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

Programmazione 1 - corso B
a.a. 2002-03

prof. Elio Giovannetti

Nota Bene: queste diapositive non sostituiscono il libro di testo.

Correzione di un pregiudizio errato

- *algoritmo* non è sinonimo di *diagramma di flusso*
- un algoritmo è un procedimento (di calcolo) specificato per mezzo di regole precise e non ambigue, in modo da poter essere eseguito "meccanicamente";
- vi sono molti modi per descrivere un algoritmo; ad esempio l'algoritmo per la divisione in colonna imparato alle elementari è un algoritmo perfettamente specificato a parole;
- il metodo dei diagrammi di flusso viene ormai usato solo in casi molto particolari: in generale, per scrivere un programma, non serve fare prima il diagramma di flusso.

Un modo di specificare un algoritmo: la programmazione funzionale.

Esempio:

f(1) = 1

f(n) = n* f(n-1)   se n > 1

Calcolo:

f(4) = 4* f(4-1) =

4* f(3) =

4* (3* f(2)) =

4* (3* (2* f(1))) =

4* (3* (2* 1)) =

4* (3* 2) =

4* 6 = 24

La programmazione imperativa

È un modo di specificare gli algoritmi che viene "capito" dallo hardware della macchina:

- Un programma è costituto da una sequenza di istruzioni o comandi (per questo si chiama imperativa!).

- La macchina esegue il programma eseguendo un'istruzione dopo l'altra (alcune istruzioni hanno l'effetto di far eseguire un'istruzione successiva piuttosto che un'altra, oppure di far ripetere una sequenza di istruzioni, ecc.).

- La macchina ha uno stato, ed ogni istruzione modifica tale stato.

Come è fatto un calcolatore?

- I primi calcolatori potevano essere considerati composti di tre parti, secondo il noto modello di von Neumann, valido - nelle sue linee generali - ancora oggi:
- una memoria (RAM), da concepirsi metaforicamente come un insieme di contenitori o "scatole";
- una unità centrale di elaborazione, o *cpu*, che è in grado di eseguire dei programmi costituiti da sequenze di ben determinati tipi di *istruzioni-di-macchina*, che in generale modificano lo stato della memoria, cioè i contenuti delle celle di memoria;
- una (o piú) unità di comunicazione con l'esterno, anticamente lettori e perforatori di schede e telescriventi, oggi tastiere, mouse, schermi, microfoni, altoparlanti, ecc.

Istruzioni di macchina

Una istruzione-di-macchina è costituita a sua volta, di solito, dalla specifica del tipo di operazione aritmetica o logica da eseguire (ad esempio somma, o sottrazione, o confronto, ecc.) e dalle identità degli operandi, cioè dai nomi (o meglio, indirizzi) dei contenitori in cui andare a prendere gli operandi (ad es. gli addendi) e in cui andare a mettere il risultato (ad es. il risultato dell'addizione); oltre a queste vi sono delle istruzioni "di salto" che permettono di "saltare" ad eseguire un'istruzione diversa da quella successiva, in particolare permettono di "saltare" o no ad un'altra parte del programma a seconda del risultato di un certo confronto od operazione (istruzioni di salto condizionato); possiamo infine assumere, per semplicità, che vi siano delle istruzioni di input/output per comunicare con l'esterno (anche se la realtà è piú complessa).

Un programma in un ipotetico linguaggio-macchina

istruz.1:	**inputint 325**	preleva un intero dal dispositivo di input e mettilo nella cella 325
istruz.2:	**inputint 326**	preleva un numero dal dispositivo di input e mettilo nella cella 326
istruz.3:	**add 325 326**	leggi i contenuti delle celle 325 e 326 fanne la somma, e metti il risultato nella cella 326
istruz.4:	**bpos 7** (branch on positive to 7)	se il risultato dell'ultima operazione è > 0, salta all'istruzione 7
istruz.5:	**mov 325 326**	copia il contenuto della cella 325 nella cella 326
iztruz.6:	**outputint 326**	leggi il contenuto della cella 326 e invialo al dispositivo di output

