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

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)

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 suddivida 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

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

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

Esempio: ricerca di CD per autore e titolo

- Problema: Dato un insieme di CD musicali si vuole cercare un CD. Si suppone che i CD siano ordinati su uno scaffale.
- Algoritmo 1:
  1) Si scorrono tutti i CD partendo dal primo fino a che non troviamo il CD desiderato
- Algoritmo 2:
  1) Si guarda il CD al centro dello scaffale e si confronta con quello che si vuole cercare
  2) Il confronto ci permette di capire se si deve cercare nella parte destra dello scaffale o in quella sinistra. Quindi ritorna a 1, concentrandoci solo sulla parte pertinente dello scaffale.

Osservazioni

- Ovviamente 2 è più veloce di 1

Proprietà degli algoritmi

- Affinché un elenco di istruzioni, possa essere considerato un algoritmo, devono essere soddisfatti i seguenti requisiti:
  - Finitezza:
    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

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?

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

Indica una possibile scelta

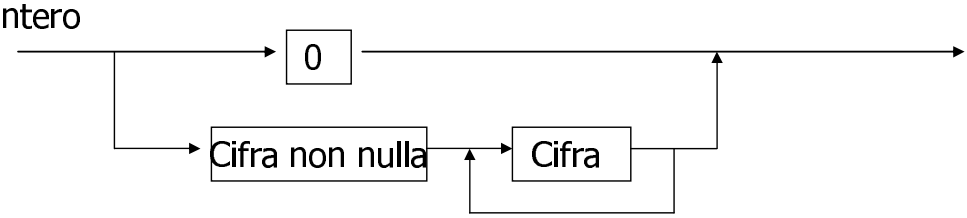
Indica la ripetizione

<intero> := 0 | <cifra-non-nulla>{<cifra>}

<cifra-non-nulla>:= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<cifra>:= 0 |1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Rappresentazione grafica di una regola



Linguaggi di basso e alto livello

Linguaggi di basso livello

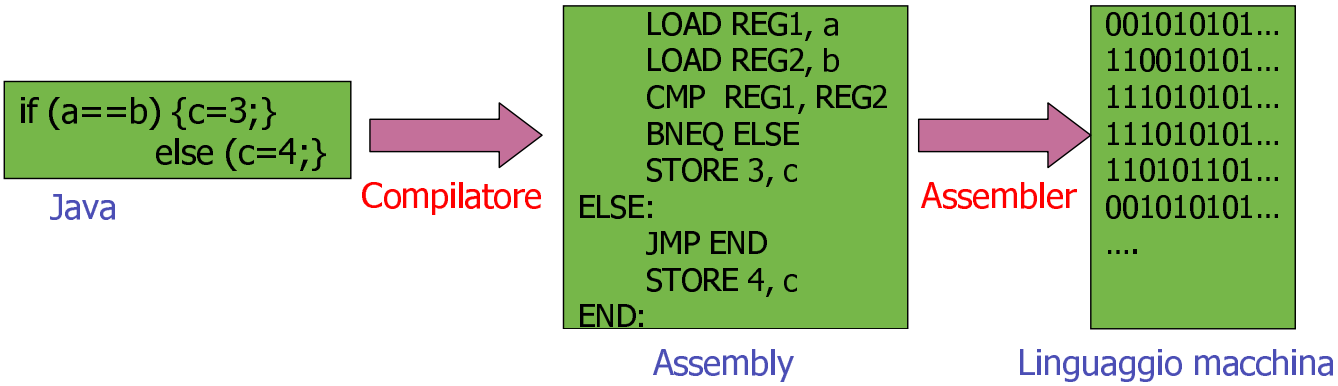
- Il linguaggio macchina è il linguaggio dalla CPU del calcolatore
- I linguaggi a basso livello sono simili al linguaggio macchina, molto diversi dal linguaggio umano
- Variano a seconda del computer su cui si scrive il programma
- Programmare con linguaggi a basso livello richiede molto tempo

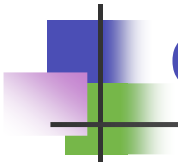
Linguaggi di alto livello

- 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, ...

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

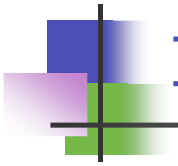




Compilatori e ambienti di sviluppo

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 editori 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 funzionalita' necessarie e dei vincoli richiesti.

- Scrittura del software (30% del tempo)

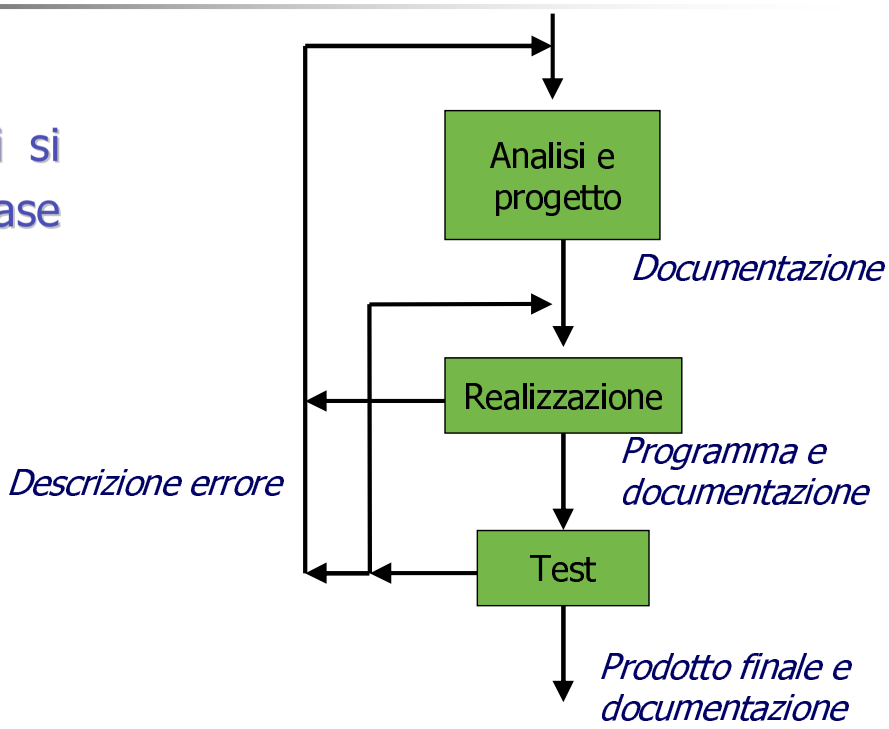
  Si scrive il programma

- Test (per applicazioni medie, 40% del tempo)

  Si prova il programma, prima la verifica viene fatta dai programmatori, poi sul campo…

Schema del processo di sviluppo del software

In caso di errori si ritorna ad una fase precedente



Il linguaggio Java

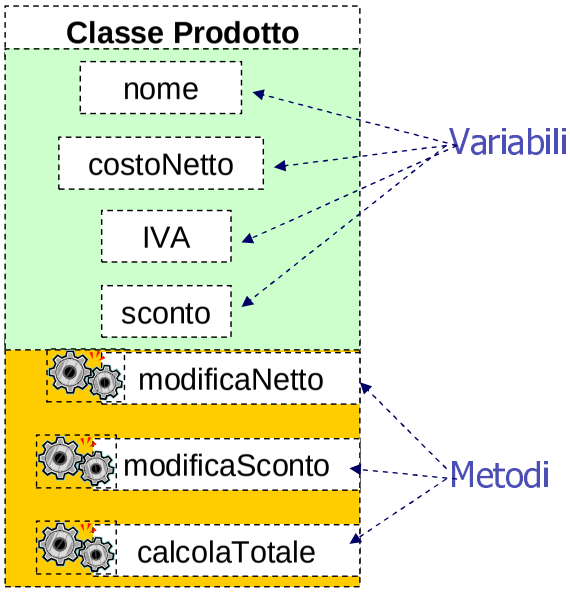
Componenti elementari di Java: classi, metodi, variabili

- Un programma Java
  - consiste di insieme di componenti semplici chiamate classi
- Una classe è definita da
  - un insieme di dati detti variabili e
  - un insieme di funzioni detti metodi, con cui si puo' operare sui quei dati

Esempio: il prodotto di un supermercato

Un prodotto

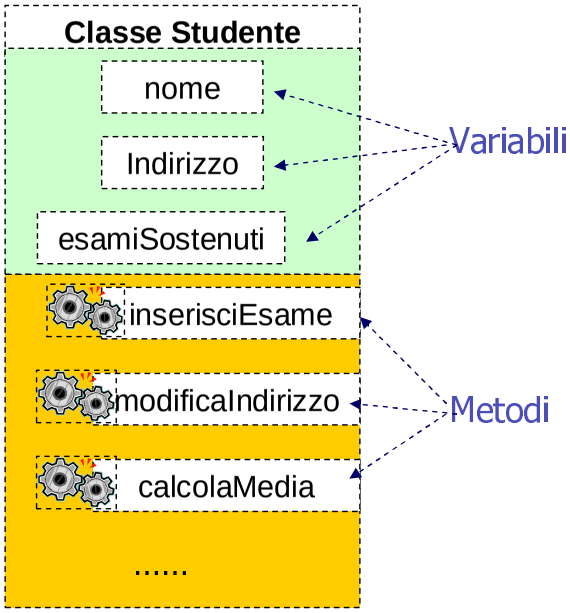
- un prodotto è caratterizzato da un nome, un costo netto, la percentuale IVA e un eventuale sconto.
- Ci è stato richiesto di sviluppare un modulo software che memorizza un prodotto, permette di modificare il costo netto, lo sconto e di calcolare il costo totale.



Esempio: uno studente

Uno studente secondo la segreteria studenti

- Uno studente ha un nome, un indirizzo e un curriculum.
- Si deve poter inserire nuovi esami, calcolarne la media e modificare l'indirizzo, ...



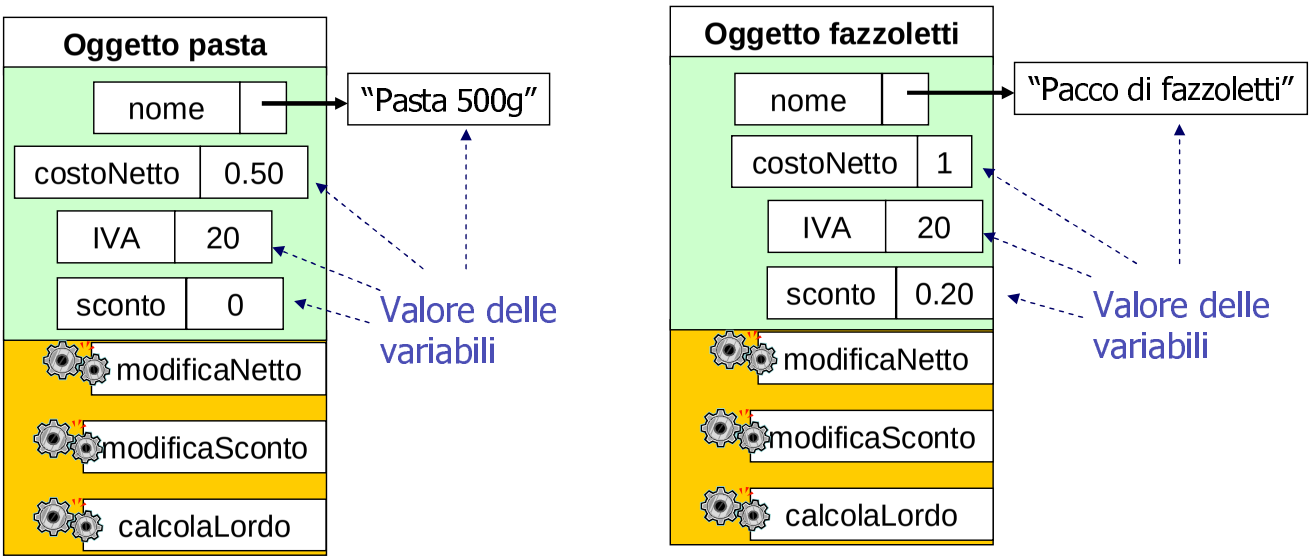
Componenti elementari di Java: oggetti

Classi e oggetti

- una classe è una definizione astratta di un concetto
- un oggetto è una particolare istanza concreta di quella classe
- negli oggetti, le variabili assumo un valore preciso
- ad ogni classe possono corrispondere più oggetti

