Università di Torino – Facoltà di Scienze MFN
Corso di Studi in Informatica

Programmazione I - corso B
a.a. 2009-2010

prof. Viviana Bono

Blocco 4 - I concetti di base della programmazione nei linguaggi imperativi, procedurali, fortemente tipati, a oggetti.
Il linguaggio Java

Java

Il linguaggio Java è un linguaggio di programmazione:

- imperativo
- fortemente tipato;
- a procedure (dette metodi)
- a oggetti

(in realtà la terminologia di cui sopra non viene più usata da tutti nello stesso senso; ad esempio le qualifiche "imperativo" e "a procedure" vengono talvolta riservate ai linguaggi tradizionali non a oggetti)

Java è un linguaggio imperativo

I concetti fondamentali della programmazione imperativa sono due nozioni-base strettamente connesse fra di loro:

variabile assegnabile e assegnazione

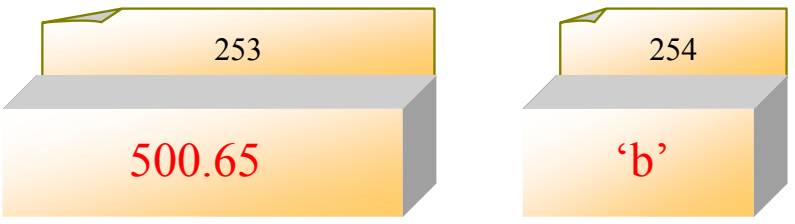
Programmazione imperativa: concetti fondamentali

variabile (o cella di memoria)

può essere pensata come:

un contenitore o scatola:

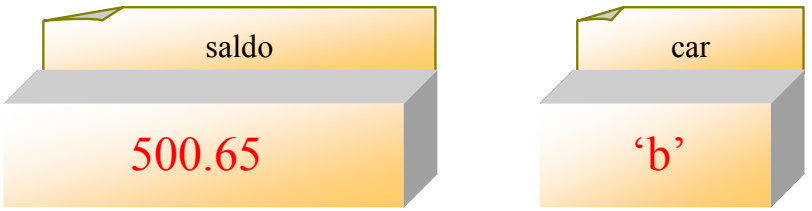
- dotata di un'etichetta esterna che è il suo indirizzo o nome,
- e contenente un valore, che è un ente astratto, come un numero, o un carattere, ecc.



Programmazione imperativa: concetti fondamentali

variabile (o cella di memoria)

Nel programma Java (o Pascal, C, ecc.) il nome della variabile non è il suo indirizzo numerico, bensì un nome simbolico dichiarato dal programmatore (o, come vedremo in seguito, un'espressione simbolica di altro genere).
La traduzione di tali nomi in indirizzi è fatta all'atto dell'esecuzione, senza intervento del programmatore Java (o C, ecc.).



Programmazione imperativa: concetti fondamentali

ASSEGNAZIONE (assignment)

è un'istruzione (cioè un comando all'esecutore) che permette di cambiare il contenuto di una cella di memoria, ad esempio:

```
saldo = saldoIniziale;
```

copia il contenuto della cella *saldoIniziale* nella cella *saldo*, cancellando il precedente contenuto di *saldo*;

```
saldo = saldo – prelievo;
```

calcola la differenza fra il contenuto della cella *saldo* e quello della cella *prelievo*, e rimette il risultato nella cella *saldo*.

Programmazione imperativa: assegnazione (cont.)

NOTA BENE:

il simbolo '=' NON indica l'uguaglianza matematica!

la scrittura

a = b;

NON è un'espressione affermante che *a* e *b* sono uguali;
è un comando o istruzione che copia in *a* il contenuto di *b*
Un simbolo più conveniente per l'assegnazione sarebbe:

a ⟵ b;

(l'informazione fluisce da destra verso sinistra)

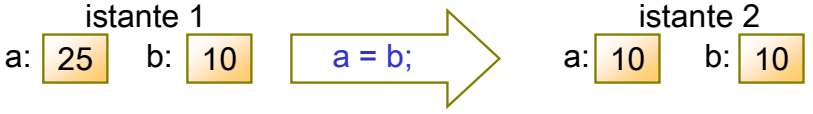
Subito dopo l'esecuzione dell'istruzione i contenuti di *a* e di *b* saranno uguali, ma poi possono di nuovo variare indipendentemente.

