

```

class Punto {
    double x;
    double y;

    Punto ( double a, double b ) {
        x = a;
        y = b;
    }

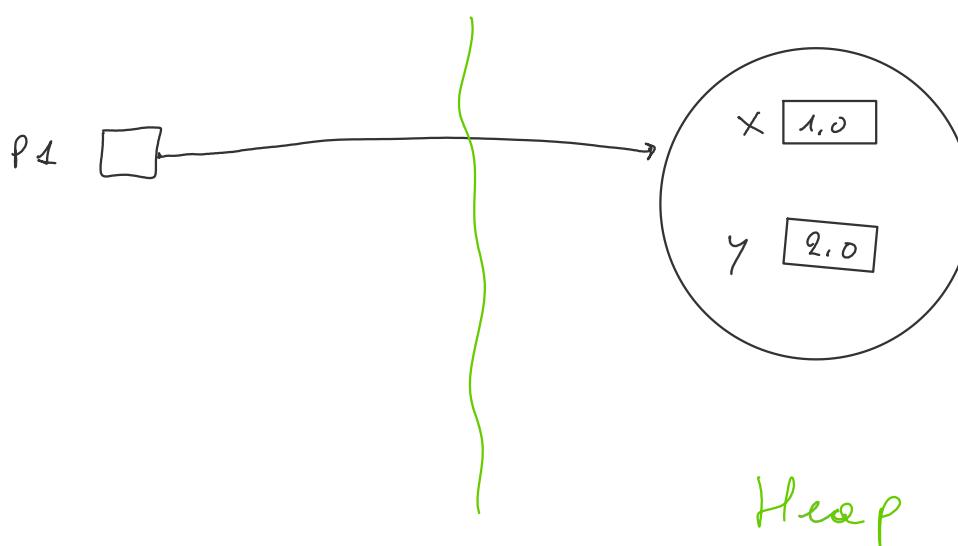
    double distanza () {
        return Math.sqrt( x * x + y * y );
    }

    void trasla ( double t ) {
        x += t;
        y += t;
    }
}

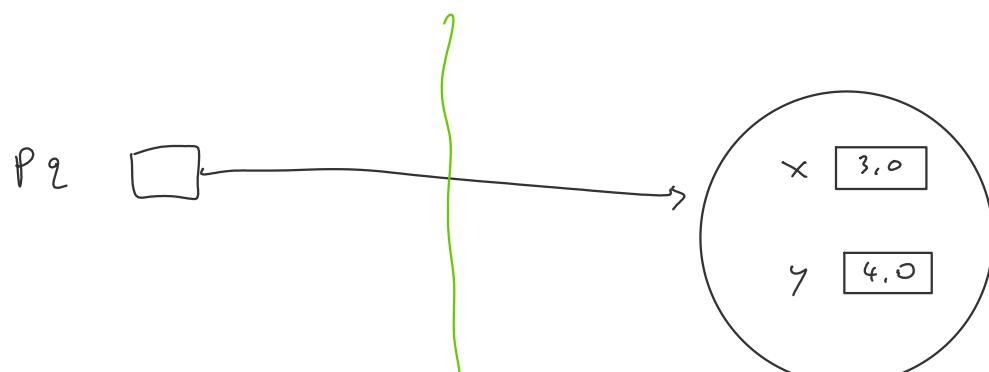
```

Punto p1;