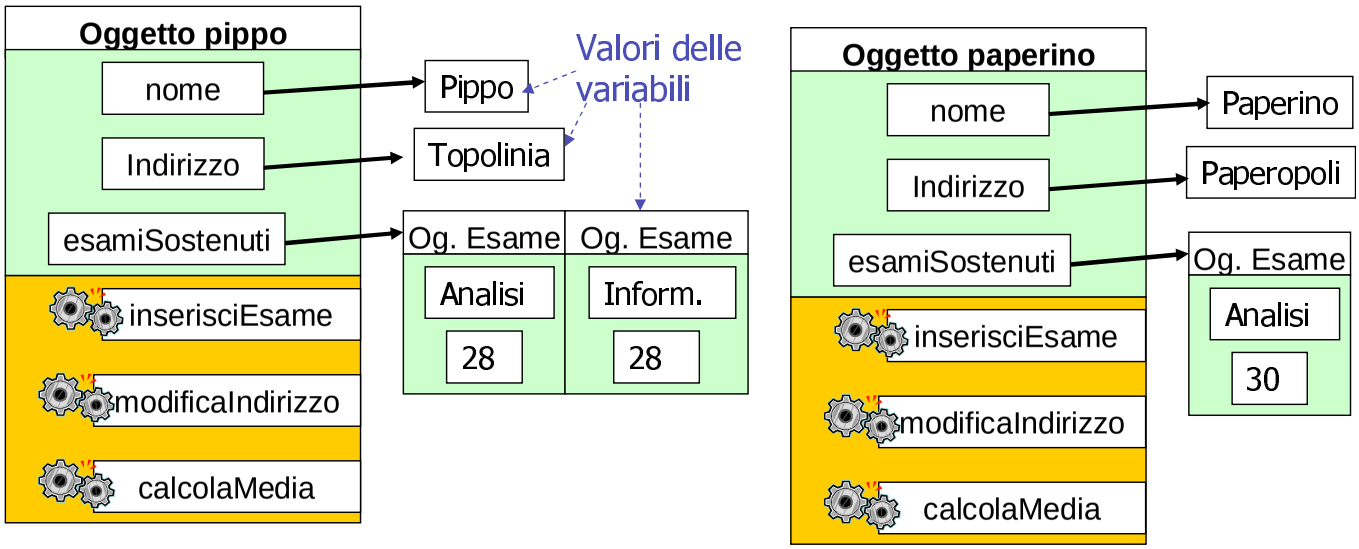
Esempio: il prodotto di un supermercato II

- pasta e fazzoletti sono istanze concreta della classe Prodotto



Esempio: uno studente II

- Le variabili dello studente assumo un valore negli oggetti concreti pippo e paperino



Componenti elementari di Java: tipi di dato

Tipi di dato

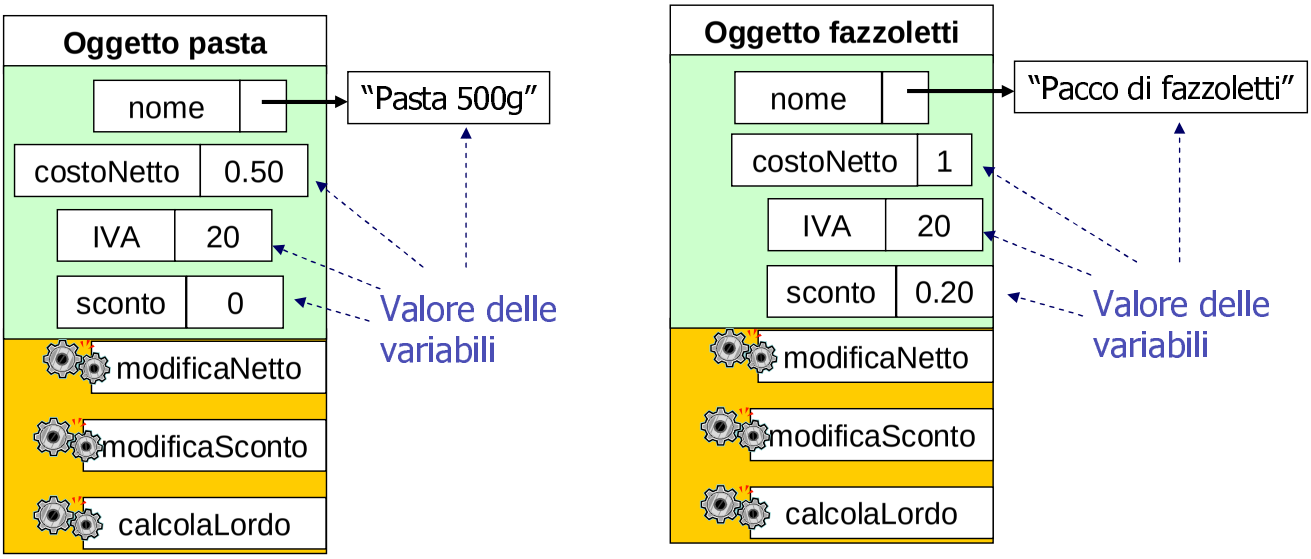
- Ad ogni variabile è associato un tipo di dato che specifica i valori che la variabile può assumere
- I tipi si dividono in
  - tipi elementari

    sono dati semplici: numero intero, numero reale, carattere, booleano,..
  - oggetti detti anche tipi strutturati

    contengono dati complessi, composti da tipi elementari

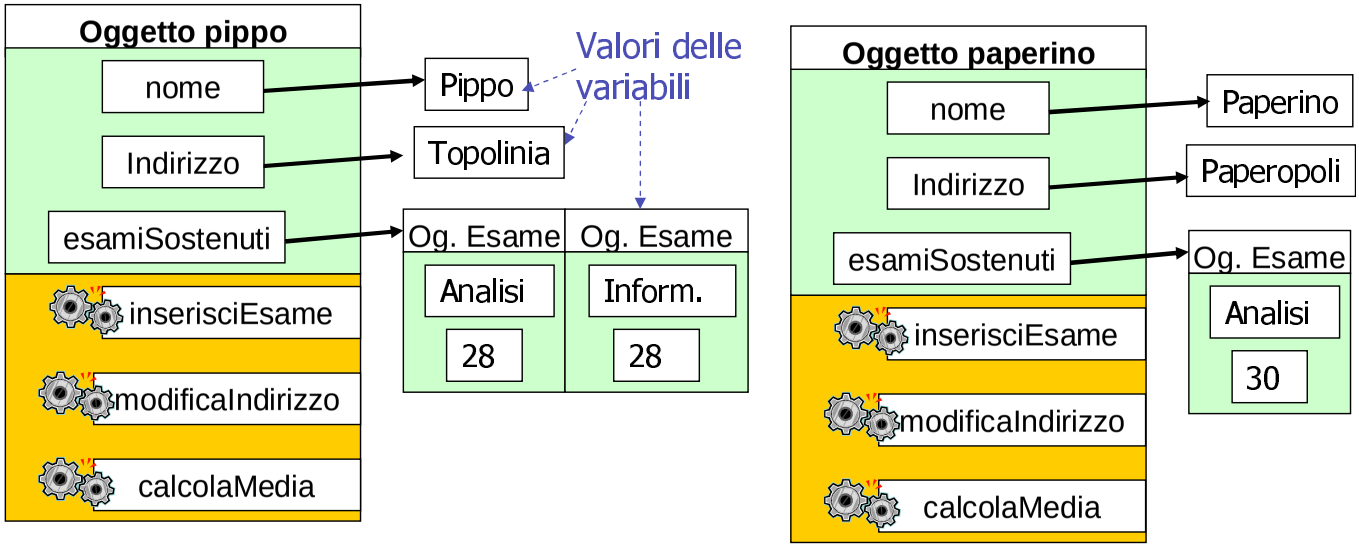
Esempio: il prodotto di un supermercato III

- Le variabili costoNetto, IVA, sconto contengono numeri reali: tipi semplici
- La variabili nome contiene una sequenza di caratteri: tipo strutturato



Esempio: uno studente II

- Le tre variabili contengono tipi strutturati
- esamiSostenuti contiene un insieme di esami: ogni esame è a sua volta un oggetto



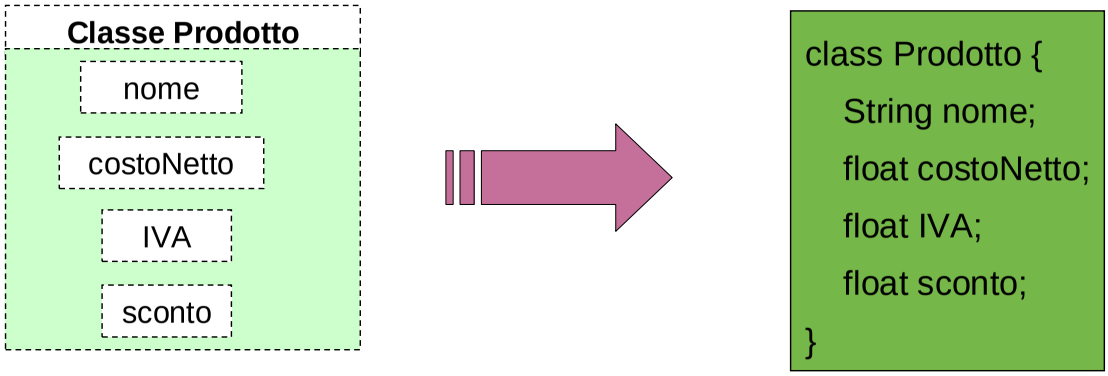
Componenti elementari di Java: tipi di dato II

Tipi di dati predefinti i Java (alcuni esempi, tanto per iniziare)

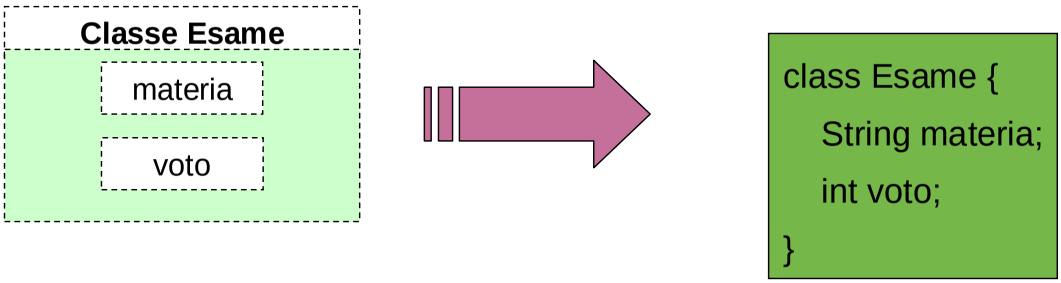
- Tipi semplici
  - gli interi: si possono definire usando la parola chiave int
  - i reali: si possono definire usando la parola chiave float
- Tipi strutturati
  - le stringhe (sequenze di caratteri): si possono definire usando la parola chiave String
  - insiemi: si possono definire usando le parentesi quadre []

Definiamo la nostra prima classe in Java ….

- In Java la definizione di una classe è racchiusa fra parentesi e preceduta dalla parola chiave class



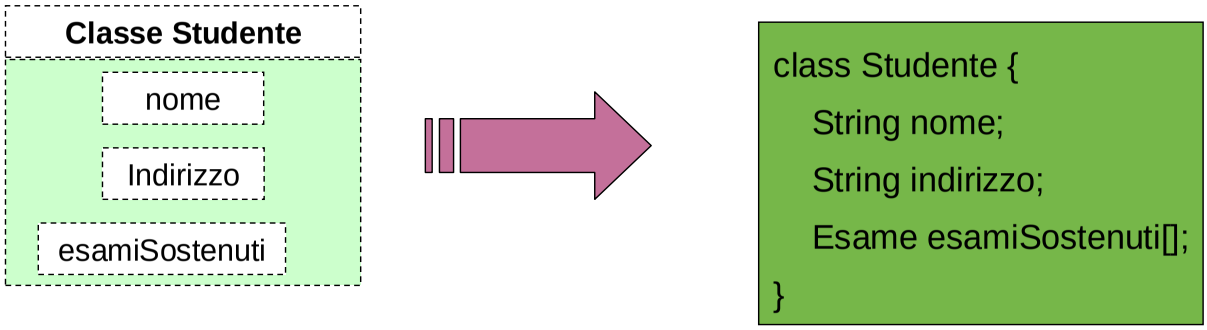
Definiamo la nostra … seconda classe in Java



……

Definiamo la nostra … terza classe in Java

- Uno studente contiene un insieme di esami, per cui la definizione delle classe Studente usa la definizione della classe Esame



.....

