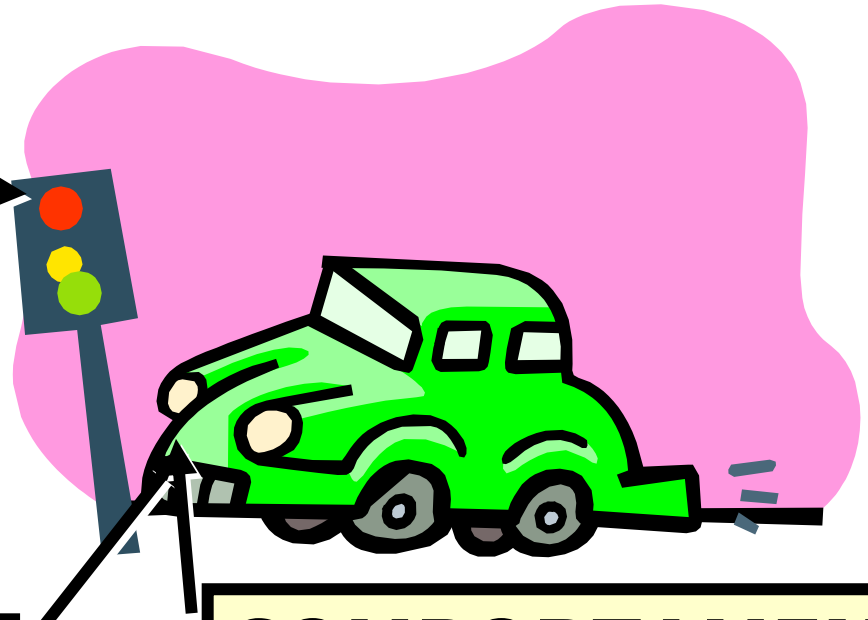
Usare gli Oggetti

- Modellare la realtà
- Le classi: il modello
  - Attributi e metodi
- Gli oggetti: il mattone
  - Costruzione e accesso
  - Riferimenti
  - Ciclo di vita degli oggetti

Modellare la realtà (1)

STATO
via1: verde
via2: rosso



STATO
motore:acceso
velocità: 0

COMPORTAMENTO
Parti!
Frena!
Sterza!

Modellare la realtà (2)

- Stato
  - L'insieme dei parametri caratteristici che contraddistinguono un oggetto in un dato istante
  - Modellato come insieme di attributi
- Comportamento
  - Descrive come si modifica lo stato a fronte degli stimoli provenienti dal mondo esterno
  - Modellato come insieme di metodi

Classi

- La classe costituisce il “progetto” di un oggetto
  - Specifica gli attributi
  - Descrive i metodi
  - Indica lo stato iniziale
- Ogni classe ha un nome
  - Deve essere univoco

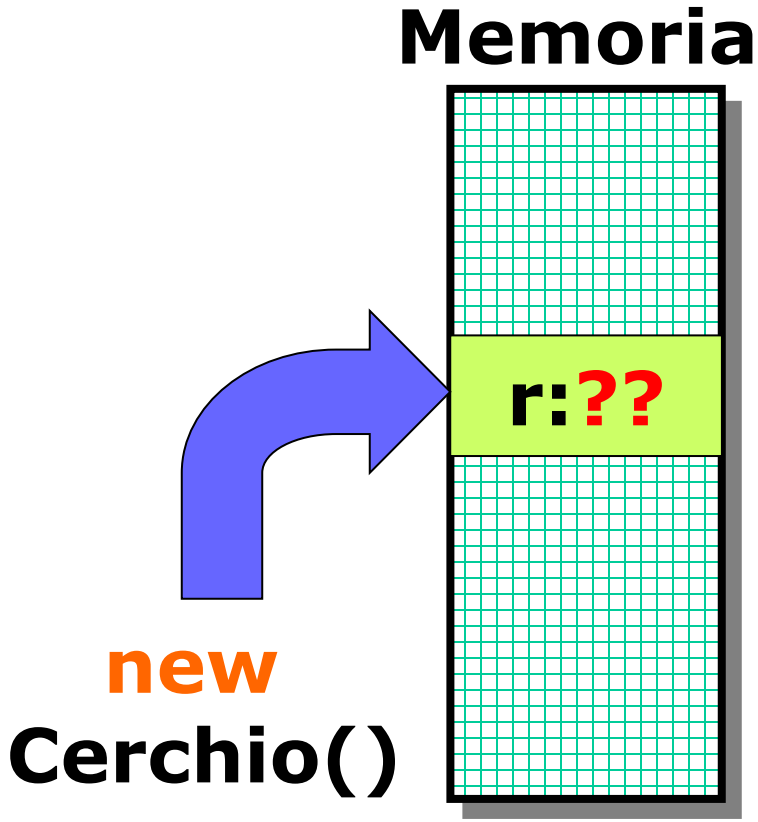
Oggetti

- Data una classe, è possibile costruire uno o più oggetti
  - Gli oggetti vengono detti “istanze” della classe
  - In Java si utilizza la notazione
    new *NomeClasse* ( );
- Ogni oggetto “vive” all’interno della memoria del calcolatore
  - Qui viene memorizzato il suo stato
  - Oggetti differenti occupano posizioni differenti

Oggetti

Cerchio
r: double
…

```
class Cerchio {

  double r;

  …

}
```



Stato di un oggetto

- Ogni oggetto ha un proprio stato:
  - Insieme dei valori assunti dagli attributi dell'oggetto
  - Operando sui metodi, si può modificare lo stato
- All'atto della costruzione di un oggetto, occorre assegnare un valore ai diversi attributi
  - È il compito di un metodo particolare, detto **costruttore**

Costruttore

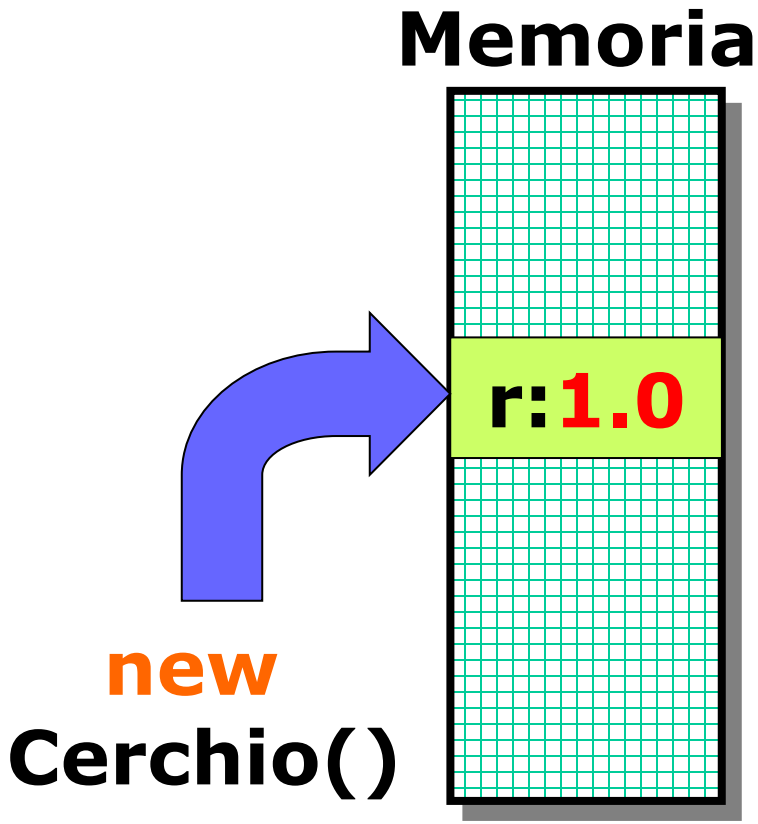
- Metodo che inizializza gli attributi di una classe
- Sintassi particolare:
  - Ha lo stesso nome della classe
  - Non indica nessun tipo ritornato

```
class Cerchio {
  double r;
  Cerchio() {
    r=1.0;
  }
}
```

