressione. Ad esempio, `Math.sqrt(4.0)` produce il valore `2.0`.

2. Scrivete un metodo che riceva come parametro un carattere (tipo `char`) e stampi sullo schermo la categoria cui la lettera appartiene, secondo la classificazione seguente:

Lettere	**Categoria**
a, e, i, o, u	vocale
j	semivocale
p, b, m	labiale
f, v	labiodentale
t, d, z, n, s, r, l	dentale
k	gutturale
altre lettere	consonante

7.3 Istruzioni iterative

I cicli, cioè l'esecuzione ripetuta di una parte di codice, sono implementati mediante le istruzioni iterative. In Java, ne esistono di tre tipi: i cicli `while`, i cicli `do-while` e i cicli `for`. Un ciclo `while` ha la seguente sintassi:

```
while (condizione)
{
  istruzioni
}
```

La condizione dev'essere un'espressione booleana. L'istruzione *while* è interpretata come segue: se la condizione è valutata `true` vengono eseguite le istruzioni fra parentesi graffe e il processo ricomincia da capo. Il ciclo ha termine non appena`condizione` è valutata `false`. È chiaro che l'esecuzione delle istruzioni (il corpo del `while`) deve in qualche modo influire sulla condizione, in modo che prima o poi il ciclo abbia termine. Se la parte `istruzioni` è composta da una singola istruzione, le parentesi graffe possono essere omesse. Il seguente esempio stampa i numeri da 1 a 10 usando un ciclo `while` (si assume che la variabile `i` sia stata precedentemente dichiarata):

```
i = 1; // Inizializza la variabile i
while (i <= 10)
{ // Finché i è minore o uguale a 10
  System.out.println(i); // stampa il valore di i
  i++; // e incrementa i di uno
}
// Attenzione! Alla fine del ciclo, i vale 11
```

La sintassi del ciclo `do-while` è:

```
do
{
  istruzioni
}
while (condizione);
```

dove `condizione` è un'espressione booleana. A differenza del ciclo `while`, in questo caso le istruzioni da ripetere sono eseguite prima della valutazione della condizione. Ciò implica che la sezione istruzioni è sempre eseguita almeno una volta. Il ciclo ha termine non appena la condizione è valutata `false`. L'esempio seguente stampa i numeri da 1 a 10 usando un ciclo `do-while`:

```
i = 0; // Inizializza i
do
{
  i++; // Incrementa i di uno
  System.out.println(i); // Stampa il valore di i
} while (i < 10); // e ripeti fintantoché i < 10
// Attenzione! Alla fine del ciclo i vale 10. 16
```

Un ciclo `for` permette di eseguire una sezione di codice un numero fissato di volte. La sintassi dell'istruzione `for` è la seguente:

```
for (inizializzazione; condizione; espressione)
{
  istruzioni
}
```

L'inizializzazione è un'istruzione di assegnamento che attribuisce ad una variabile, detta variabile di controllo, un valore iniziale. La condizione è un'espressione di tipo booleano che tipicamente confronta il valore della variabile di controllo con un valore limite. Infine, `espressione` specifica il modo in cui la variabile di controllo deve essere modificata prima della successiva iterazione del ciclo. Se la parte `istruzioni` è composta da una singola istruzione, le parentesi graffe possono essere omesse. Il seguente esempio mostra come usare un ciclo for per stampare i numeri da 1 a 10:

```
for (i = 1; i < 11; i++) // Ripeti, per i che varia da 1 a 10:
  System.out.println(i); // stampa il valore di i
// Attenzione! Alla fine del ciclo, i vale 11
```

Si può interpretare l'istruzione precedente come segue: si assegni a `i` il valore `1`; finché il contenuto di `i` è minore di `11` si esegua l'istruzione di stampa e si incrementi il contenuto di `i` di uno. Volendo stampare i numeri da 1 a 10 in ordine decrescente, si userebbe un ciclo for di questo tipo:

```
for (i = 10; i > 0; i--) // Ripeti, per i che varia da 10 a 1
  System.out.println(i); // stampa il valore di i
                         // e decrementa i di uno
// Alla fine del ciclo, i vale 0.
```

Si rifletta attentamente sul modo in cui la variabile di controllo è inizializzata e sul valore finale che essa assume, nonché sul tipo di condizione da imporre in ciascun tipo di ciclo.