Componenti elementari di Java: metodi

I metodi

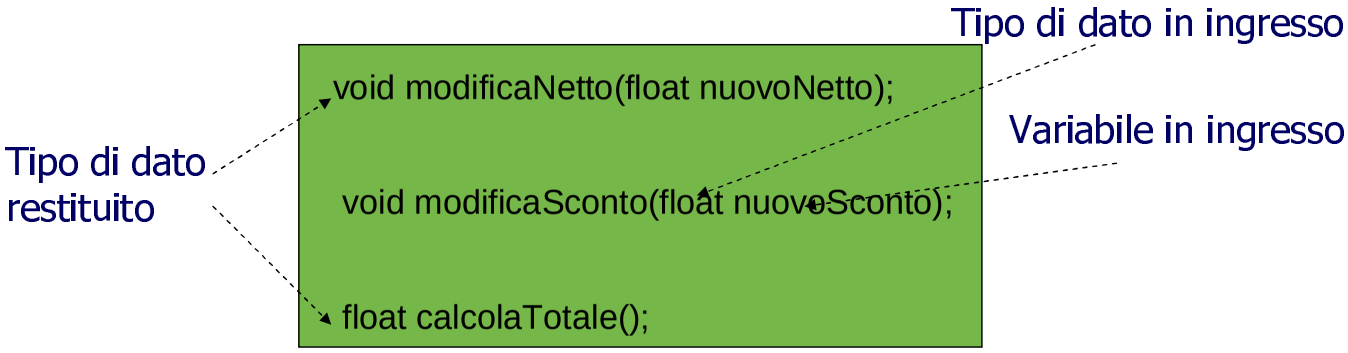
- sono funzioni che operano sui dati della classe
- un metodo riceve dei dati in ingresso e restituisce altri dati
- sono l'interfaccia verso l'esterno della classe
- sono operazioni messe a disposizione "degli altri" per fare modifiche e calcoli sui dati: … per operare sui dati conviene chiedere (si dice chiamare…) ai metodi

Componenti elementari di Java: tipi di dato e metodi

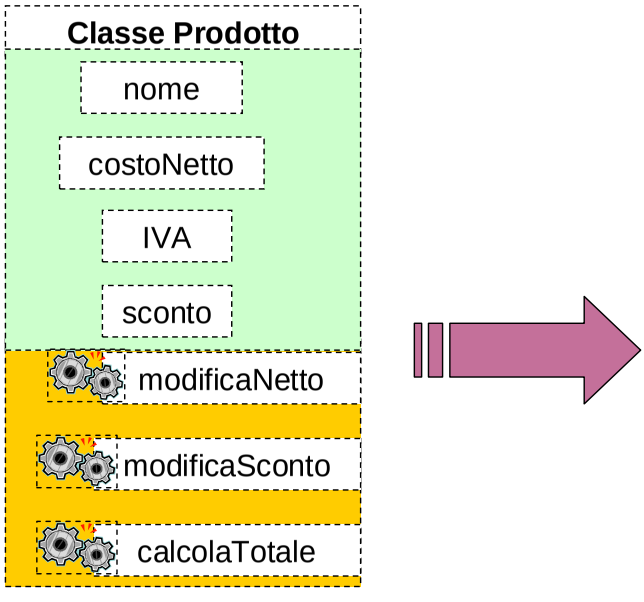
Tipi di dati associati ai metodi definiscono

- cosa il metodo prende in ingresso:
  - ad ogni ingresso (parametro) è associato una variabile e un tipo
- cosa restituisce

Il tipo speciale void specifica che un metodo non restituisce niente

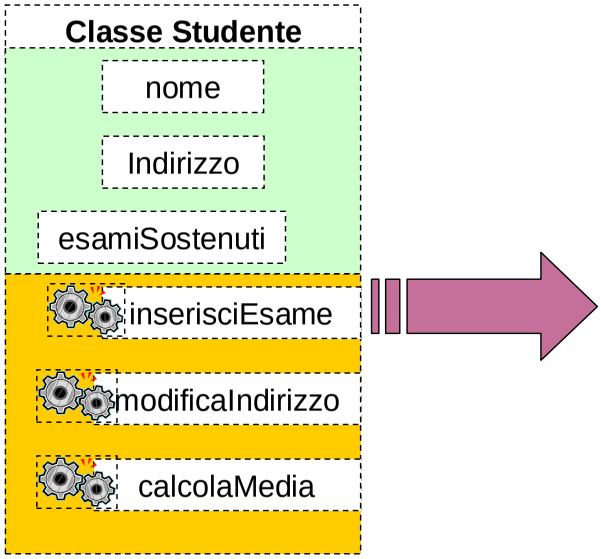


Definizione Java della classe Prodotto




```
class Prodotto {
    String nome;
    float costoNetto;
    float IVA;
    float sconto;

   void modificaNetto(float nuovoNetto);

   void modificaSconto(float nuovoSconto);

   float calcolaTotale(float nuovoSconto);
}
```

Definizione Java della classe Studente




```
class Studente {
    String nome;
    String indirizzo;
    Esame esamiSostenuti[];

    void inserisciEsame(Esame nuovoEsame)
    void modificaIndirizzo(String nuovoIndirizzo)
    float calcolaMedia()
}
```

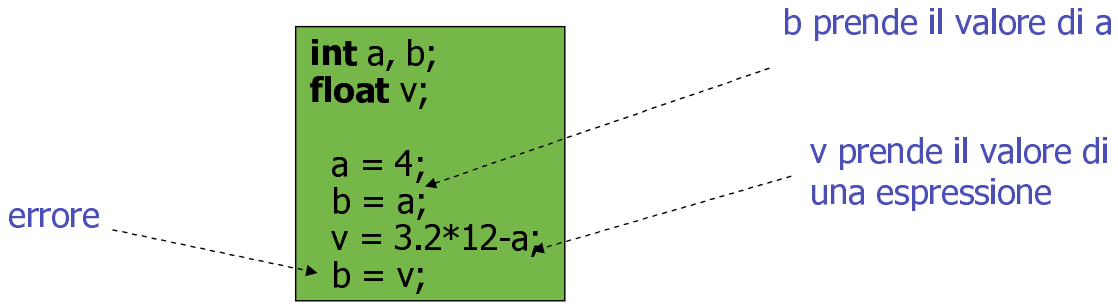
Data manipulation

Manipolazione dei dati

- Abbiamo visto come definire variabili e classi
- Dobbiamo capire come manipolare variabili e classi
- Ogni linguaggio di programmazione contiene istruzioni per inizializzare o modificare il valore delle variabili
  - ad es., fare in modo che il nome del prodotto è "pasta"
- Ogni linguaggio di programmazione contiene un meccanismo per creare un'istanza di una classe, cioè un oggetto
  - ad es., fare in modo da creare l'oggetto Pasta a partire dalla definizione della classe Prodotto

Istruzioni di manipolazione dei dati: cominciamo con … l'assegnamento

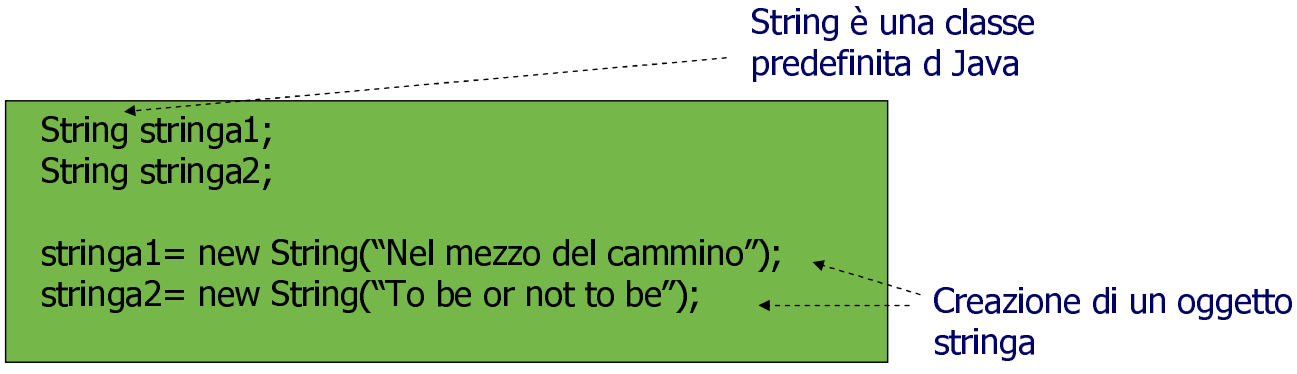
- L'assegnamento è l'istruzione più semplice
  - attribuisce alla variabile a sinistra del simbolo "=" il valore dell'espressione a destra.
  - Il valore assegnato deve essere compatibile col tipo della variabile, altrimenti si genera un errore …. quasi sempre



Istruzione new e oggetti

L'istruzione new

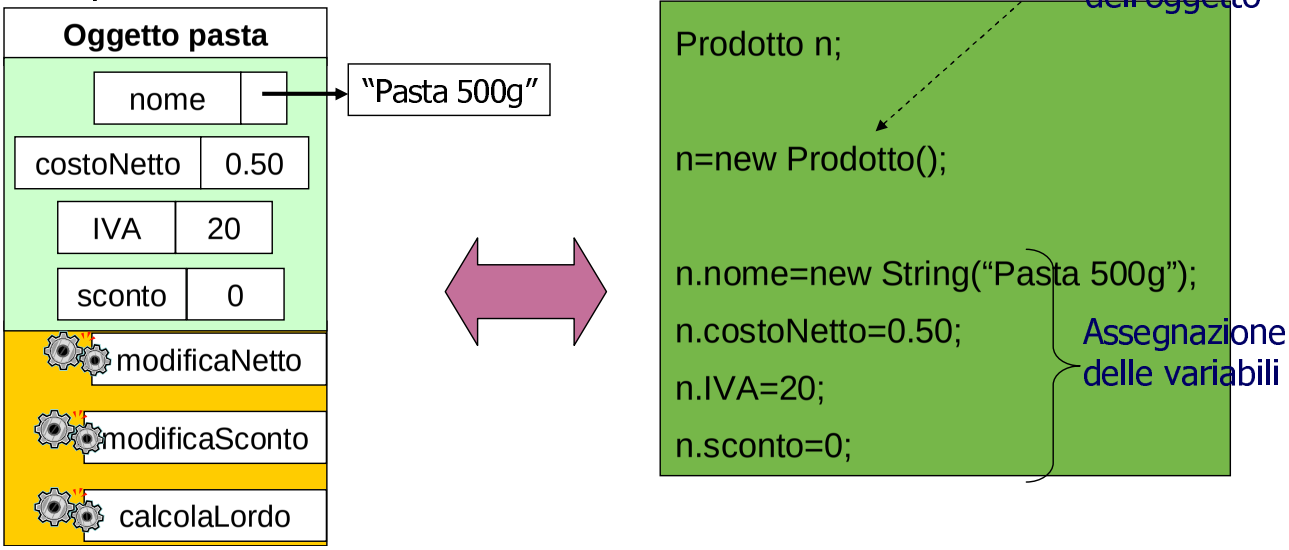
- serve a costruire un oggetto a partire dalla classe
- deve essere seguita dal nome della classe ed eventuali parametri che specificano di che oggetto si tratta



Accesso diretto alle variabili di un oggetto

Una volta creato un oggetto

- si può accedere alle sue variabili, usando il nome dell'oggetto seguito da un punto e il nome della variabile



Osservazione importante

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

Entriamo nei dettagli: nozioni di base

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

- Puo contenere lettere (minuscole o maiuscole), cifre e il carattere '_'
- Non può essere una delle **parole riservate del linguaggio** (if, else, while, return ecc...)
- Non puo' iniziare con una cifra
- E' case sensitive: nome è diverso da Nome

mia eta 12stipendio co*ca repeat	nomi scorretti
miaEta stipendio12 co_ca ripetizione	nomi corretti

