

Algoritmi e programmazione

- Definizioni
 - Definizione di algoritmo
 - I linguaggi di programmazione
 - Linguaggi compilati e linguaggi interpretati
- Nozioni fondamentali di programmazione i JAVA
 - I tipi di dato
 - Gli oggetti
 - Il controllo del flusso



Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Algoritmi



Algoritmi

- **Algoritmo:** sequenza di istruzioni attraverso le quali si risolvere un problema di una data classe; non è direttamente eseguibile dall'elaboratore
- **Programma:** sequenza di operazioni atte a predisporre l'elaboratore alla soluzione di una determinata classe di problemi
 - Il programma è la descrizione di un **algoritmo** in una forma comprensibile all'elaboratore
- L'elaboratore è una **macchina universale**: cambiando il programma residente in memoria, è in grado di risolvere problemi di natura diversa (una classe di problemi per ogni programma)

Franco Scariselli

Fondamenti di Informatica I, 2005-2006



Esempio: raggruppamento per seme delle carte

- **Problema:** Sia dato un mazzo di carte da ordinare in modo che le cuori precedano le quadri, che a loro volta precedono fiori e picche; le carte di uno stesso seme sono ordinate dall'asso al re
- **Algoritmo:**
 - 1) Si suddivide il mazzo in 4 mazzetti, ciascuno costituito da tutte le carte dello stesso seme
 - 2) Si ordinino le carte di ciascun mazzetto dall'asso al re
 - 3) Si prendano nell'ordine i mazzetti delle cuori, quadri, fiori e picche
- **Osservazioni**
 - L'algoritmo ordina le carte qualunque sia il modo in cui sono ordinate le carte inizialmente e sia per i mazzi da 0 carte che per quelli da 52: l'algoritmo risolve una classe di problemi

Franco Scariselli

Fondamenti di Informatica I, 2005-2006

Esempio: calcolo delle radici di un'equazione di secondo grado

- **Problema:** Calcolo delle radici reali di $ax^2+bx+c=0$

- **Algoritmo:**

- 1) Acquisire i coefficienti a, b, c
- 2) Calcolare $\Delta = b^2 - 4ac$
- 3) Se $\Delta < 0$ non esistono radici reali, eseguire l'istruzione 7)
- 4) Se $\Delta = 0$, $x_1 = x_2 = -b/2a$, poi eseguire l'istruzione 6)
- 5) $x_1 = (-b + \sqrt{\Delta})/2a$, $x_2 = (-b - \sqrt{\Delta})/2a$
- 6) Comunicare i valori x_1, x_2
- 7) Fine

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Esempio: ordinamento dei CD musicali per autore e titolo

- **Problema:** Dato un insieme di CD musicali si vogliono ordinare per il nome dell'autore e , in caso di CD dello stesso autore, per titolo e inserirli in un contenitore

- **Algoritmo:**

- 1) Si scorrono tutti i CD ricordandosi dell'autore con il nome che precede tutti gli altri
- 2) Si scorrono nuovamente tutti i CD cercando quello dell'autore individuato al passo 1) e con il titolo che precede gli altri
- 3) Si inserisce il il CD individuato dai passi 1) e 2) nella prima posizione vuota del contenitore
- 4) Se l'insieme non è vuoto si ripetono i passi 1), 2) e 3)

- **Osservazioni**

- Si poteva definire un algoritmo più efficiente possono esistere più algoritmi per risolvere lo stesso problema

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



Proprietà degli algoritmi

- Affinché un elenco di istruzioni, possa essere considerato un algoritmo, devono essere soddisfatti i seguenti requisiti:
 - **Finitzza:**
ogni algoritmo deve essere finito, cioè ogni singola istruzione deve poter essere eseguita in tempo finito ed un numero finito di volte
 - **Generalità:**
ogni algoritmo deve fornire la soluzione per una classe di problemi; deve pertanto essere applicabile a qualsiasi insieme di dati appartenenti **all'insieme di definizione o dominio dell'algoritmo** e deve produrre risultati che appartengano **all'insieme di arrivo o codominio**
 - **Non ambiguità:**
devono essere definiti in modo univoco e chiaro tutti i passi da eseguire

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



Programmare:
cosa è un linguaggio, compilatore....



Algoritmi e Linguaggi di programmazione

Caratteristiche intuitive dei linguaggi di programmazione

- Intuitivamente, permettono di specificare algoritmi in modo comprensibile ad un calcolatore
- Sono dotati di strutture linguistiche che garantiscono precisione e sintesi:
 - il linguaggio naturale non ha queste caratteristiche perché alcune frasi possono essere ambigue e la stessa cosa può essere detta in maniere diverse
- Sono sufficientemente precisi da poter di rispondere a domande come:
 - quali sono le frasi lecite?
 - si può stabilire se una frase appartiene al linguaggio?
 - come si stabilisce il significato di una frase?
 - quali sono gli elementi linguistici primitivi?

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



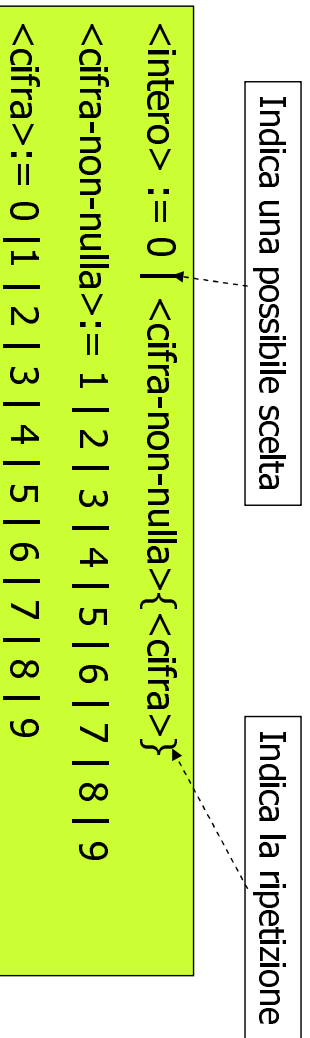
Linguaggi di programmazione

In questo corso non ci soffermeremo su questo aspetto, ma

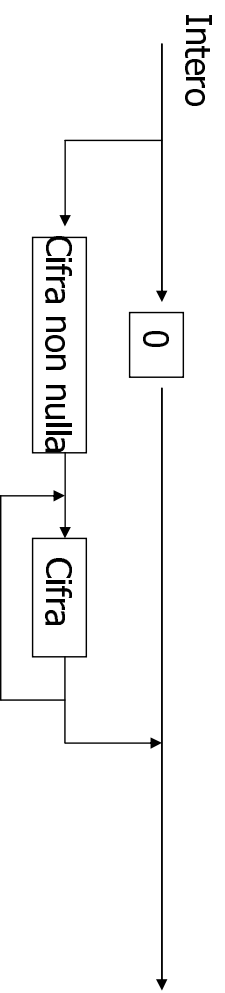
- I linguaggi di programmazione sono sistemi matematici definiti **formalmente** da tre componenti
 - **Lessico**: un insieme di regole formali per la scrittura delle parole (i componenti elementari)
 - **Sintassi**: un insieme di regole formali per la scrittura di frasi, che stabiliscono cioè la grammatica del linguaggio stesso
 - **Semantica**: un insieme dei significati da attribuire alle frasi (sintatticamente corrette) costruite nel linguaggio
- **Nota**: una frase può essere sintatticamente corretta e tuttavia non avere significato!

Esempio

- Definizione formale di numero intero positivo per mezzo di regole sintattiche



Rappresentazione
grafica di una regola



Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Linguaggi di basso e alto livello

Linguaggi di basso livello

- L'assembly è un linguaggio di basso livello
- Simili al linguaggio macchina, molto diversi dal linguaggio umano
- Variano a seconda del computer su cui si scrive il programma
- Programmare con linguaggi a basso livello richiede molto tempo

Linguaggi di alto livello

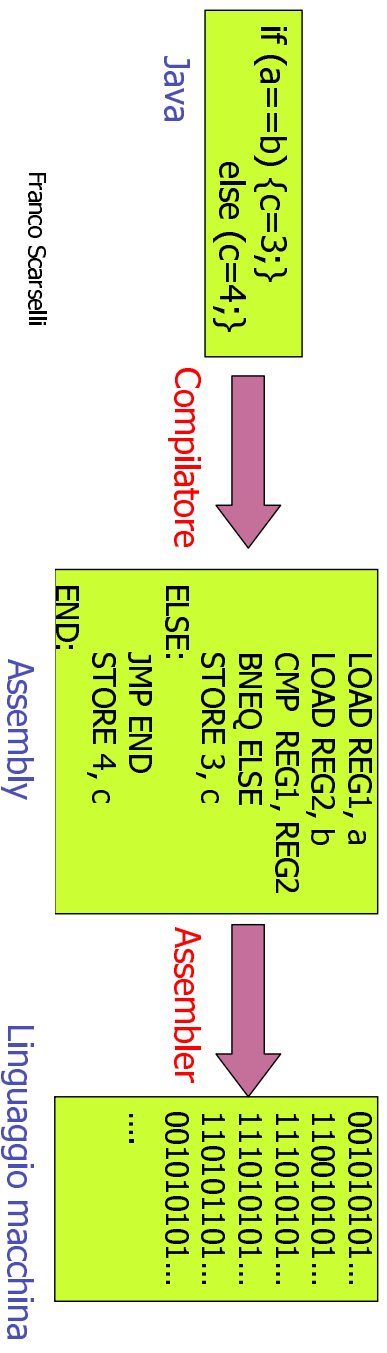
- Sono indipendenti dalla macchina
- Sono piu' vicini al linguaggio umano, meno al linguaggio macchina
- Programmare con linguaggi ad alto livello richiede meno tempo
- Java, C++, Basic,

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Linguaggi di alto livello e compilazione

- Contengono istruzioni piu' vicine al linguaggio umano
 - "se QUESTA CONDIZIONE È VERA, FAI QUESTO, altrimenti FAI QUEST'ALTRO"
 - "ripeti QUESTO fino a che si verifica QUESTA CONDIZIONE"
- Poiche' i computer non possono eseguire i linguaggi ad alto livello, occorre **compilare** (tradurre) il programma prima di poterlo eseguire



Compilatori e ambienti di sviluppo

Gli ambienti di sviluppo

- includono un compilatore e aiutano il programmatore nella scrittura del codice
- quelli piu' moderni si chiamano RAD (Rapid Application Development Tools) e sono sofisticatissimi che forniscono una grande quantita' di strumenti
- includono editor specializzati, strumenti per controllare la correttezza del programma, strumenti per testare il programma, documentazione, strumenti per lo sviluppo visuale del software,
- In laboratorio ne useremo uno: **JBuilder**