La sequenza di istruzioni-di-macchina costituente un programma, per poter essere eseguita dal calcolatore, deve anch'essa essere codificata in forma numerica; in particolare devono esserlo i nomi simbolici delle istruzioni, come ADD, MOV, BPOS, ecc.: ad esempio ADD sarà codificato con il numero 214, MOV con il numero 215, BPOS con il numero 216, ecc.; l'istruz.3 sarà allora codificata come 214 325 326, ecc. Un programma completamente codificato in forma numerica e direttamente eseguibile dallo hardware di una macchina viene detto programma in *linguaggio-macchina*; vi sono naturalmente tanti linguaggi-macchina diversi quanti sono i tipi di cpu esistenti.

Come si vede, la strutturazione di questo tipo di programma - cioè il modo in cui le istruzioni debbono o possono venire "disposte" - è molto povera, poiché coincide con la semplice sequenza (tanto che in tal caso si parla di programmi non strutturati).

Anche oggi la forma finale di programma che viene eseguita dallo hardware di una macchina è simile a quella sopra ricordata. Tuttavia raramente i programmi vengono scritti in tale forma; vengono invece scritti, per lo piú, in un cosiddetto *linguaggio di alto livello*, di piú facile concezione e comprensione per l'uomo, con una strutturazione piú ricca ed articolata che la semplice sequenza, e indipendente (almeno in linea di principio) dalla particolare macchina.

In un linguaggio di alto livello il programma precedente verrebbe scritto in un modo simile al seguente:

```
...
readint(a);
b = a + b;
if(b == 0) b = a;
printint(b);
...
```

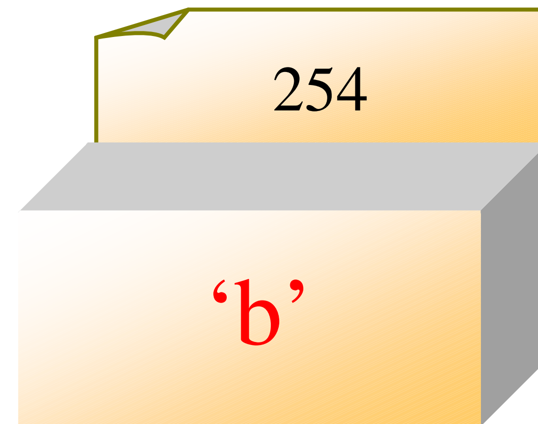
Programmazione imperativa: concetti fondamentali

variabile (o cella di memoria)

può essere pensata come:

un contenitore o scatola:

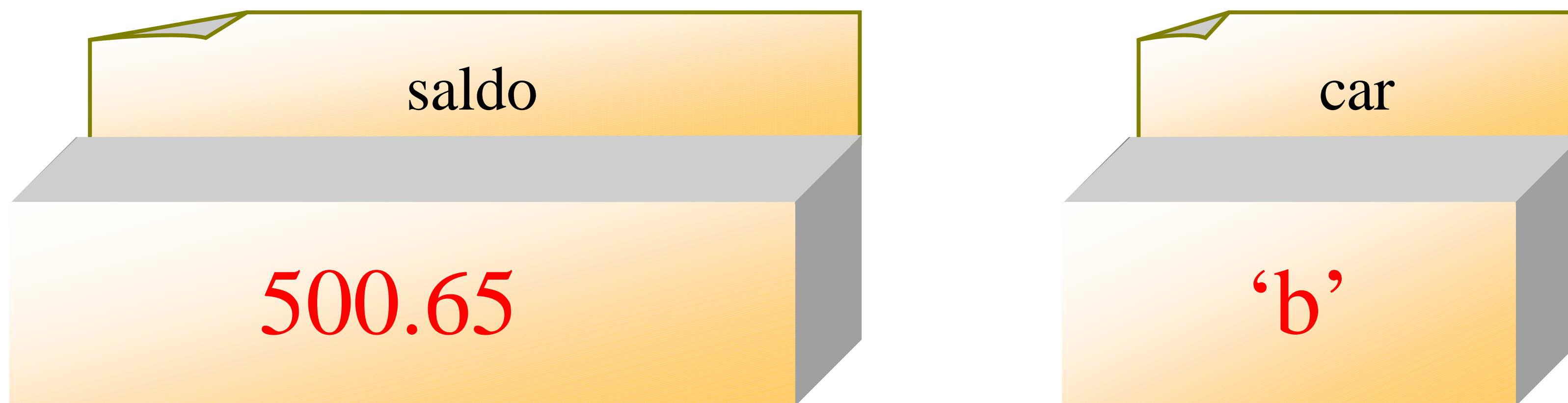
- dotata di un'etichetta esterna che è il suo indirizzo o nome,
- e contenente un valore, che è un ente astratto, come un numero, o un carattere, ecc.