I commenti e le istruzioni

Tutte le istruzioni

- sono terminate da un punto e virgola

I commenti

- sono il testo contenuto fra i simboli "/*" e "*/"
- sono il testo da "//" fino alla fine della linea
- commentare un programma è essenziale per documentarlo

```
// Questo è un commento ….
// definizione
 int a, b;
 float v;

/* Un'altro commento ….
assegnazioni */

  a = 4;
  b = a;
  v = 3.2*12-a;
  b = v;
```

Stampare a video

- Esistono molti modi per stampare dei valori a video
- Il modo più semplice è quello di usare i metodi della classe predefinita

  System.out

```
String a;
String b;
String c;

a=new String("Paperon");
b=new String("dei Paperoni");
c=new String("Paperopoli");


System.out.print(a);
System.out.println(b);
System.out.print(c);
```

stampa la stringa aggiungendo un fine linea

Stampa

```
Paperon dei Paperoni
Paperopoli
```

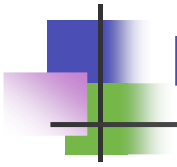
Entriamo nei dettagli: le variabili

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 puo' accettare

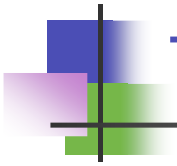
Intero con segno	int eta;
Numero reale	float stipendio;
Stringa, sequenza di caratteri	String cognome;
Data, è classe Java per la rappresentazione delle date	Date dataDiNascita;



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 per memorizzarci i dati della variabile
- Se una variabile viene dichiarata di tipo T, essa potrà memorizzare valori di tipo T
- Il compilatore dara' un messaggio di errore sia in caso di mancata dichiarazione della variabile che in caso gli si assegni un valore di tipo sbagliato



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

Tipi di dato semplici

Tipo	*Descrizione*	*Min*	*Max*
boolean	vero o falso	-	-
char	carattere	0	$2^{16}-1$
byte	intero	-128	+127
short	intero	-2^{15}	$+2^{15}-1$
int	intero	-2^{31}	$+2^{31}-1$
long	intero	-2^{63}	$+2^{63}-1$
float	reale	precisione semplice	
double	reale	doppia precisione	
void	-	-	-

Un carattere ossia l'intero positivo che lo codifica

Tipo speciale per indicare un valore non definito

Assegnamento

- L'assegnamento permette di definire il valore di una variabile:
  - ad una variabile puo' essere assegnato un valore anche contemporaneamente alla sua definizione
  - ad una variabile puo' essere assegnato il valore di una espressione ottenuta con gli operatori "/", "+", "-", "*", "%" e altri ..

```
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

errore

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

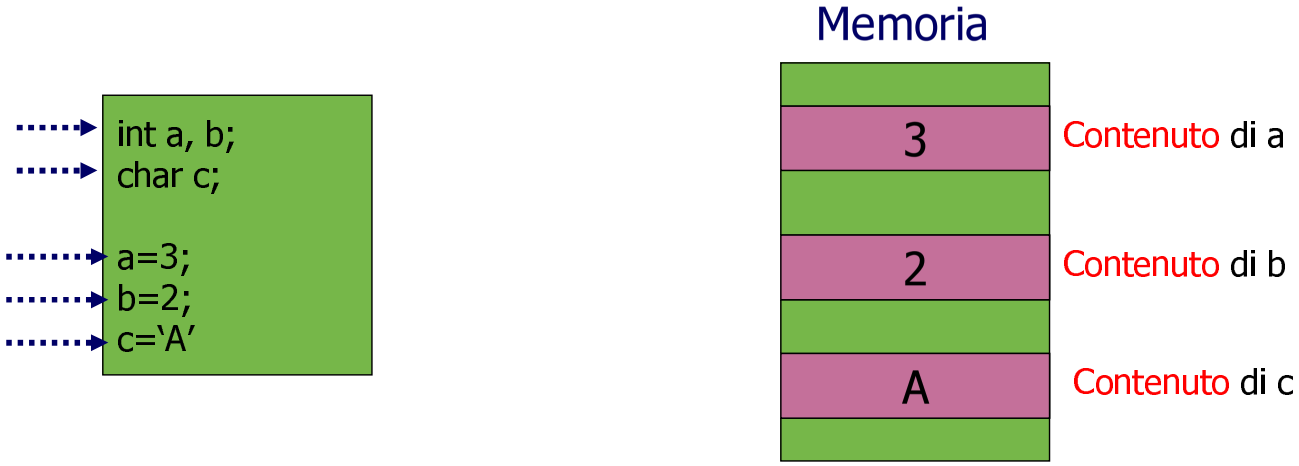
- È corretto o no?

```
short  a=1.2;      ------> errore
int b;
long c=10;
float d=3;         ------> ok
double e;
char f='M';

a=c;               ------> errore
a=(short) c;       ------> ok, ma se c fosse troppo grande...
e=d;               ------> ok
d=c;               ------> ok
d=e;               ------> errore
d=(float) e;       ------> ok, ma ci potrebbe essere perdita di precisione
```

Tipo	***Desc***	***Min***	***Max***
boolean	vero falso	-	-
char	carattere	0	2^{16}-1
byte	intero	-128	+127
short	intero	-2^{15}	$+2^{15}$-1
int	intero	-2^{31}	$+2^{31}$-1
long	intero	-2^{63}	$+2^{63}$-1
float	reale	precisione semplice	
double	reale	doppia precisione	
void	-	-	-

Memoria e variabili di tipi semplici

- Le variabili di tipi semplici corrispondono a locazioni di memoria dove vengono memorizzati i dati
  - La locazione viene predisposta quando la variabile è definita
  - Il contenuto viene modificato quando la variabile è modificata



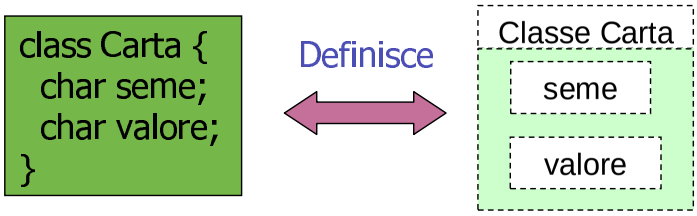
Tipi di dato strutturati e oggetti

- Gli oggetti sono tipi di dato composti da più "campi", da cioè uno o più altri tipi di dato a loro volta semplici o strutturati
- Per definire una classe si usa la parola chiave class
- Per usare un oggetto occorre
  - crearlo con l'operatore new: l'operatore new produce un handle

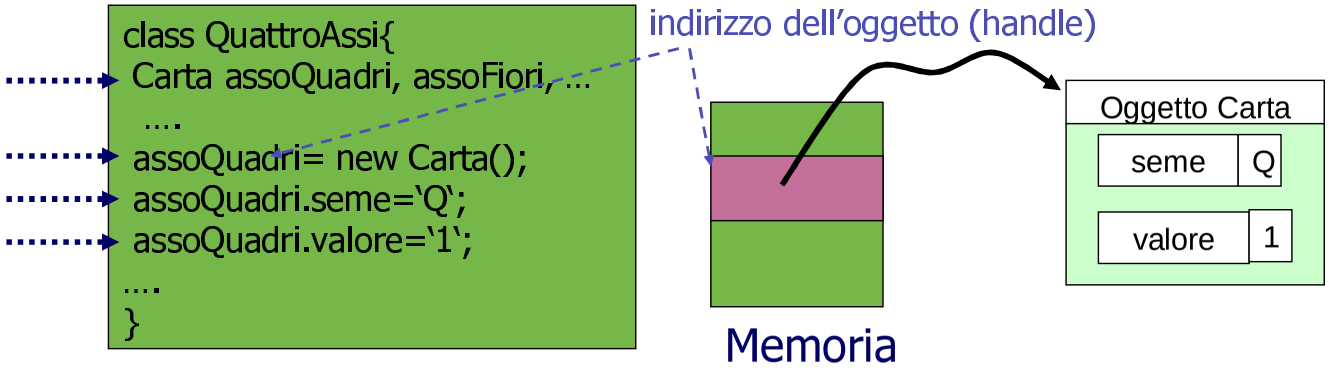
    (l'handle è l'indirizzo (puntatore) dell'area di memoria dove è allocato l'oggetto)
  - Le variabili di tipi oggetti contengono l'handle
  - l'informazione interna all'oggetto puo' essere acceduta postponendo un punto al nome dell'oggetto

Esempio di un oggetto: una carta

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

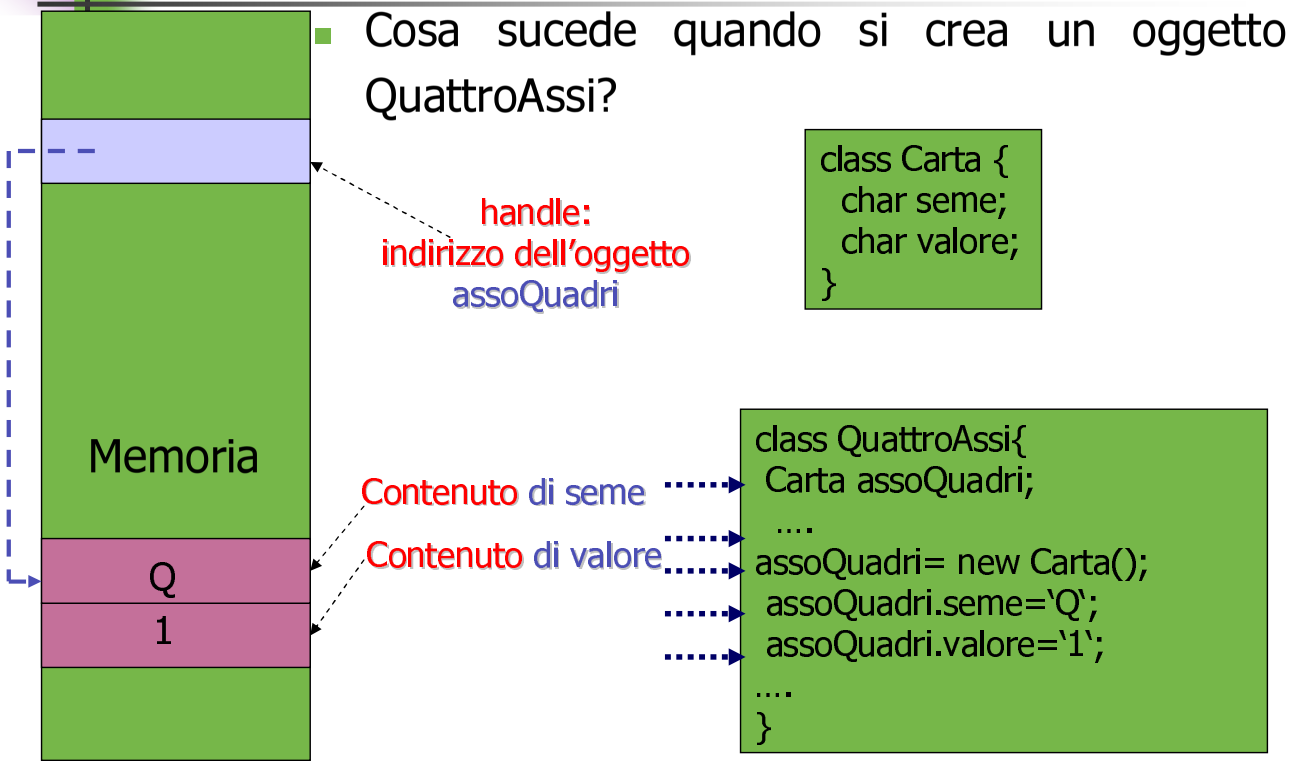


- Nella classe QuattroAssi potrebbe esserci



Esempio di creazione di un oggetto quattroAssi

- Cosa sucede quando si crea 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 puo' essere iterata se un oggetto contiene un oggetto che contiene un oggetto che contiene….

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

In qualche classe potrebbe esserci scritto…

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

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

Definizione, inizializzazione e uso delle variabili

La vita di una variabile è caratterizzata da

- dichiarazione

  istruzione in cui si definisce il tipo
- inizializzazione

  istruzione in cui viene assegnato il primo valore
- uso

  ogni istruzione in cui si accede il contenuto della variabile

In Java non è ammesso

- usare una variabile non inizializzata
- inizializzare o usare una variabile non definita

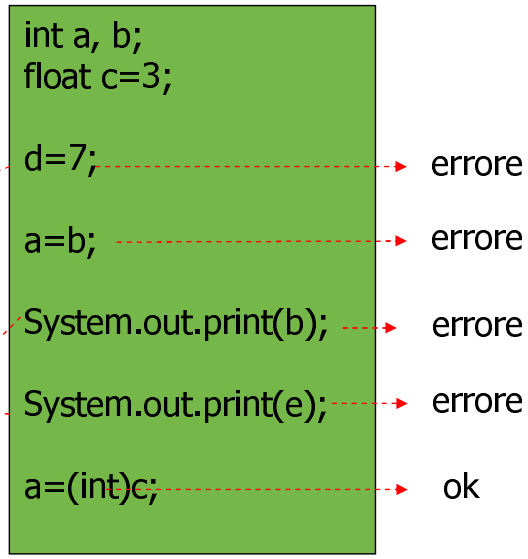
Vita di una variabile di tipo semplice

Errori con una variabile di un tipo semplice

- 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



La classe String e gli array

- Java contiene numerose classi predefinite che permettono di realizzare le componenti software piu` 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, cioe' 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.

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 puo' 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

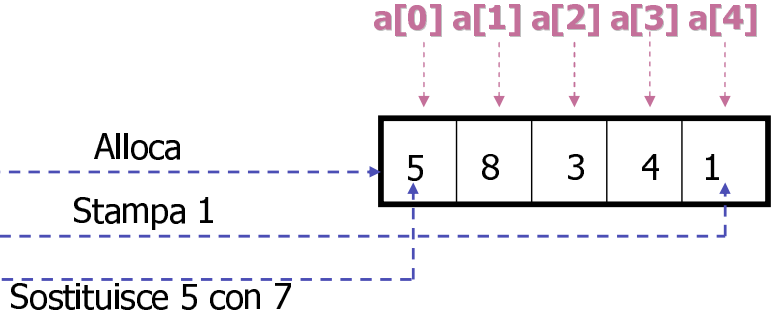
La stringa vuota

La stringa "Paolino Paperino"

Array

- Permettono di memorizzare sequenze di dati elementari o oggetti
- Si dichiarano aggiungendo alla variabile due parentesi quadre
- Un array può essere inizializzato usando 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