Il processo di sviluppo del software

Il processo di sviluppo del software è costituito da tre fasi

- **Analisi e progetto (per applicazioni medie, 30% del tempo)**
Si studia il problema documentandosi. Si realizza e si raffina il progetto e fino a giungere ad una definizione dettagliata delle funzionalità necessarie e dei vincoli richiesti.
 - **Scrittura del software (30% del tempo)**
Si scrive il programma
 - **Test (per applicazioni medie, 40% del tempo)**
Si prova il programma, prima la verifica viene fatta dai programmatori, poi sul campo...
- 

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Introduzione alla programmazione attraverso Java

Componenti elementari di Java

- Un programma Java consiste di insieme di componenti semplici chiamate **classi**
- Ogni classe definisce un insieme di dati (**variabili**) e un insieme di funzioni con cui si può operare sui quei dati (**metodi**)
- Una classe contiene un tipo di dato astratto (**Abstract Data Type**) che può corrispondere a più **oggetti** concreti
- Ogni variabile ha associato un **tipo** che definisce quali tipo di valori può assumere: sono ammessi
 - **tipi semplici** (interi, Floating point, caratteri, ..)
 - e **tipi strutturati**, ad esempio oggetti, composti da più tipi elementari

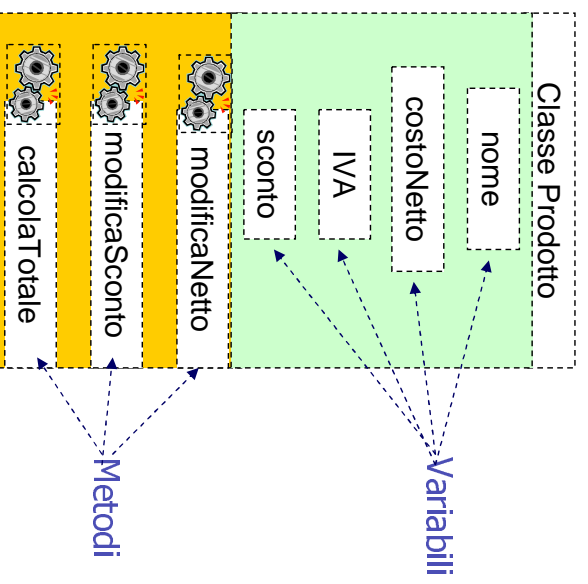
Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Esempio: il prodotto di un supermercato

Un prodotto

- un prodotto è caratterizzato da un nome, un costo netto, la percentuale IVA e un eventuale sconto. Ci è stato richiesto di sviluppare un modulo software che memorizza un prodotto, permette di modificare il costo netto, lo sconto e di calcolare il costo totale.

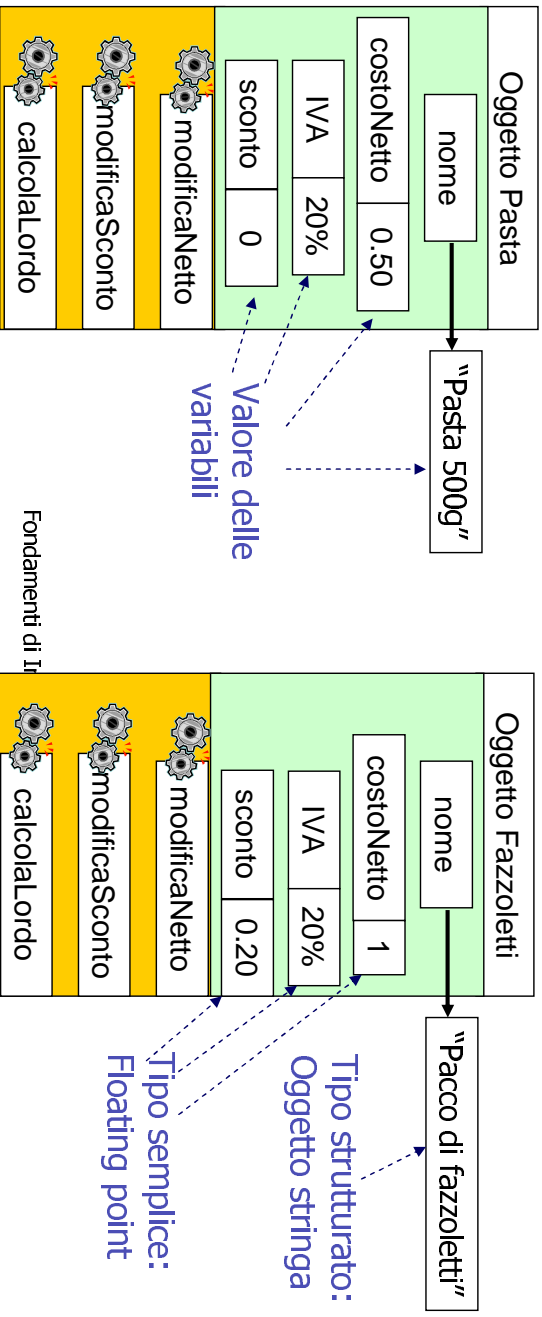


Franco Scarselli

Fondamenti di Informatica I, 2005-2006

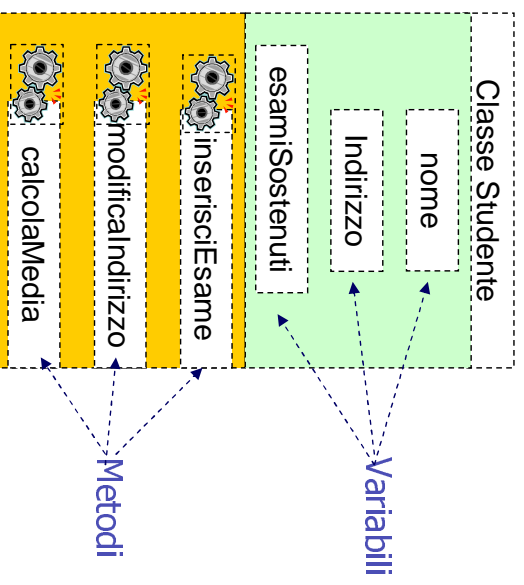
Esempio: il prodotto di un supermercato II

- Negli oggetti concreti le variabili assumono un valore
- Le classi sono le definizioni e gli oggetti sono moduli software funzionanti
- Ad ogni classe possono corrispondere più oggetti (istanze della classe)



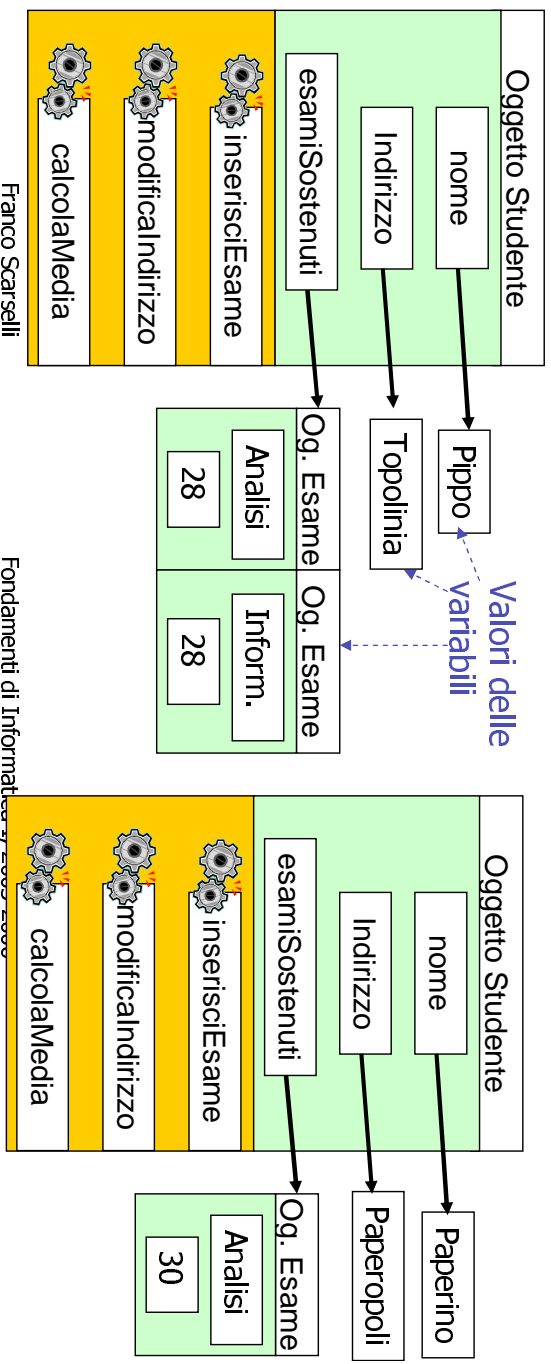
Esempio: uno studente

- Uno studente ha un nome, un indirizzo e un curriculum. Ci è stato richiesto di sviluppare un modulo software che permette di inserire nuovi esami, calcolarne la media e modificare l'indirizzo.