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.).



Astratto e concreto

Non si confondano i valori, che sono degli enti astratti, con le loro rappresentazioni sia esterne (sullo schermo o sulla tastiera) che interne (nella memoria del calcolatore), sia concrete che astratte: il numero 15 non è né la coppia di cifre 1 e 5, né la sua rappresentazione binaria per mezzo di dispositivi elettronici all'interno del calcolatore. Analogamente, il carattere "a minuscolo", che in Java viene indicato come 'a', non coincide né con il segno d'inchiostro impresso dalla stampante su questo foglio, né con il numero 97 con cui è rappresentato all'interno del calcolatore in base al codice ASCII o UNICODE.

||||| ||||| ||||| 15 17_8 1111_2 F   XV

sono rappresentazioni diverse dello STESSO NUMERO. In un certo senso, la prima rappresentazione è la più fedele, cioè quella più vicina all' "essenza" del numero.

La rappresentazione di un ente astratto può essere realizzata per mezzo di enti astratti di un altro tipo: ad esempio il numero 3, che ovviamente non è un oggetto materiale, può essere rappresentato dal carattere '3', che è un ente astratto il quale può essere a sua volta rappresentato da un numero, ad esempio nella codifica UNICODE o ASCII dal numero 51, che a sua volta è rappresentato dalla sequenza di cifre binarie 00110011, la quale a sua volta è rappresentata dallo stato di certi microscopici circuiti elettronici all'interno di un frammento di silicio... Ma il numero 3 non deve essere confuso con il carattere '3'!

Nota Bene: come vedremo, nel linguaggio Java i caratteri, per ragioni di comodità di programmazione, *sono* dei numeri: sono tuttavia dei particolari "tipi" di numero, che quando vengono mandati ad esempio sullo schermo vengono visualizzati come caratteri (ad esempio: il numero 97 verrà visualizzato come 'a', il numero 65 come 'A', il numero 51 verrà visualizzato come il carattere 3, ecc.)

A questo punto ci si potrà domandare: se nel calcolatore tutto è rappresentato per mezzo di numeri, come fa la macchina a distinguere ad esempio fra il numero 51 inteso per se stesso e il numero 51 inteso come rappresentazione del carattere 3? entrambi sono rappresentati dalla sequenza binaria 00110011!

La risposta è che

sono diverse le operazioni che vengono compiute su di essi!

Cioè: nei linguaggi ad alto livello come Java il numero 51 e il carattere '3' vengono espressi in due modi diversi; ma, proprio per questo, lo stesso comando di visualizzazione viene tradotto nel linguaggio della macchina in due comandi diversi: nel primo caso il comando visualizza (in notazione decimale) proprio 51, nel secondo caso consulta la tabella UNICODE e visualizza 3.

Similmente, la sequenza di caratteri o *stringa* "62" è una cosa diversa dal numero 62; come vedremo, "62" + "51" è diverso da 62 + 51 !

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; esempi:

. . .

