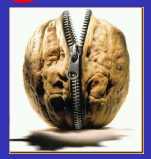
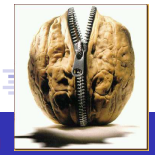
Programmazione
Gli oggetti in un



Eugenio Omodeo

Università degli Studi di Trieste.

Trieste, 20.10.2015

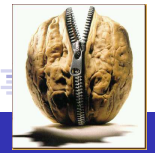


Come è costituito un oggetto

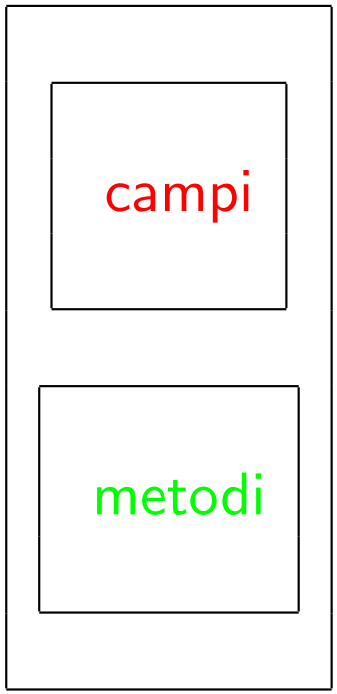
L'oggetto come esemplare di una classe

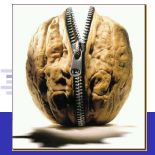
- Campi
- Metodi dinamici
- Stato

Variabili che designano oggetti

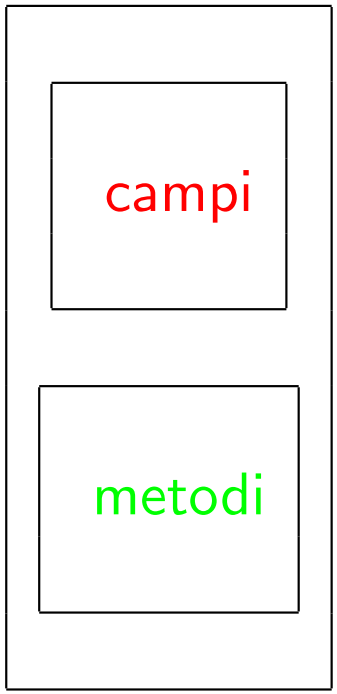


Un oggetto è una mescolanza di dati e comportamenti...

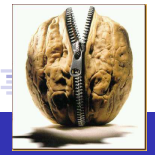




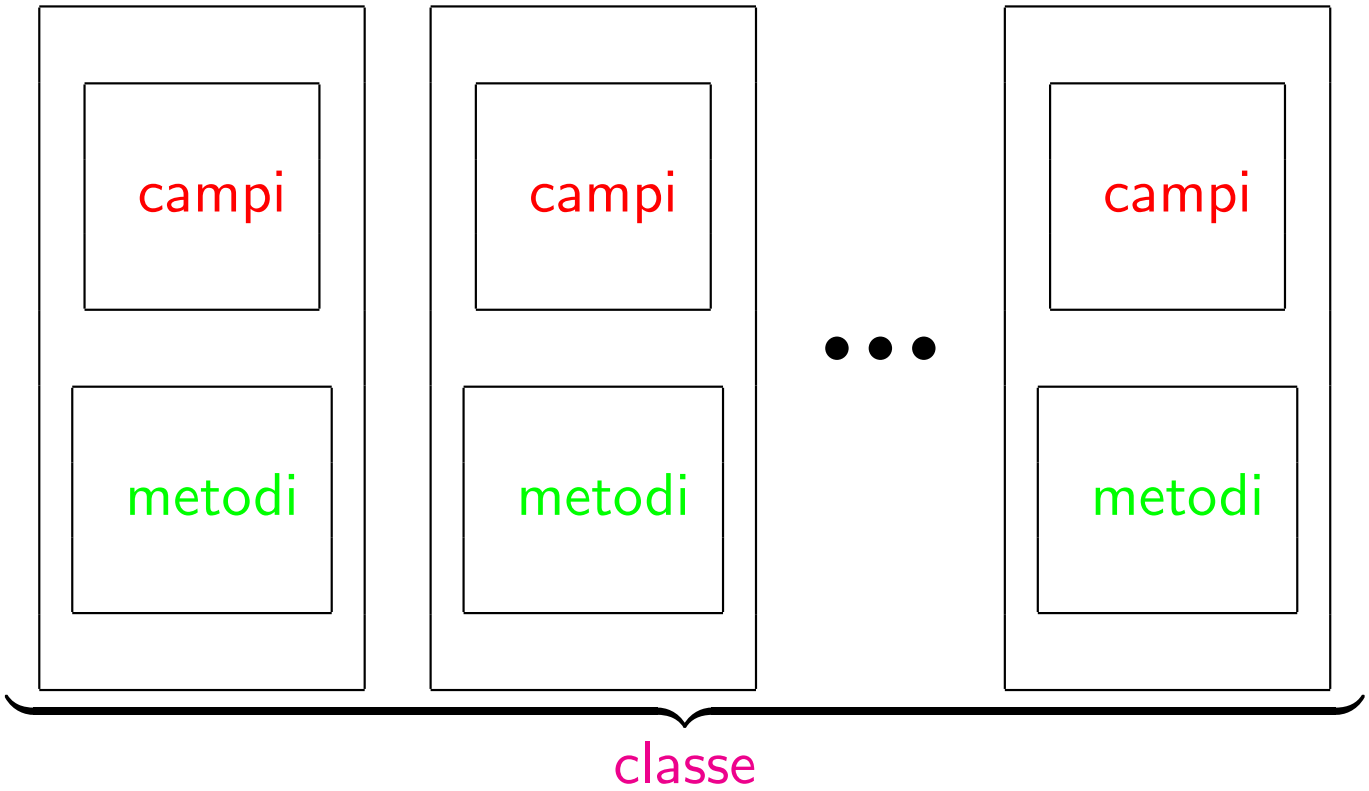
Un oggetto è una mescolanza di dati e comportamenti...