Esempio: uno studente II

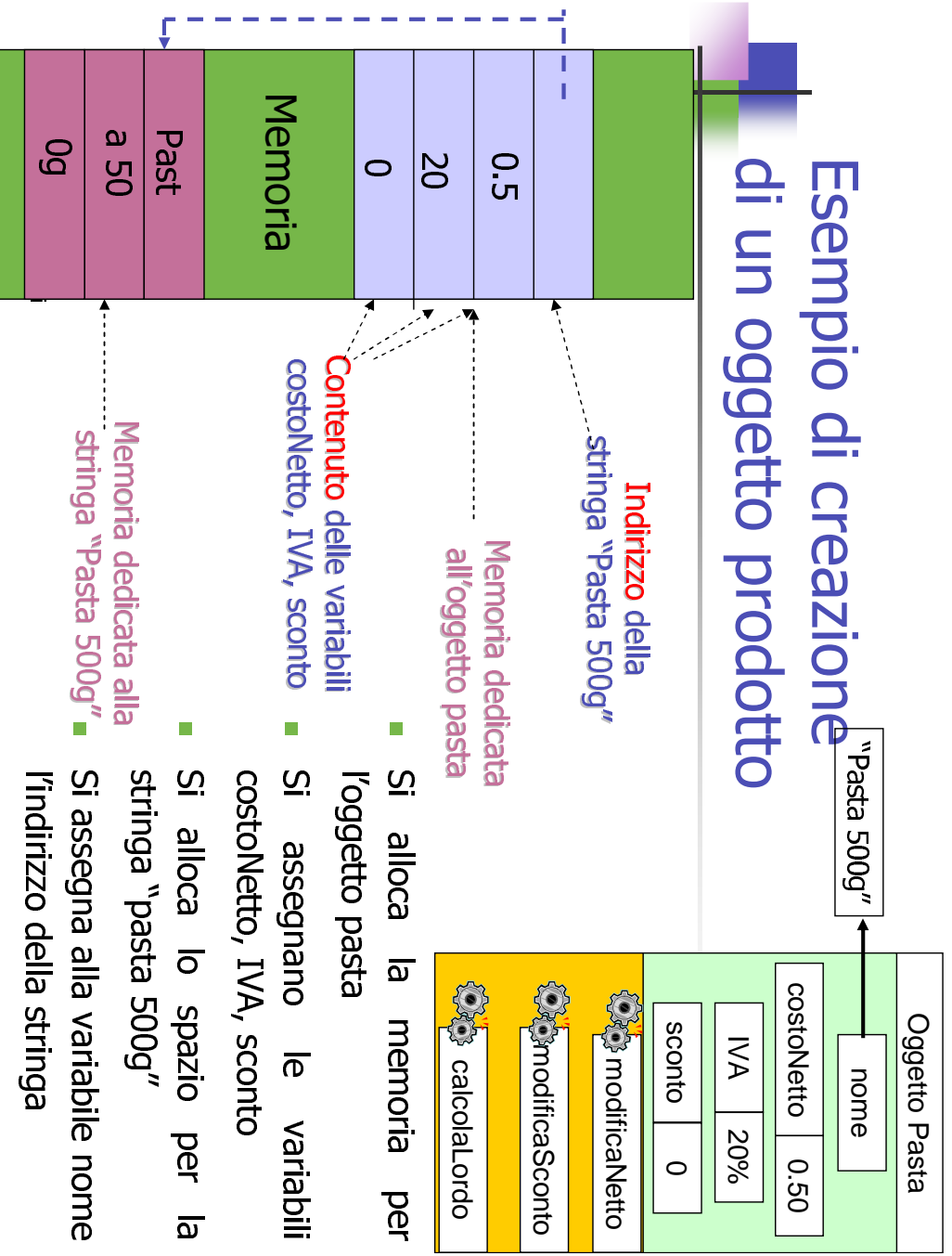
- Negli oggetti concreti le variabili assumo un valore
- Ad ogni classe possono corrispondere più oggetti (**istanze** della classe)



Classi e oggetti

- Una classe è una definizione astratta. Un oggetto è un'implementazione concreta di una classe.
- Java mette a disposizione un'istruzione (**new**) per **creare un oggetto** a partire dalla sua classe
- Nel momento in cui un oggetto viene creato
 - uno **spazio di memoria** viene **predisposto** (allocato) per la sua memorizzazione
 - Per ogni variabile semplice si alloca lo spazio per memorizzarne il **valore**
 - Per ogni variabile strutturata si alloca lo spazio per memorizzare un **handler** (un puntatore) al contenuto della variabile strutturata
 - Il contenuto delle variabili può essere assegnato **al momento della creazione o modificato successivamente**

Esempio di creazione di un oggetto prodotto



Osservazione importante

- La programmazione attraverso oggetti e classi si chiama **Object Oriented Programming (OOP)**
- Ogni oggetto è un'unità software che racchiude (**incapsula**) un insieme di dati omogeneo e offre i metodi per poterci lavorare
- Se un oggetto è fatto bene, può essere riusato in più applicazioni
 - ad esempio, l'oggetto studente può essere usato nell'anagrafica della segreteria, nella gestione dei tirocini, ...
- Se un programma è diviso bene in oggetti, più persone possono lavorare sulla stessa applicazione contemporaneamente: ognuno realizza alcuni oggetti



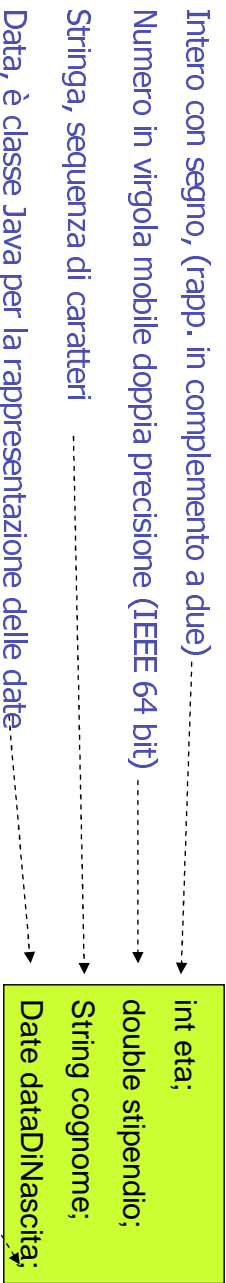
Le variabili



Le variabili

Una variabile

- rappresenta un'area di memoria riservata dal compilatore per memorizzare dati
- è definita da un nome univoco che la identifica e da un tipo che definisce i tipi di dati che può accettare



Il punto e virgola



Le variabili

In Java una variabile deve essere dichiarata prima di essere usata (altri linguaggi non hanno questo vincolo)

- Dichiarare una variabile significa avvertire il compilatore che riservi un'area di memoria perché in futuro essa verrà utilizzata con quel nome
- Se una variabile viene dichiarata di tipo T, necessariamente essa potrà memorizzare valori di tipo T
- Il compilatore darà un messaggio di errore sia in caso di mancata dichiarazione della variabile che in caso gli si assegni un valore di tipo sbagliato

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



I nomi

- I nomi delle variabili, dei metodi e di tutti gli altri elementi di un linguaggio devono rispettare delle regole ben precise

In Java il nome di una variabile

- Può contenere lettere (minuscole o maiuscole), cifre e il carattere ``_``
- Non può essere una delle **parole riservate del linguaggio** (if, else, while, return ecc...)
- Non può iniziare con una cifra

```
mia eta
12stipendio
co*ca
repeat
```

nomi scorretti

```
miaEta
stipendio12
co_ca
ripetizione
```

nomi corretti

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Tipi di dato

- I tipi di dato si distinguono in tipi semplici e tipi strutturati:
 - tipi di dato semplici** – sono tipi di dato a cui può essere associato un singolo valore (numerico o carattere) ed un riferimento alla variabile è un riferimento al contenuto
 - tipi di dato strutturati** – sono tipi di dato composti da più “campi”, da cioè uno o più altri tipi di dato a loro volta semplici o strutturati

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Tipi di dato semplici

<i><u>Tipo</u></i>	<i><u>Dimensione</u></i>	<i><u>Min</u></i>	<i><u>Max</u></i>
boolean	1 bit	-	-
char	16 bit	0	$2^{16}-1$
byte	8 bit	-128	+127
short	16 bit	- 2^{15}	+ $2^{15}-1$
int	32 bit	- 2^{31}	+ $2^{31}-1$
long	64 bit	- 2^{63}	+ $2^{63}-1$
float	32 bit	IEEE 754	
double	64 bit	IEEE 754	
void	-	-	-

Un carattere ossia
l'intero positivo che
lo codifica

Tipo speciale per
indicare un valore
non definito

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Assegnamento

- L'assegnamento permette di definire il valore di una variabile:
 - il valore assegnato deve essere compatibile col tipo della variabile, altrimenti si genera un errore quasi sempre
 - ad una variabile può essere assegnato un valore anche contemporaneamente alla sua definizione
 - ad una variabile può essere assegnato il valore di una espressione ottenuta con gli operatori `"/"`, `"+"`, `"-"`, `"*"`, `"%"` e altri ..

errore

```
int a, b;  
float v;  
char c = 'M';  
a = 4;  
b = a;  
v = 3.2*12-a;  
b = v;
```

definizione e assegnamento
b prende il valore di a
v prende il valore di una espressione

Franco Scarselli
Fondamenti di Informatica I, 2005-2006

Casting

- Con **cast** (forzatura) si indica l'operazione di forzare la trasformazione di un tipo di dato ad un altro
- Il cast può essere esplicito o implicito:
 - **cast esplicito** - l'utente specifica il tipo di dato finale in cui desidera lavorare (tra parentesi tonde)
 - **cast implicito** - il compilatore automaticamente decide il tipo di dato finale in base al tipo di dato della variabile in cui il valore va ad essere memorizzato. Le conversioni ammesse implicitamente sono quelle con cui non si perde informazione:
byte → short → int → long → float → double

Casting: esempi

- È corretto o no?

```

short a=1.2;
int b;
long c=10;
float d=3;
double e;
char f='M';

a=c;
a=(short) c;
e=d;
d=c;
d=e;
d=(double) e;

```

errore

ok

errore

ok, ma se c fosse troppo grande...

ok

errore

ok, ma ci potrebbe essere perdita di precisione

<i>Type</i>	<i>Dim</i>	<i>Min</i>	<i>Max</i>
char	16 bit	0	$2^{16}-1$
byte	8 bit	-128	+127
short	16 bit	-2^{15}	$+2^{15}-1$
int	32 bit	-2^{31}	$+2^{31}-1$
long	64 bit	-2^{63}	$+2^{63}-1$
float	32 bit	IEEE 754	
double	64 bit	IEEE 754	

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Casting: esempi

- Cosa stampano?

```

int b=32768;
short a=(short) b;
System.out.println(a);

double e=4E+38; // piu' grande del
                // massimo numero per
                // float

float d=(float)e;
System.out.println(e);
System.out.println(d);

e=1+1E-9;
d=(float)(1+1E-9); // 1E-9 è piu' piccolo
                  // di 2^-23
System.out.println(e);
System.out.println(d);

```

//uguale a 2^{15}

// massimo numero per float

Commenti, il compilatore non li considera

<i>Type</i>	<i>Dim</i>	<i>Min</i>	<i>Max</i>
char	16 bit	0	$2^{16}-1$
byte	8 bit	-128	+127
short	16 bit	-2^{15}	$+2^{15}-1$
int	32 bit	-2^{31}	$+2^{31}-1$
long	64 bit	-2^{63}	$+2^{63}-1$
float	32 bit	IEEE 754	
double	64 bit	IEEE 754	

Notazione scientifica

-32768

4.0E38

Infinity

1.0000000001

1.0

Tipi di dato strutturati e oggetti

- Sono tipi di dato composti da più "campi", da cioè uno o più altri tipi di dato a loro volta semplici o strutturati
- In Java i tipi di dati strutturati sono oggetti
- Per definire una classe si usa la parola chiave **class**
- Per usare un oggetto occorre
 - crearlo istanziando una classe con l'operatore **new**
 - l'operatore **new** produce un **handle** che può essere assegnato ad una variabile
 - (l'handle è l'indirizzo (puntatore) dell'area di memoria dove è allocato l'oggetto)
- L'informazione interna all'oggetto può essere acceduta postponendo un **punto all'handle**