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*

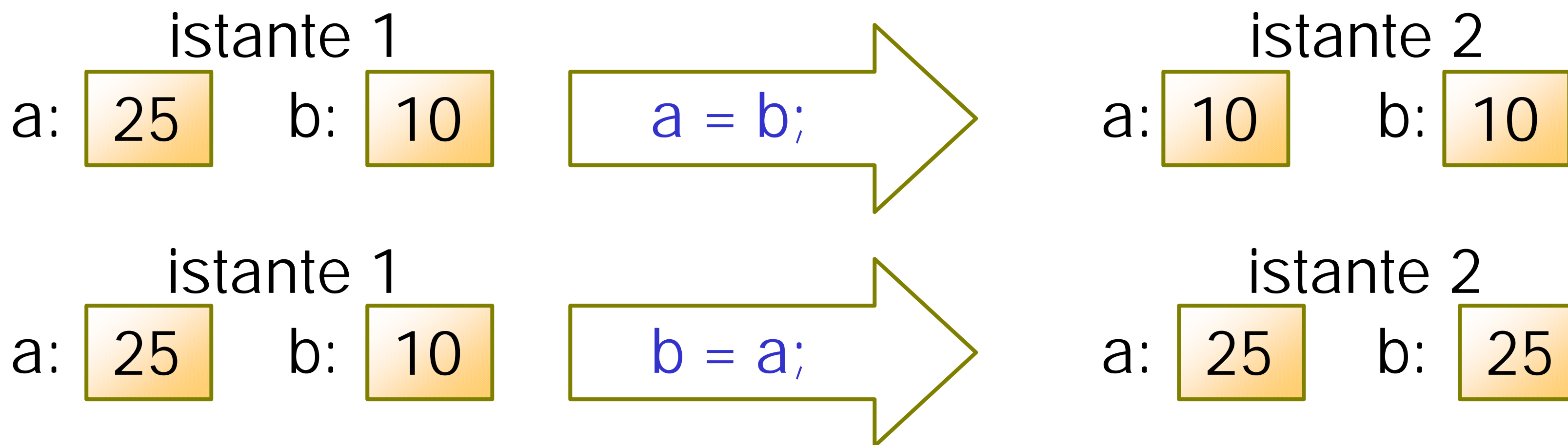
subito dopo l'esecuzione dell'istruzione i contenuti di *a* e di *b* saranno uguali, ma poi possono di nuovo variare indipendentemente.

Programmazione imperativa: assegnazione (cont.)

NOTA BENE:

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

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 e b “si riferiscano allo stesso oggetto”

Il significato preciso di tale affermazione
sarà chiarito nelle prossime lezioni.

Procedure e 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* 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:

```
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.

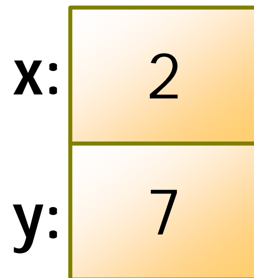
Procedure e funzioni (cont.)

Interagendo con un interprete Javascript, se dopo aver immesso la definizione:

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

si immette l'espressione `f(2,7)`, nella memoria centrale del calcolatore succede la cosa seguente:

in un'area chiamata *stack* o *pila* viene creato un contenitore chiamato *frame di f* (*o record di attivazione di f)*, composto di due celle di nomi rispettivamente x e y, e in tali celle vengono depositati i valori 2 e 7.



In Java succede la stessa cosa, sia pure con alcune complicazioni in più.

Nota Bene. In inglese *return* vuol dire *restituisci*, quindi la definizione Javascript

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

o l'analoga Java

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

vuol dire:

- il mio nome è f;
- posso venire chiamato (o attivato), ma all'atto della chiamata devo ricevere una sequenza di due valori, che deposito il primo nel contenitore x e il secondo nel contenitore y;
- il comando che eseguo quando vengo chiamato è:
  restituisci al chiamante il valore che si ottiene facendo

  3* (contenuto di x) + 5* (contenuto di y)

Procedure o funzioni (cont.)

In generale, una procedura o funzione o 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 mette i dati che serviranno alla procedura 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, oppure quello di operare dei cambiamenti dello stato della memoria (in senso lato, comprendente lo stato dello schermo, ecc.) senza restituire un valore.

Terminologia

In linguaggi come il Pascal, il termine *funzione* indica un sottoprogramma che restituisce qualcosa al 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.

Esempio di procedura che non restituisce nulla