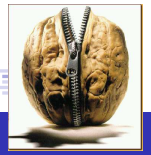


...realizzati rispettivamente dai campi dell'oggetto e dai suoi metodi



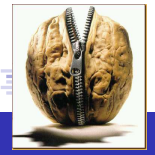
Ogni oggetto è esemplare, "istanza", di una qualche classe





Campi di un oggetto

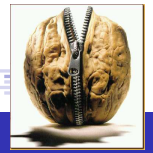
Implementando una classe, dovremo specificare *quali campi* caratterizzano ciascun esemplare della classe.



Campi di un oggetto

Implementando una classe, dovremo specificare *quali campi* caratterizzano ciascun esemplare della classe.

I campi di un oggetto sono perciò chiamati anche sue variabili d'istanza, perché sono dote esclusiva di ciascun esemplare.

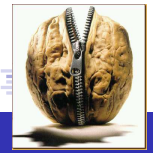


Campi di un oggetto

Implementando una classe, dovremo specificare *quali campi* caratterizzano ciascun esemplare della classe.

È possibile che due oggetti, pur essendo distinti, abbiano tutti i campi uguali. Pertanto:

quasi mai ha senso confrontare due oggetti tramite l' == !

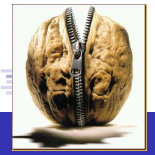


Campi di un oggetto – 1^o esempio

Potremmo, nel definire una classe

Rettangolo ,

attribuire agli oggetti suoi esemplari quattro campi:



Campi di un oggetto – 1^o esempio

Potremmo, nel definire una classe

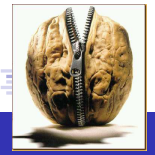
Rettangolo ,

attribuire agli oggetti suoi esemplari quattro campi:

x , y coordinate di un suo prestabilito vertice

altezza

larghezza



Campi di un oggetto – 1^o esempio

Potremmo, nel definire una classe

Rettangolo ,

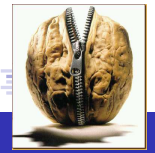
attribuire agli oggetti suoi esemplari quattro campi:

x , y coordinate di un suo prestabilito vertice

altezza

larghezza

Perché questi e non altri ? Sta al programmatore scegliere



Campi di un oggetto – 1^o esempio

Potremmo, nel definire una classe

Rettangolo ,

attribuire agli oggetti suoi esemplari quattro campi:

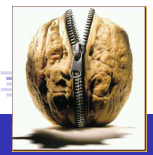
x , y coordinate di un suo prestabilito vertice

altezza

larghezza

Perché questi e non altri ? Sta al programmatore scegliere

(Avete una proposta alternativa?)