Franco Scarselli

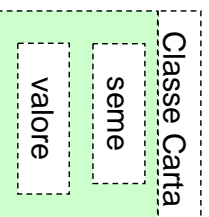
Fondamenti di Informatica I, 2005-2006

Esempio di un oggetto: una carta

- Una carta è definita da un seme e dal suo valore

```
class Carta {  
    char seme;  
    char valore;  
}
```

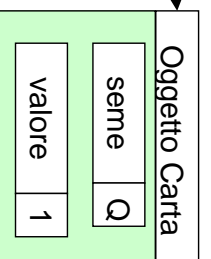
Definisce



- Nella classe QuattroAssi potrebbe esserci

```
class QuattroAssi {  
    ....  
    Carta assoQuadri= new Carta();  
    ....  
    assoQuadri.seme='Q';  
    assoQuadri.valore='1';  
    ....  
}
```

handle: indirizzo dell'oggetto

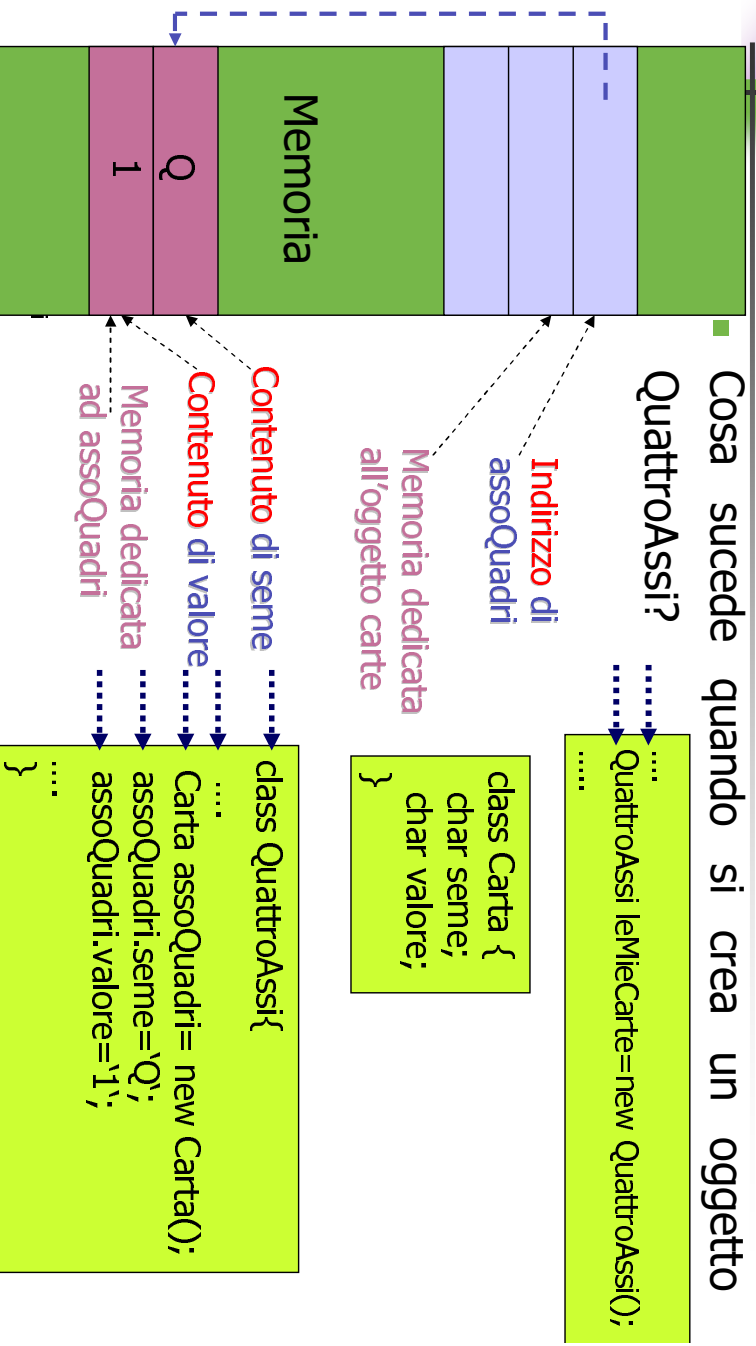


Inizialmente
l'oggetto non
esiste viene
creato con
l'istruzione **new**

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Esempio di creazione di un oggetto quattroAssi



Accesso alle variabili di un oggetto

- Per accedere ad una variabile contenuta in un oggetto occorre scrivere il nome dell'oggetto seguito da un punto e il nome della variabile
- La notazione può essere iterata se un oggetto contiene un oggetto che contiene un oggetto che contiene....

In qualche classe potrebbe esserci scritto...

```
QuattroAssi leMieCarte=new QuattroAssi();  
System.out.println(leMieCarte.assoQuadri.seme);  
System.out.println(leMieCarte.assoQuadri.valore);
```

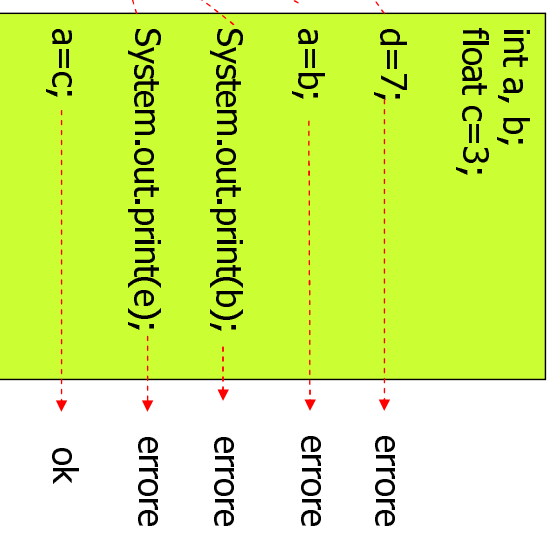
```
class Carta {  
    char seme;  
    char valore;  
}
```

```
class QuattroAssi {  
    ....  
    Carta assoQuadri= new Carta();  
    assoQuadri.seme='Q';  
    assoQuadri.valore='1';  
    ....  
}
```

Variabili non definite e non inizializzate

La vita della variabile di un tipo semplice

- **definizione**
 - si alloca lo spazio per la variabile
- **inizializzazione**
 - primo assegnamento di un valore
 - se non è stata definita ... **errore**
- **uso**
 - accesso al contenuto della variabile
 - se non è stata definita ... **errore**
 - se non è stata inizializzata ... **errore**



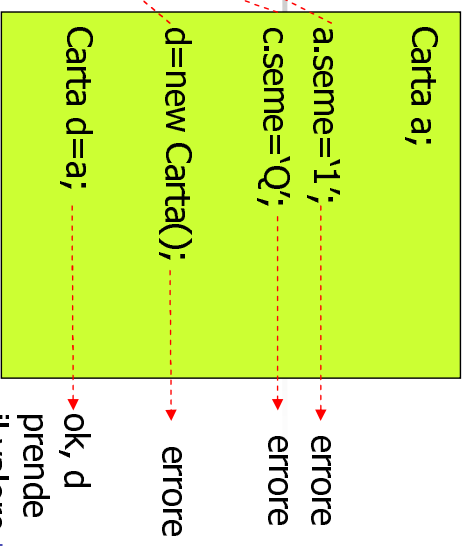
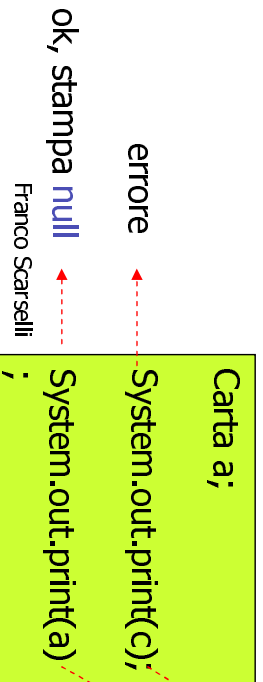
Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Valori nulli

La vita della variabile di un oggetto

- **definizione**
 - si alloca lo spazio per l'handler
 - l'handler assume lo speciale valore **null**
- **inizializzazione**
 - primo assegnamento di un valore e allocazione dell'oggetto
 - se non è stata definita ... **errore**



- **uso**
 - accesso solo alla variabile
 - se non è stata definita ... **errore**
 - se non è stata inizializzata ... **ok**
 - accesso al contenuto dell'oggetto
 - se non è stata definita ... **errore**
 - se non è stata inizializzata ... **errore**

Fondamenti di Informatica I, 2005-2006

Franco Scarselli

La classe String e gli array

- Java contiene numerose classi predefinite che permettono di realizzare le componenti software più frequentemente usate
- Due classi di queste sono speciali, in quanto implementano oggetti molto comuni e JAVA fornisce per loro costrutti sintattici ad hoc, diversi rispetto a tutti gli altri oggetti

- **String**

Implementa le stringhe, cioè le sequenze di caratteri. Permette di memorizzare pezzi di testo, di concatenarli, di paragonarli, etc.

- **Array**

Implementa sequenze di elementi che possono contenere tipi semplici e oggetti. Permette di realizzare degli insiemi, di conoscerne la dimensione, etc.