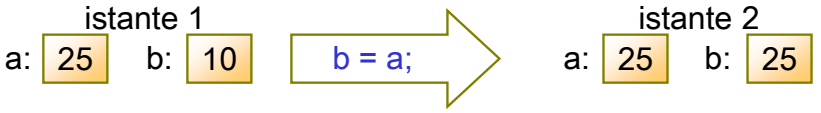
Programmazione imperativa: assegnazione (cont.)

NB

Dalla spiegazione precedente segue che
l'operazione di assegnazione, a differenza dell'uguaglianza matematica, NON è simmetrica:

a = b NON è la stessa cosa che b = a





L'assegnazione non è simmetrica (cont.)

a = 3; è un'operazione perfettamente legale, che mette il valore 3 nel contenitore a, cancellando il valore precedente

3 = a; è una scrittura priva di senso, perché 3 non è un contenitore! non si può mettere qualcosa nel numero 3!

a = b+c; è un'operazione perfettamente legale, che mette in a il risultato della somma dei valori contenuti in b e in c

b+c = a; è una scrittura priva di senso, perché l'*espressione* b+c non denota un contenitore, bensì un numero:
la somma dei contenuti di b e c (si possono sommare i valori contenuti, NON i contenitori)

L'assegnazione non è simmetrica (cont.): significato destro e significato sinistro

Si noti che il nome di una variabile ha un significato diverso a seconda che compaia a sinistra o a destra del simbolo di assegnazione:

- a = …; a sinistra denota proprio il contenitore, in cui andare a mettere il risultato del calcolo dell'espressione di destra
- … = a; a destra denota invece il contenuto della cella, cioè un valore;

… = a + b ; i nomi a e b denotano i contenuti delle risp. celle.

A sinistra ci può stare solo il nome di un contenitore (o, come vedremo, un'espressione che denota un contenitore).

A destra ci può stare un'espressione ordinaria.

L'assegnazione non è simmetrica (cont.): significato destro e significato sinistro

Vi sono dei linguaggi di programmazione (di un genere diverso da quelli imperativi) in cui i due significati sinistro e destro di una variabile vengono distinti anche nella sintassi;

in cui cioè vi è un simbolo (indichiamolo con "!") che significa "contenuto di".

Così, invece di scrivere ad esempio

r = s + t

si deve scrivere esplicitamente

r = !s + !t

cioè: nel contenitore r *mettici* la somma dei contenuti di s e t (invece di *mettici* si dice *assegnato-uguale a,* evitando così anche la sgrammaticatura …)

Guardando avanti nel corso…

ATTENZIONE!

Nella programmazione a oggetti, come vedremo, se a e b sono due "riferimenti a oggetti", l'effetto dell'istruzione a = b è quello di far sì che a "si riferisca allo stesso oggetto" cui si riferisce b

Il significato preciso di tale affermazione sarà chiarito in lezioni successive.

Java è un linguaggio fortemente tipato

Tipi

- che cos'è un linguaggio (fortemente) tipato
- che cosa vuol dire programmare con i tipi

Tipi: una forma per i dati

- Variabili, parametri,valori ritornati hanno un ne precisa/limita l'uso
  - Possono contenere solo valori conformi al proprio tipo
- Esistono tipi
  - Semplici (**primitivi**)
  - Composti (**classi**) – descritti nelle lezioni successive

Programmazione tipata: concetti fondamentali

tipi di valore e di variabile

- i valori – cioè gli enti astratti – che si mettono nelle celle sono di diversi tipi: intero, reale, carattere, riferimento a oggetto di una certa classe, ecc.;
- anche le celle vengono dichiarate di diversi tipi, e dentro una cella dichiarata di un dato tipo ci può stare solo un valore di un tipo compatibile con il tipo della cella;
- ad esempio, in una cella di tipo intero non si può mettere un reale:

```
int n; …
n = 400.65;
ERRORE!
```


Java: tipi

Un tipo può essere:

- *primitivo*: int, double, boolean, ecc.;
- *array*;
- *riferimento-ad-oggetto*-di-una-data-classe: è dato semplicemente dal nome di una classe.