Example 3. Si scriva una classe e un metodo opportuno per stampare a video la seguente figura:

```
*********
*********
*********
*********
*********
*********
*********
```

Soluzione.

```
public class Disegnetti
{
   //stampa un quadrato di asterischi di lato l
   // Es. stampaquadrato(4) stampa la seguente figura:

   // ****
   // ****
   // ****
   // ****
```

```
    public static void stampaquadrato(int l){
        int i,j;
        for (i=0;i<l; i++){

          for(j=0;j<l;j++)
            System.out.print("*");

          System.out.println("");
        }
    }
 }
```

7.4 Esercizi

Si estenda la classe dell'esempio precedente, aggiungendo dei metodi per stampare a video le seguenti figure:

```
*********       *     *       *********
********         *   *        *       *
*******            * *        *       *
******               *         *       *
*****              * *        *       *
****               *   *       *       *
***               *     *      *********
```

8 Classi, interfacce e package

8.1 Classi

Un programma Java è un insieme di definizioni di classi. La sintassi per la definizione di una classe è

```
class nome
{
  corpo della classe
}
```

dove `nome` dev'essere un identificatore Java valido. Il corpo della classe è costituito da dichiarazioni di variabili e da definizioni di metodi. Un metodo ha la seguente sintassi:

```
tipo nome-metodo(lista parametri)
{
  corpo metodo
}
```

dove `tipo` è il tipo del valore di ritorno del metodo e `lista parametri` è una sequenza (eventualmente vuota) di coppie `tiponome parametro` separate da una virgola e dette parametri `formali`. Il nome di un parametro è ancora un identificatore. Il corpo del metodo è costituito da un insieme di dichiarazioni e da una sequenza di istruzioni. Se il tipo del valore di ritorno è diverso da `void` (parola chiave che denota l'assenza di un valore di ritorno), nel corpo del metodo dev'essere presente almeno un'istruzione `return` seguita dal valore che deve essere restituito. Ad esempio:

```
int somma(int m, int n)
{
  return m + n; // Restituisce la somma di due interi
}
```

Le variabili dichiarate nel corpo di un metodo sono dette variabili *locali*, perché la loro esistenza è limitata all'ambito del metodo, ossia tali variabili possono essere usate soltanto nel corpo del metodo. Le variabili locali non rappresentano attributi di una classe, ma sono usate come "contenitori temporanei" nelle computazioni eseguite dal metodo. Anche i parametri hanno una "vita" limitata e possono essere usati solo nel corpo del metodo. Quello che segue è un esempio di metodo per stampare i numeri da 1 a n.

```
void conta(int n)
{
  int i; // i è una variabile locale
  for (i = 1; i < n; i++) // Istruzione iterativa
     System.out.println(i); // Stampa i numeri da 1 a n
}
// Al di fuori del metodo, le variabili
// i e n non possono essere usate
```

Si può specificare che una classe specializza (o eredita da) un'altra mediante la clausola `extends`, come segue:

```
class nome extends superclass
{
  corpo della classe
}
```

In Java, ogni classe eredita automaticamente (cioè senza bisogno di specificare una clausola `extends`) dalla classe `Object`, definita nelle API di Java. Quindi anche la classe minimale, cioè la classe vuota, cosí definita:

```
class Vuota { }
```

sa "fare qualcosa", perché eredita attributi e metodi della classe `Object`. Si veda la documentazione di `Object` nelle API di Java.

Gli oggetti di una data classe si creano mediante l'uso dell'operatore `new`, secondo la sintassi seguente:

```
new nome classe(lista parametri)
```

dove l`lista parametri` è una lista, eventualmente vuota, di parametri *attuali* (cioè valori). Il numero e il tipo dei parametri attuali dipende dalla definizione dei costruttori nella classe. Un *costruttore* è un metodo speciale che ha lo stesso nome della classe e nella cui definizione non compare il tipo del valore di ritorno. In altre parole è definito come segue:

```
public nome classe(lista parametri)
{
  corpo del costruttore
}
```

Ad esempio:

```
public Vuota
{
  // Beh, se la classe è vuota, non c'è niente da fare
}
```

La parola chiave `public` è un modificatore d'accesso (si veda la Sezione successiva) che, tipicamente, è associato al costruttore. Il costruttore è invocato al momento dell'istanziazione di una classe e il suo compito è quello di inizializzare lo stato dell'oggetto che si sta per creare. Se nessun costruttore è definito in una classe, allora viene automaticamente usato il costruttore della superclasse.