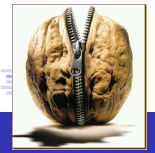


Campi di un oggetto – 2^o esempio

Potremmo, nel definire una classe

Frazione ,

attribuire agli oggetti suoi esemplari due campi:



Campi di un oggetto – 2^o esempio

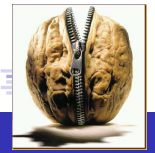
Potremmo, nel definire una classe

Frazione ,

attribuire agli oggetti suoi esemplari due campi:

numeratore l'intero *da* dividere

denominatore l'intero *per cui* dividere



Campi di un oggetto – 2^o esempio

Potremmo, nel definire una classe

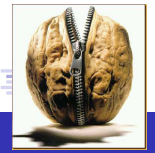
Frazione ,

attribuire agli oggetti suoi esemplari due campi:

numeratore l'intero *da* dividere

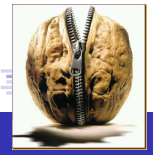
denominatore l'intero *per cui* dividere

Ridotta ai minimi termini ? Sta al programmatore decidere



Visibilità dei campi

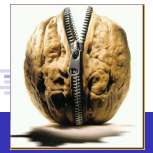
È bene che i campi siano *privati*, altrimenti potrebbero venir modificati dall'esterno.



Visibilità dei campi

È bene che i campi siano *privati*, altrimenti potrebbero venir modificati dall'esterno.

Fanno eccezione a questa regola i campi dichiarati `final`, visto che una volta inizializzati non possono piú cambiar valore.

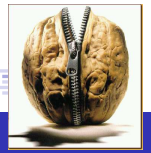


Visibilità dei campi

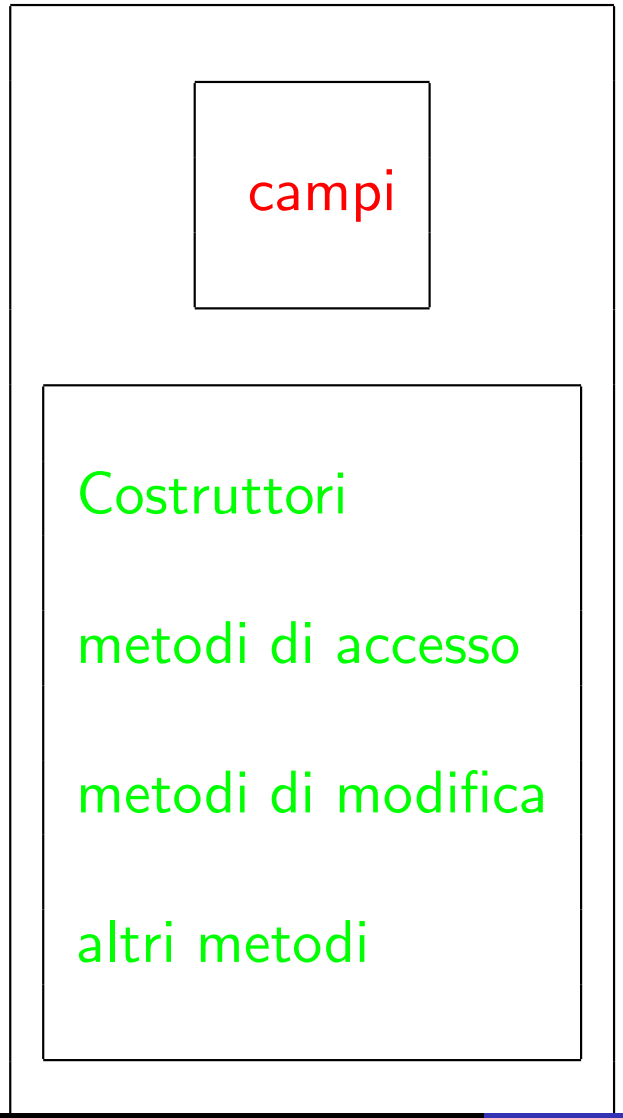
È bene che i campi siano *privati*, altrimenti potrebbero venir modificati dall'esterno.

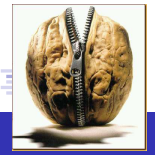
Fanno eccezione a questa regola i campi dichiarati `final`, visto che una volta inizializzati non possono piú cambiar valore.

Esempio: Il campo `length` di qualunque `array`.



Sommaria classificazione dei metodi



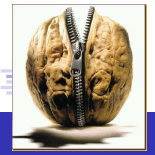


Cosa specificano i metodi non-statici ?