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

La classe String

- La classe **String** realizza le stringhe, sequenze di caratteri
- Una stringa si rappresenta con una sequenza racchiusa fra doppi apici
- Una stringa può essere inizializzata sia
 - usando l'operatore new
 - assegnando alla variabile una sequenza di caratteri racchiusi fra apici
- L'operatore "+" rappresenta la concatenazione

String a,b,c,d,e;

```
a = new String("Paperino");  
b = "Paolino",  
c = new String();  
d = b + " " + a;
```

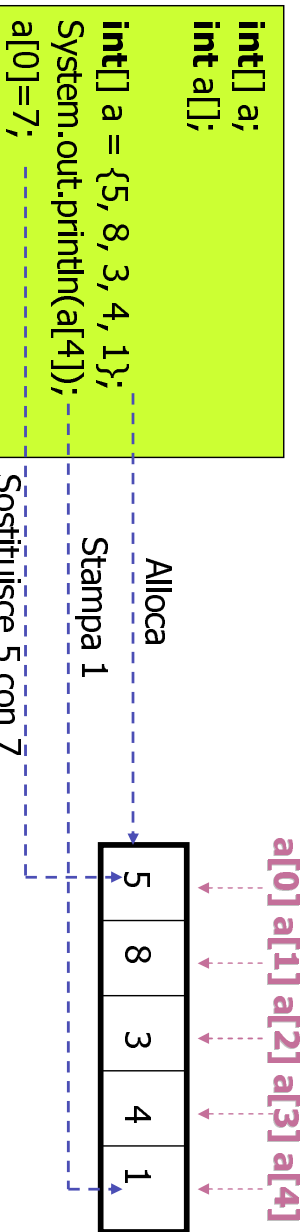
Due assegnamenti corretti
La stringa vuota
La stringa "Paolino Paperino"

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Array

- Permettono di memorizzare sequenze di dati elementari o oggetti
- Si dichiarano aggiungendo alla variabile **due parentesi quadre**
- Un array è rappresentato come una sequenza separata da virgole e racchiusa fra parentesi quadre
- Per accedere ad un elemento si usano le parentesi quadre e un **indice**
 - l'indice del **primo elemento** è 0



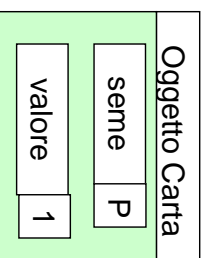
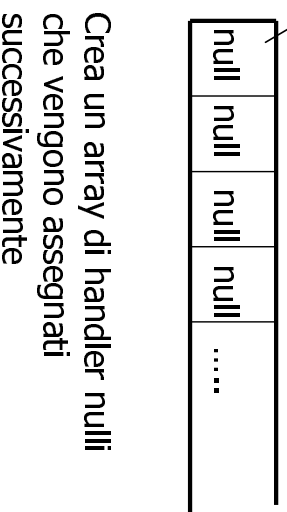
Franco Scarselli

Fondamenti di Informatica I, 2005-2006

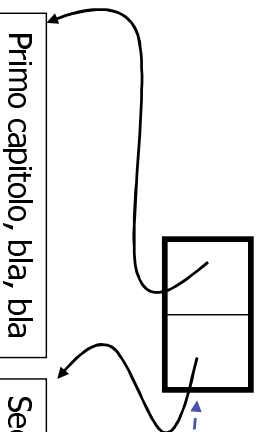
Array II

- Gli array possono contenere anche oggetti

```
Carta mazzoDiCarte = new Carta[40];  
mazzoDiCarte[0]=new Carta();  
mazzoDiCarte[0].seme='P';  
mazzoDiCarte[0].valore='1';  
.....
```



```
String libro[]={"Primo capitolo, bla bla",  
               "Secondo capitolo, bla bla"}
```



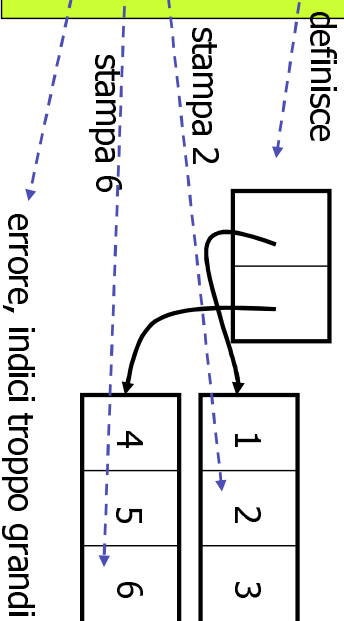
Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Array III

- Gli array possono contenere anche altri array (array multidimensionally)

```
int matrice[][] = {{1,2,3},  
                   {4,5,6}};  
  
System.out.println(matrice[1][2]);  
System.out.println(matrice[2][3]);  
System.out.println(matrice[3][3]);
```



Questi programmi
creano lo stesso array

Franco Scarselli

Fondamenti di Informatica I, 200

```
int matrice[][];  
  
matrice = new int[2][3];  
matrice[1][1]=1;  
matrice[1][2]=2;  
matrice[1][3]=3;  
....
```

I metodi

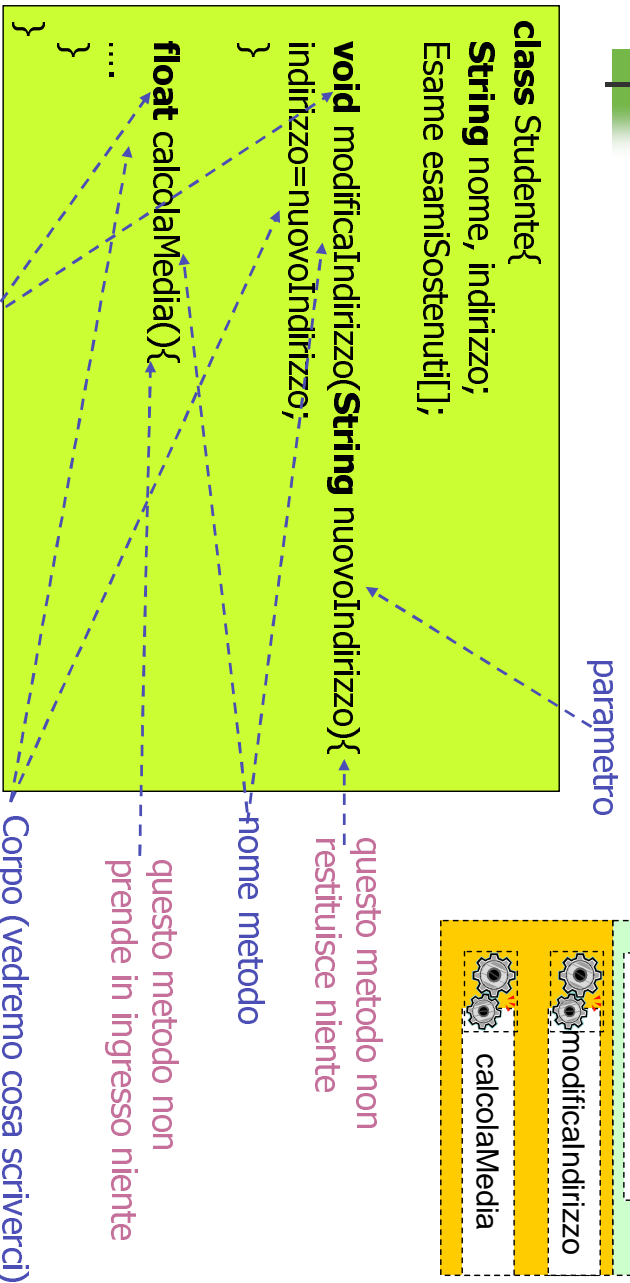
I metodi

- I **metodi** sono **funzioni** che permettono di fare operazioni o derivare informazioni sui dati di un oggetto
- Ogni metodo è caratterizzato da
 - un **nome** che serve ad identificare il metodo fra i vari disponibili nell'oggetto
 - i **parametri** di ingresso, che specificano le informazioni da inviare affinché il metodo possa funzionare. Ogni parametro consiste in una variabile e nel suo tipo
 - il **tipo di dato restituito** dal metodo che specifica quale tipo di informazione il metodo produce
 - un **corpo** che specifica come il metodo funziona

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

I metodi: esempi



Franco Scarselli

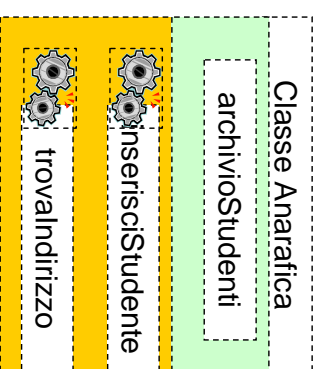
Fondamenti di Informatica I, 2005-2006

I metodi: esempi II

- Un metodo può avere zero o più parametri di ingresso
- Un metodo deve avere uno ed un solo tipo di uscita: se la funzione non restituisce niente allora si usa il tipo **void**

```
class Anagrafica{
    Sudente archivioStudenti[]=new Sudente[1000];

    void inserisciSudente(String nome, String indirizzo){
        ....
    }
    ....
    String trovaIndirizzo(String nome){
        ....
    }
}
```



Franco Scarselli

Fondamenti di Informatica I, 2005-2006

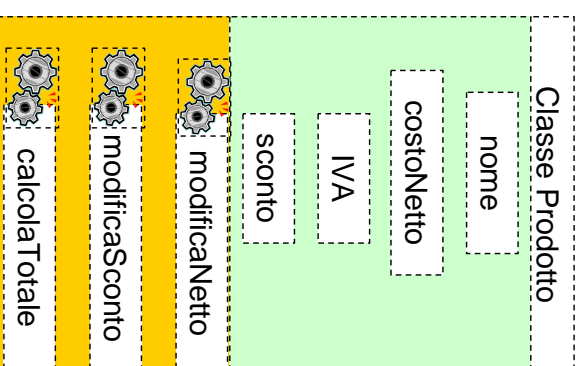
I metodi: esempi III

- L'istruzione **return** definisce il valore restituito da un metodo

```
class Prodotto{
    String Nome;
    float costoNetto, IVA, sconto;

    float calcolaTotale(){
        float totale= costoNetto+IVA-sconto;
        return totale;
    }

    String trovaIndirizzo(String nome){
        ....
    }
}
```



I costruttori

- Sono metodi particolari che servono ad inizializzare un oggetto
- Hanno lo **stesso nome della classe**
- A differenza degli altri metodi **non viene indicato il tipo restituito**
- Si richiamano con **new** e restituiscono l'oggetto

```
class Carta {  
    char seme;  
    char valore;  
  
    Carta(char s, char v) {  
        seme=s;  
        valore=v;  
    }  
}
```

Franco Scarselli

Il costruttore
di Carta

```
class QuattroAssi{  
    Carta assoQuadri= new Carta('Q','1');  
    Carta assoCuori= new Carta('C','1');  
    Carta assoPicche= new Carta('P','1');  
    Carta assoFiori= new Carta('F','1');  
}
```

Fondamenti di Informatica I, 2005-2006

I costruttori II

- Se una classe non ha un costruttore si assume che abbia un costruttore vuoto

```
class Carta {  
    char seme;  
    char valore;  
}
```