Example 4. Il seguente esempio definisce una classe che modella alcune caratteristiche di una persona. Inoltre a partire dalla classe `Persona` si deriva una sottoclasse `Studente` che modella il comportamento e alcune proprietà di uno studente.
Soluzione.

```
public class Persona
{
   protected String nome;
   protected String cognome;
   protected int eta;

   public Persona(String n, String c, int e)
   {
     nome=n;
     cognome=c;
     eta=e;
   }

    public void stampaDati()
    {
      System.out.println("Salve, sono una persona e mi chiamo "+nome);
      System.out.println("Ho "+eta+" anni");
```

```
    }

    public void modificaNome(String n)
    {
      nome=n;
    }

    public String ottieniNome()
    {
      return nome;
    }
}

public class Studente extends Persona
{
    // sottoclasse derivata dalla classe Persona

    private int matricola;

    // costruttore della sottoclasse Studente
    // si richiama il costruttore della sopraclasse
    // Persona mediante l'istruzione super
    public Studente(String n, String c, int e, int m)
    {
       super(n,c,e);
       matricola=m;
    }

    public void stampaDati()
    {
      System.out.println("Salve, sono uno stuente e mi chiamo "+
                         nome+" "+cognome);
      System.out.println("Ho "+eta+" anni"+ " e il mio numero di
                         matricola e' "+matricola);
    }

    public void modificaMatricola(int matr)
    {
      matricola=matr;
    }
}
```

8.2 Esercizi

Create una classe vuota in BlueJ, compilatela e istanziatela. Nel menú dell'oggetto potete notare un'opzione *inherited* from `Object`: si tratta dei metodi au-

tomaticamente ereditati dalla classe `Object`, e che potete invocare sull'oggetto. Per una spiegazione del significato di tali metodi, è possibile consultare la documentazione delle API di Java. Il modo piú semplice è selezionare l'opzione *Java Class Libraries* dal menú *Help* di BlueJ: un browser si aprirà automaticamente e sarà avviata una connessione alla pagina web contenente la documentazione delle API. La consultazione delle API è una pratica abituale quando si programma in Java.

8.3 Interfacce

La sintassi per la definizione di un'interfaccia è

```
interface nome
{
  corpo dell'interfaccia
}
```

Il corpo dell'interfaccia è una lista di segnature, vale a dire intestazioni di metodi, terminate da un punto e virgola. Ad esempio, la segnatura corrispondente al metodo precedente è

```
void conta(int n)
```

Una classe può implementare un numero arbitrario di interfacce, includendone i nomi, separati da una virgola, dopo una clausola `implements`, in accordo alla seguente sintassi:

```
class nome implements interfaccia1, interfaccia2,...
{
  corpo dell'interfaccia
}
```

Una classe che implementi una o piú interfacce deve fornire le implementazioni per tutti i metodi le cui segnature sono contenute nelle interfacce. Quando si implementa il metodo di un'interfaccia è obbligatorio dichiararlo `public` (si veda la Sezione 8.6).

Tutti i metodi di un'interfaccia sono pubblici, anche se non esplicitamente dichiarato nella segnatura del metodo all'interno dell'interfaccia.

Example 5. Si consideri il seguente esempio. Entrambe le classi `Rettangolo` e `Cerchio` implementano la stessa interfaccia `Figura`.
Soluzione.

```
public interface Figura
{
   double area();
   double perimetro();
}
```

```
public class Cerchio implements Figura
{
  double x;
  double y;
  double raggio;

  public Cerchio(double xcord,double ycord, double r)
  {
     x=xcord;
     y=ycord;
     raggio=r;
  }

  public double ottieniRaggio()
  {
     return raggio;
  }

  public void stampaDati()
  {
     System.out.println("Raggio: "+raggio);
     System.out.println("X del centro: "+x);
     System.out.println("Y del centro: "+y);
  }

  public double area()
  {
     double PIGRECO=3.1456;
     return raggio*raggio*PIGRECO;
  }

  public double perimetro()
  {
     double PIGRECO=3.1456;
     return (2*raggio*PIGRECO);
  }
}

public class Rettangolo implements Figura
{
    private double x;
    private double y;
    private double base;
    private double altezza;
```

```
    public Rettangolo(double xcord,double ycord, double b, double a)
    {
       x=xcord;
       y=ycord;
       base=b;
       altezza=a;
    }

    public double ottieniAltezza()
    {
       return altezza;
    }

    public double ottieniBase()
    {
       return base;
    }

    public void stampaDati()
    {
       System.out.println("Altezza: "+altezza);
       System.out.println("Base: "+base);
       System.out.println("X angolo superiore sx: "+x);
       System.out.println("Y angolo superiore sx: "+y);
    }

    public double area()
    {
       return base*altezza;
    }

    public double perimetro()
    {
       return (2*base+2*altezza);
    }
}
```