```
void salutaSuSchermo(String s)
{   istruzioni che hanno l'effetto di visualizzare sullo schermo la
    stringa "ciao" seguita dalla stringa passata come parametro;
}
```

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

Invece:

```
int lunghezza(String s)
{   istruzioni che computano la lunghezza della stringa,
    mettendola in una variabile di nome lun;
    return lun;
}
```

restituisce al programma chiamante un numero intero.

I nomi dei parametri valgono solo dentro la procedura

Se al di fuori della definizione di funzione

```
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 !!!

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: da quanto precede dovrebbe essere chiaro che

Non bisogna confondere
la definizione di una procedura (o funzione)
con la sua invocazione!

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

void salutaSuConsolle(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);   salutaSuConsolle(“Carlo”);
double a = f(4,6);   f(a,1); ...
```

Domanda: chi è che può chiamare una procedura? Risposta: un'altra procedura!

Un moderno prodotto software è di solito un programma costituito di molti pezzi (cioè procedure) 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 una procedura vi sono spesso invocazioni di altre procedure, e non bisogna confondere le definizioni con le invocazioni.

Esempio di procedure che invocano procedure

definizione della procedura salutaSuConsolle

```
void salutaSuConsolle(String nome) {
  Consolle.println("Ciao " + nome);
}
```

definizione della procedura salutaDaFinestra

```
void salutaDaFinestra(String nome) {
  SimpleWindows.showMessageDialog("Ciao " + nome );
}
```

definizione della procedura salutaElio

```
void salutaElio() {
  salutaSuConsolle(“Elio");
  salutaDaFinestra(“Elio");
}
```

salutaElio invoca salutaSuConsolle 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, possiamo immaginare che Consolle.println e SimpleWindows.showMessageDialog, di cui non abbiamo riportato le definizioni, siano procedure predefinite di libreria.

Anche in questo corso utilizzeremo procedure predefinite delle librerie Java.

Ma chi comincia il tutto?
Il main (cioè il principale)!

In ogni programma C o C++ o Java ci deve essere una procedura 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(...)   {
        salutaElio();
        ...
}
```

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 sono di diversi tipi, e dentro una cella 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!

Dati e procedure

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

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.

Programmazione a oggetti.

- oggetto: è una "entità" software (in memoria centrale) costituita da un insieme di celle di memoria dette campi dell'oggetto, o variabili dell'oggetto, o variabili di istanza, e da un insieme di procedure dette metodi, con cui operare su tali celle;
- classe: è una specie di "stampo" per costruire oggetti aventi caratteristiche comuni, cioè è un pezzo di software contenente:
  - *la descrizione* dei campi che ogni oggetto della classe deve avere;
  - le istruzioni che compongono i metodi che permettono di agire sugli oggetti;
  - delle speciali procedure dette costruttori, aventi lo stesso nome della classe, che permettono di (creare e) inizializzare gli oggetti di quella classe.

ATTENZIONE: IMPERFEZIONI DI TRADUZIONE

Nel libro di Horstmann, nella traduzione italiana:

i campi, o variabili ***di*** istanza (o variabili ***dell'*** oggetto), vengono chiamati variabili istanza.

Si tratta di una **traduzione errata** dell'inglese instance variables:

istanza (di una classe) è sinonimo di oggetto (di una classe);

un campo è una variabile ***di*** un'istanza, cioè ***di*** un oggetto, **non è una variabile-oggetto**!

Lo stesso testo, poi, chiama variabili oggetto (pag. 37) le variabili di tipo riferimento-a-oggetti (di una data classe).

Anche questa è una terminologia non-standard e fuorviante.