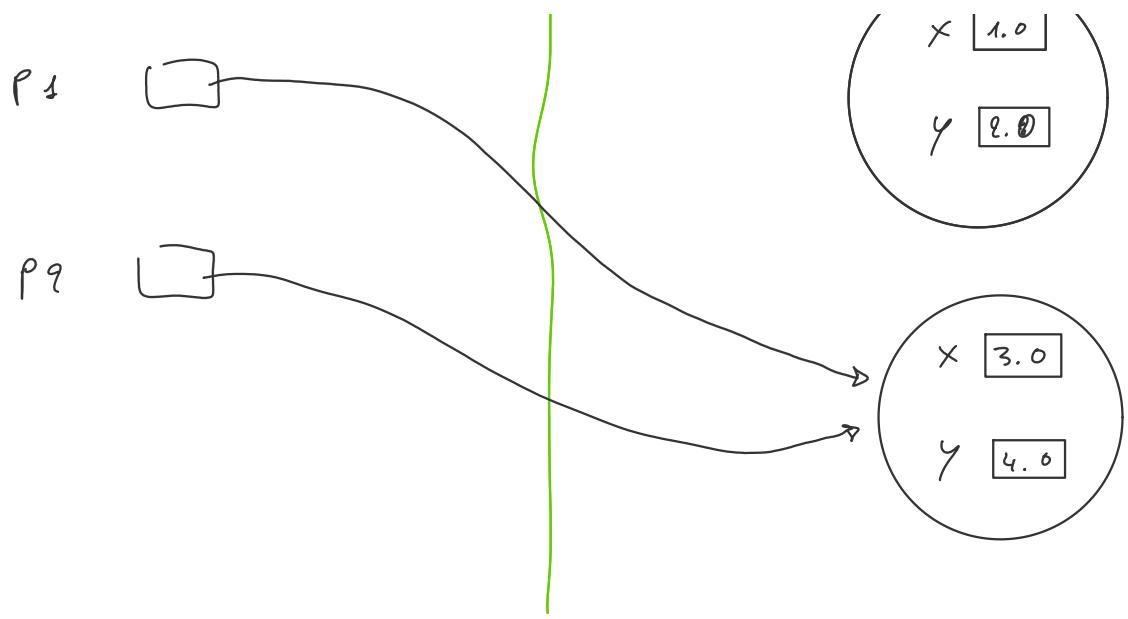
p1 = new Punto ( 1.0, 2.0 );



Punto p2 = new Punto ( 3.0, 4.0 );



`p1 = p2;`



Per invocare un metodo su un oggetto,  
si usa la notazione a punto

nomeDelRiferimento. nomeDelMetodo ( )  
 ↑  
 eventuali  
 argomenti

Per es.

```
double d1 = p1. distanza();  

double d2 = p2. distanza();  

p1. trasla (5,3);
```

La notazione a punto può essere usata anche  
per accedere alle variabili istanza

nomeDelRiferimento. nomeDellaVariabile

Per esempio:

```
p1. x = 7.7;  

double z = p2. y;
```

```
class Esempio {  

    float uno, due;
```

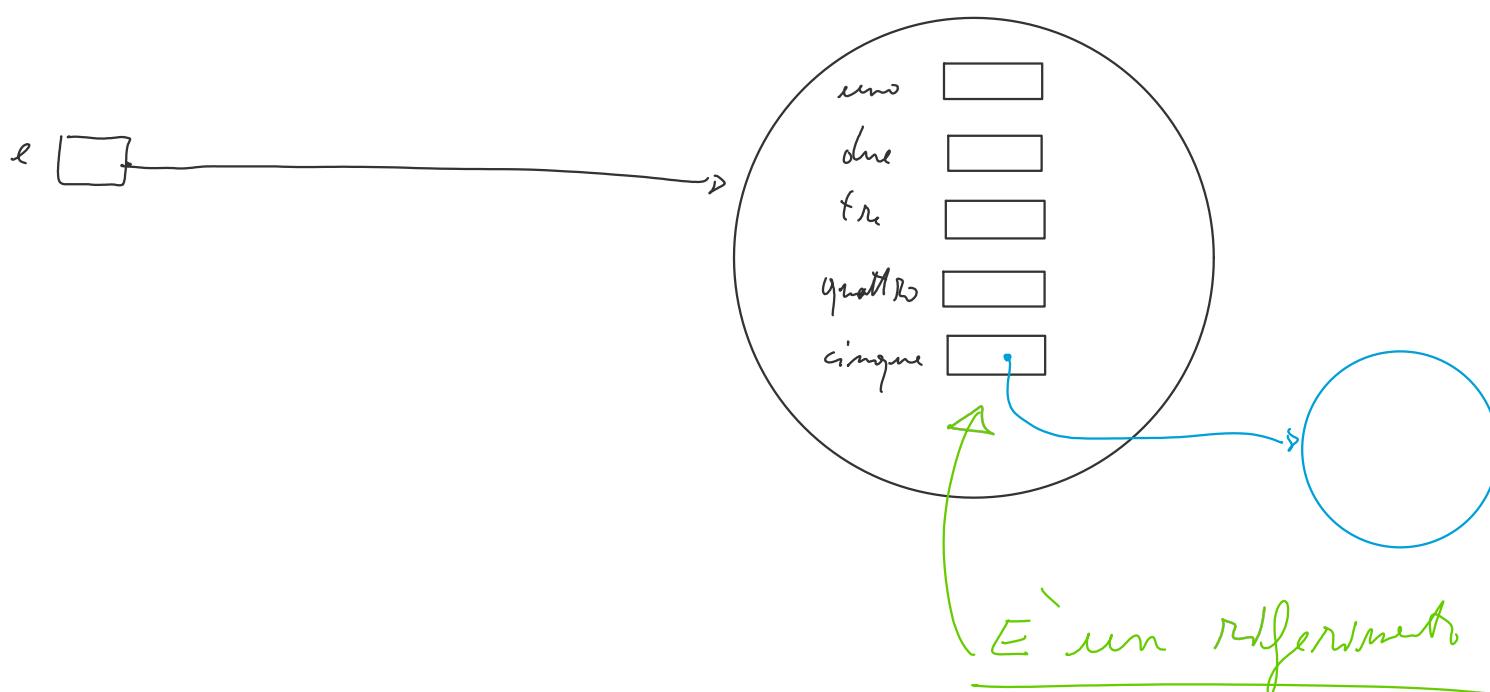
...  
char quattro;