8.4 Package

I *package* sono semplicemente gruppi di classi e interfacce correlate. Le API di Java, ad esempio, sono tutte raggruppate in package, con nomi come `java.awt`, `java.io`, `java.math`..., secondo la loro funzione (classi correlate all'interfaccia grafica, classi per l'input/output, classi con funzioni matematiche, etc.). Per specificare che una classe fa parte di un package, è sufficiente scrivere, all'inizio del file sorgente, un'istruzione del tipo

```
package nome-package
```

dove `nome package` è un identificatore che rappresenta il nome che si vuole dare al package. In questo corso, non utilizzeremo mai l'istruzione package. Per poter utilizzare classi che appartengono ad un package (ad esempio, le API di Java) è necessario importarle, mediante un'istruzione `import` del tipo:

```
import nome-package
```

inserita all'inizio del file. I package possono essere annidati gli uni dentro gli altri. Ad esempio, il nome `java.awt` significa che è definito un package di nome `awt` all'interno di un altro package il cui nome è `java`. Il nome `java.awt.Color` identifica la classe `Color` che appartiene al package `awt` all'interno del package `java`. Quindi, l'istruzione

```
import java.awt.Color;
```

importa (e quindi consente di utilizzare) la classe `Color`.

È possibile usare un carattere "jolly" `*` che permette di importare tutte le classi di un package. Ad esempio, l'istruzione

```
import java.awt.*;
```

importa (e quindi consente di utilizzare) tutte le classi del package `java.awt`.

8.5 Esercizi

Esaminate la documentazione delle API di Java. Cercate informazioni sulle classi che avete incontrato finora (`Object`, `String`, `StringBuffer`, `Math`, `System`, `Color`). Cercate di capire com'è strutturata la documentazione e che tipo di informazioni fornisce.

8.6 Modificatori

È possibile regolamentare l'accesso dall'esterno ai membri (variabili e metodi) di una classe attraverso l'uso dei *modificatori* di accesso. I modificatori di accesso sono scritti immediatamente prima del tipo di una variabile o del tipo del valore di ritorno di un metodo. Ad esempio:

```
private int n; public void conta(int n)
{ ... }
```

I modificatori d'accesso piú usati sono:

1. `private`: specifica che la variabile, o il metodo, che segue può essere acceduto solo dall'interno della classe;
2. `protected`: consente l'accesso anche alle sottoclassi e alle classi nello stesso package;
3. `public`: consente l'accesso a chiunque. In assenza di modificatori d'accesso, si applica il criterio `protected`.

Nota: una variabile dichiarata `public` può essere acceduta e modificata direttamente dall'esterno. Si consideri la classe seguente:

```
class Accesso
{
  public int x; // La variabile x è definita pubblica
}
```

Se accesso è il nome di un'istanza di `Accesso`, allora l'attributo `x` può essere acceduto usando la sintassi con il punto, come segue:

```
System.out.println(accesso.x); // Stampa il valore di x
```

L'uso del modificatore `public` davanti a variabili è solitamente da evitare, perché va contro il principio dell'incapsulamento dei dati. Se volete che una vostra classe renda accessibile il valore di una variabile e/o lo renda modificabile, scrivete metodi adatti allo scopo.

Concludiamo con lo studio di altri due modificatori: `final` e `static`. Il modificatore final posto in testa alla dichiarazione di una variabile specifica che il contenuto della variabile non può essere modificato. Per questo motivo, una variabile `final` è inizializzata al momento della dichiarazione. Ad esempio:

```
final int VALORE_MASSIMO = 10;
```

Convenzionalmente, le variabili `final` (che hanno valore costante) si scrivono tutte in maiuscolo. Il modificatore `final` posto davanti all'intestazione di un metodo specifica che quel metodo non può essere ridefinito in una sottoclasse.