I tipi *primitivi* di Java

- boolean 1 bit true, false
- tipi numerici
  - tipi interi
    - con segno:
      - byte 8 bit da -128 a 127
      - short 16 bit da -32768 a 32767
      - int 32 bit da ~ -2 miliardi a ~ 2 miliardi
      - long 64 bit ... miliardi di miliardi ...
    - assoluti:
      - char 16 bit da 0 a 65535
  - tipi con virgola mobile (floating point)
    - float 32 bit ~ 7 cifre decimali significative
    - double 64 bit ~ 15 cifre decimali signific.

Valori ed espressioni

Espressione è una porzione di programma che ha un valore, il quale viene calcolato durante l'elaborazione (compilazione o più spesso esecuzione) del programma.

I valori dei tipi primitivi sono rappresentati in Java da cosiddetti literals (o letterali): 3, -15, 1.4, 'a', true, false, ecc.

Esempi di espressioni:

(3 + 2)*7 espressione di tipo intero

m contenuto della variabile m (il suo tipo è il tipo di m)

5==7 espressione di tipo booleano

a >= 5 espressione di tipo booleano

a < m && m < b espressione di tipo booleano

Dichiarazioni di variabili

- Introducono l'utilizzo di una variabile
- Formato generale
  - *<Tipo> <Nome>* '**;**'
  - *<Tipo> <Nome>* '=' *<ValoreIniziale>* '**;**'
- Esempi
  - char c;
  - int i = 10;
  - long l1, l2;
- La mancata dichiarazione è un errore sintattico
- Alcuni linguaggi non sono "tipati" (o non hanno una struttura di tipi rigida). In questi linguaggi si possono fare operazioni ed assegnamenti "strani" e anche flessibili, ma spesso si perde il controllo dei risultati

Usare le variabili

- Si può dichiarare una variabile ovunque nel codice
  - Bisogna farlo, però, prima di usarla
  - Visibilità limitata al blocco di istruzioni in cui è dichiarata
- Le variabili devono essere inizializzate!
  - int a:
  - int b = a;

Errore!
La variabile "a" non è stata inizializzata, e il suo valore è casuale

Conversioni (NB. programmare funzioni numeriche è difficile!)

Alcuni linguaggi, tra cui Java, effettuano conversioni (cast implicito) tra tipi "simili". per es.

int x;
double y;
double z = x + y (ok)

Nel caso in cui si voglia fare un'operazione che *fa perdere di precisione* occorre mettere un cast esplicito:

```
double x = 3.3;
int y = x;  // ERRORE A COMPILE TIME!
int y = (int) x; // OK! (se questo è ok per noi)
```


I booleani: operatori di confronto

>	maggiore
>=	maggiore o uguale
<	minore
<=	minore o uguale

= =	uguale
!=	diverso
s_1.equals(s_2)	uguale (stringhe)
!s_1.equals(s_2)	diverso (stringhe)

le stringhe sono *oggetti* (dopo…)

I booleani: operatori logici

Per verificare se sia la condizione A che lacondizione B sono vere, verifica se è vera la condizione A oppure la condizione B, ecc.

&&	AND
||	OR
!	NOT

→

A	B	A && B
V	V	V
V	F	F
F	V	F
F	F	F

A	B	A || B
V	V	V
V	F	V
F	V	V
F	F	F

A	!A
V	F
F	V

dove: V = vero (*true*) e F = falso (*false*)

I booleani

NB

In programmazione, a differenza che in logica:

i booleani sono valori, esattamente come i numeri !

>, <, <=, == (che in logica si scrive =), != (cioè ≠), ecc. sono operatori, NON simboli di relazioni !