```
int[] a;
int a[];

int[] a = {5, 8, 3, 4, 1};
System.out.println(a[4]);
a[0]=7;
```



Array II

Un altro modo di inizializzare i gli array

- consiste nell'usare new seguito dal nome della classe della dimensione dell'array racchiuso fra parentesi quadre
- in tal caso gli elementi dell'array vengono automaticamene inizializzati ad un valore che dipende dal tipo: 0 per i numeri, blank per i caratteri, null per gli oggetti

```
int[] a = {5, 8, 3, 4, 1};
```

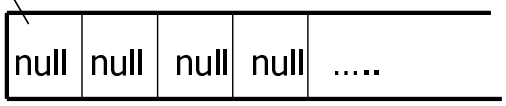
sono equivalenti

```
int[] a = new int[5];
a[0]=5;
a[1]=8;
a[2]=3;
a[3]=4;
a[4]=1;
```

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';
……..
```



Crea un array di handler nulli che vengono assegnati successivamente

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

Primo capitolo, bla, bla

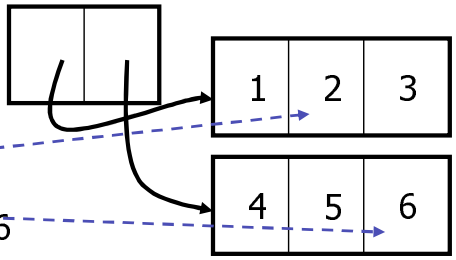
Secondo capitolo, bla, bla

Array III

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

```
int  matrice[][] = {{1,2,3},
                    {4,5,6}};

System.out.println(matrice[0][1]);
System.out.println(matrice[1][2]);
System.out.println(matrice[2][2]);
```

definisce



stampa 2

stampa 6

errore, indici troppo grandi

Questi programmi
creano lo stesso array

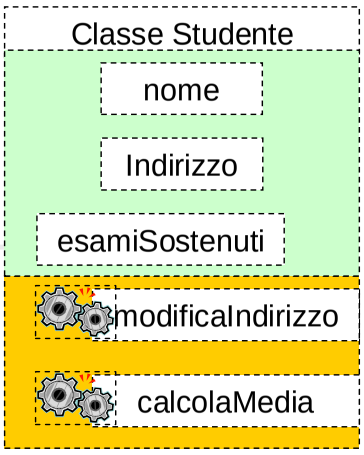
```
int  matrice[][] ;

matrice= new int[2][3];
matrice[0][0]=1;
matrice[0][1]=2;
matrice[0][2]=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 affinche' 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

I metodi: esempio




```
class Studente{
  String nome, indirizzo;
  Esame esamiSostenuti[];

  void modificaIndirizzo(String nuovoIndirizzo){
  indirizzo=nuovoIndirizzo;
  }

  float calcolaMedia(){
  ….
  }
}
```

parametro

questo metodo non restituisce niente

nome metodo

questo metodo non prende in ingresso niente

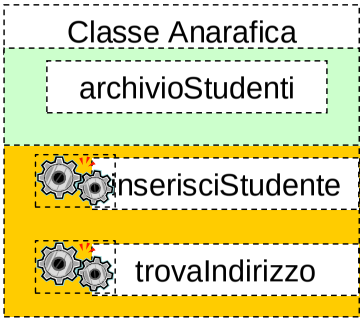
Corpo (vedremo cosa scriverci)

tipo di dato restituito

I metodi: parametri di ingresso e uscita

- 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

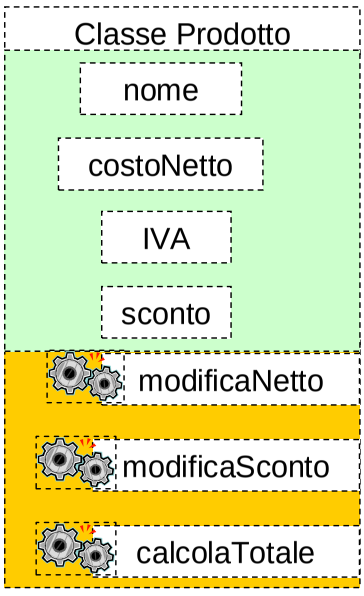
```
class Anagrafica{
  Sudente archivioStudenti[]=new Studente[1000];

  void inserisciStudente(String nome, String indirizzo){
  ....
  }

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



I metodi: l'istruzione return

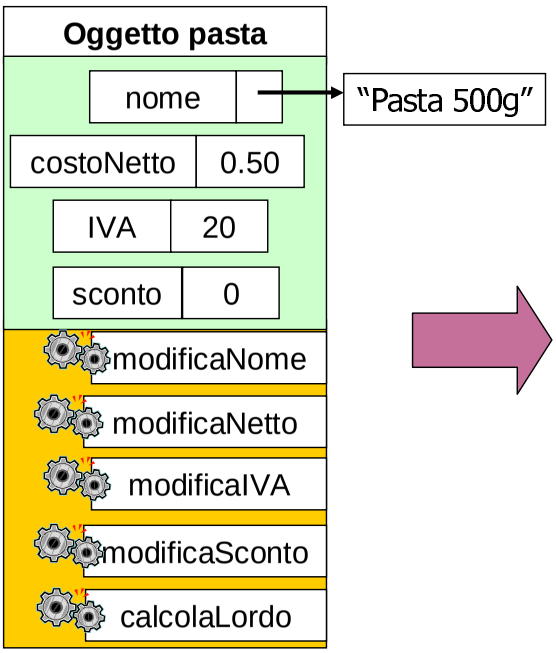
- L'istruzione return definisce il valore restituito da un metodo

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

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

.....
}
```



Uso dei metodi

- Per chiamare i metodi si usa l'handler seguito da un punto e il nome




```
Prodotto pasta;
pasta=new Prodotto();

pasta.modificaNome("Pasta 500g");
pasta.modificaNetto(0.50);
pasta.modificaIVA(20);
pasta.modificaSconto(0);

System.out.println(pasta.calcolaLordo());
```

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;
  }
}
```

Il costruttore di Carta

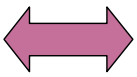
```
class QuattroAssi{
   Carta assoQuadri= new Carta('Q','1');
   Carta assoCuori= new Carta('C','1');
   Carta assoPicche= new Carta('P','1');
   Carta assoFiori= new Carta('F','1');
}
```

I costruttori II

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

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

Sono equivalenti



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

   Carta(){
   }

}
```

Creazione di un oggetto: maggiori dettagli

Quando si crea un oggetto (istruzione new)

1) Viene considerato 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) Viene eseguito il codice del costruttore

```
class Studente{
  String nome, indirizzo;
  int numeroEsamiSostenuti=0;

 Studente(String n, String i){
    nome=n;
    indirizzo=i;
  }
}
```

Si allocano le variabili e si assegna 0

Poi si esegue

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 comadi 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

If ... then .... else

- L'istruzione if contiene una condizione booleana racchiusa fra parentesi tonde che permette di scegliere fra due blocchi di istruzioni racchiusi fra parentesi graffe

```
if (condizione_booleana) {
    blocco 1
    }
else{
       blocco 2
       }
```

```
if (vendite > media) {
    premio = 100;
    }
else {
       premio = 0;
       }
```

Espressioni booleane

- Le espressioni boleane si costruiscono a partire da operatori di confronto
  - uguale ==
  - diverso !=
  - maggiore >
  - maggiore o uguale >=
  - minore <
  - minore o uguale <=

```
(c != 23)
```
è **true** se c è diverso da 23

```
(b == 's')
```
è **true** se b è il carattere 's'

Espressioni booleane II

- Le espressioni possono essere combinate utilizzando gli peratori booleani:
  - And &&
  - Or ||
  - Not !

```
( !((c == 23) || (c ==24)) )
( (b == 's') || (b =='S') )
( !((a >= 0) && (a < 0.5)) )
```

c non è uguale a 23 nè a 24

b è il carattere 's' o 'S'

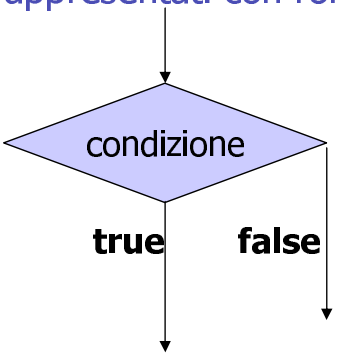
Non è vero che a è compreso tra 0 (compreso) e 0.5 (escluso)

Diagrammi di flusso

I diagrammi di flusso sono

- rappresentazioni grafiche di sequenze di istruzioni
- tre tipi di forme

unità di scelta
rappresentati con rombi



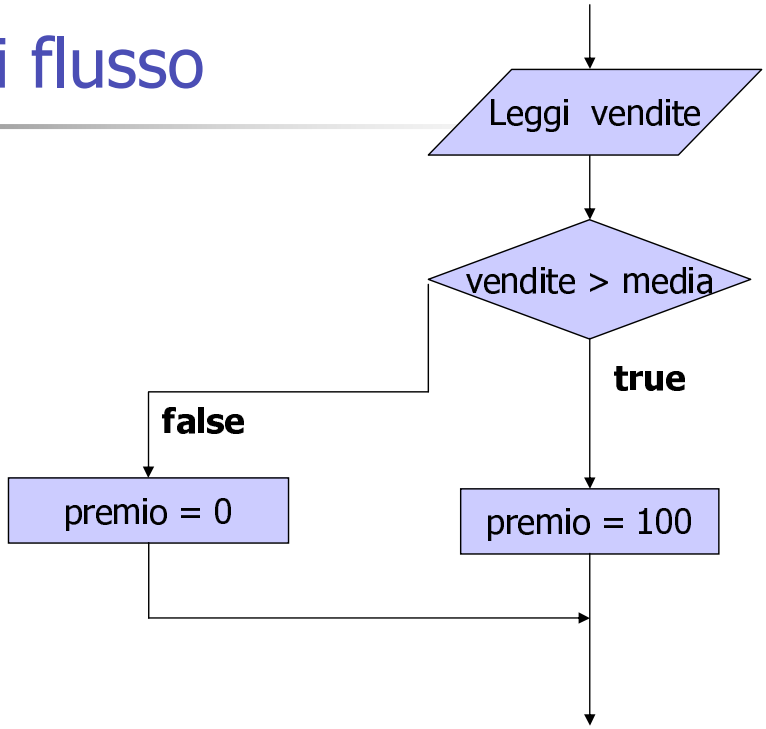
blocchi di istruzioni
rappresentati da rettangoli