Sono equivalenti

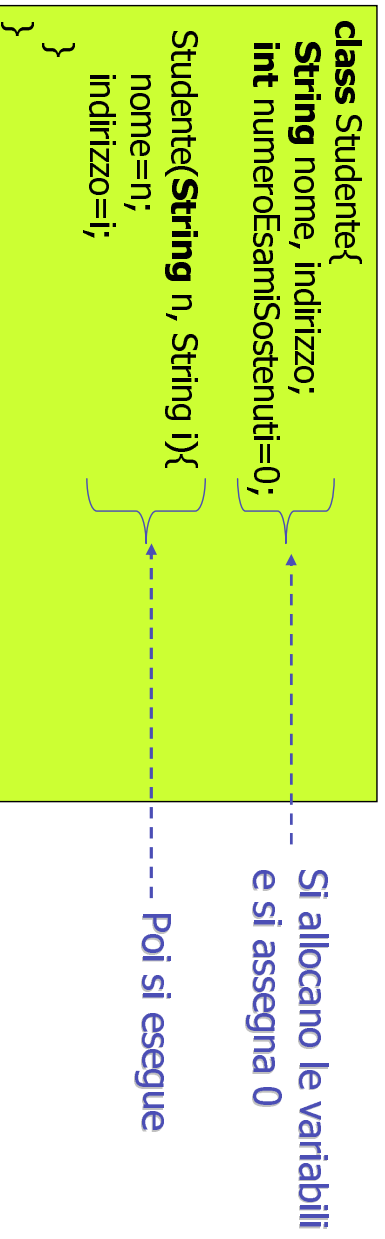


```
class Carta {  
    char seme;  
    char valore;  
    Carta()  
}
```

Inizializzazione oggetto: riepilogo

Quando si inizializza un oggetto (istruzione new)

- 1) Si considera il codice esterno ai metodi e ai costruttori:
 - si allocata la memoria per tutte le variabili dell'oggetto e si eseguono le relative inizializzazioni
- 2) Si esegue il codice del costruttore



Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Controllo del flusso di esecuzione



Controllo del flusso di esecuzione

- I metodi contengono codice che definisce le operazioni che vengono eseguite sui dati
- Il codice, oltre agli assegnamenti, include dei comandi che permettono di realizzare scelte, cicli, ...
- Tali istruzioni permettono di controllare il **flusso di esecuzione**
- Vedremo quali sono le istruzioni Java per il controllo del flusso di esecuzione e li visualizzeremo con i **diagrammi di flusso**

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



If ... then else

- L'istruzione **if** contiene una condizione booleana che permette di scegliere fra due blocchi di istruzioni racchiusi fra parentesi quadre

```
if (boolean-cond) {  
    blocco 1  
}  
else {  
    blocco 2  
}
```

```
if (vendite > media) {  
    bonus = 0.1;  
    guadagno = guadagno + quota * bonus;  
}  
else {  
    guadagno = 0;  
}
```

Espressioni booleane

- Le espressioni booleane si costruiscono a partire da operatori di confronto

- uguale == , diverso !=
- maggiore > , maggiore e uguale >=
- minore < , minore e uguale <=

`(c != 23)` è **true** se c è diverso da 23
`(b == 's')` è **true** se b è il carattere 's'

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Espressioni booleane II

- Le espressioni possono essere combinate utilizzando gli operatori booleani:

- And &&
- Or ||
- Not !

`!(c == 23) || (c == 24)` c non è uguale a 23 nè a 24
`(b == 's') || (b == 'S')` b è il carattere 's' o 'S'
`!((a >= 0) && (a < 0.5))` Non è vero che a è compreso tra 0 (compreso) e 0.5 (escluso)

Franco Scarselli

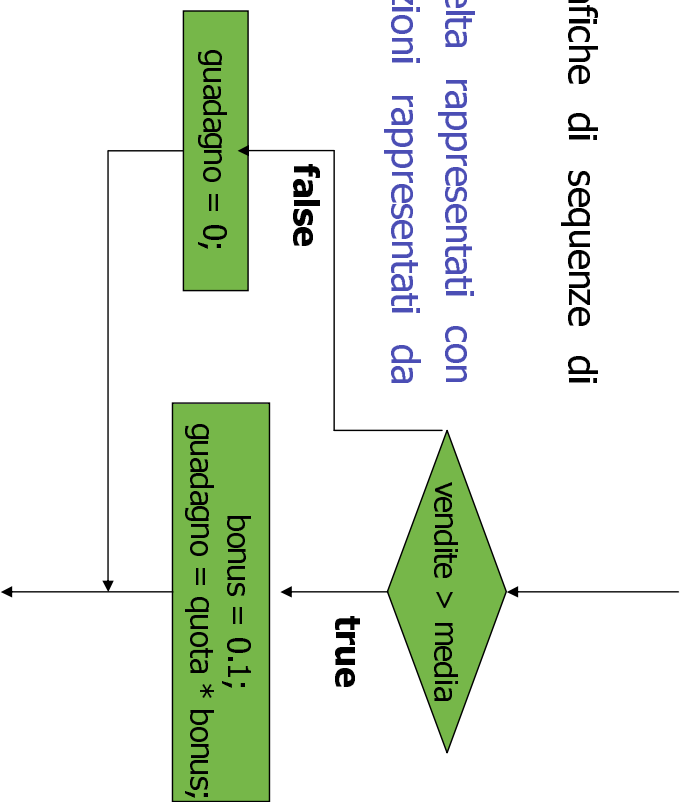
Fondamenti di Informatica I, 2005-2006

Diagrammi di flusso

Diagrammi di flusso

- sono rappresentazioni grafiche di sequenze di istruzioni
- composti da unità di scelta rappresentati con rombi e insiemi di istruzioni rappresentati da rettangoli

```
if (vendite > media) {  
    bonus = 0.1;  
    guadagno = quota * bonus;  
}  
else {  
    guadagno = 0;  
}
```



Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Ciclo while

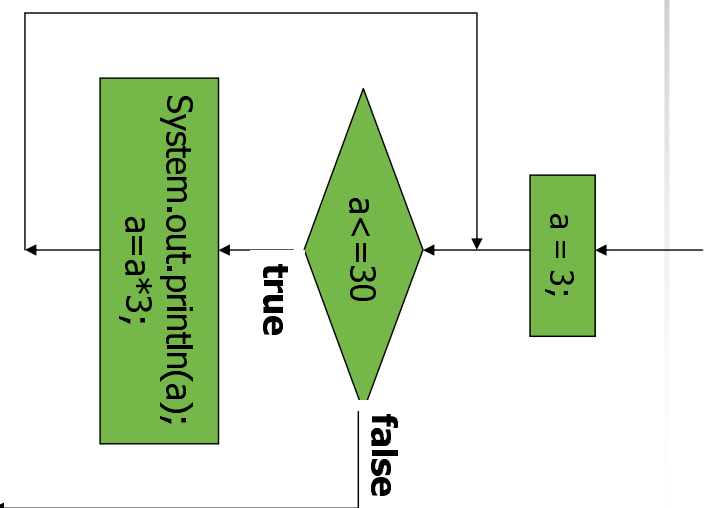
- L'istruzione **while** permette di ripetere una certa operazione

Schema generale

```
while (boolean-cond) {  
    blocco  
}
```

Esempio: stampa le potenze di 3

```
int a=3;  
while (a<=30) {  
    System.out.println(a);  
    a=a*3;  
}
```



Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Ciclo do while

Schema generale

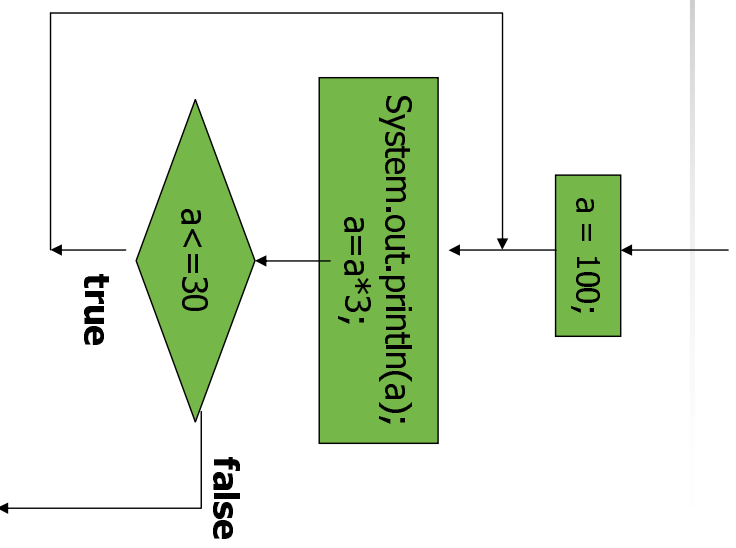
```
do {  
    blocco  
} while (boolean-cond)
```

Esempio: stampa solo 100

```
int a=100;  
do {  
    System.out.println(a);  
    a=a*3;  
} while (a<=30)
```

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



Ciclo for

- L'istruzione **for** permette di realizzare naturalmente cicli determinati, dove il numero di ripetizioni è predefinito

Schema generale

```
for (inizializzazione; boolean-cond; incremento) {  
    blocco  
}
```

Equivale a
 $i=i+1$

Esempio: loop infinito

```
for (;;) {  
    System.out.println("Infinito");  
}
```

Esempio: loop ripetuto 5 volte

```
for (int i = 0; i < 5; i++) {  
    System.out.println(i);  
}
```

Ciclo for: esempio

Riempi un array con i quadrati

```
int a[]=new int[10];  
for (int i=0; i<a.length; i++) {  
    a[i]=i*i;  
}
```

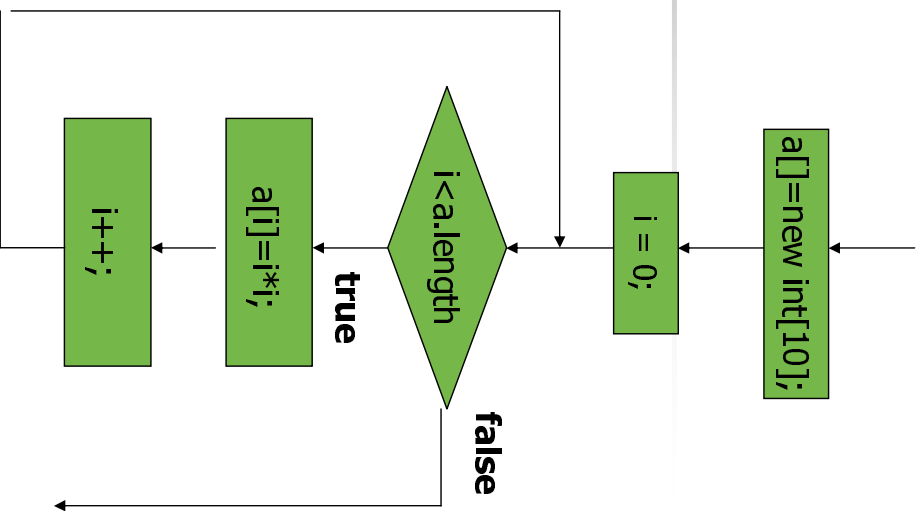
Stampa un array in ordine inverso

```
for (int i=a.length-1; i>=0; i--) {  
    System.out.println(a[i]);  
}
```