7 <= 5 è un'espressione esattamente come 7+5:

- 7+5 è un'espressione di tipo intero, perché il + è un operatore che applicato a due interi produce un intero;
- 7 <= 5 è un'espressione di tipo booleano, perché il <= è un operatore che applicato a due interi produce un booleano

I booleani (cont.)

5 <= 7 && m == 3

è un' espressione booleana, perché 5 <= 7, m == 3 sono due espressioni booleane, e && (che in logica si scrive AND) è un operatore che prende due booleani e restituisce un booleano.

Il valore di un'espressione booleana può essere memorizzato in una variabile di tipo booleano, ad esempio:

```
boolean b = 5 <= 7;
b = 3==7;
boolean uguali =   m == n;
boolean diversi =  m != n;
b  =  (b   &&  m <= n);
```


Java è un linguaggio procedurale

Funzioni e Procedure
(dette Metodi in Java)

Programmazione procedurale: funzioni

Una definizione matematica di funzione, ad esempio

f(x,y) = 3*x + 5*y

se *applicata* ad una coppia di argomenti, fornisce un risultato:

f(2,7) = 3*2 + 5*7 = 41

Il risultato si ottiene sostituendo ai *parametri formali* x e y gli *argomenti effettivi* (o *parametri attuali*) 2 e 7, e valutando poi l'espressione così ottenuta. Analogamente f(0,1) = 5, f(1,1) = 8, ecc.

In Javascript (un altro linguaggio a oggetti) si scriverebbe:

```
function f(x,y) {return 3*x + 5*y; }
```

in Java, invece, non si scrive la parola function, ed occorre dichiarare i tipi dei parametri e del risultato, ad esempio:

```
static double f(double x, double y) {return 3*x + 5*y; }
```

Sia in Javascript che in Java, come in tutti i linguaggi *imperativi*, i parametri formali sono dei contenitori in cui vengono automaticamente depositati gli argomenti effettivi.

Funzioni e procedure

In generale, una procedura o funzione (in Java, metodo) è un insieme strutturato di istruzioni (cioè un sottoprogramma, un pezzo di programma) dotato di:

- un nome, con cui il sottoprogramma può essere *chiamato*, cioè attivato, mandato in esecuzione;
- eventualmente una sequenza di parametri formali, che sono dei contenitori in cui il programma che chiama la procedura o funzione mette i dati che serviranno alla procedura/funzione stessa per eseguire il suo compito;
- il compito del sottoprogramma può essere quello di computare un valore (numerico o di altro tipo) da restituire al chiamante (funzione), oppure quello di operare dei cambiamenti dello stato della memoria (in senso lato, comprendente lo stato dello schermo, ecc.) senza restituire un valore (procedura).

Terminologia: ricapitoliamo

In linguaggi come il Pascal, il termine funzione indica un sottoprogramma che restituisce qualcosa al programma chiamante, il termine procedura un sottoprogramma che non restituisce nulla.

In generale, però, i due termini – e soprattutto il termine procedura – sono usati indifferentemente come sinonimi di *sottoprogramma*. In C e nei linguaggi da esso derivati (come C++, Java, Javascript, ecc.) tutti i sottoprogrammi sono formalmente delle funzioni, anche se non restituiscono nulla (in tal caso, al posto del tipo del risultato, deve comparire la parola void, che vuol dire *vuoto*)

Nei linguaggi a oggetti, come Java, le procedure o funzioni vengono chiamate metodi, e hanno alcune caratteristiche particolari che vedremo.

Nota

void può essere pensato come un tipo coincidente con l'insieme vuoto: quindi non esistono valori di tipo void, ed una procedura che ha void come tipo del risultato è una procedura che non restituisce alcun valore.

In realtà in Java void non è formalmente un tipo (ad esempio, non si possono dichiarare variabili di tipo void, che d'altra parte non servirebbero a molto, non potendo contenere nulla ...)