AltraClass cinqne;

;

}

Esempio `e = new Esempio();`



`e.cinqne = new AltraClass();`

### Initializzazione variabili istanza

È possibile fornire un iniziatore

- un valore costante
- il valore di un'altra variabile istanza
- il valore restituito da un metodo  
(o fatto che non possono lanciare eccezioni di tipo checked)
- un'espressione che combina i punti precedenti

Esempio:

```
class MiaClass {  
    double a = 3.1;  
    double b = a;}
```

ordine  
di  
inizializzazione

```

double c = Math.sqrt(5.0);
double d = 11.3 + a * Math.sqrt(5.0);
:
}

```



Le variabili istanza hanno un valore di default  
(se non specificato diversamente)

<u>Tipo</u>	<u>Valore di default</u>
boolean	false
char	'\u0000'
byte, short, int, long	0
float, double	0.0
riferimento	null

## Costruttori

I costruttori sono blocchi di istruzioni utili a definire lo stato iniziale di un oggetto.

Si usano in due casi:

- le possibilità fornite dagli inizializzatori non sono sufficienti
- lo stato iniziale è noto solo al tempo in cui l'oggetto viene creato

## Un costruttore

- ha lo stesso nome della classe
- non restituisce alcun valore
- può avere zero o più parametri
- può essere mandato in esecuzione solo insieme a new

In Java una classe può avere

zero o più costruttori

Se la classe ha zero costruttori:

c'è il costruttore di default, che non prende argomenti e non esegue operazioni

Se la classe ha uno o più costruttori  
allora quelli di default non è più disponibile

Se la classe ha più costruttori:

devono essere distinguibili per numero e/o tipo  
dei parametri

```
class A {  
    double uno;  
    double due;  
    ;  
}  
A a = new A();
```

 posso usare il  
costruttore implicito

```
class A {  
    double uno;  
    double due;  
    A (double u, double d){  
        uno=u;  
        due=d;  
    }  
}
```

A a1 = new A(1.1, 2.2); OK

A a2 = new A(); Error: il costruttore di default  
non è più disponibile

```

class A {
    double uno;
    double due;
    A () {
    }
    A ( double u ) {
        uno = u;
    }
    A ( double u, double d ) {
        uno = u;
        due = d;
    }
}