Costruire oggetti

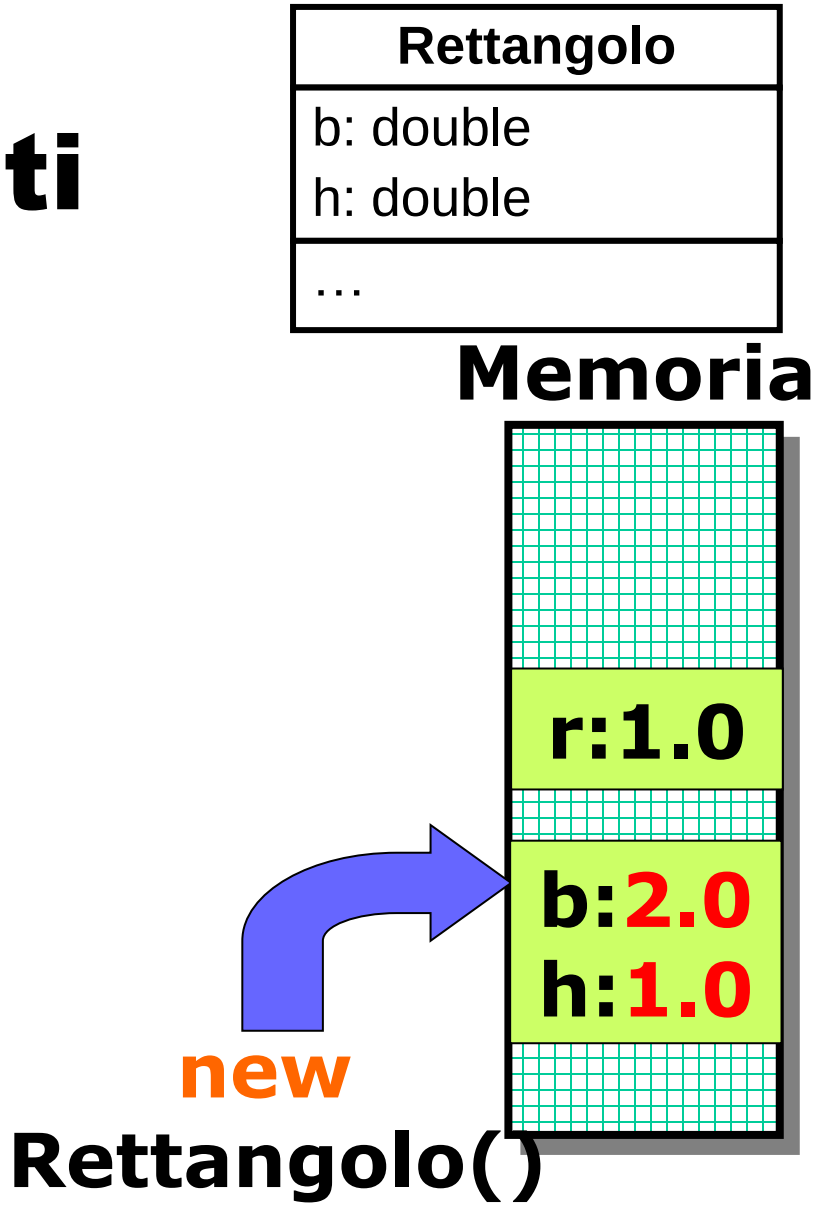
Cerchio
r: double
…

```
class Cerchio {
  double r;

  Cerchio() {
    r=1.0;
  }
  …
}
```



Costruire oggetti

```
class Rettangolo {
  double b,h;

  Rettangolo() {
    b=2.0;
    h=1.0;
  } …
}
```



Costruttore e parametri

- Normalmente un costruttore assegna valori “standard” agli attributi
- Se ha dei parametri, può differenziare gli oggetti costruiti
  - Chi invoca il costruttore deve fornire i parametri richiesti
- Una classe può avere molti costruttori
  - Occorre che siano distinguibili attraverso il numero ed il tipo di parametri richiesti

Costruttore e parametri

```
class Poligono {
  double lato;
  int numeroLati;
  Poligono(int n) {
     numeroLati=n;
     lato=1.0;
  }
}
```

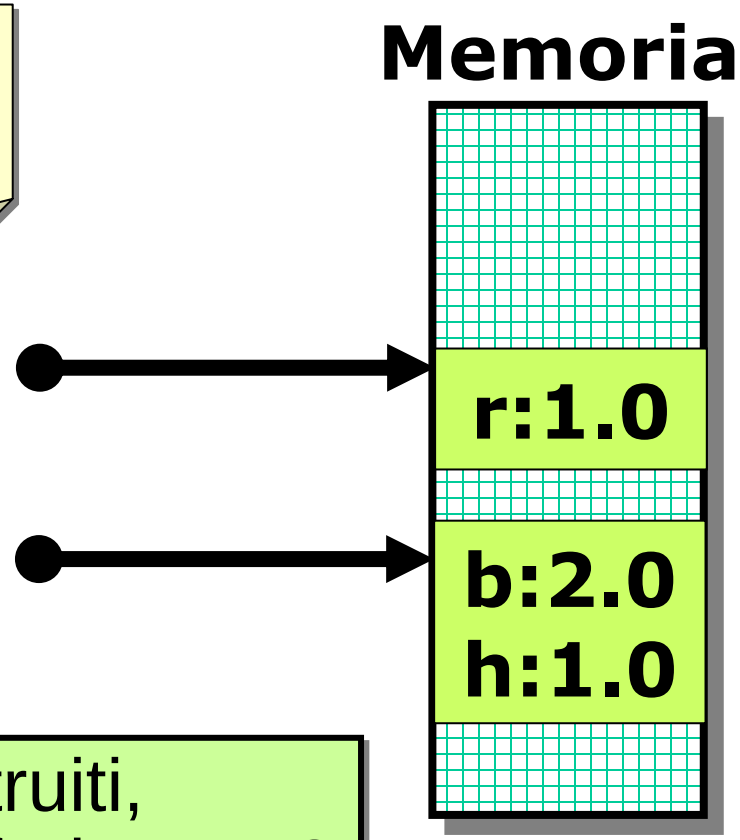
```
Poligono p;

p= new Poligono(3);
```

Riferimenti

- Si opera su un oggetto attraverso un riferimento
  - Indica la posizione in memoria occupata dall'oggetto
- All'atto della costruzione, l'operatore **new**:
  - Alloca un **blocco** di memoria sufficiente a contenere l'oggetto
  - Invoca il **costruttore**, determinandone la corretta inizializzazione
  - Restituisce il **riferimento** (indirizzo) del blocco inizializzato

Riferimenti

```
new Cerchio();
new Rettangolo();
```



Riferimento al cerchio

Riferimento al rettangolo

Gli oggetti vengono costruiti,
ma dove finisce il loro riferimento?

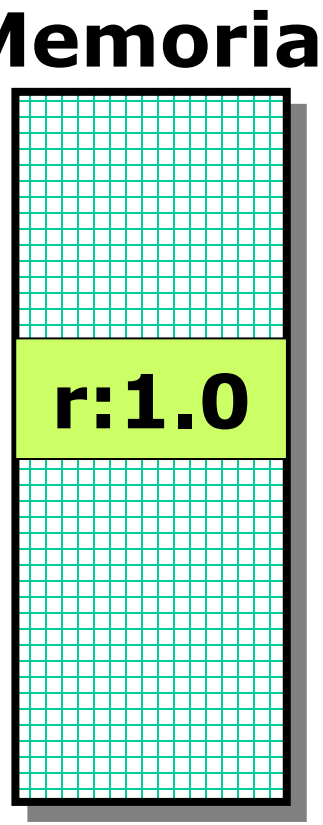
Variabili

- I riferimenti possono essere memorizzati all'interno di **variabili locali**
  - Devono avere un tipo compatibile con il tipo di riferimento che si intende memorizzare al loro interno
  - Devono essere dichiarate prima di essere usate

```
{…
  Cerchio c;
  c= new Cerchio();
…}
```

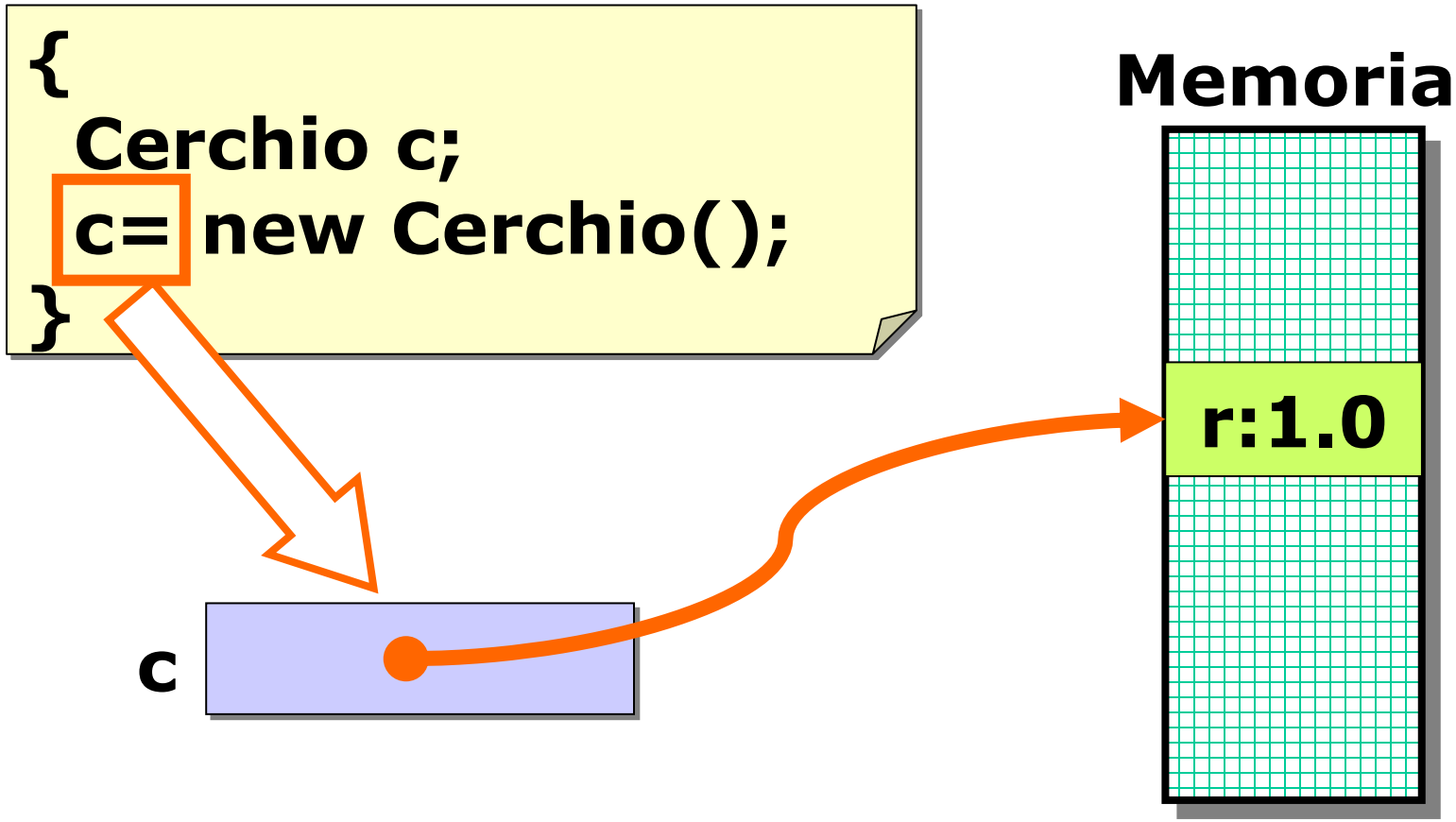
Variabili

```
{
  Cerchio c;
  c= new Cerchio();
}
```