```
istruzione1;
istruzione2;
...
```

Istruzioni di lettura/scrittura
rappresentati da parallelogrammi

Diagrammi di flusso

```
Leggi vendite;

if (vendite > media) {
    premio = 100;
    }
else {
       premio = 0;
       }
```

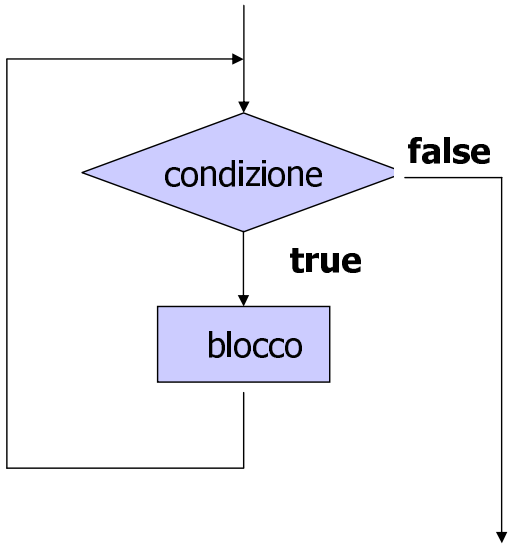


Ciclo while

- L'istruzione while permette di ripetere una certa operazione

Schema generale

```
while (condizione_booleana) {
       blocco
       }
```

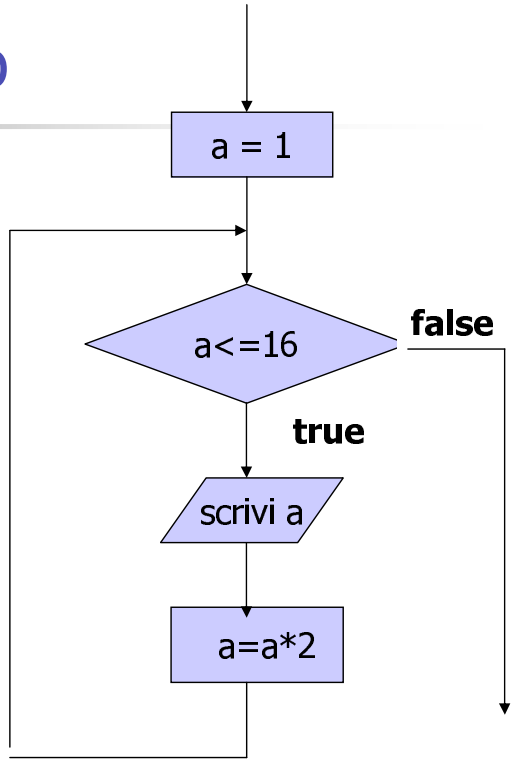


Ciclo while: esempio

Esempio: stampa le potenze di 2
l'ultimo valore stampato è 16

```
int a=1;
while (a<=16) {
      System.out.println(a);
      a=a*2;
      }
```

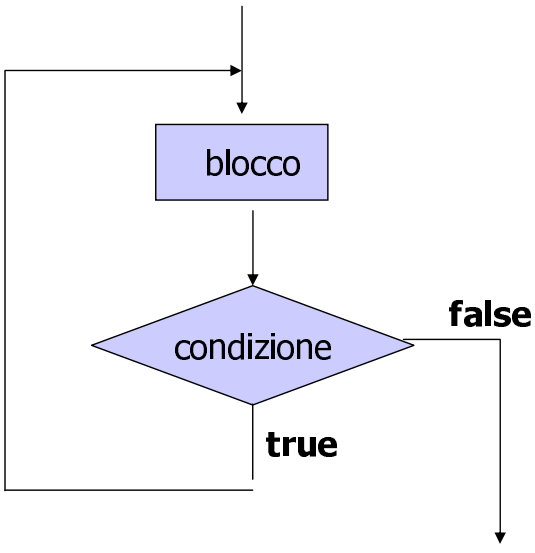
Stampa 1, 2, 4, 8, 16



Ciclo do .... while

Schema generale

```
do {
    blocco
    } while (condizione_booleana)
```

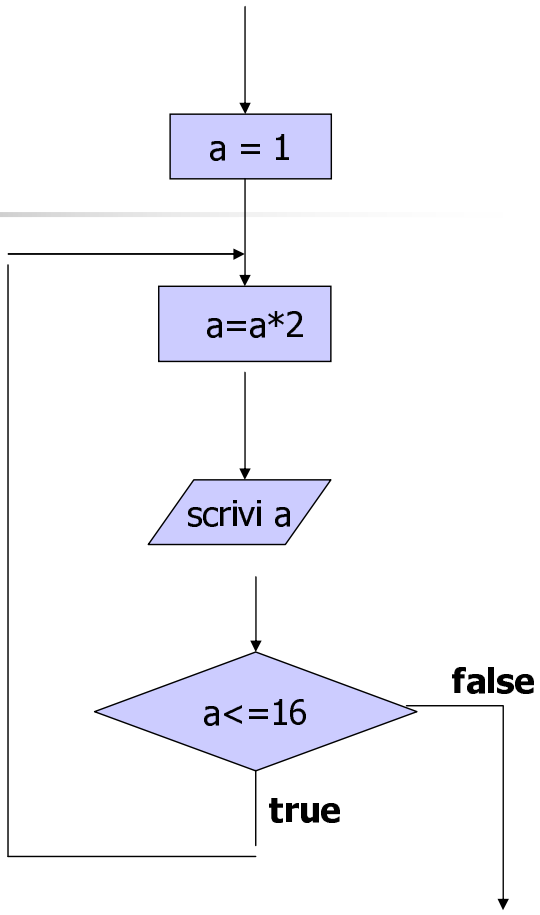


Ciclo do .... while: esempio

Esempio: stampa le potenze di 2
l'ultimo valore stampato è 32

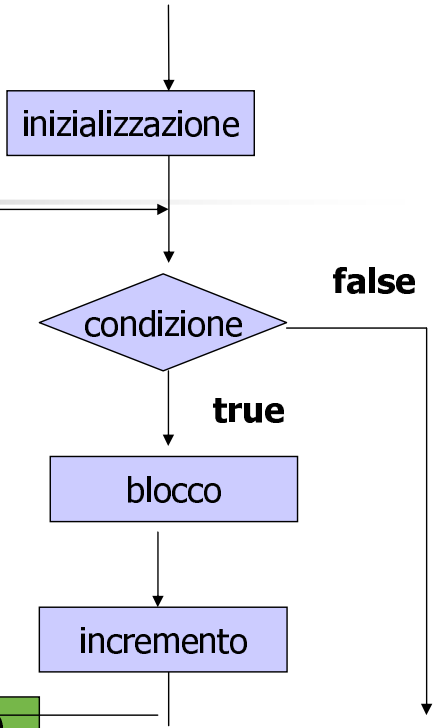
```
int a=1;
do {
     a=a*2;
     System.out.println(a);

   } while (a<=16)
```

Stampa 2, 4, 8, 16, 32



Cliclo for

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



Schema generale

```
for (inizializzazione; condizione_booleana; incremento)
   {
    blocco
    }
```

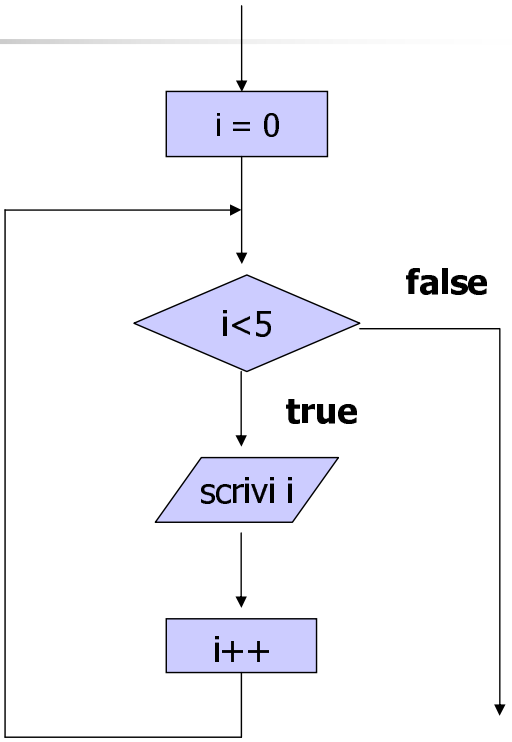
Ciclo for: esempio

Esempio: loop ripetuto 5 volte

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

stampa 0, 1, 2, 3, 4

i++ è una notazione coincisa per i=i+1



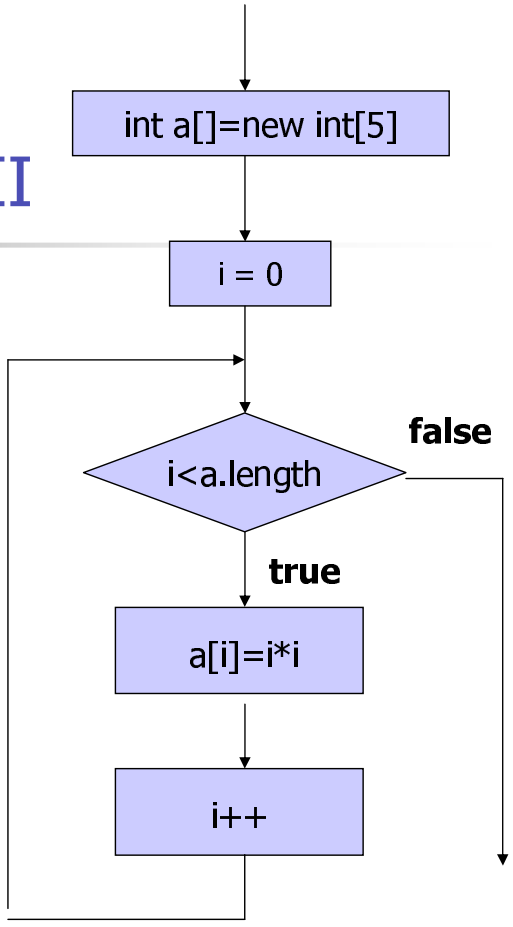
Ciclo for: esempio II

Riempie un array con i quadrati

```
int a[]=new int[5];
for (int i=0; i<a.length; i++) {
     a[i]=i*i;
}
```

length è una variabile che contiene la lunghezza di un array:
esiste per ogni array!!

0	1	4	9	16



Cliclo for: esempio III

Stampa il contenuto di un array in ordine inverso

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

Rappresentazione coincisa di i=i-1

Cliclo for: esempio IV

- Inizializzazione e incremento possono contenere più operazioni separate da virgole

Riempie un array con le potenze del 2