```

3 costruttori

Quando creiamo un oggetto

- 1) viene allocato lo spazio in memoria (heap)
- 2) le variabili istanza assumono il valore specificato dagli inizializzatori
- 3) viene eseguito il codice del costruttore

```

PA1819_inizializzazione
Main.java
1 class Main {
2
3     public static void main(String[] args) {
4         Prova p1 = new Prova(10);
5         Prova p2 = new Prova(20);
6         p1.stampa();
7         p2.stampa();
8     }
9 }

java version "1.8.0_31"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)

```

b vale sempre 2  
perché il costruttore viene eseguito solo dopo che la fase di inizializzazione è terminata

## Metodi

- Il chiamante di un metodo deve fornire un valore per tutti gli argomenti formali

- Il tipo di ogni argomento attuale deve essere lo stesso del corrispondente argomento formale (o implicitamente convertibile).
- E' possibile avere overloading dei metodi:
  - più metodi possono avere lo stesso nome
  - devono essere distinguibili per numero e/o tipo degli argomenti
  - il tipo del valore restituito non è utile a differenziare

Per esempio la class Points potrebbe avere un secondo metodo trasla

```
void trasla (double dx, double dy) {
    x += dx;
    y += dy;
}
```

p1. trasla (1.0);

p1. trasla (2.0, 3.0); viene eseguito questo

Quando un metodo viene invocato

- 1) viene elaborato spazio nello stack per gli argomenti formalis e eventuali variabili locali.
- 2) gli argomenti formalis vengono inizializzati con il valore dei corrispondenti arg. attuali
- 3) al termine dell'esecuzione del metodo gli arg. formalis e var. locali vengono deallocati

Il passaggio dei parametri è sempre per valore.

```

class A {
    void m1( int x ) {
        x++;
    }
}

```

```

A a = new A();
int y = 10;
a.m1(y);
y vale 10

```

Altro esempio:

```

class Uno {
    int valore;
    Uno( int v ) {
        valore = v;
    }
    void setValore( int v ) {
        valore = v;
    }
}

```

```

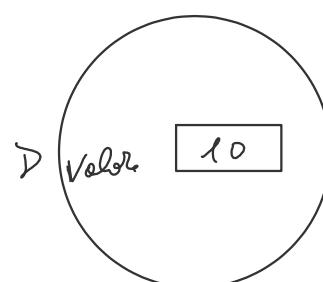
class Due {
    void metodo1( Uno u ) {
        u = new Uno( 100 );
    }
}

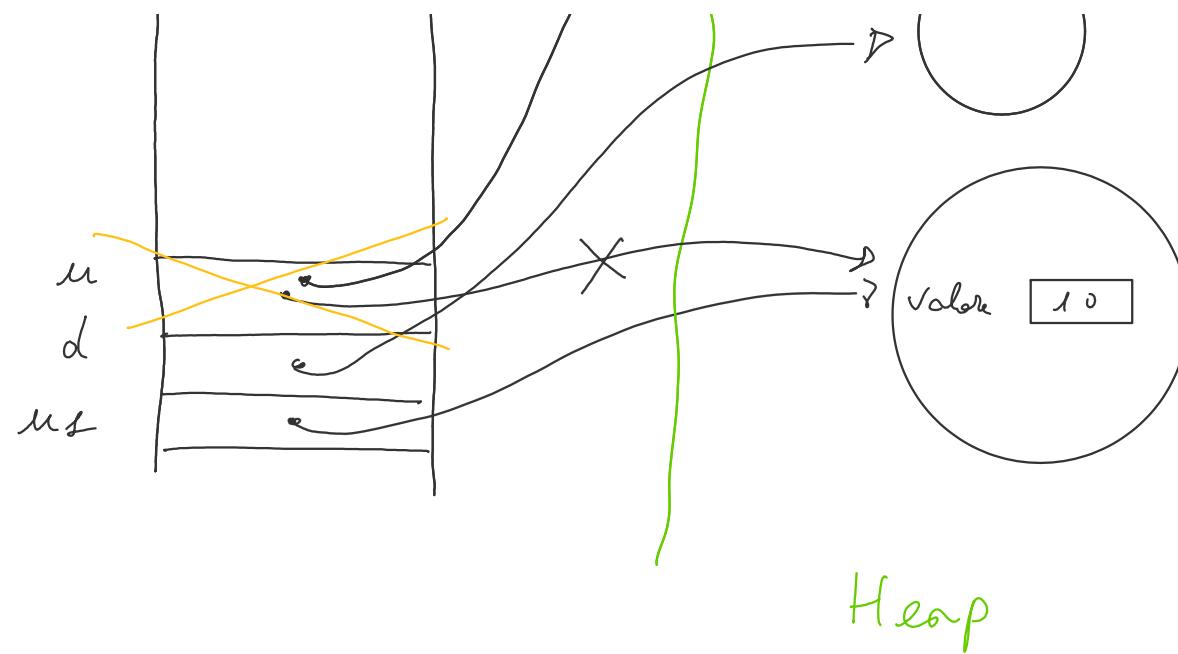
```

```

Uno u1 = new Uno( 10 );
Due d = new Due();
d.metodo1( u1 );