c

Variabili

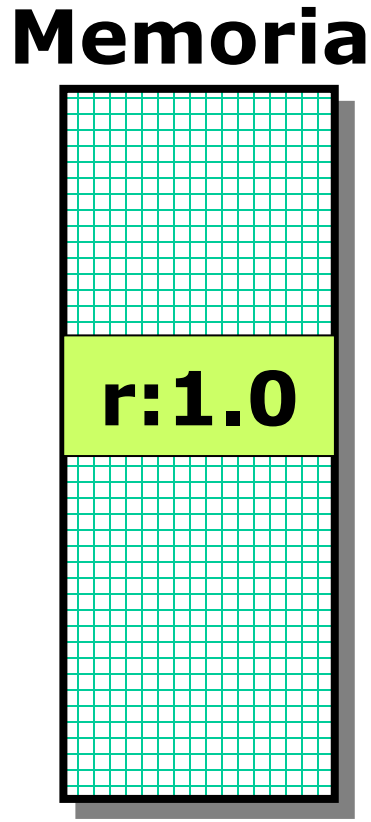


Ciclo di vita delle variabili locali

- Le variabili locali “esistono” finché il metodo (blocco di codice) che le definisce è in esecuzione
  - Quando si incontra la loro definizione, il sistema riserva un’area di memoria per ospitarne il contenuto
  - Quando il metodo (blocco) termina, l’area viene rilasciata ed il contenuto della variabile perso
  - La memoria viene prelevata da una zona apposita detta “stack” (quella in cui si trovano gli oggetti, invece, si chiama “heap”)

Ciclo di vita delle variabili locali

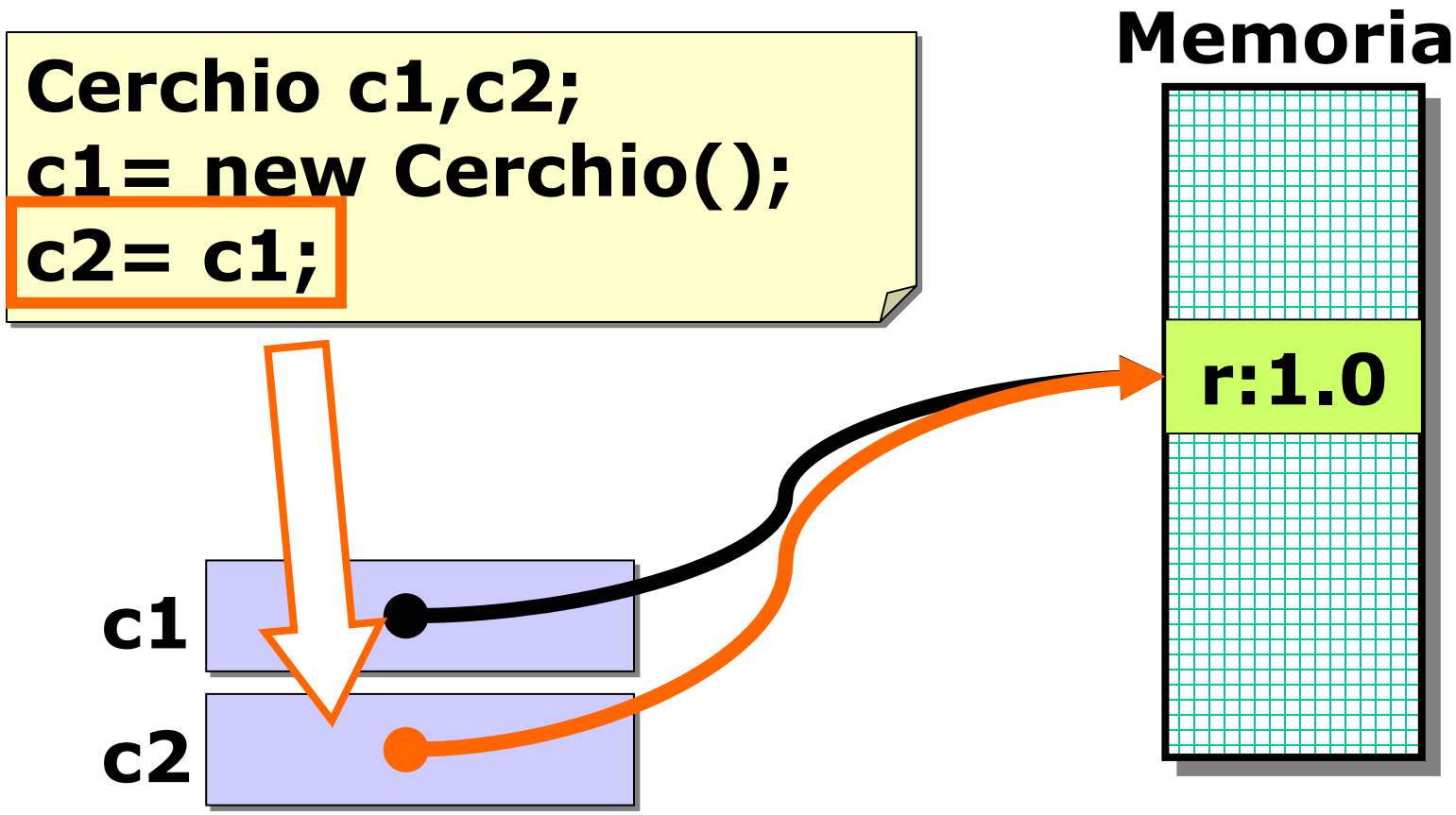
```
{
  Cerchio c;
  c= new Cerchio();
}
```



Riferimenti multipli

- Uno stesso oggetto può essere denotato da più variabili
  - Succede quando si assegna ad un variabile il valore contenuto in **un'altra** variabile
  - Le variabili condividono il riferimento allo **stesso** oggetto
- Se si opera sull'oggetto attraverso la prima variabile, le modifiche sono visibili da tutte le altre variabili coinvolte