```
int a[]=new int[5];
for (int i=0, int j=1; i<a.length; i++, j=2*j) {
      a[i]=j;
}
```

inizializzazione

condizione boleana

incremento

1	2	4	8	16

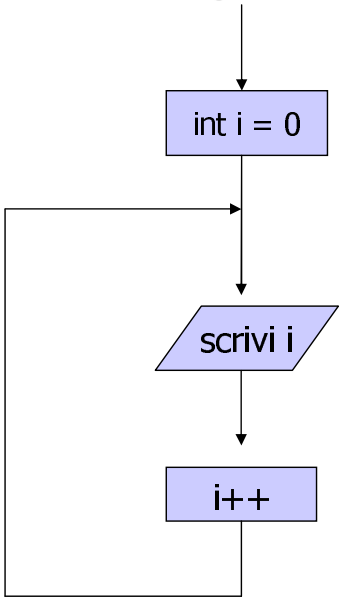
Cliclo for infinito

- Se la condizione è vuota allora il for è eseguito all'infinito

Esempio: loop infinito

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

Stampa 0, 1, 2, 3, 4, 5, 6, 7,....



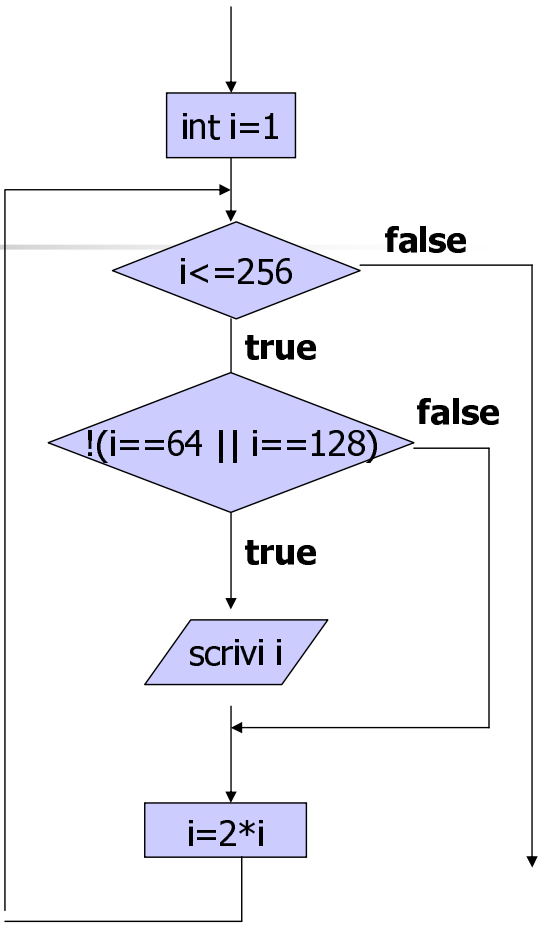
Annidamento

- 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);
}
```

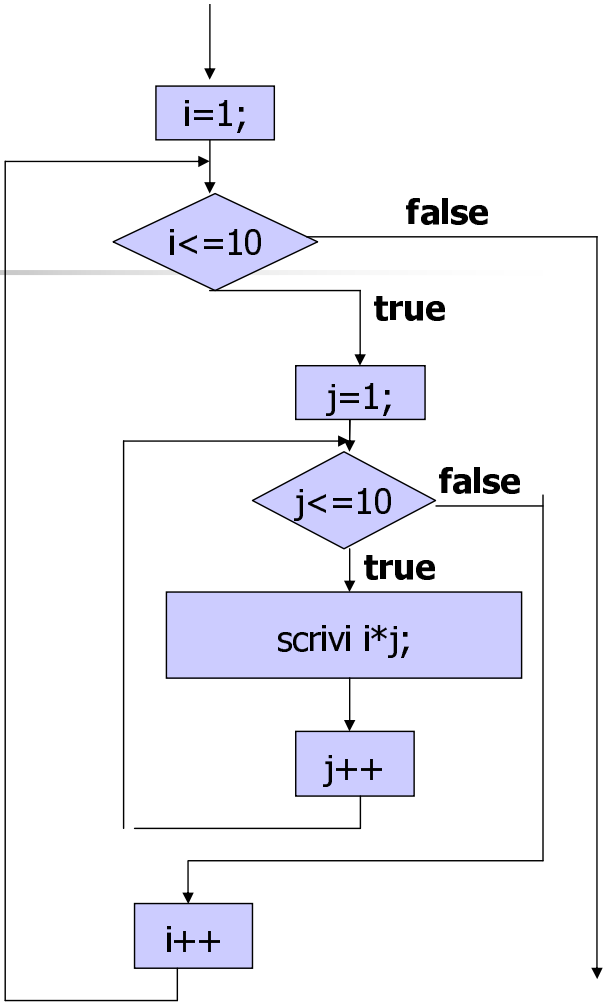
Stampa 1, 2, 4, 8, 16, 32, 256



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);
 }
}
```

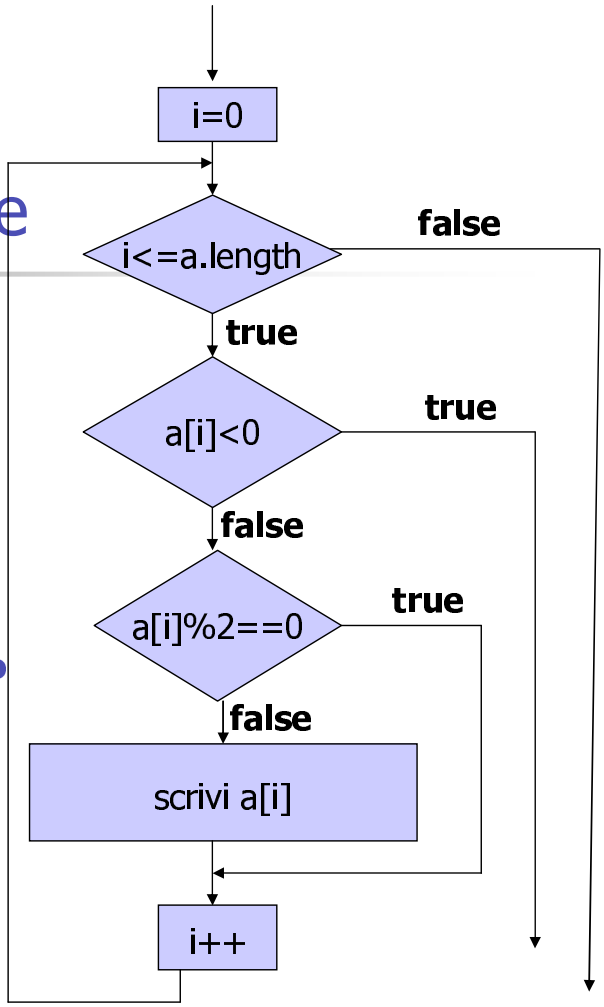


Break and continue

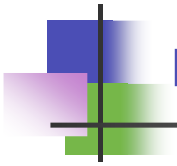
- break interrompe l'attuale istruzione di iterazione ed esce definitivamente dalla iterazione
- continue interrompe la corrente iterazione e ritorna all'inizio del ciclo, iniziandone un'altra

Stampa il contenuto di un vettore:
- **fermandosi se trova un numero negativo**
- **non stampando i numeri pari**

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



Ancora su Java



Costruttori e metodi con lo stesso nome (Overloading)

- Una classe puo' avere piu' costruttori o metodi con lo stesso nome purche' abbiano parametri di ingresso diversi
- Il compilatore determina automaticamente quale metodo/costruttore chiamare

Si usa un costruttore diverso a seconda che lo studente sia lavoratore o meno

```
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;
 }
}
```

Costruttori e metodi con lo stesso nome (Overloading) II

Un esempio di metodi con lo stesso nome

```
class Segreteria {
  Studente  archivioStudenti =new Studente[10000];
  Corso archivioCorsi=new Corso[1000];
  int numeroStudenti=0, numeroCorsi=0;

  void  inserisci(Corso a){
      archivioCorsi[numeroCorsi]=a;
      numeroCorsi++;
   }

void  inserisci(Studente a){
      archivioStudenti[numeroStudenti]=a;
      numeroStudenti++;
   }

}
```

Chiama

Chiama

```
Segreteria s=new Segreteria;

Corso c=…
s.inserisci(c);

Studente  b= ….
s.inserisci(b);
```

Costruttori e metodi con lo stesso nome (Overloading) III

Attenzione

- Non sono ammessi metodi che differiscono solo per i parametri di uscita

```
class MathTool {

  float somma(float a, float b){
     return a+b;
  }

  int somma(int a, int b){
     return a+b;
  }

  String somma(int a, int b){
     return (String)(a+b);
  }

}
```

ok

errore

Blocchi

- Un blocco è una parte di programma delimitata da parentesi graffe
- sono blocchi
  - Una classe
  - Un metodo
  - Un for (inclusi inizializzazione, condizione e incremento)
  - Un while (inclusa la condizione)
  - .....

blocchi

```
Class MathTool{
  int maxInt= ....
  int minInt=...

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

  int factorial(int numero){
     int  risultato=1;
      for(int i=1;i<=numero;i++){
              risultato=risultato*i;
          }
      return risultato;
  }

  int getMaxInt(){
     return maxInt;
  }
}
```

Visibilita' delle variabili

- Una variabile è visibile (accessibile)
  - solo dopo la sua definizione
  - solo all'interno del blocco in cui è definita

Esempi

- maxInt è visibile in tutta la classe
- questa variabile numero è visibile solo in factorial
- questa variabile i è visibile solo all'interno del for

```
Class MathTool{
  int maxInt= ....
  int minInt=...

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

  int factorial(int numero){
     int  risultato=1;
      for(int i=1;i<=numero;i++){
              risultato=risultato*i;
          }
      return risultato;
  }

  int getMaxInt(){
     return maxInt;
  }
}
```

Visibilita' delle variabili II

```
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, ultimoRisultato è definita nella classe, ma dopo

ok

errore, i è definita solo nel for

errore, risultato è definito solo in power

Variabili con lo stesso nome

Due variabili possono avere lo stesso nome se

- sono definite in blocchi diversi
- va bene anche se uno dei blocchi contiene l'altro, ma in questo caso la variabili esterna non è più visibile (la variabile interna copre l'esterna)
  - Se la variabile esterna è definita nella classe, allora si può sempre accedere usando this

Vantaggi del meccanismo di visibilità

- non occorre inventarsi nomi diversi per ogni oggetto trattato
- molte persone possono collaborare alla realizzazione di un'applicazione senza doversi trovare d'accordo sul nome di tutte le variabili
  - Se capita che persone diverse scelgano lo stesso nome per variabili diverse, non ci sono problemi purché saranno definite in blocchi diversi

Variabili con lo stesso nome: esempio

```
Class Studente{
  String nome;
  String cognome;
  String luogoDiResidenza;
  String cognome;

  void modificaNome(String nome){
      this.nome=nome;
  }

  void modificaCognome(String cognome){
      cognome=cognome;
  }

   void modificaResidenza(String nome){
      luogoDiResidenza=nome;
  }

}
```

this indica la classe in cui ci si trova

Errore, cognome già definito

ok, modifica correttamente il nome

nessun errore, ma il metodo non fa niente

ok, è strano che si chiami nome il luogo di residenza, ma il metodo funziona correttamente

Variabili con lo stesso nome: esempio II

```
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"

Variabili con lo stesso nome: esempio II

```
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"

Modificatori di visibilità

- I modificatori di visibilità sono parole chiave che definiscono la visibilità, dall'esterno della classe, di una variabile o un metodo: ne vedremo alcuni ...
- public: la variabile/metodo è visibile a tutte le altre classi
- private: la variabile/metodo è visibile sono all'interno della classe
- static: la variabile/metodo è una proprietà della classe,
  - è visibile anche senza aver creato l'oggetto,
  - esiste una sola copia della variabile per tutti gli oggetti della classe
- final: una variabile dichiarata final è una costante che può essere inizializzata, ma non ulteriormente modificata

Modificatori di visibilità: public, private

- È buona norma rendere accessibile dall'esterno della classe solo lo stretto necessario!!!

Esempio: un conto corrente dove il bilancio può essere modificato/letto solo attraverso gli appositi metodi

```
Class CC{
  private float bilancio=0;

   public void deposito(float quantita){
        bilancio= bilancio+quantita;
       }

  public void getBilancio(){
        return bilancio;
       }
}
```

```
CC mioCC=new CC();
mioCC.bilancio= mioCC.bilancio+100;
```


Il compilatore genera un errore

Modificatori di visibilità: static, final

Informazioni predefinite sono memorizzate in variabili/metodi dichiarati static e final

- Le variabili di Float (ne esistono corrispondenti in Integer, Double, ...)
  - static final float maxValue, minValue

    (restituiscono il massimo/minmo valore rappresentabile)
- I metodi
  - static float parseFloat(String s)

    prende una stringa e restituisce il corrispondente float
  - static boolean isNaN(float f)

    prende una float e dice se rappresenta il valore NaN (not a number)

```
System.out.println(Float.maxValue);
System.out.println(Float.isNaN(1/0));
```

Tali metodi/variabili sono usabili direttamente dalla classe senza creare l'oggetto!!!