```





```

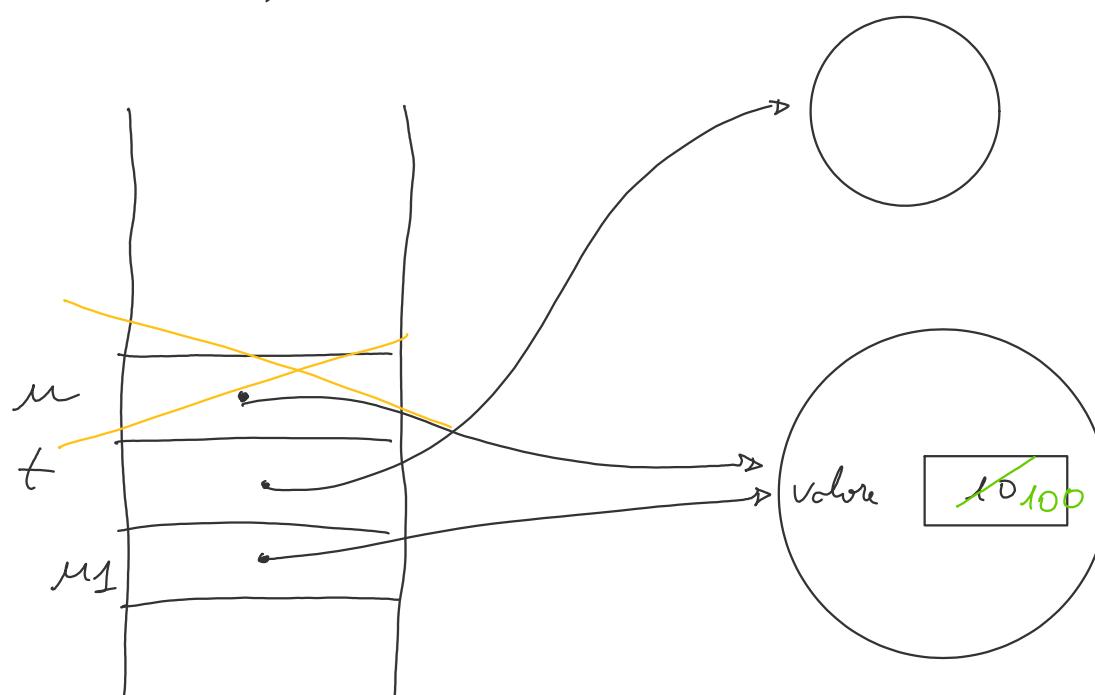
class Tre {
    void metodo2(Um u) {
        u.setValore(100);
    }
}

```

Ums ms = new Ums(10);

Tre t = new Tre();

t.metodo2(ms);



Ricapitolando:

- un metodo non può modificare un parametro attuale (dell'chiamante) di tipo primitivo
- un metodo non può modificare un parametro attuale (dell'chiamante) di tipo riferimento

- un metodo può modificare un oggetto  
old cui ha riferito il riferimento

Initializzazione varia lib local

A differenza delle variabili istanza non è previsto un valore di default.