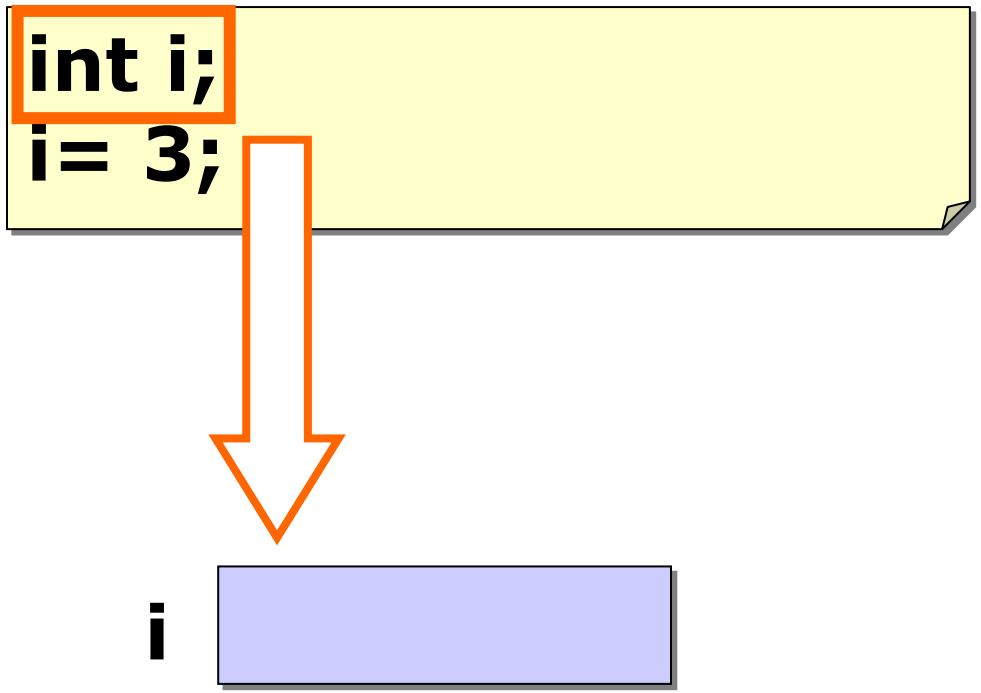
Riferimenti multipli

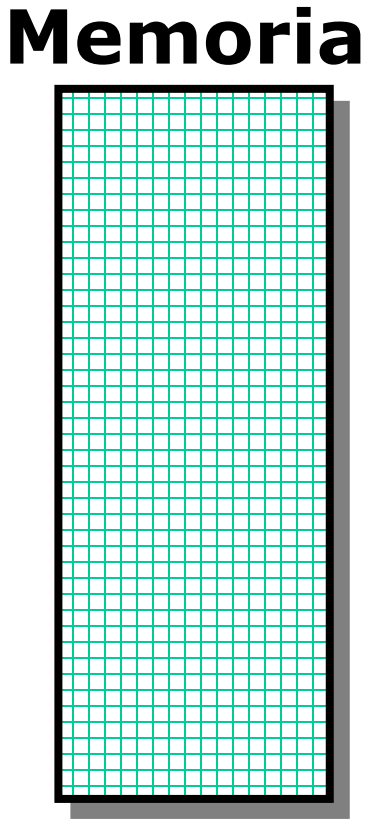


Variabili elementari

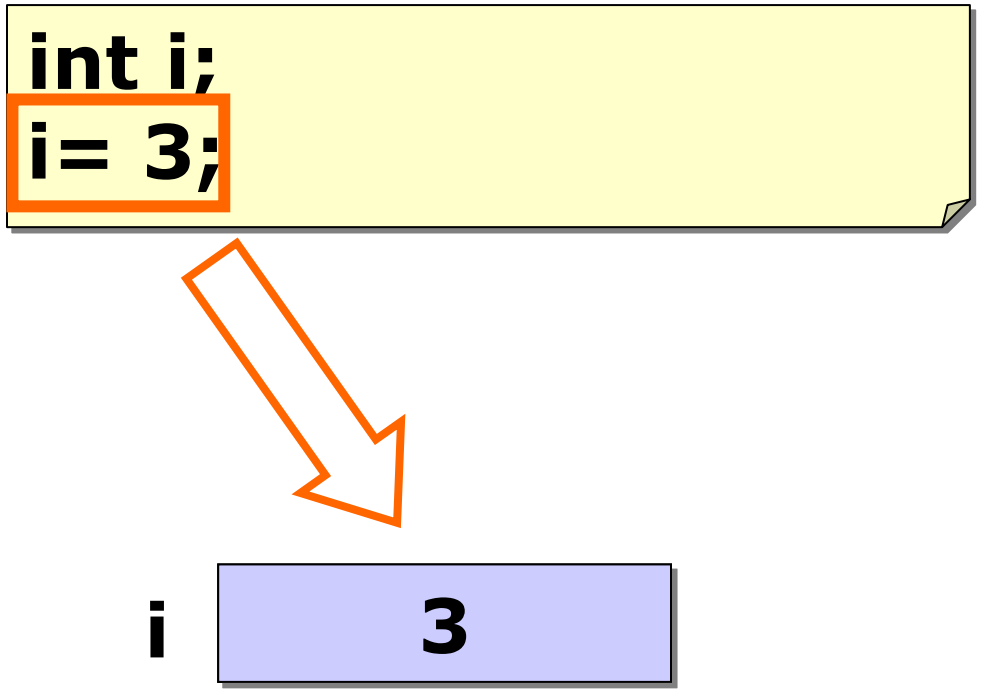
- Non tutte le variabili contengono un riferimento
  - Le informazioni più semplici possono essere memorizzate direttamente nella variabile
  - È il caso delle variabili il cui tipo è elementare (detto anche primitivo)

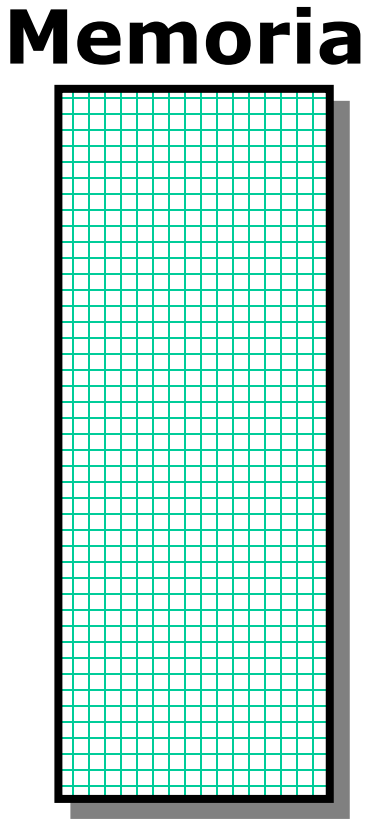
Variabili elementari





Variabili elementari

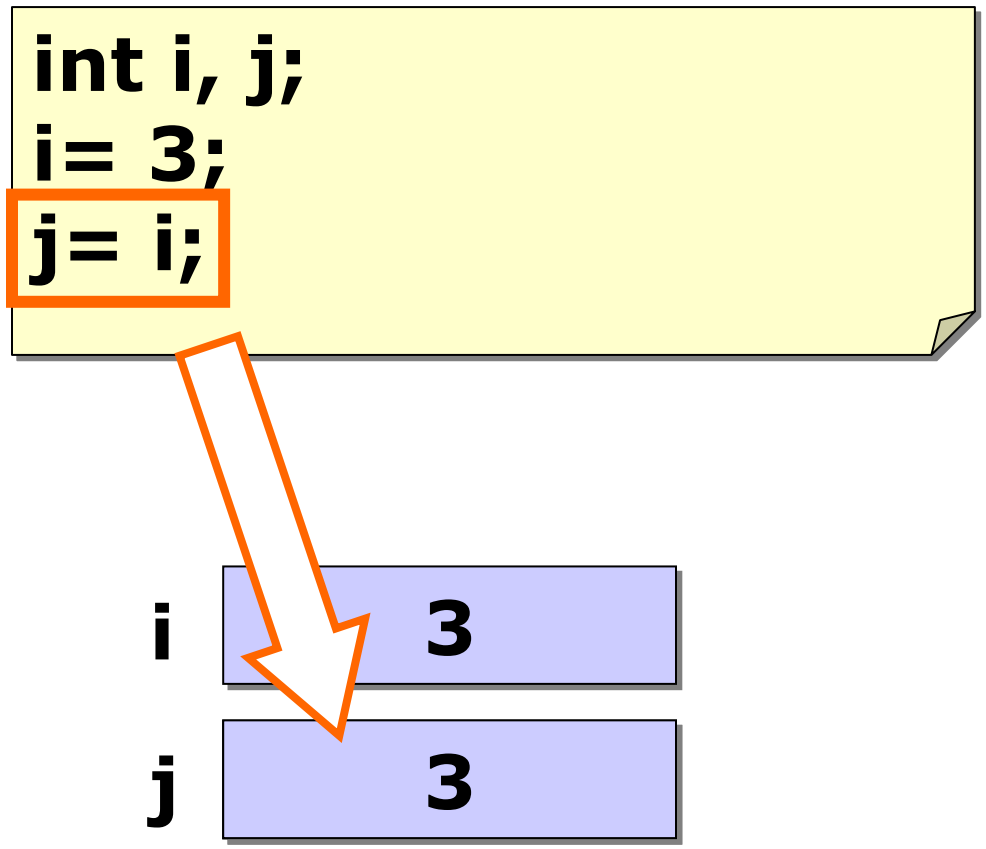


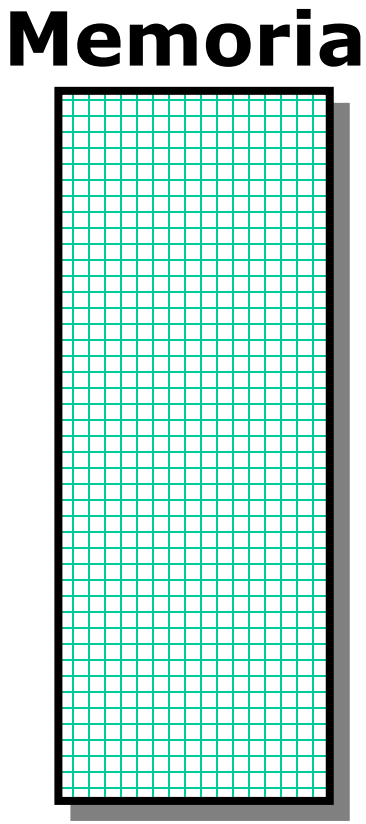


Copia di variabili elementari

- Se si assegna ad una variabile elementare il valore di un'altra variabile viene eseguita una **copia** del valore
  - I due valori diventano **disgiunti** e potranno evolvere indipendentemente