I metodi *non-statici* di una classe definiscono

- fisionomia
- comportamento

che accomunano gli oggetti esemplari di quella classe



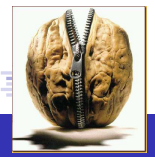
Cosa specificano i metodi non-statici ?

I metodi *non-statici* di una classe definiscono

- fisionomia
- comportamento

che accomunano gli oggetti esemplari di quella classe

Questo discorso comprende anche i *costruttori*; perché se è vero che un costruttore non può riferirsi a un oggetto che *non c'è* (essendo, al momento, in costruzione), è pur vero che i campi dell'oggetto vengono *allocati* subito, non appena il costruttore viene invocato (con la `new`).



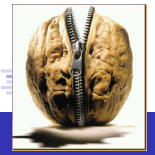
Metodi non-statici: Esempio (Di nuovo le frazioni)

Nel caso della

Frazione ,

il costruttore può

- effettuare la riduzione ai minimi termini ed
- attribuire l'eventuale segno negativo al solo numeratore.

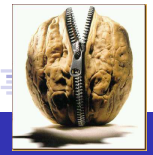


Metodi non-statici: Esempio (Di nuovo le frazioni)

Nel caso della

Frazione ,

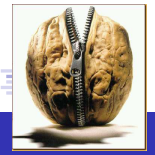
metodi di somma, prodotto, inversione stabiliranno come effettuare queste operazioni e garantiranno—se cosí è convenuto—il mantenimento dei 'minimi termini'.



Stato di un oggetto

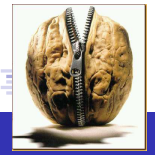
Talvolta i metodi dinamici di un oggetto consentono di modificare 'dall'interno' i campi di un oggetto.

Modificano —cosí usa dire— lo stato dell'oggetto.



Accesso ai dati di un oggetto

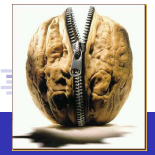
Talvolta i metodi dinamici consentono semplicemente di acquisire dall'esterno informazioni sul suo stato.



Accesso ai dati di un oggetto

Talvolta i metodi dinamici consentono semplicemente di acquisire dall'esterno informazioni sul suo stato.

Ciò è utile, in quanto i campi sono per lo piú privati.



Accesso ai dati di un oggetto

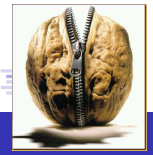
Talvolta i metodi dinamici consentono semplicemente di acquisire dall'esterno informazioni sul suo stato.

Ciò è utile, in quanto i campi sono per lo piú privati.

Si pensi, ad es., al metodo

`lenght( )`

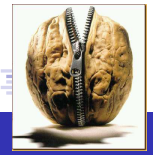
della classe `String`



Un utilizzo sconsigliato dei campi

I campi:

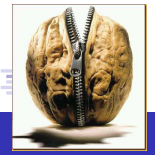
- rappresentano lo stato: possono essere attributi primari dell'oggetto (ad es. la larghezza di un rettangolo) o—in qualche caso—attributi secondari (quali ad es. perimetro o area);



Un utilizzo sconsigliato dei campi

I campi:

- rappresentano lo stato: possono essere attributi primari dell'oggetto (ad es. la larghezza di un rettangolo) o—in qualche caso—attributi secondari (quali ad es. perimetro o area);
- *non* vanno utilizzati come espediente per far condividere informazioni accessorie ai metodi: per questo fine, lo strumento appropriato è il passaggio di parametri.

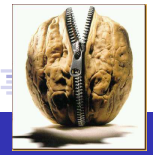


Valori per le variabili di una certa classe

Una variabile, `xxx`, dichiarata come

```
Tipo xxx
```

(dove `Tipo` è una certa classe) può assumere come valori



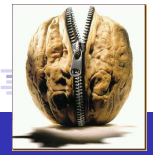
Valori per le variabili di una certa classe

Una variabile, `xxx`, dichiarata come

```
Tipo xxx
```

(dove `Tipo` è una certa classe) può assumere come valori

- gli oggetti di classe `Tipo`



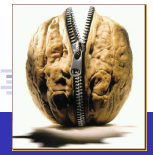
Valori per le variabili di una certa classe