Il compilatore controlla che ogni variabile  
locale abbia un valore prima che venga  
usata

class Esempio {

```
void m() {  
    int a;  
    a++;      Errore a tempo di compilazione  
}
```

void  $n()$  {

```
int q = 10;  
q ++;  
;
```

```
void p() {
```

int ej

三

if ( condition )  
    a = 1;

else

$$\varrho = 2j$$

OK : il capitatore  
vede che in ogni caso  
ad a viene assegnato  
un valore pr

suo uso.

In alcuni casi il compilatore non è così intelligente...

## Parola chiave this

La parola chiave this può essere usata quando abbiamo bisogno di un riferimento all'oggetto implicito

- quando abbiamo passato l'oggetto implicito come argomento di un metodo
- quando abbiamo restituito un riferimento all'oggetto implicito
- come risolutore di ambiguità

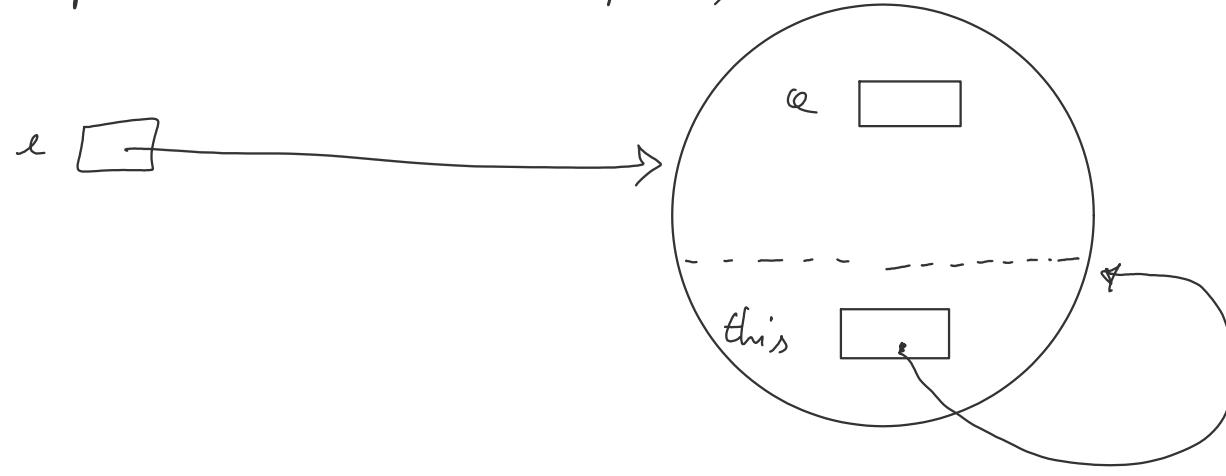
```
class Esempio {  
    int a;  
    void incrementa () {  
        a++;  
    }  
    void m () {  
        incrementa ();  
    }  
}
```

```
class Esempio {  
    int a;  
    void incrementa () {  
        this.a++;  
    }  
    void m () {  
        this.incrementa ();  
    }  
}
```

Analogia al codice precedente  
ma meno leggibile  
e più lungo

Non usare this in questi modi

Esempio `e = new Esempio();`



class Punto {

```
double x;  
double y;
```

Punto (double x, double y) {

this.x = x;  
 this.y = y; }  
 ;

uso `this` come risolutore  
di ambiguità (è la variabile  
istanza `x` e non l'argomento `x`)