Modificatori di visibilità: static, final

Una classe di cui possono essere creati solo 5 oggetti

```
Class Connessione{
  private final int maxConnesioni=5;
  private static int  numConnessioni=0;

  private Connessione(){}

  public static Connessione getConnessione(){
      if (numConnessioni<=maxConnesioni){
            System.out.println("Connesione aperta");
            numConnessioni++;
            return new Connessione();
            }
          else {
                System.out.println("Non esistono connessioni disponibili");
               return null;
            }
      }
}
```

Gestione delle eccezioni

- Le eccezioni sono anomalie nel funzionamento di un programma
- Possono dipendere da errori di programmazione, dati sbagliati e guasti del computer
- Le eccezioni possono essere classi definite dal programmatore o classi predefinite in Java: AritmeticException, IOException,….
- Java fornisce i comandi try e catch per gestire (catturare) le eccezioni

  se c'è un'eccezione nel blocco del try si esegue ciò che c'è nel catch

```
float div(float a, float b){
   float result=0;
    try{
            result=a/b;
   }catch(AritmeticException e){
            System.err.println("Si è verificato un errore:"+e.getMessage())}
  return result;
}
```

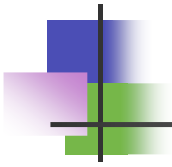
Gestione delle eccezioni

- Per alcune eccezioni, il compilatore Java segnala un errore se non vengono gestite
- Le eccezioni all'interno di un metodo possono essere anche rilanciate con il comando throws:
  - in tal caso sarà il programma che chiama il metodo a dover gestire l'eccezione

```
float div(float a, float b) throws AritmeticException {
   float result=a/b;
   return result;
}
```

IO (ingresso, uscita)

- Un'applicazione può scrivere e leggere da periferiche (detti flussi, streams) diverse:
  - tre di esse sono predefinite, gli standard in, out and error
- Java contiene classi predefinite per la gestione dell'ingresso e l'uscita verso queste
  - System.out ➔ E' la classe per scrivere sullo standard out
  - System.in ➔ E' la classe per scrivere sullo standard input
  - System.err ➔ E' la classe per scrivere sullo standard error



Standard Output/Error

- System.out e System.err (classe PrintStream) forniscono una serie di metodi per scrivere dati sullo stream:
  - print()
  - println()

- Per scrivere sullo standard output si usa quindi la funzione:

  System.out.println()

- Per scrivere sulllo standard error:

  System.err.println()

Standard output: esempio

- Implementare un metodo che prende in ingresso un vettore di stringhe e le stampa

  void stampaArray(String A[])

```
public void stampaArray(String A) {
    for (int i=0; i<A.length;i++) {
            System.out.println(A[i]);
            }
}
```

Standard Input

- System.in (classe InputStream) fornisce una serie di metodi per leggere dati sullo stream:
  - read()
- Per leggere dati da tastiera (standard input) si utilizza la riga di comando:

  int System.in.read() throws IOException
- Questa funzione legge un carattere per volta (int) quindi per leggere una sequenza deve essere inserita in un ciclo.

Standard Input: esempio

- Implementare un metodo che legge una stringa da tastiera e la restituisce
- Si devono leggere tutti i caratteri digitati fino ad al prtimo invio (il carattere '\n')
- per concatenare due stringhe si usa il simbolo +
- per convertire un intero in un carattere c si può scrivere String.valueOf(c)
- Catturare IOException

```
public String leggiDaTastiera() {
  String s="";
  try{
        char c=System.in.read();
        while (c!='\n') {
                s=s+String.valueOf(c);
                c=System.in.read();}
  catch(IOException e){
        System.err.println(e.getMessage());}
   return s;
}$
```

Compilare ed eseguire programmi Java

Java e files

File e classi

- Ogni classe Java si trova in un file il cui nome è esattamente uguale al nome della classe seguito da ".java":
  - ad esempio, la classe Studente si troverà in Studente.java
  - se nome della classe e nome del file non corrispondono, si ha un errore

File e compilati

- Un programma Java per essere eseguito deve essere prima compilato (tradotto)
- Al compilato viene dato il nome della classe seguito da "class"
  - ad esempio, il compilato della classe Studente si troverà in Studente.class

File: Prova.java

Compilazione e esecuzione

File e classi

- il compilatore Java produce un codice intermedio detto Java bytecode
- il Java bytecode è contenuto nei file ".class"
- il Java bytecode consiste in un insieme istruzioni per una macchina virtuale detta Java Virtual Machine (JVM)

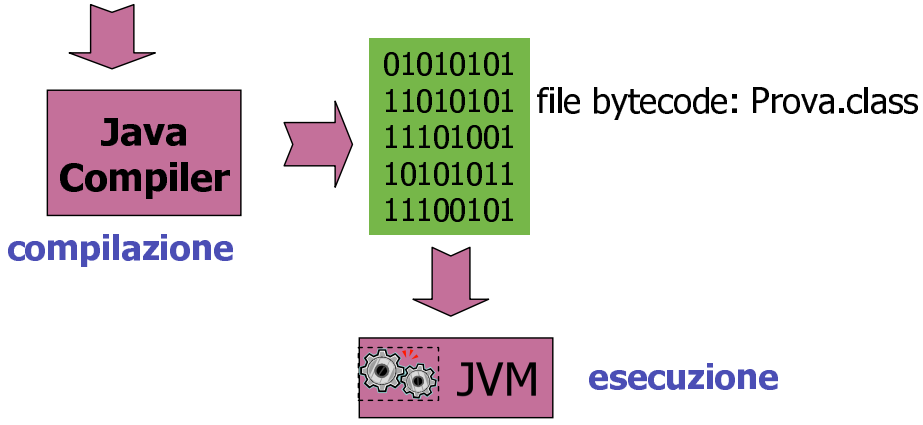
Java Virtual Machine

- è un programma che simula un calcolatore virtuale
- esistono JVM per ogni sistema operativo, nei browser, sui cellulari ... in questo modo il Java bytecode è eseguibile praticamente su ogni architettura
  - massima portabilità!!!

Compilazione e esecuzione

File: Prova.java

```
class Prova {
  public static void main(String argv[]) {
     System.out.println("Hello World!");
  }
}
```



Il metodo main: tutto inizia da quì

Il metodo che si chiama main

- è un metodo speciale che serve ad inizializzare e far partire un programma
- è la prima cosa che la JVM esegue
- può trovarsi in qualsiasi classe
- è definito come
  - public: viene chiamato dall'esterno della classe
  - static: viene chiamato all'inizio, quando ancora nessun oggetto è stato creato
  - void: non restituisce niente
  - può ricevere in ingresso un insieme di stringhe

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

Il metodo main: un esempio

Si supponga che siano state già implementate le seguenti classi

```
class Facolta{
   Facolta(String Nome, String indirizzo);
}
```

```
class Archivio{
   Archivio();
   void add(Facolta f);
}
```

```
class graphicalInterface{
 graphicalInterface(Archivio a);
}
```

Il metodo main: un esempio

Nel programma che gestisce la segreteria studenti il main potrebbe:

- creare l'oggetto che contiene l'archivio dei dati
- inserire le facoltà nell'archivio
- far partire l'interfaccia grafica che interagisce con l'utente e attraverso la quale sarà possibile inserire e modificare corsi, studenti, esami .....

```
Class Segreteria{
   public static void main(String arg[]){
      Archivio ds = new Archivio();
      ds.add(new Facolta("Ingegneria","Via Roma 56", …));
      ds.add(new Facolta("Fisica","Via Roma 56", …));
      graphicalInterface gi= new graphicalInterface(ds);
  }
}
```

Le classi e l'organizzazione in package

- Un programma Java compilato è composto da un insieme di file:
  - ogni file ".class" può contenere la definizione di una sola classe ed ha il nome di tale classe
- I file sono organizzati in packages (librerie)
- La libreria fornita con Java (Java Delopment Kit, JDK) è costituita da centinaia organizzati in packages:
  - java.lang , java.awt, java.io , ecc…
- L'istruzione package permette di definire il package a cui appartiene una classe

```
package Universita

Class Rettore{
   String nome;
    .....

}
```

L'istruzione Import

L'istruzione import

- permette di usare una classe contenuta in un package
- la classe si indica con il nome del package seguito da un punto e il nome della classe
- si comporta come se copiasse la classe nella punto in cui si trova il comando

```
import Universita.Rettore;

Rettore r=new Rettore();
```

I package e l'organizzazione gerarchica

- I package possono essere organizzati in una gerarchia
  - un package può essere contenuto in un altro
- Per indicare che una package è contenuto in un'altro si usa la notazione con i punti

Definizione di una classe di Studente che si trova in Anagrafica che si trova in Didattica …

```
package Universita.Didattica.Anagrafica;

Class Studente{
   String nome;
    …….
}
```

Uso della classe Studente …

```
import Universita.Didattica.Anagrafica.Studente;

Studente s=new Studente();
```

I package e l'organizzazione gerarchica II

- Una classe della gerarchia può essere usata anche senza l'istruzione import, purché si usi il nome completo

```
Universita.Didattica.Anagrafica.Studente  s;
s=new Universita.Didattica.Anagrafica.Studente();
```

- Inoltre è possibile importare tutte le classi di un package usando il simbolo *

```
import Universita.Didattica.*;
```

Esercizi

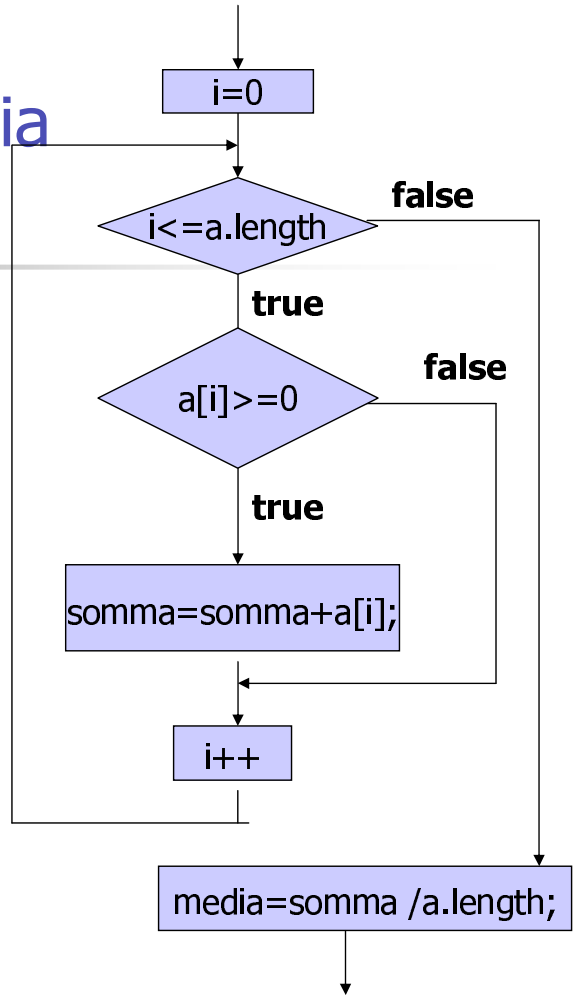
La somma e la media di un array

- 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];
```

La somma e la media di un array

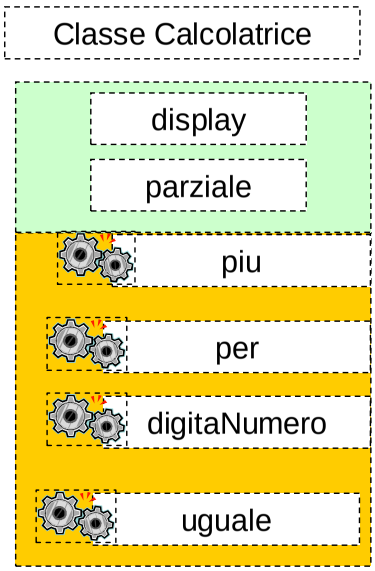
```
float somma=0;
float media;

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



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



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 uguale(){
      display=parziale;
   }
}
```

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=' ';
                       }
          }
    }
```