Variabili elementari

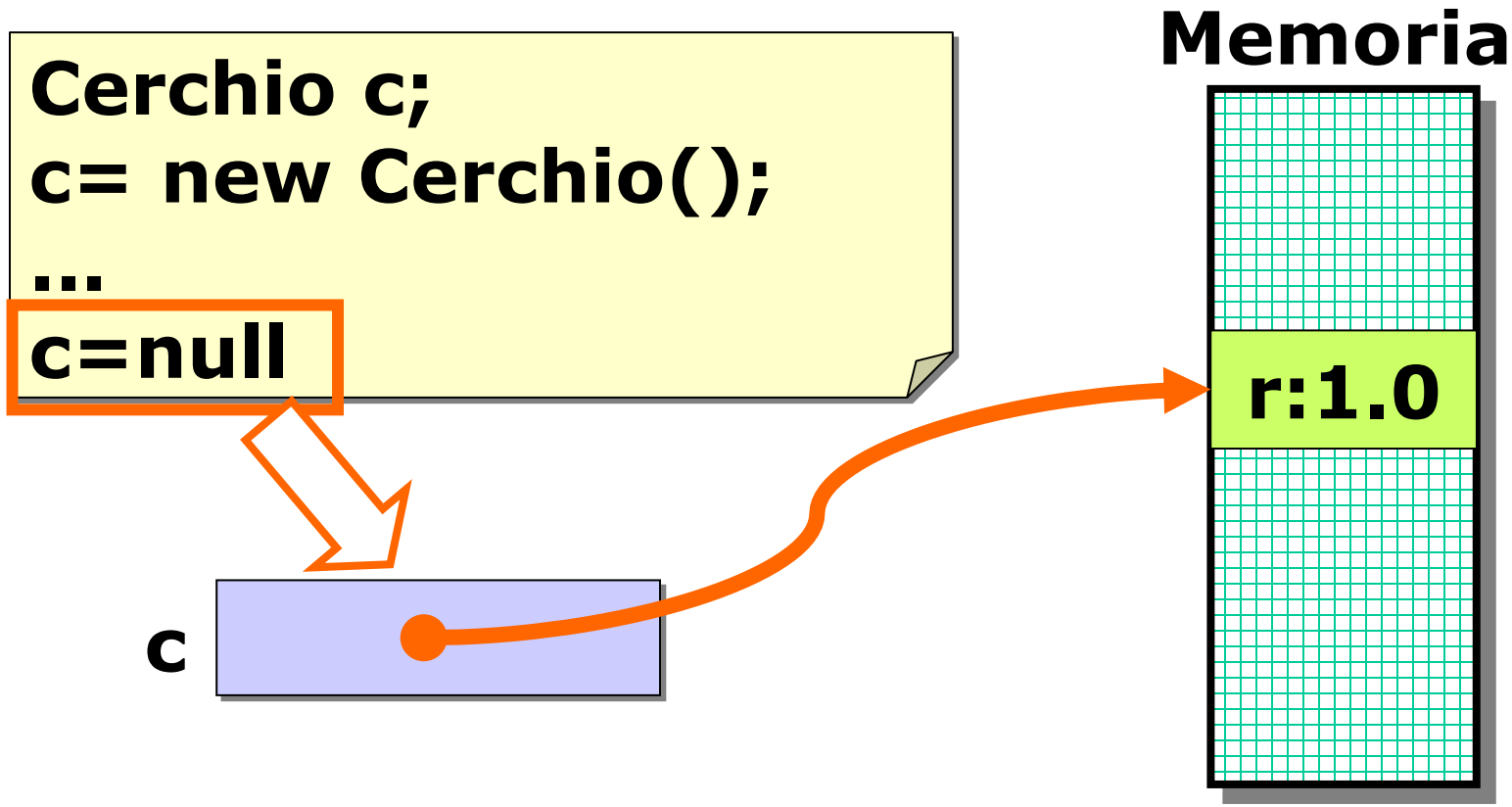




Riferimenti nulli

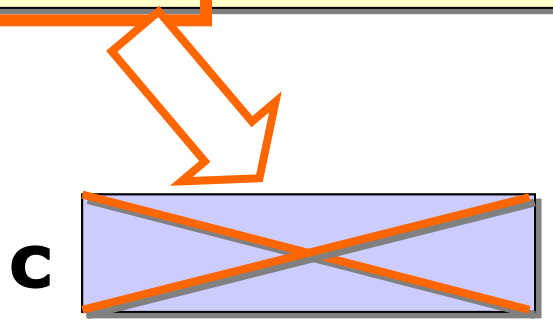
- Nel caso di variabili di tipo classe, a volte occorre indicare che non contengono nessun valore
  - Si usa la parola chiave **null**
- Se una variabile vale null, non è possibile accedere ai metodi o agli attributi

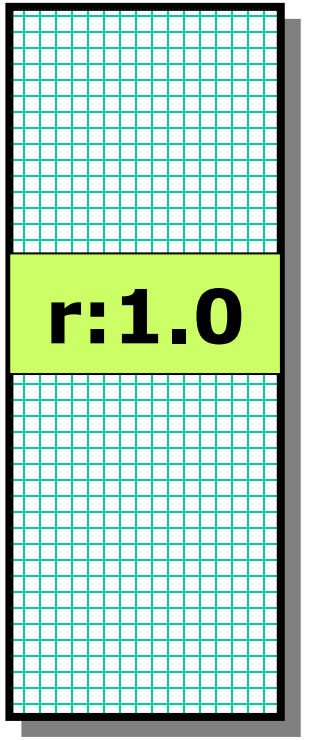
Riferimenti nulli



Riferimenti nulli

```
Cerchio c;
c= new Cerchio();
...
c=null
```





Accedere agli oggetti

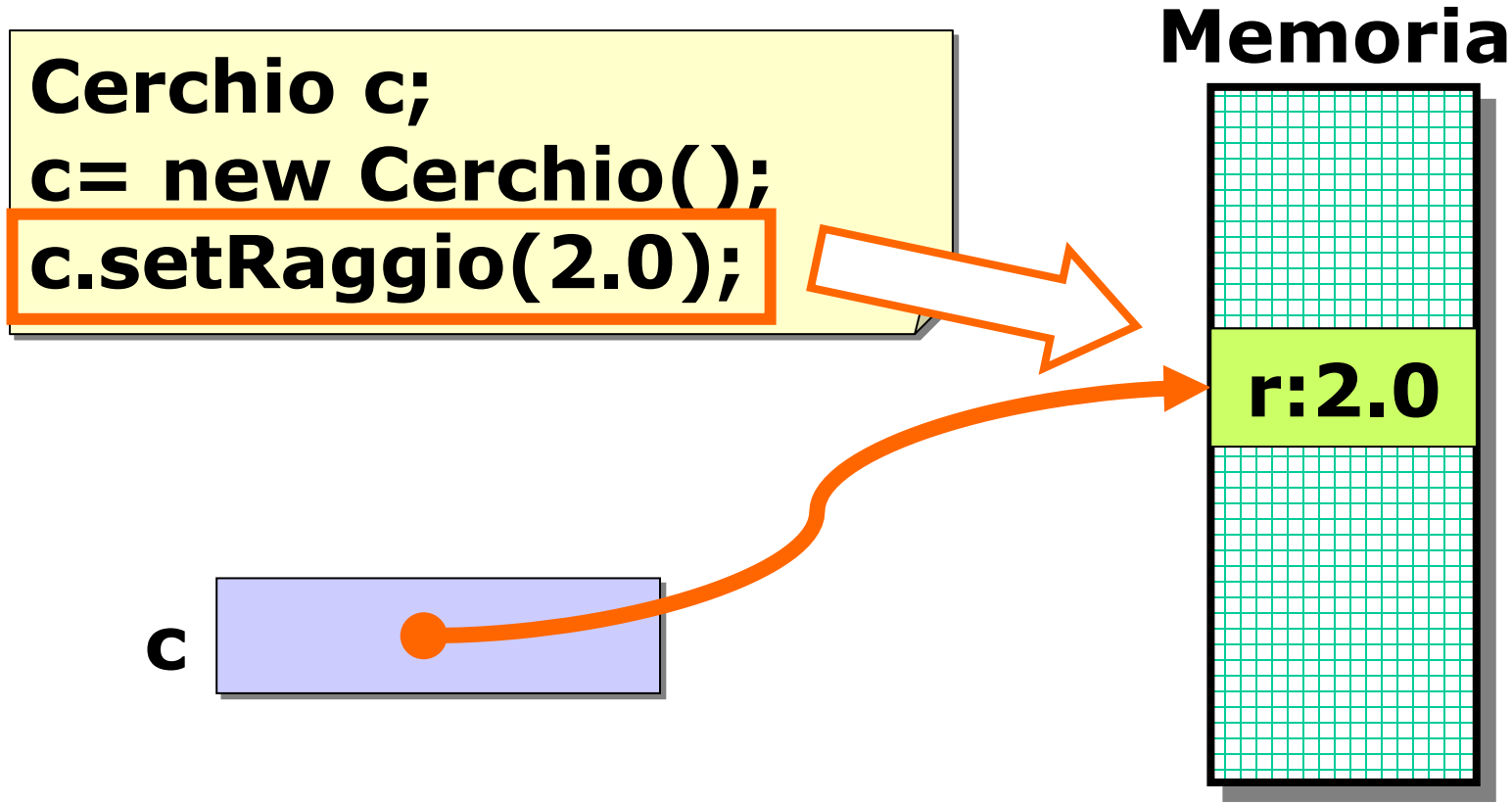
- Noto il riferimento ad un oggetto, è possibile invocarne i metodi
  - Si usa la notazione

    variabile.metodo(...);
  - Nelle parentesi si indicano gli eventuali parametri
- Se la variabile contiene un riferimento nullo si genera un errore di esecuzione
- Il metodo è costituito da un insieme di istruzioni
  - Il comportamento è simile ad una chiamata a procedura
  - Il chiamante attende il completamento del metodo, poi prosegue la propria elaborazione

Parametri

- Un metodo può avere parametri
  - Internamente, appaiono come variabili locali
  - Il loro valore, però, è fornito dal chiamante
- Come le variabili, i parametri hanno un tipo
  - Elementare
  - Complesso
- Nel caso di parametri di tipo elementare
  - All'interno del parametro viene memorizzata una copia del valore indicato dal chiamante
  - Eventuali modifiche compiute dal metodo sul valore **non si ripercuotono sul chiamante**
- Se il tipo è complesso
  - Il parametro contiene una copia del riferimento all'oggetto passato come parametro
  - Le eventuali modifiche sullo stato dell'oggetto **sono visibili al chiamante**