CORREZIONI AL LIBRO

Sostituire ovunque:

variabile istanza → variabile **di** istanza, o campo

variabile oggetto → variabile di tipo riferimento-a-oggetto

Vedremo in seguito che le variabili di tipo riferimento-a-oggetto possono essere di tre generi:

- riferimento ad oggetto-array;
- riferimento ad oggetto di una data classe;
- riferimento ad un oggetto avente una data interfaccia.

Un pezzo di programma Java. Nel file Conto.java:

```
public class Conto {
  private double saldo; // campo
  private double tasso; // campo
  public Conto() {…}  // costruttore
  public Conto(double saldoIniziale)
  {…}                        // costruttore
  public void deposita(double importo)
  {…}    // metodo (che cambia lo stato)
  public void preleva(double importo)
  {…} // metodo (che cambia lo stato)
  public double getSaldo() {…}// metodo
}
```

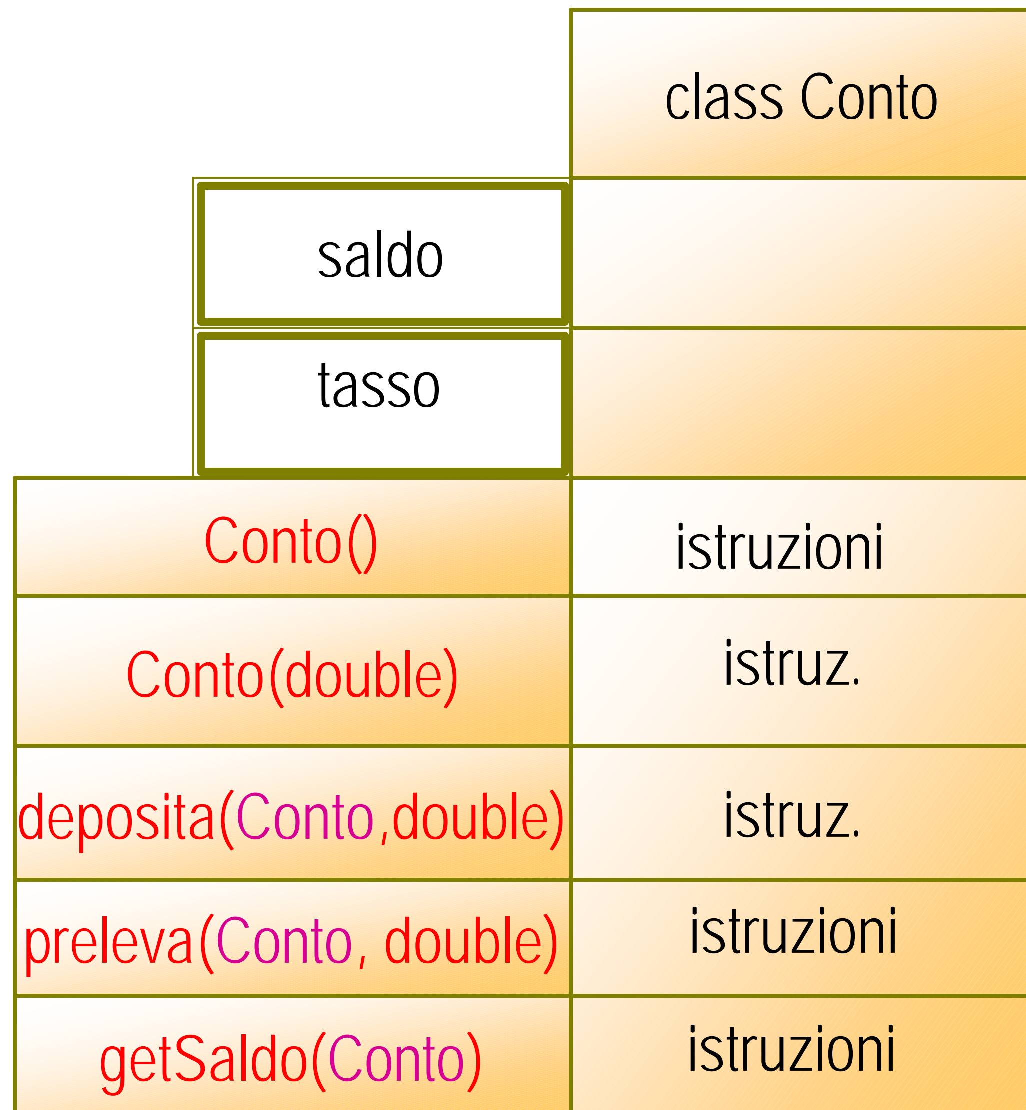
Classi e oggetti

	class Conto
saldo	
tasso	
Conto()	istruzioni
Conto(double)	istruz.
deposita(double)	istruz.
preleva(double)	istruzioni
getSaldo()	istruzioni

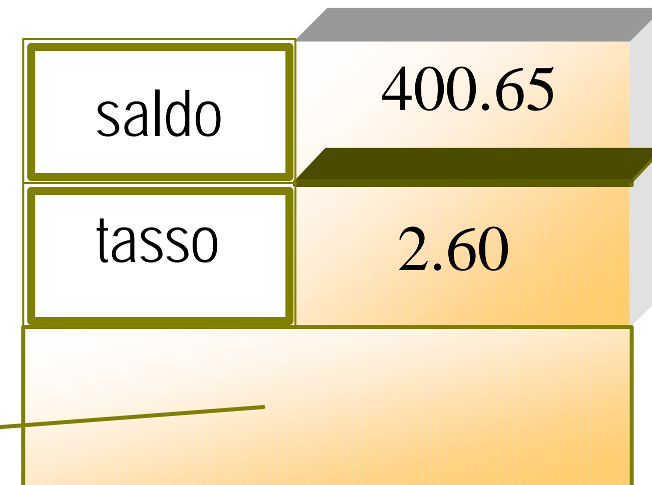
oggetto:

saldo	400.65
tasso	2.60
Conto()	istruzioni
Conto(double)	istruzioni
deposita(double)	istruzioni
preleva(double)	istruzioni
getSaldo()	istruzioni

o meglio:



oggetto:



le istruzioni costituenti i vari metodi non vengono duplicate in ogni oggetto: esse stanno solo nella classe, e nell'oggetto vi è solo l'indirizzo (rappresentato dalla freccia) a cui trovare tali istruzioni.

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* **[ ]**

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!