class Numero {

```
int n;
```

Numero (int n) {

```
    this.n = n;
```

}

Numero maggiore (Numero x) {

```
if (n > x.n)
```

```
    return this;
```

```

        else
            return x;
    }

void stampa(){
    System.out.println(n);
}

;

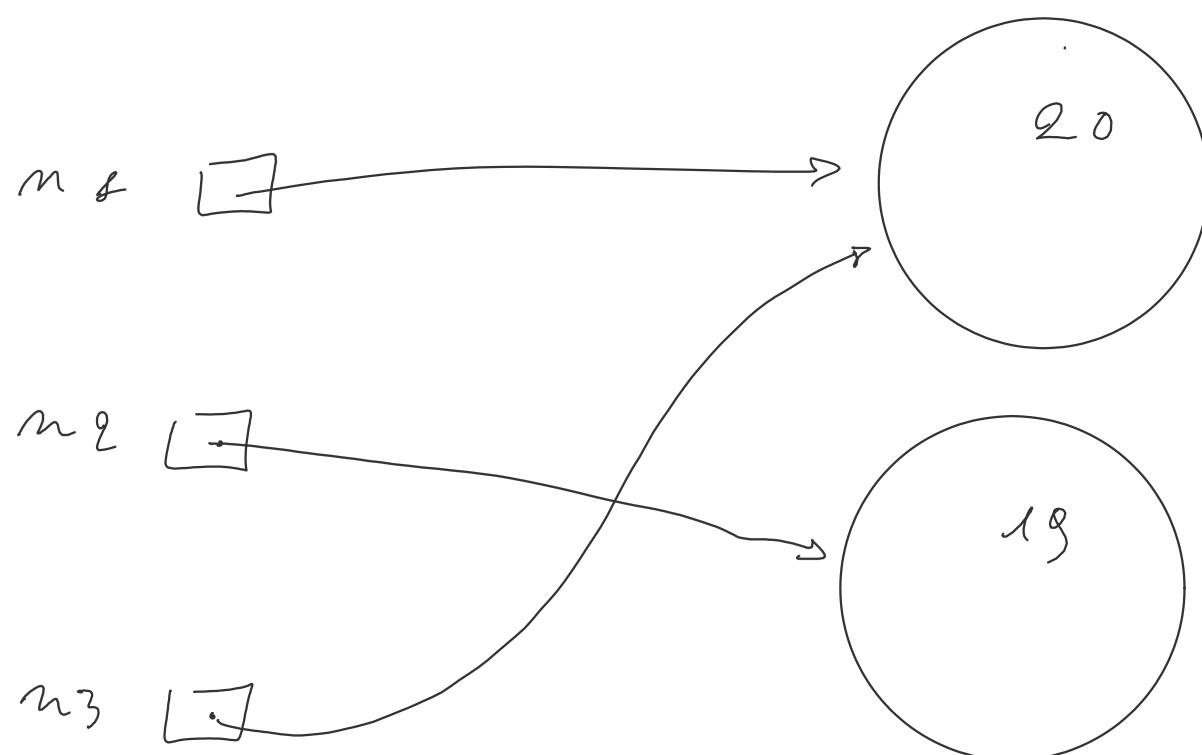
```

```

Numeri n1 = new Numeri(20);
Numeri n2 = new Numeri(19);
Numeri n3;

n3 = n1. maggiore(n2);

```



`n3.stamp()`; stampa 20

Altro uso di this:

evitare duplicazione del codice nel costruttore

```

class ContoBancario {
    double bilancio;
    String intestato;
}

```

```

int numero;

ContoBancario (double bilancio, String intestatario, int numero) {
    this.bilancio = bilancio;
    this.intestatario = intestatario;
    this.numero = numero;
}

ContoBancario (String intestatario, int numero) {
    this.intestatario = intestatario;
    this.numero = numero;
}

};

}

```

} Codice  
duplicato

Il secondo costruttore è meglio scriverlo così

```

ContoBancario (String intestatario, int numero) {
    this(0.0, intestatario, numero);
}

};


```

 richiamo il costruttore con 3 argomenti

La "chiamata a this" deve essere la prima istruzione del costruttore in via di definizione.

Potrò avere altre istruzioni a seguire.

### Campi e metodi statici

A volte è necessario che dei campi di una classe esistano in singola copia (per esempio come "deposito centralizzato" delle informazioni di quella classe).