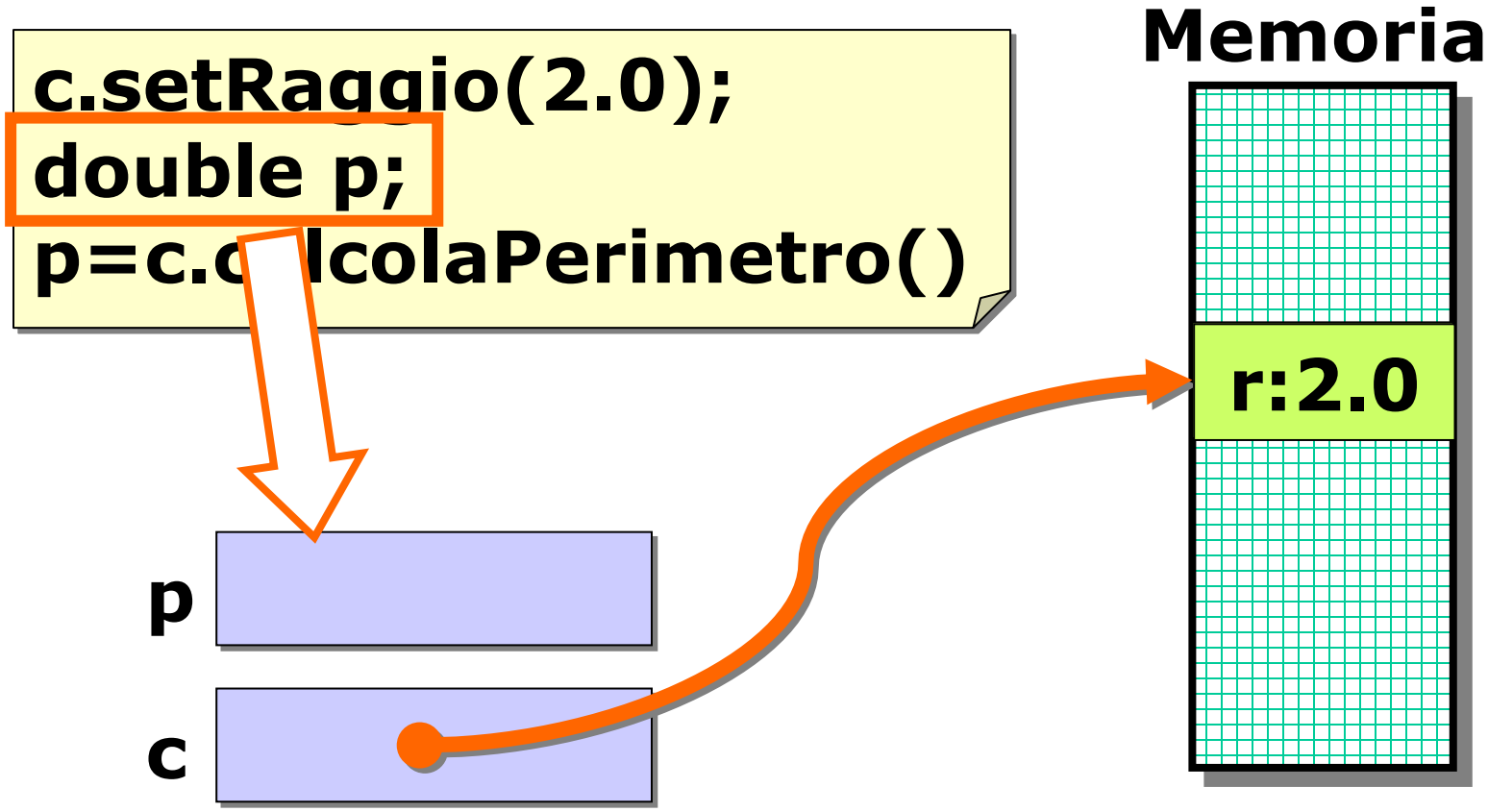
Invocare metodi



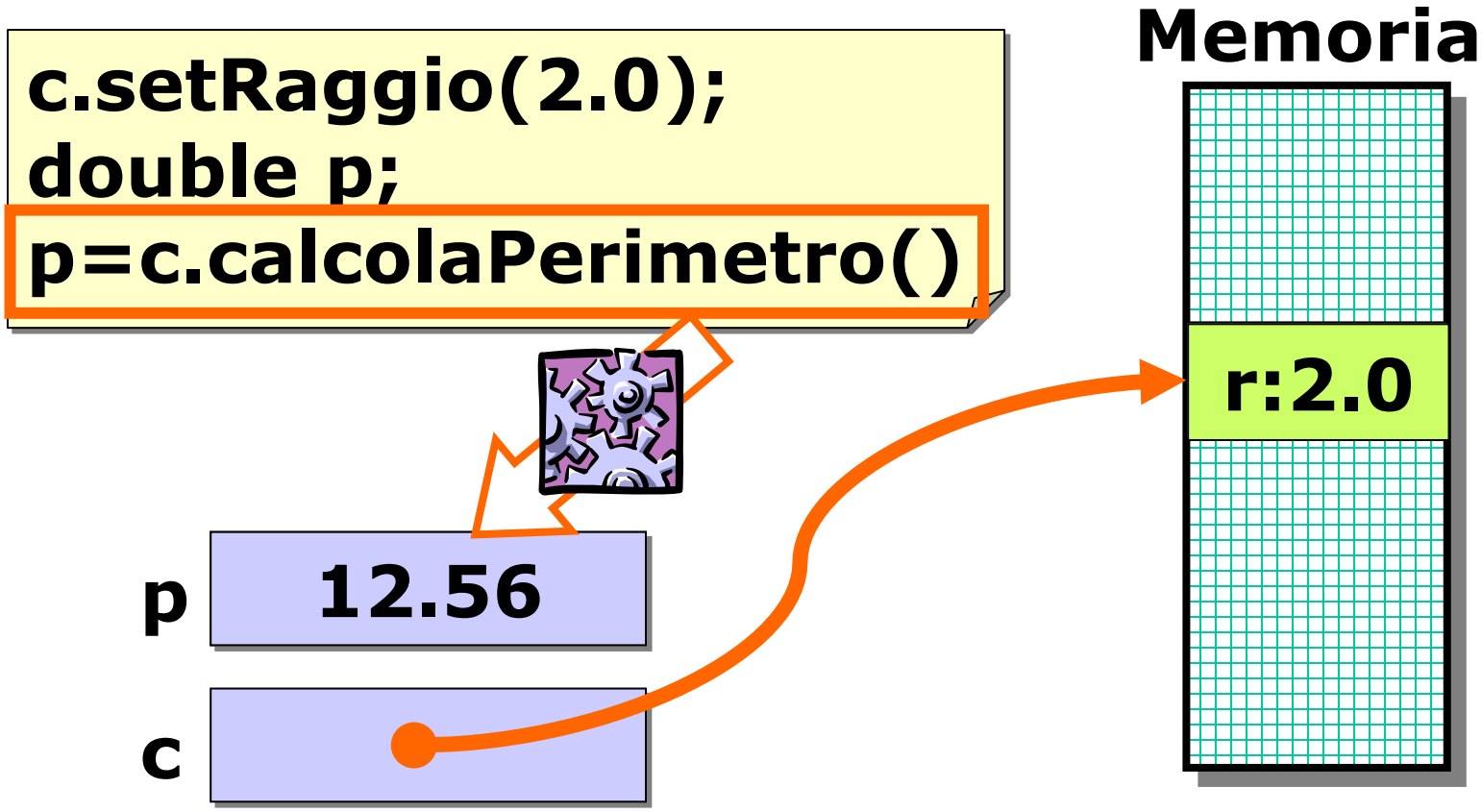
Valori di ritorno

- Alcuni metodi restituiscono un valore
  - Il tipo del valore ritornato viene dichiarato prima del nome del metodo
  - **double calcolaPerimetro() { … }**
- Il valore ritornato può essere assegnato ad una variabile
  - Occorre che la variabile sia compatibile con il tipo del valore ritornato
- Altri metodi non ritornano nulla
  - In questo caso dichiarano di ritornare il tipo predefinito **void**
  - **void setRaggio(double r) { … }**

Valori di ritorno



Valori di ritorno



Tipologie di metodi

- Un metodo, in generale, può operare liberamente sull'oggetto su cui viene invocato…
  - Modificandone lo stato
  - Invocando metodi su altri oggetti conosciuti
  - Effettuando calcoli
  - Restituendo risultati
- Spesso, però, si progettano metodi specializzati in una delle funzioni citate

Tipologie di metodi

- Modificatori *(setter)*
  - Servono ad alterare, **in modo controllato**, lo stato dell'oggetto (o una sua parte specifica)
  - Di solito hanno parametri e non ritornano nulla
  - Per convenzione, in Java, hanno un nome simile a **void setXyz(…);**
- Lettori *(getter)*
  - Permettono di conoscere lo stato (o una sua parte specifca) di un oggetto
  - Di solito, non hanno parametri, e ritornano il valore letto
  - Per convenzione, in Java, hanno un nome simile a **<tipoRitornato> getXyz();**