Il linguaggio Java.
Il solito primo programma. Nel file Ciao.java:

```
public class Ciao {
  public static void main(String[] args) {
    System.out.println("Ciao, ragazzi!");
  }
}
```

Compilazione ed esecuzione

Un file Ciao.java è un file di testo (che sta sullo hard disk).
Il compilatore legge tale file e genera il file Ciao.class, che è una traduzione del programma in una forma più conveniente per l'esecuzione.
L'esecutore esegue (o interpreta) il file Ciao.class

Nota Bene: il compilatore e l'esecutore non sono macchine fisiche, bensì dei programmi che vengono eseguiti dalla macchina fisica. Durante l'esecuzione del programma Ciao la macchina esegue il programma java.exe che esegue il programma Ciao.

Come si scrive un programma Java

- adottare un font a spaziatura fissa (ad es. Courier)
- fissare il tab (cioè la tabulazione) a 2 o 3 spazi
- 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

Il MS Arial Unicode

Questo è
un tipo di carattere (font)
a spaziatura *variabile*,
senza grazie (sans serif).

Il **Times New Roman**

Questo è
un tipo di carattere (font)
a spaziatura *variabile*,
con grazie (with serifs).

Il **Courier New**

Questo è

un tipo di carattere

a spaziatura *fissa*

(ingl. *monospaced*).

RISORSE

java.sun.com/j2se/downloads.html

(il Java ufficiale: scaricare la versione

J2SE 1.4.1 o J2SE 1.4.0;

sul sito java.sun.com vi è una grande quantità di documentazione e di materiale)

www.bluej.org

(scaricare l'ultima versione; sul sito c'è anche un libro scaricabile, di introduzione alla programmazione con Java e BlueJ)