Una variabile, `xxx`, dichiarata come

`Tipo xxx`

(dove `Tipo` è una certa classe) può assumere come valori

- il valore fittizio *null*—l'indefinito



Valori per le variabili di una certa classe

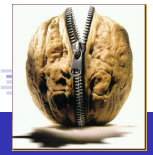
Una variabile, `xxx`, dichiarata come

`Tipo xxx`

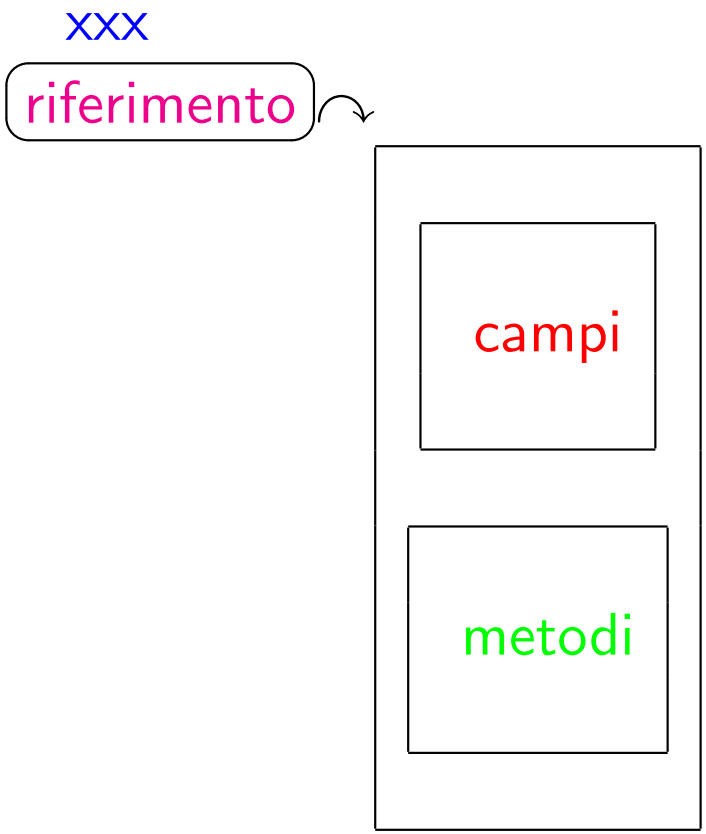
(dove `Tipo` è una certa classe) può assumere come valori

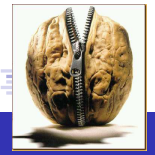
- gli oggetti di classe `Tipo`
- il valore fittizio *null*—l'indefinito

Ma allora una var. di tipo *Boolean* ammette tre possibili valori ?

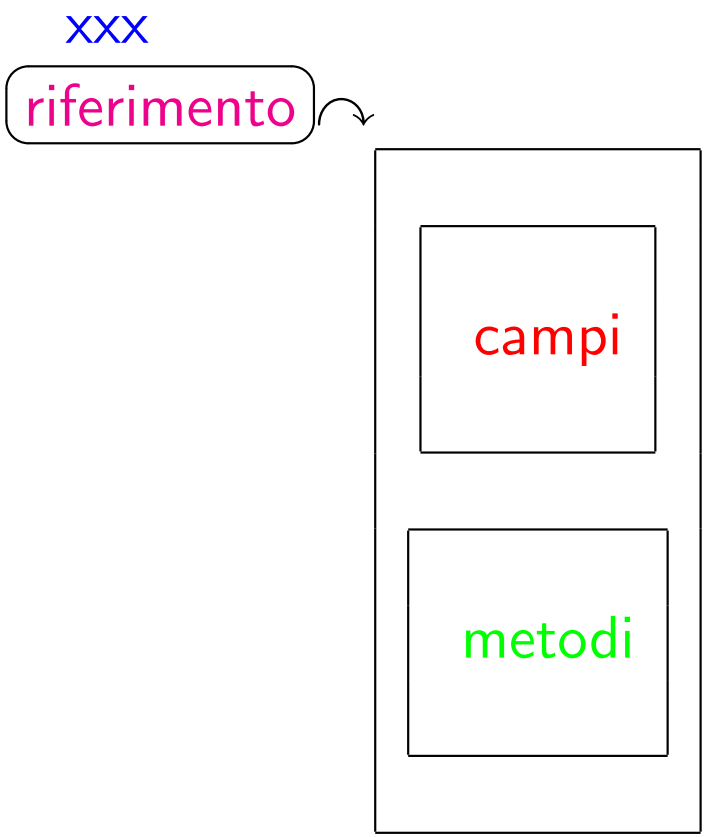


Un oggetto viene *riferito* (non posseduto come valore)