Esempio di metodo che non restituisce nulla (procedura)

```
static void salutaSuConsolle(String s) {
  System.out.println(s);
}
```

NB Questa procedura produce l'effetto della scrittura sullo schermo, ma non restituisce nulla al programma chiamante (infatti non esegue alcuna istruzione return)

Invece la procedura schematizzata sotto:

```
static int numParole(String s)  {
  istruzioni che computano il numero di parole nella stringa,
  mettendolo in una variabile di nome numPar;
  return numPar;
}
```

restituisce al programma chiamante un numero intero.

I nomi dei parametri valgono solo dentro il metodo

Se "al di fuori' della definizione di funzione

static double f(double x, double y) {return 3*x + 5*y; }

scriviamo l'istruzione x = 2, tale x non può essere il contenitore che compare nella definizione di f.

L'unico modo per mettere dei valori nei contenitori x e y è quello di chiamare la funzione f, ad esempio con l'espressione f(2,7)

NON si può ottenere tale effetto con le istruzioni x=2, y=7 "fuori" dal metodo!!!

x e y sono contenitori propri della funzione f, ed esistono solo durante *le sue attivazioni*; i loro nomi x e y non sono visibili né utilizzabili al di fuori di f; se si parla di x e di y fuori di f, si tratterà di altri contenitori che casualmente hanno lo stesso nome!

Attenzione, quindi: non bisogna confondere
la definizione di un metodo
con la sua invocazione!

```
static double f(double x, double y) {return 3*x + 5*y; }

static void salutaSuSchermo(String nome) {
  System.out.println("ciao "+ nome);
}
```

sono due definizioni di procedure; il *corpo* di ciascuna di esse, cioè la sequenza di istruzioni compresa fra le graffe, viene eseguito solo quando (e tutte le volte che) la procedura viene chiamata o invocata:

```
f(2,7);   salutaSuSchermo("Carlo");
double a = f(4,6);   f(a,1); ...
```


Domanda: chi è che può chiamare un metodo?
Risposta: un altro metodo!

Un moderno prodotto software è di solito un programma costituito di molti pezzi (cioè procedure e/o funzioni, ma ormai li chiamamo tutti metodi) distinti, ognuno dei quali ne invoca (cioè ne attiva) qualcun altro; è cioè simile ad una squadra di lavoratori specializzati in cui ognuno, per portare a termine il proprio lavoro, deve in generale chiedere l'aiuto di qualcun altro, che a sua volta delegherà certi compiti ad altri ancora, e così via; oppure ad una macchina fatta di molti componenti, dove ogni componente interagisce con altri.

Quindi dentro la definizione di un metodo vi sono spesso invocazioni (dette anche chiamate) di altri metodi, e non bisogna confondere le definizioni con le invocazioni.

Esempio di procedure che invocano procedure

definizione della procedura salutaSuSchermo

```
static void salutaSuSchermo(String nome) {
  System.out.println("Ciao " + nome);
}
```

definizione della procedura salutaDaFinestra

```
static void salutaDaFinestra(String nome) {
   JOptionPane.showMessage("Ciao " + nome );
}
```

definizione della procedura salutaCarlo

```
static void salutaCarlo() {
   salutaSuSchermo(“Carlo");
   salutaDaFinestra(“Carlo");
}
```

salutaCarlo invoca salutaSuSchermo e salutaDaFinestra

Librerie di procedure predefinite

I compilatori o interpreti dei linguaggi di programmazione vengono forniti con un ricco insieme di procedure già definite per le più svariate necessità, in modo che il programmatore che deve realizzare un nuovo prodotto non debba ogni volta reinventare la ruota, o comunque partire da zero.

Nello stesso modo, il costruttore di una qualunque macchina di solito non parte da zero, ma da componenti che gli sono forniti da altri costruttori (bulloni, circuiti elettronici, cuscinetti a sfera, ecc.)

Un insieme di procedure utili per risolvere una certa classe di problemi si chiama *libreria* (dall’inglese *library*, che vuol dire più propriamente biblioteca).

Così vi sono librerie matematiche, librerie di grafica, ecc.

Nell'esempio precedente le procedure println e showMessage, di cui non abbiamo riportato le definizioni, sono metodi predefiniti (di libreria).