Equivale a $i=i-1$

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



Ciclo for: esempio II

- Inizializzazione e incremento possono contenere più operazioni separate da virgole

Riempi un array con le potenze del 2

```
int a[]=new int[10];  
for (int i=0, int j=1; i<a.length; i++, j=2*j) {  
    a[i]=j;  
}
```

inizializzazione

condizione
booleana

incremento

Franco Scarselli

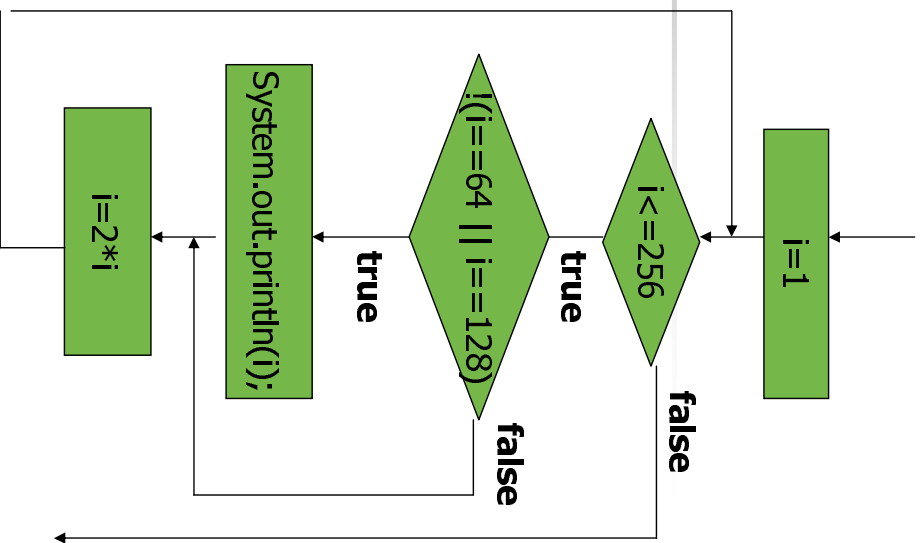
Fondamenti di Informatica I, 2005-2006

Annidamento

- Le istruzioni per il controllo di flusso possono essere annidate (*nesting*)

Stampa le potenze del 2 fino a 256 eccetto 64 e 128

```
for (int i=1; i<=256; i=2*i) {  
    if (!(i==64 || i==128))  
        System.out.println(i);  
}
```



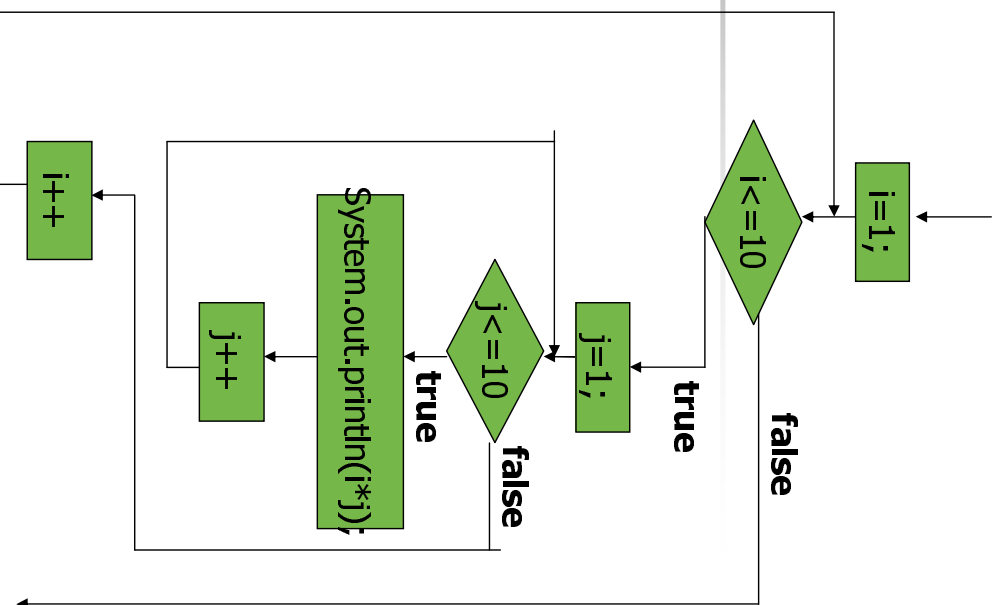
Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Annidamento II

Stampa la tabellina di tutti i numeri Da 1 a 10

```
for (int i=1; i<=10; i++) {  
    for (int j=1; j<=10; j++)  
        System.out.println(i*j);  
}
```



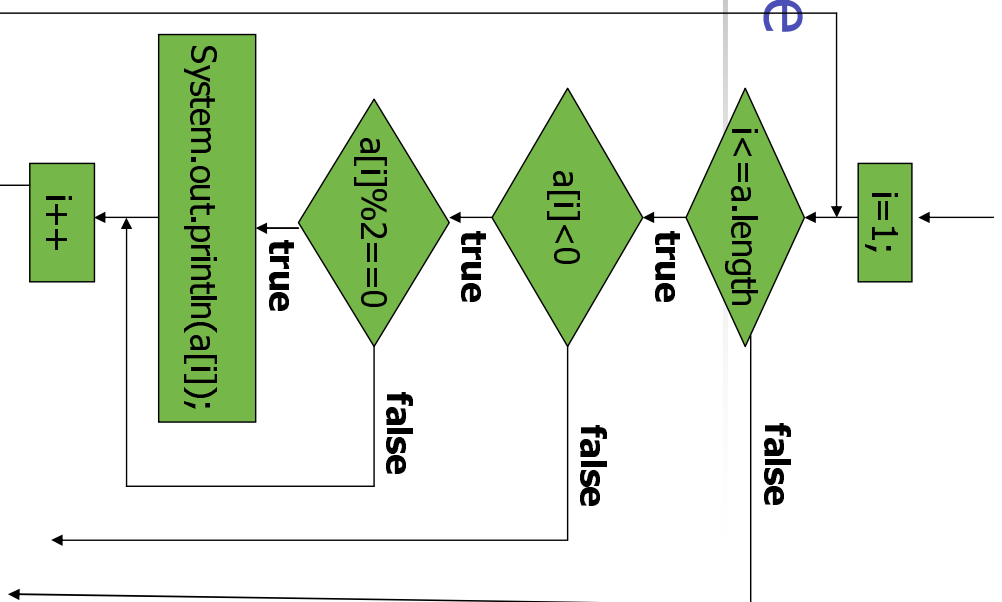
Franco Scarselli

Break and continue

- **break** interrompe l'attuale istruzione di iterazione ed esce definitivamente dalla iterazione
- **continue** interrompe la corrente iterazione e ritorna all'inizio del ciclo, iniziandone un'altra

Legge un vettore e...

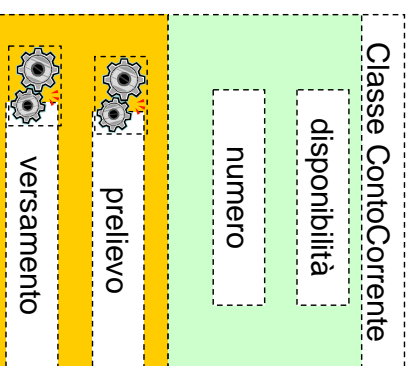
```
for (int i = 0; i < a.length; i++) {  
    if (a[i] < 0)    break;  
    if (a[i] % 2 == 0) continue;  
    System.out.println(a[i]);  
}
```



Implementazione dei metodi: il conto corrente

Il conto corrente

- Si vuole implementare una classe che funzioni come un conto corrente. Il conto corrente ha un numero. Dal conto corrente è possibile versare o prelevare una cifra.
- Per semplicità non importa ricordarsi delle operazioni fatte, ma solo della disponibilità attuale
- Il prelievo deve essere possibile solo se c'è sufficiente disponibilità
- L'oggetto deve rispondere indicando se l'operazione ha avuto successo oppure no



Implementazione dei metodi: il conto corrente

```
Class ContoCorrente{
String numero;
float disponibilit =0;

ContoCorrente(int n){numero=n;}

String prelievo(float quantita){
if (disponibilita- quantita>=0) {
disponibilita= disponibilita- quantita;
return "ok";}
else return "unsuccessful";
}

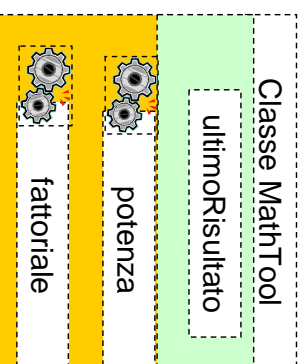
String deposito(float quantita) {
disponibilita= disponibilita+quantita;
return "ok"; }
}
```

Return  
l'istruzione che
definisce il valore
restituito dal
metodo

Implementazione dei metodi: Le potenze e il fattoriale

Strumento per calcolare le potenze e fattoriale

- Si vuole implementare un modulo in grado di calcolare potenze e fattoriale.
- Lo strumento dovr  anche essere in grado di ricordare il risultato dell'ultima operazione



Implementazione dei metodi: Le potenze e il fattoriale

```
Class MathTool{
    int ultimoRisultato=0;

    int power(int numero, int esponente){
        risultato=1;
        for(int i=1;i<=esponente;i++){
            risultato=risultato*numero;
        }
        ultimoRisultato=risultato;
        return risultato;
    }
}
```

```
int fattoriale(int numero){
    risultato=1;
    for(int i=numero;i<=2;i--){
        risultato=risultato*i;
    }
    ultimoRisultato=risultato;
    return risultato;
}
}
```

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Visibilita' delle variabili

- Un **blocco** è una parte di programma delimitata da parentesi graffe
- Una variabile è **visibile** (accessibile) solo all'interno del blocco in cui è definita e dopo la sua definizione
- **ultimoRisultato** è visibile in tutta la classe MathTool
- **numero** è visibile solo all'interno del metodo power
- **i** è visibile solo all'interno del ciclo for

```
Class MathTool{
    int ultimoRisultato=0;

    int power(int numero, int esponente){
        risultato=1;
        for(int i=1;i<=esponente;i++){
            risultato=risultato*numero;
        }
        ultimoRisultato=risultato;
    }
}
```