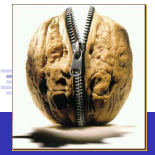




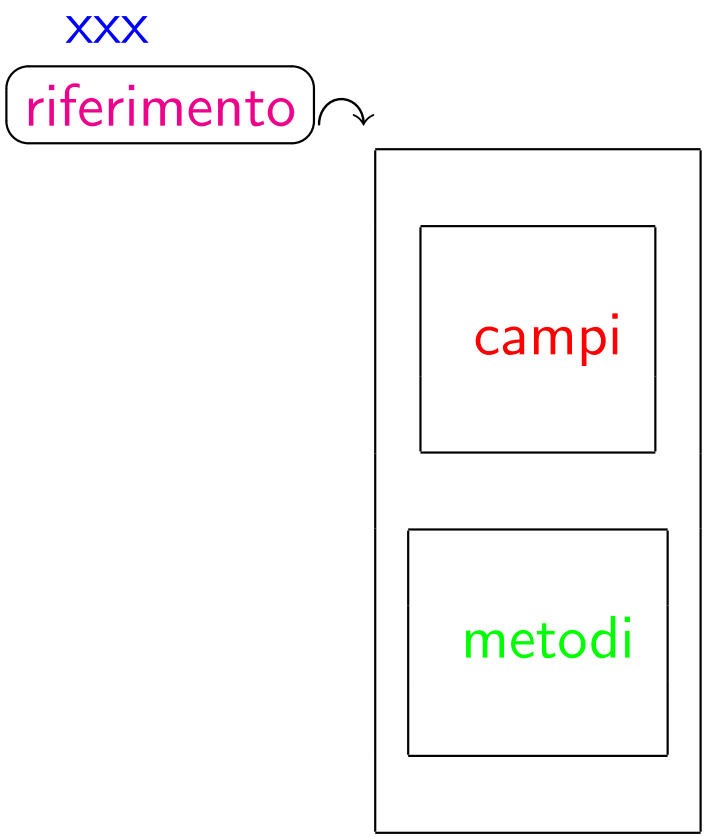
Un oggetto viene *riferito* (non posseduto come valore)



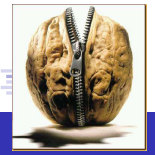
Questa la situazione che
si instaura con una
`Tipo` *xxx* = **new** *Tipo*(···);



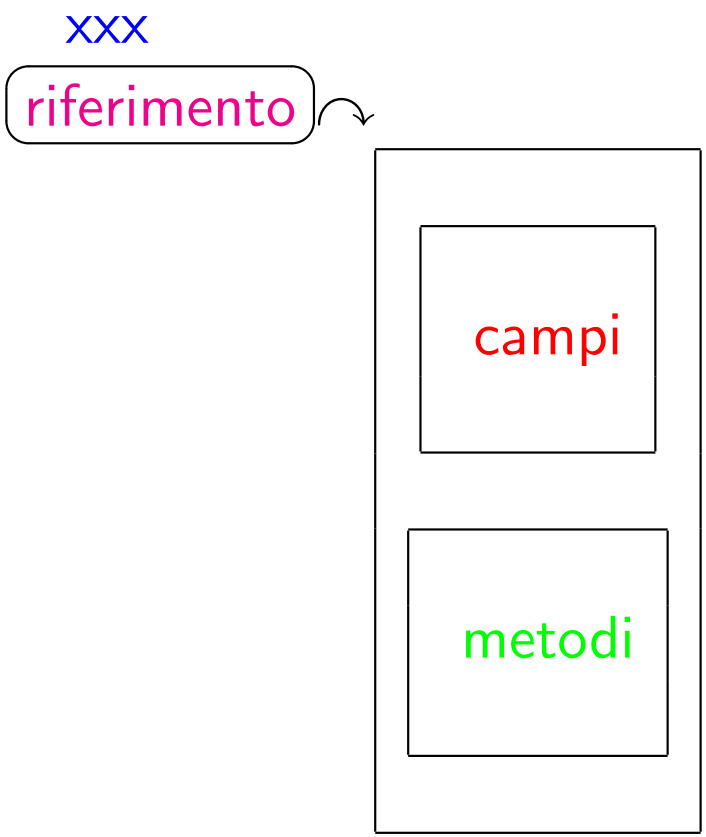
Un oggetto viene *riferito* (non posseduto come valore)



Stessa situazione quando,
in un'invocazione,
xxx funge da
parametro formale.