Il modificatore `static` è usato davanti alla dichiarazione di una variabile per specificare che di quella variabile esiste un'unica "copia" condivisa da tutte le istanze. Di norma, ogni istanza ha un proprio stato (cioè, un proprio insieme di valori associati agli attributi). Se una variabile è dichiarata `static`, tuttavia, il valore di tale variabile è lo stesso per tutte le istanze della classe. Le variabili statiche sono dette variabili di *classe*, mentre le variabili non statiche sono dette variabili d'*istanza*.

Un caso tipico è costituito dalle variabili `final`: poiché il loro valore non può essere cambiato, è inutile averne una copia per ciascuna istanza: tutti gli oggetti possono tranquillamente condividere un'unica copia della variabile. Per questo, i valori costanti sono tipicamente dichiarati `static`:

```
static final int VALORE_MASSIMO = 10;
```

Un modificatore `static` può anche essere associato a un metodo. Un metodo statico può essere invocato senza aver creato alcuna istanza della classe che contiene tale metodo. I metodi statici sono detti metodi di *classe*, mentre i metodi non statici sono detti metodi d'*istanza*. Un caso tipico di metodo statico è il metodo `main()` di un'applicazione, che è il metodo automaticamente invocato quando il programma è eseguito. Tale metodo deve avere la seguente segnatura:

```
public static void main(String[] s)
```

Invocare un metodo senza aver creato un oggetto può sembrare strano a prima vista. Il modo di interpretare la situazione è il seguente: quando un programma Java viene eseguito, le definizioni delle classi del programma sono caricate in memoria. Queste definizioni di classi in memoria sono di fatto oggetti software: usando una terminologia in apparenza paradossale, possono essere chiamati *oggetti-classe*. I metodi statici possono essere invocati usando il nome della classe (l'oggetto-classe) come ricevitore.

Data la loro natura, i metodi statici non possono usare variabili d'istanza. In modo analogo, le variabili statiche possono essere accedute usando la sintassi

```
nome-classe.nome-variabile
```

In quali situazioni è conveniente usare metodi statici? Si consideri la classe `Math` delle API di Java. Tale classe implementa alcune comuni operazioni matematiche (come la radice quadrata o l'elevamento a potenza). Non ha senso avere molti oggetti di tipo `Math`, visto che tutti farebbero le stesse cose: un oggetto è piú che sufficiente. Perciò, tutti i metodi della classe `Math` sono definiti statici, e possono essere invocati direttamente sull'oggetto-classe `Math` (automaticamente creato dalla JVM quando si esegue un programma che usa la classe `Math`). Si può dire che la classe `Math` è il progetto di un singolo oggetto software e che la definizione della classe coincide essenzialmente con la costruzione dell'oggetto (aver definito la classe rende disponibile un oggetto con le caratteristiche della classe).

Vediamo un altro esempio: la classe `System` delle API di Java possiede una variabile `out` che è definita `public static final`. Il modificatore d'accesso `public` la rende accessibile direttamente dall'esterno. Il modificatore `final` la rende immodificabile. Il modificatore `static` fa sí che `out` possa essere acceduta senza istanziare `System`. La classe `System` modella alcune caratteristiche del calcolatore su cui si opera, tra cui il flusso di output diretto su terminale, rappresentato dalla variabile `out` (che è di tipo `PrintStream`). Tale flusso di uscita è unico, perciò la variabile `out` è dichiarata statica.

8.7 Esercizi

1. Perché un metodo statico non può usare variabili d'istanza?

2. Spiegate la struttura dell'invocazione di metodo seguente, tenendo presente che l'interpretazione avviene da sinistra a destra:

   ```
   System.out.println("Ciao!");
   ```

3. In BlueJ create tre classi, diciamo `A`, `B` e `C`, e imponete che `B` sia sottoclasse di `A`. Nella classe `A` dichiarate le quattro seguenti variabili:

   ```
   int x = 0;
   public int y = 1;
   protected int w = 2;
   private int z = 3;
   ```

Nella classe `B` scrivete un metodo che stampi il valore delle quattro variabili. Quale di queste non può essere stampata? Nella classe `C` scrivete un metodo in cui viene creata un'istanza di `A` e si prova ad accedere direttamente alle variabili di `A` per stamparne il valore. Di quali tra queste variabili può essere stampato il contenuto?

4. Dichiarate, nella classe `C`, una variabile statica (di qualunque tipo) e definite due metodi, uno che restituisca il valore della variabile e uno che permetta di modificarlo. Create due istanze di `C` e provate a modificare il valore della variabile in un'istanza e a controllare il valore della variabile nell'altra istanza. Togliete il modificatore `static` e ripetete l'esperimento.