Anche in questo corso utilizzeremo metodi predefiniti delle librerie di Java.

Ma chi comincia il tutto? Il `main`!

In ogni programma C o C++ o Java ci deve essere un metodo che si chiama *main* (che vuol dire principale): quando si lancia l'esecuzione del programma si invoca automaticamente tale procedura, che a sua volta ne chiamerà altre, e così via.

```
... void main(...)    {
     salutaCarlo();
          ...
}
```

Dati e metodi

Come abbiamo visto nell'esempio precedente, ogni programma o modulo di programma, durante l'esecuzione, è costituito da:

- un insieme di celle contenenti i dati su cui si lavora;
- un insieme strutturato di istruzioni operanti su tali dati

Come vedremo, la programmazione cosiddetta a oggetti rende l'associazione fra dati e procedure che operano su di essi molto stretta, attraverso appunto la nozione di oggetto.

Java è un linguaggio a oggetti

Introduzione alla programmazione a oggetti:
creazione e uso di oggetti di classi predefinite

In questa prima parte del corso scriveremo programmi Java che non definiscono nuove classi di oggetti, ma al più utilizzano oggetti di classi predefinite dalla libreria di Java.

Oggetti: una prima, vaga, idea

Un oggetto è un'entità software costituita da dati interni (campi dell'oggetto) e da procedure (metodi dell'oggetto) che operano su tali dati e che possono essere invocate dall'esterno.

Un oggetto è cioè un'entità dotata di uno stato, i suoi campi e di un comportamento, i suoi* metodi, che possono modificare il suo stato o produrre effetti di vario genere.

Ad esempio ogni oggetto della classe Scanner possiede, fra gli altri, il metodo nextInt che restituisce il primo intero del *flusso di dati* cui l'oggetto è "attaccato".

*: In Java ci sono due specie di metodi: quelli associati agli oggetti e quelli non associati agli oggetti (detti *static*)… Teniamolo a mente…

Creazione di oggetti (esempio per l'input da tastiera)

Come vedremo, un valore di tipo riferimento-a-oggetto può essere ottenuto come risultato dell'invocazione della primitiva new su un costruttore di oggetti.

Un costruttore è una procedura che ha lo stesso nome della classe di cui costruisce gli oggetti; essa restituisce al chiamante l'indirizzo dell'oggetto costruito.

Esempio. L'espressione

```
new Scanner(System.in)
```

costruisce un oggetto della classe Scanner operante sull'input da tastiera; il metodo nextInt di tale oggetto restituisce il successivo intero digitato sulla tastiera che appare sulla consolle sullo schermo.

NB Per "consolle" intendiamo, in Windows, una finestra di comando oppure la finestra del programma cmd.exe aperta automaticamente da TextPad (che useremo in lab.).

Invocazione di metodi di oggetti

Per invocare un metodo di un oggetto si deve usare la notazione col punto (*dot notation*):

riferimento-a-un-oggetto.nomeMetodo(Argomenti)

Esempi:

out.println("Ciao"); (out contiene un riferim. a oggetto)

tastiera.nextInt();

"Programmazione".subString(7);

(come vedremo, anche le stringhe sono oggetti)

...

Esempio di creazione e uso di oggetto

```
import static java.lang.System.*;
import java.util.Scanner;

public class CiaoNome {

  public static void main(String[] args) {
    out.print("Come ti chiami? ");
    String nome = new Scanner(System.in).nextLine();
    out.println("Benvenuto, " + nome);
  }
}
```

viene invocato il metodo nextLine dell' oggetto creato dall'espressione **`new Scanner(System.in)`**

il metodo legge da tastiera e restituisce una stringa che il main memorizza nella variabile locale **`nome`**

Si noti che in tal modo non possiamo usare l'oggetto una seconda volta, perché non ne abbiamo memorizzato l'indirizzo.

```
import static java.lang.System.*;
import java.util.Scanner;

public class CiaoNome {

  public static void main(String[] args) {
    out.print("Come ti chiami? ");
    Scanner tastiera = new Scanner(System.in);
    String nome1 = tastiera.nextLine();
    String nome2 = tastiera.nextLine();
    out.println("Ciao, " + nome1);
    out.println("Ciao, " + nome2);
  }
}
```