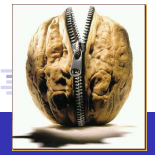


Un oggetto viene *riferito* (non posseduto come valore)



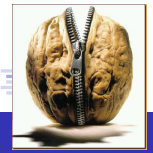
Questa la situazione che
si instaura con una
`Tipo` *xxx* = **new** *Tipo*(···);

Stessa situazione quando,
in un'invocazione,
xxx funge da
parametro formale.



Potere dei riferimenti

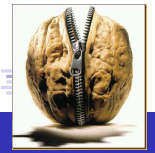
Copiando un oggetto in un parametro formale, all'invocazione di un metodo, si conferisce al metodo la possibilità di modificare lo stato di quell'oggetto.



Potere dei riferimenti

Copiando un oggetto in un parametro formale, all'invocazione di un metodo, si conferisce al metodo la possibilità di modificare lo stato di quell'oggetto.

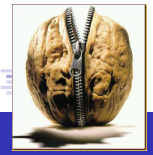
Qualora, però, il metodo poi cambi il valore del parametro formale, con ciò rinuncia a tale facoltà.



Potere dei riferimenti

Copiando un oggetto in un parametro formale, all'invocazione di un metodo, si conferisce al metodo la possibilità di modificare lo stato di quell'oggetto.

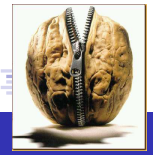
Un metodo potrà dunque restituire modificato il contenuto (non la lunghezza) di un **array** ricevuto tramite parametro;



Potere dei riferimenti

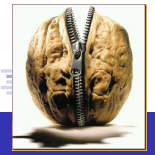
Copiando un oggetto in un parametro formale, all'invocazione di un metodo, si conferisce al metodo la possibilità di modificare lo stato di quell'oggetto.

Un metodo potrà dunque restituire modificato il contenuto (non la lunghezza) di un **array** ricevuto tramite parametro; non potrà, invece, in alcun modo modificare una `String` né un `Integer`



Una radicale differenza fra `String` ed **array**

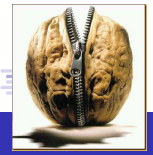
La classe `String` non offre alcun metodo di modifica.



Una radicale differenza fra `String` ed **array**

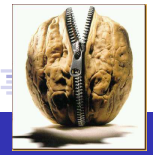
La classe `String` non offre alcun metodo di modifica.

Per gli **array** è prevista anche la modifica dello stato.
Consideriamo:



Una radicale differenza fra `String` ed **array**

- `String a = "Ciao";`
- `char[ ] b = new char[4];`
  `b[0] = 'C'; b[1] = 'i'; b[2] = 'a'; b[3] = 'o';`
- `char x, y;`

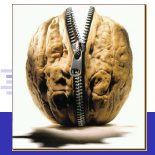


Una radicale differenza fra `String` ed **array**

- `String a = "Ciao";`
- `char[ ] b = new char[4];`
  `b[0] = 'C'; b[1] = 'i'; b[2] = 'a'; b[3] = 'o';`
- `char x, y;`

Accesso al primo carattere di b e di a:

x = b[0]; y = a.charAt(0);

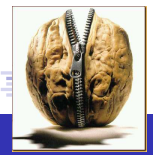


Una radicale differenza fra `String` ed **array**

- `String a = "Ciao";`
- `char[ ] b = new char[4];`
  `b[0] = 'C'; b[1] = 'i'; b[2] = 'a'; b[3] = 'o';`
- `char x, y;`

Modifica del primo carattere di b e di a:

b[0] = 'c'; a = 'c' + a.substring(1);



Una radicale differenza fra `String` ed **array**

- `String a = "Ciao";`
- `char[ ] b = new char[4];`
  `b[0] = 'C'; b[1] = 'i'; b[2] = 'a'; b[3] = 'o';`
- `char x, y;`

Accesso al primo carattere di b e di a:

$$x = b[0]; \quad y = a.charAt(0);$$

Modifica del primo carattere di b e di a:

$$b[0] = 'c'; \quad a = 'c' + a.substring(1);$$

Si tratta di due meccanismi ben diversi: in che senso ?