Potrò usare i campi statici

- singola copia
- condivisi da tutte le istanze della classe (se ce ne sono)

Esempio:

```

class Veicolo {
    static int contatore = 1;
}

```

```

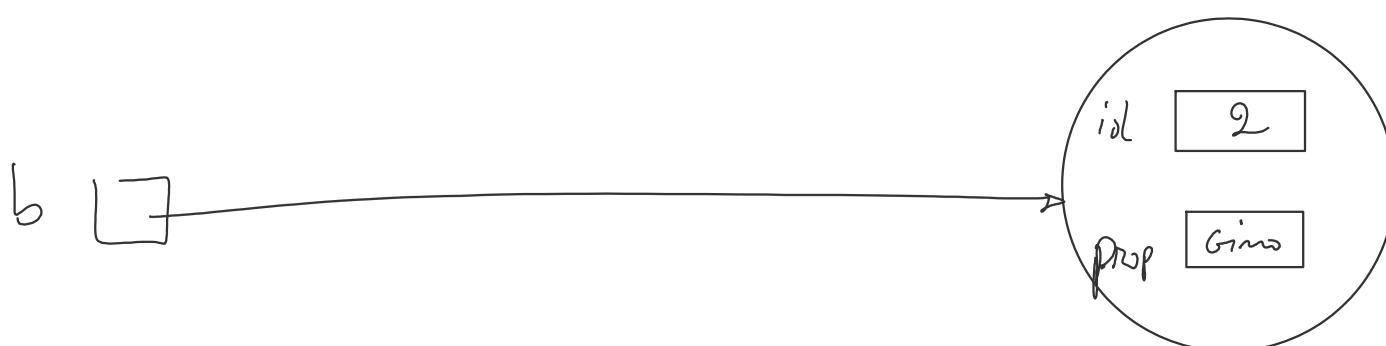
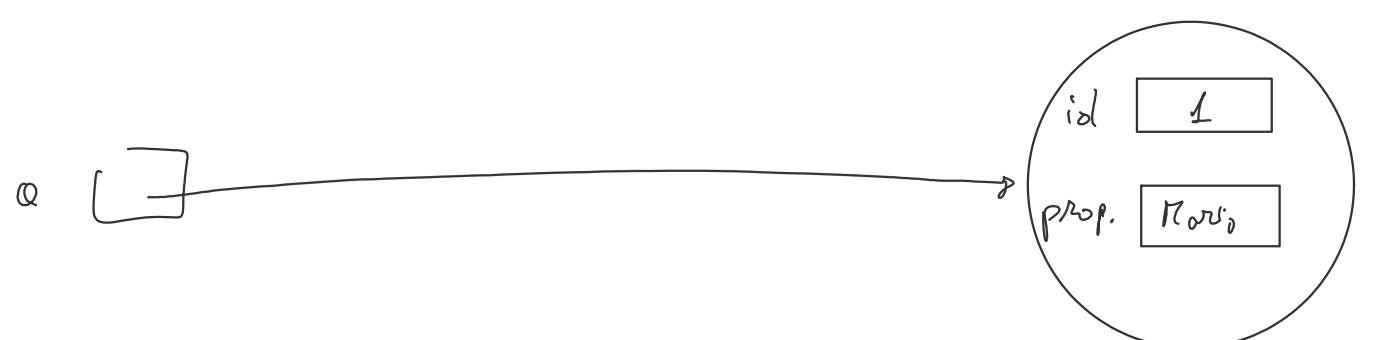
String proprietario;
Veicolo ( String p ) {
    id = contatore++;
    proprietario = p;
}
void stampa () {
    System.out.println( "proprietario = " + proprietario +
        " id = " + id );
}

```

Dentro la classe  
 possiamo usare il  
 nome "semplice" del  
 campo statico

Veicolo a = new Veicolo ("Morì");

Veicolo b = new Veicolo ("Gino");



Contatore 1 2 3

Da fuori la classe :

NameDellaClasse.nomeDelCampoStatico

```

class AltraClasse {
    public static void main ( String [ ] args ) {
        Veicolo v1 = new Veicolo ( "Sera" );
    }
}

```

```
int quanti = Veicolo::contatore - 1;  
int z = v1::contatore;  
}  
}
```

Brutto: non  
si capisce che contatore  
è un campo della  
classe e non di v1

I campi statici vengono elencati quando, nell'esecuzione del programma, viene incontrata per la prima volta la classe in cui sono definiti