Attributi

- Come le variabili, anche gli attributi possono avere tipi
  - Elementari
  - Complessi (riferimenti ad oggetti)
- Un attributo di tipo elementare
  - Contiene direttamente il valore
- Un attributo di tipo complesso
  - Contiene il riferimento ad un oggetto (oppure null)

Esempio

```
class Disegno{
int x;
int y;

Cerchio c;
...
}
```

Tipo elementare

Tipo composto

Attributi e costruttore

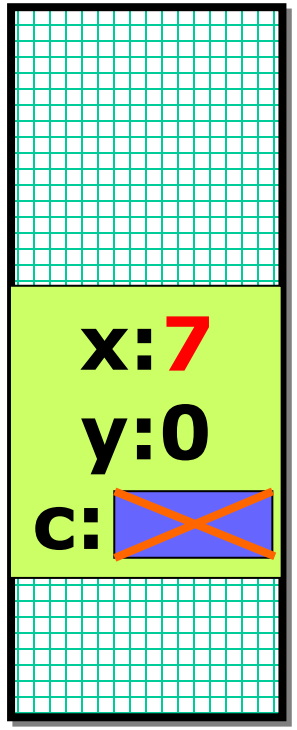
- All'atto dell'invocazione del costruttore, tutti gli attributi hanno un valore di default
  - Gli attributi semplici valgono **0** (false, nel caso dei valori booleani)
  - Quelli complessi, valgono **null**
- È compito del costruttore assegnare valori opportuni se quelli di default non sono adatti
  - Questo può comportare anche la creazione di oggetti

Attributi e costruttore

```
class Disegno {
  double x,y;
  Cerchio c;

  Disegno() {
    x=7.0;
    y=3.0;
    c= new Cerchio()
  }
  …
}
```

Disegno
x, y: double c: Cerchio
…

Memoria

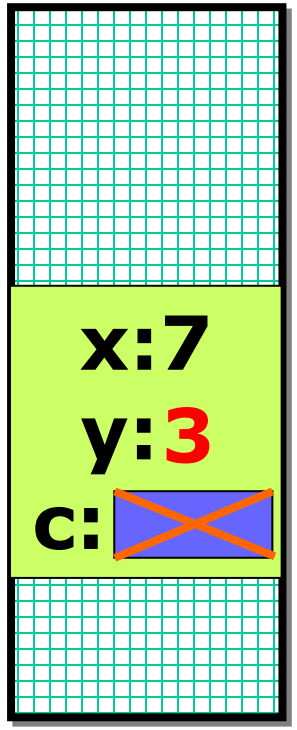


Attributi e costruttore

```
class Disegno {
  double x,y;
  Cerchio c;

  Disegno() {
    x=7.0;
    y=3.0;
    c= new Cerchio()
  }
  …
}
```

Disegno
x, y: double c: Cerchio
…

Memoria

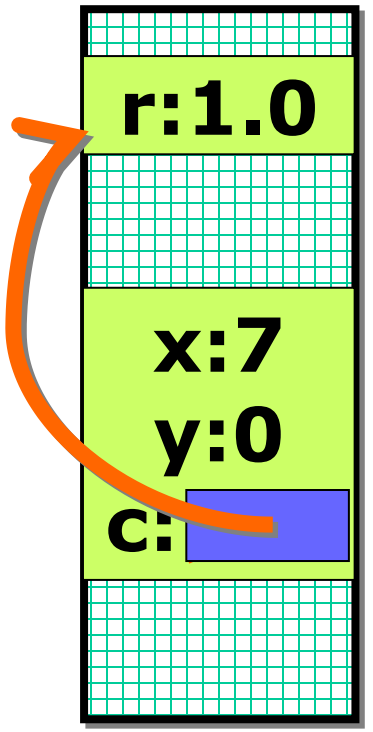


Attributi e costruttore

```
class Disegno {
  double x,y;
  Cerchio c;

  Disegno() {
    x=7.0;
    y=3.0;
    c= new Cerchio()
  }
  ...
}
```

Disegno
x, y: double c: Cerchio
…

Memoria



Ciclo di vita di un oggetto

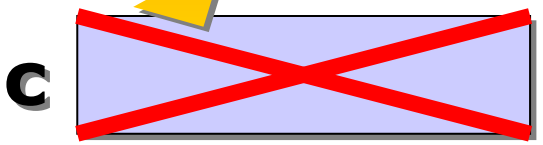
- L'operatore new, su richiesta del programmatore, alloca la memoria necessaria a contenere un oggetto
  - D1: quando viene rilasciata?
  - R1: quando l'oggetto non serve più!
  - D2: chi decide che non serve più?
  - R2: l'ambiente di esecuzione (!?!)

Accessibilità

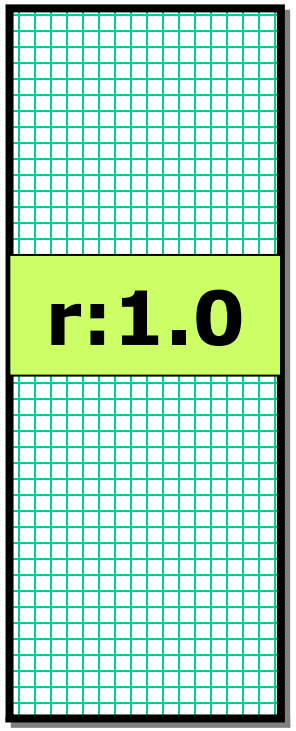
- Un oggetto è accessibile fino a che ne esiste un riferimento
- Nel momento in cui non esistano più riferimenti, l'oggetto può essere eliminato
  - Rilasciando la memoria che occupa
- I riferimenti sono memorizzati in variabili e attributi
  - Si cancellano quando la variabile cessa di esistere (fine del blocco)
  - Oppure assegnando esplicitamente il valore **null**