NB il riferimento all'oggetto viene memorizzato nella variabile tastiera e può quindi essere usato più volte

NOTA BENE: metodi statici e metodi degli oggetti

Al di fuori della classe in cui è definito, un metodo statico viene invocato premettendogli il nome della classe (e il punto):

```
JOptionPane.showInputDialog("digita un intero");
```

oppure

importandolo staticamente, e poi usandone il nome semplice:

```
import javax.swing.JOptionPane.*;
 ...
showInputDialog("digita un intero");
```

Un metodo di un oggetto viene sempre invocato premettendogli un riferimento ad un oggetto (ed il punto):

```
Scanner input = new Scanner("System.in");
input.next();
```

Il linguaggio Java

La sintassi (o grammatica) di Java

Esempi di regole:

ClassDeclaration →
  ClassModifiers$_{opt}$ `class` *Identifier* *Super*$_{opt}$ *ClassBody*

ClassModifiers →
  ClassModifier | *ClassModifiers ClassModifier*

ClassModifier → `public` | `private` | `static` | *...*

ClassBody → { *ClassBodyDeclarations*$_{opt}$ }

ClassBodyDeclarations →
  ClassBodyDeclaration |
  ClassBodyDeclarations ClassBodyDeclaration

ClassBodyDeclaration →
FieldDeclaration |
ConstructorDeclaration |
MethodDeclaration |
…
FieldDeclaration →
FieldModifiers$_{opt}$ *Type* *VariableDeclarators* ;
…
VariableDeclarator →
VariableDeclaratorId | *VariableDeclaratorId* = *Initializer*

VariableDeclaratorId → *Identifier* | *Identifier* [ ]

C: una classe Java corretta

ClassDeclaration →
ClassModifiers$_{opt}$ `class` *Identifier* *Super*$_{opt}$ *ClassBody*
→ `class` *Identifier ClassBody*
→ `class C` *ClassBody*
d'altra parte:
ClassBody → { *ClassBodyDeclarations*$_{opt}$ } → { }

quindi:

ClassDeclaration → `class C { }`

La definizione di classe Java più corta del mondo!

```
class C {}
```

(10 caratteri!)

Il programma Java completo più corto del mondo:

```
class C {
  public static void main(String[] a) {
  }
}
```

sono risp. un pezzo di programma ed un programma completo perfettamente corretti, che si compilano senza errore; inoltre il secondo si esegue senza errore.

Ma non sono molto utili … non fanno niente!

Convenzioni universali di scrittura di programmi

- indentare i costrutti sintattici annidati (vedi esempi a lezione) e allineare ogni graffa chiusa con la corrispondente aperta o meglio con l'inizio del corrispondente costrutto sintattico
- i nomi dei campi, delle variabili e dei metodi sono tutti minuscoli, eventualmente con maiuscole in mezzo, per distinguere paroleDiverse nelloStessoNome
- i nomi delle classi iniziano con una maiuscola, seguita da minuscole, eventalmente con maiuscole in mezzo, per distinguere ParoleDiverse NelloStessoNome
- le costanti sono tutte maiuscole, eventualmente con carattere di sottolineatura per distinguere PAROLE_DIVERSE NELLO_STESSO_NOME
- ATTENZIONE: Java è *case-sensitive*.

Buone norme stilistiche

- dopo aver scritto tutte le definizioni dei campi in righe successive (un campo per riga), lasciare una riga bianca prima del primo metodo, e poi separare un metodo dall'altro per mezzo di una riga bianca.
- mettere i campi statici prima dei campi ordinari (cioè non statici), e analogamente mettere i metodi statici prima degli altri metodi.
- Infine, dopo aver scritto tutte le definizioni di metodi, *naturalmente* si chiude il corpo della classe con una *chiusa graffa*.