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

Visibilit  delle variabili

```
Class MathTool{
    int a=ultimoRisultato; .....
    int ultimoRisultato=0;
    int b=ultimoRisultato; .....
    int power(int numero, int esponente){
        int risultato=1;
        for(int i=1;i<=esponente;i++){
            risultato=risultato*numero;
        }
        int d=i; .....
        ultimoRisultato=risultato;
    }
    int c=risultato; .....
}
```

errore
ok
errore
errore

Informatica I, 2005-2006

Visibilit  dall'esterno della classe

- JAVA fornisce delle parole chiavi che permettono di specificare se una variabile o un metodo puo' essere acceduta dall'esterno di un oggetto: **public**, **private**
- È buona norma rendere accessibile dall'esterno solo lo stretto necessario!!!

```
Class ContoCorrente{
    private float disponibilit =0;
    public String prelievo(float quantita){
        if (disponibilita- quantita>=0) {
            disponibilita= disponibilita- quantita;
            return "ok";}
        else return "unsuccessful";
        ....
    }
}
```

```
ContoCorrente mioCC=new ContoCorrente("232");
mioCC.disponibilita= mioCC.disponibilita-100;
```

Il compilatore genera un errore

Ancora sulla visibilità

- Sebbene nello stesso blocco la stessa variabile non possa essere definita due volte, in blocchi diversi la stessa variabile può essere ridefinita
- Se i blocchi sono uno dentro l'altro la variabile interna nasconde quella esterna

```
Class paperino{  
    private String discorso="sono  
    sfortunato...";  
  
    public void parla(){  
        System.out.println(discorso);  
    }  
  
    public void imitaPluto(){  
        String discorso="Bau Bau..."  
        System.out.println(discorso);  
    }  
}
```

Stampa "sono sfortunato"

Stampa "Bau Bau"

F. I., 2005-2006

Ancora sulla visibilità II

- Per accedere dall'interno di un oggetto ad una variabile nascosta si può usare **this** che rappresenta lo stesso oggetto in cui ci si trova

```
Class paperino{  
    private discorso="... odio fare il cane";  
  
    public void imitaPluto(){  
        String discorso="Bau Bau..."  
        System.out.println(discorso);  
        System.out.println(this.discorso);  
    }  
}
```

Stampa "Bau Bau..."

Stampa "...odio fare il cane"

Un metodo speciale: il main

- Gli oggetti sono strumenti passivi, si attivano solo quando vengono chiamati i loro metodi: come fa allora un programma ad essere eseguito?
- Deve esistere una classe con un metodo chiamato **main** ... è possibile chiamare quel metodo dal sistema operativo dando il via all'esecuzione

```
Class inizio{  
    ....  
    public void main(String arg[]){  
        ....  
    }  
}
```

tutto parte di qui'

05-2006

Costruttori e metodi con lo stesso nome (Overloading)

- Una classe può avere più costruttori o metodi con lo stesso nome purché abbiano parametri di ingresso diversi
- Il compilatore determina automaticamente quale metodo/costruttore chiamare

Studente pluto=new Studente("pluto", "cane da guardia")

Studente pippo=new Studente("pippo");

```
class Studente {  
    String nome;  
    boolean lavoratore;  
    String lavoro;  
  
    Studente(String n, String l){  
        nome=n;  
        lavoro=l;  
        lavoratore=true;  
    }  
  
    Studente(String n){  
        nome=n;  
        lavoratore=false;  
    }  
}
```

Le classi e l'organizzazione in file e directory

- Un programma Java è composto da un insieme di file:
 - ogni file può contenere la definizione di una sola classe (...più le classi interne) ed ha il nome di tale classe
- I file (compilati) sono organizzati in directory
- Esiste una corrispondenza diretta fra il modo con cui si accede ad una classe e il file in cui si trova
- Per accedere ad una classe occorre prendere l'indirizzo del file e sostituire al simbolo di directory il punto

università/didattica/anagrafica/studentelavoratore.class

Posto dove si potrebbe trovare la classe studentelavoratore

Creare un oggetto
studentelavoratore
Franco Scarselli

new università.didattica.anagrafica.studentilavoratore()

I package

- Le classi che appartengono ad una directory e a tutte le sue sottodirectory formano un pacchetto (**package**).
- Tipicamente un pacchetto è costituito da una serie di classi correlate
- I package servono da una parte a organizzare le classi, dall'altra a definire gruppi di classi che possono essere riusate
- La libreria JDK è costituita da centinaia di classi organizzate in dozzine di pacchetti:
 - `java.lang` , `java.awt`, `java.io` , ecc...



Import

- Per usare una classe si può usare il path completo oppure si può importare la classe: nel secondo caso, si potrà usare il nome direttamente

```
import universita.didattica.anagrafica.studentelavoratore
```

```
new studentelavoratore()
```

- Inoltre, è anche possibile importare tutte le classi di un package

```
import universita.didattica.*
```

Importa tutte le classi del
package didattica

```
new anagrafica.studentelavoratore()
```

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



Package

- Infine, l'istruzione package permette di definire il package a cui appartiene una classe

```
Package universita.didattica
```

```
....
```



I package e l'organizzazione

- I package hanno una corrispondenza “fisica” sul progetto che si sta sviluppando:
 - Overo se una classe fa parte di un package, allora **deve** risiedere in una cartella il cui percorso di directory coincide con la gerarchia del package

package prova.bankaccount;

- Es



La classe deve essere salvata in
`[CLASSPATH]prova/bankaccount/BankAccount.class`

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



Definire il package

- Per inserire le classi in un pacchetto si usa la seguente sintassi all'inizio del file:
package NOME_PACCHETTO;

- Il nome del pacchetto può essere costituito da una serie di identificatori, separati da punti:
 - L'idea è quella di organizzare i package in maniera gerarchica e quindi dare la possibilità di creare un albero di pacchetti

- Ogni punto individua un “ramo” dell'albero

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



Gestione delle eccezioni

- Le eccezioni sono eventi eccezionali che interrompono il normale flusso del programma
- Le istruzioni per la gestione delle eccezioni permettono al programmatore di gestire tali eventi:
 - Il programmatore può specificare cosa fare quando accade un'eccezione
- La gestione delle eccezioni permette di scrivere programmi più facili da leggere e più robusti

```
Studente pluto=new Studente("pluto", "cane da guardia");  
Studente pippo=new Studente("pippo");
```

Franco Scarselli

Fondamenti di Informatica I, 2005-2006



Esercizi

La somma e la media di un array

- Si deve realizzare il codice che calcola la somma e la media dei valori positivi contenuti in un array.
- Dalla somma e dalla media devono essere esclusi i numeri negativi
- Disegnarne il diagramma di flusso

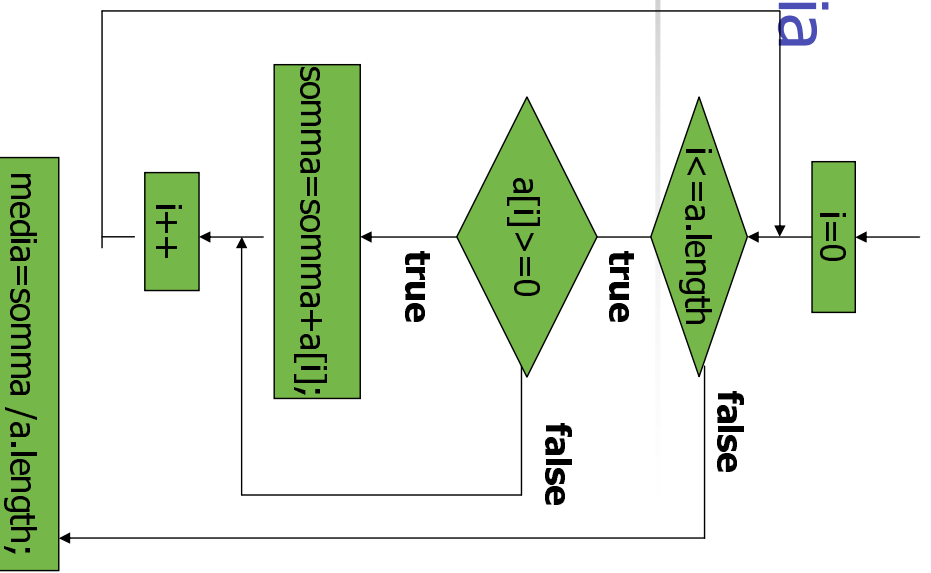
```
Int a=new int[100];
```

Franco Scarselli

Fondamenti di Informatica I, 2005-2006

La somma e la media di un array

```
float somma=0;  
float media;  
for (int i=0; i<=a.length; i++) {  
    if (a[i]>=0) {  
        somma=somma+a[i];  
    }  
}  
media=somma /a.length;
```



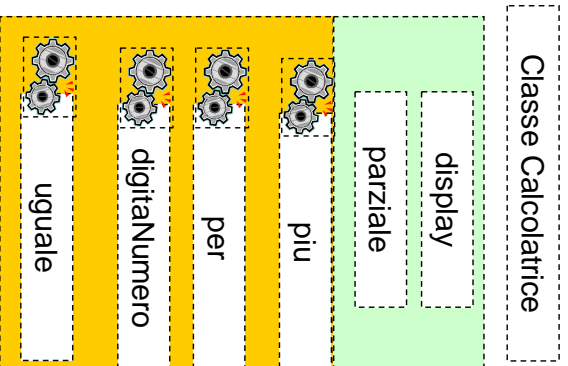
Franco Scarselli

Fondamenti di Informatica I, 2005-2006

La calcolatrice

La calcolatrice

- Si vuole implementare una calcolatrice che realizzi la somma e la moltiplicazione
- La calcolatrice ha un display che mostra il risultato
- La calcolatrice fornisce il modo di visualizzare il risultato di un'operazione usando un metodo che corrisponde a premere il simbolo "="
- La calcolatrice si mantiene internamente il parziale delle operazioni fatte



Franco Scarselli

Fondamenti di Informatica I, 2005-2006

La calcolatrice II

```
Class Calcolatrice{
    public float display=0;
    private float parziale=0;
    private char opCorrente='';

    void per(){
        opCorrente ='x';
    }

    void piu(){
        opCorrente ='+';
    }

    void digitaNumero(float numero){
        display=numero;
        aggiornaParziale();
    }
}
```

```
void aggiornaParziale(){
    if (opCorrente==''){
        parziale=display;
    }
    else {
        if (opCorrente=='x'){
            parziale=parziale*numero;
            opCorrente='';
        }
        else {
            parziale=parziale+numero;
            opCorrente='';
        }
    }
}

void aggiornaParziale(){
    display=parziale;
}
}
```

damer



La calcolatrice II

```
Class Calcolatrice{  
    float display=0;  
    char opCorrente='';  
  
    void uguale(){  
        if (opCorrente=='')  
            { display=numero;}  
        else {  
            if (opCorrente=='x'){  
                display=display*numero;  
                opCorrente='';  
            }  
            else {display=numero;  
                opCorrente='';  
            }  
        }  
    }  
}
```

Franco