Riferimenti nulli

```
Cerchio c;
c= new Cerchio();
...
c=null;
```



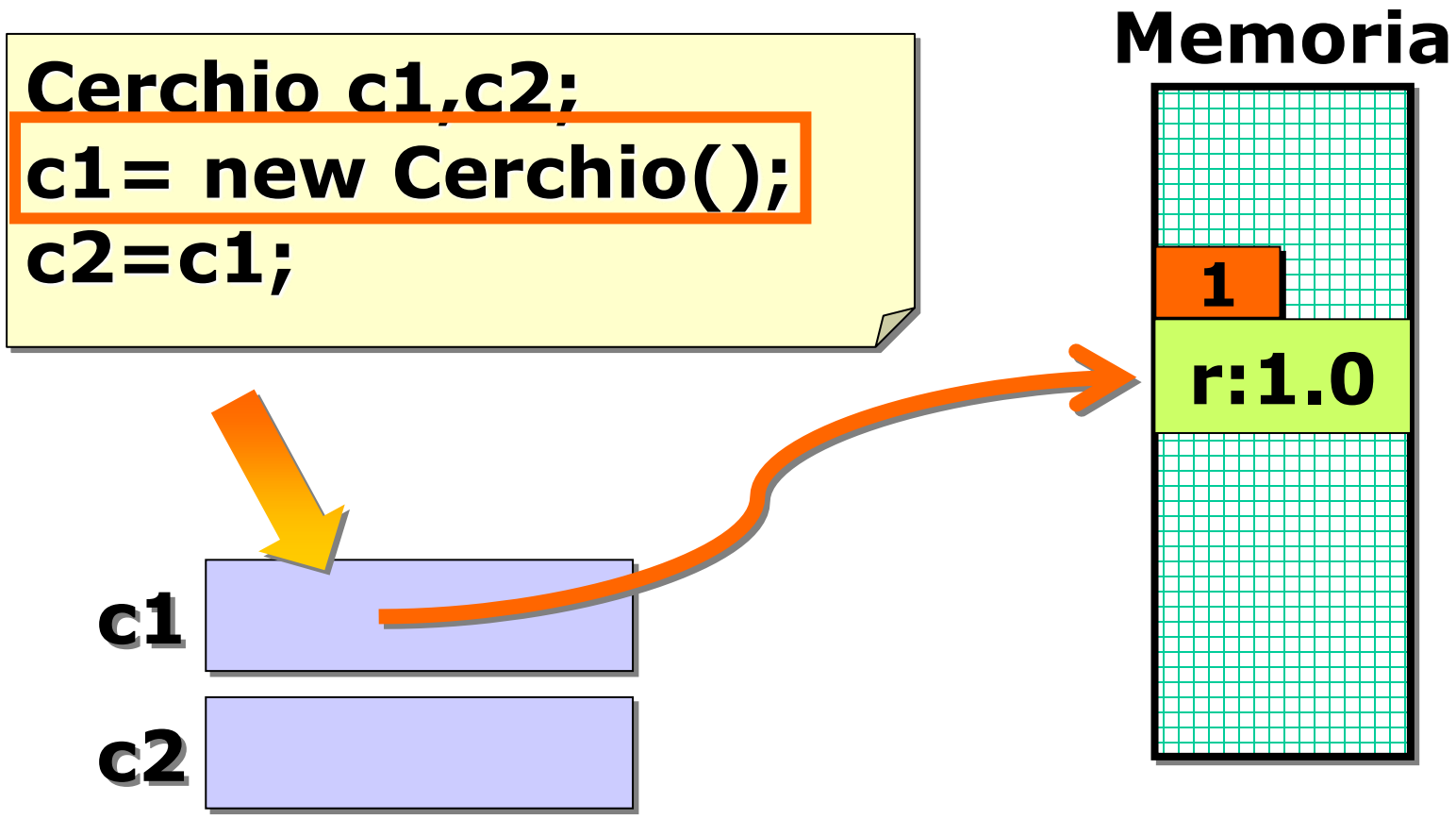
Memoria



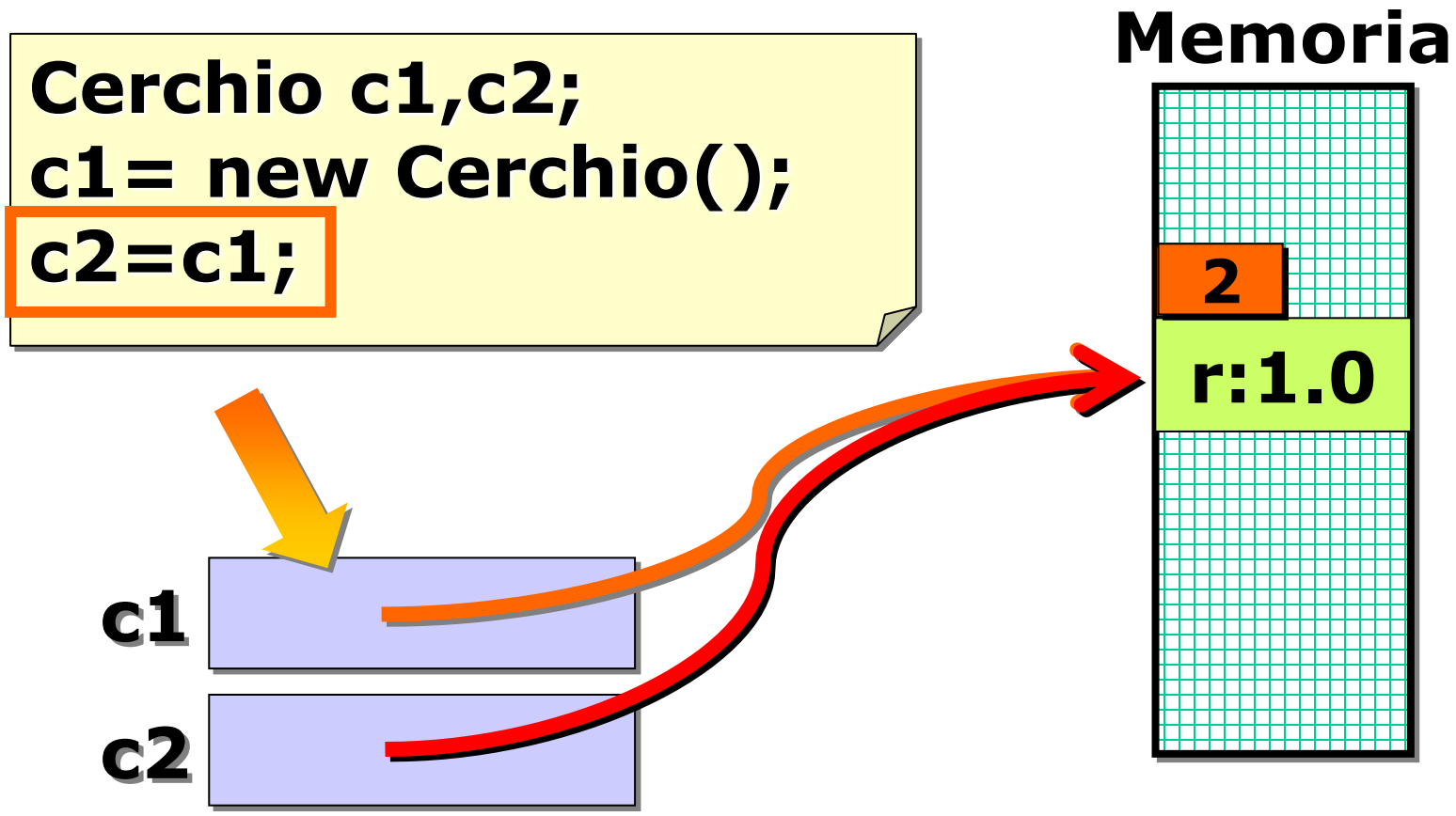
Conteggio dei riferimenti

- All'interno di ogni oggetto, Java mantiene un contatore nascosto
  - Indica il numero di riferimenti esistenti a quello specifico oggetto
  - Quando il suo valore scende a 0, indica che l'oggetto può essere eliminato, rilasciando la memoria che occupa
- Un particolare sottosistema, il **garbage collector**, si occupa, periodicamente, di riciclare la memoria degli oggetti eliminati
  - Viene eseguito automaticamente dalla macchina virtuale Java

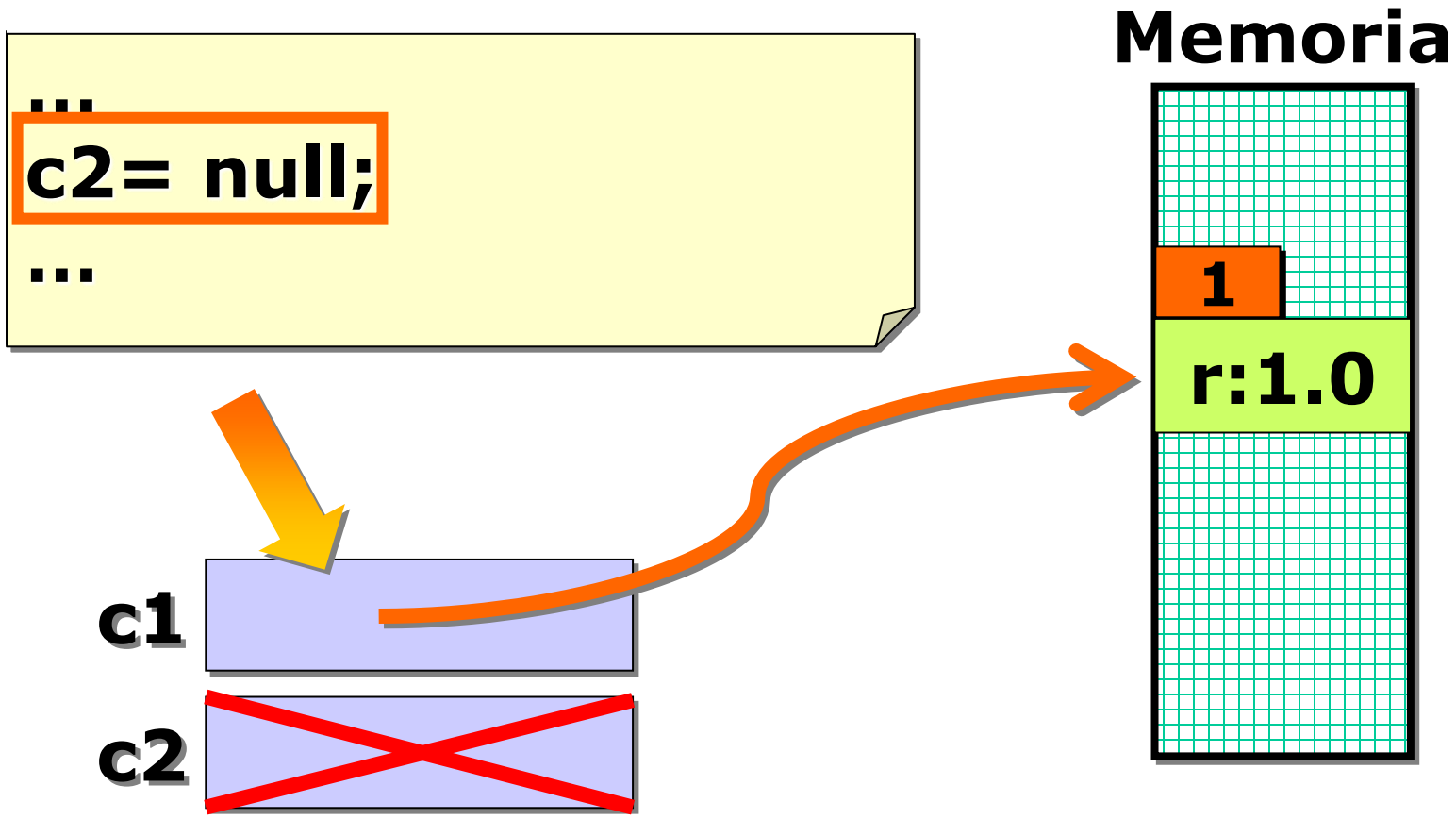
Conteggio dei riferimenti



Conteggio dei riferimenti

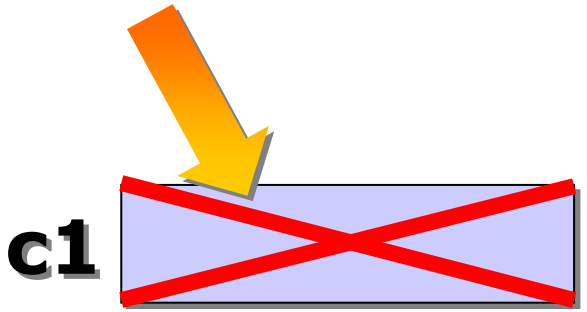


Conteggio dei riferimenti



Conteggio dei riferimenti

```
...
c1= null;
...
```



Memoria