# Classi Wrapper

Tipo di dato in Java che **ingloba** (***wraps***)
un valore primitivo all’interno di un oggetto.

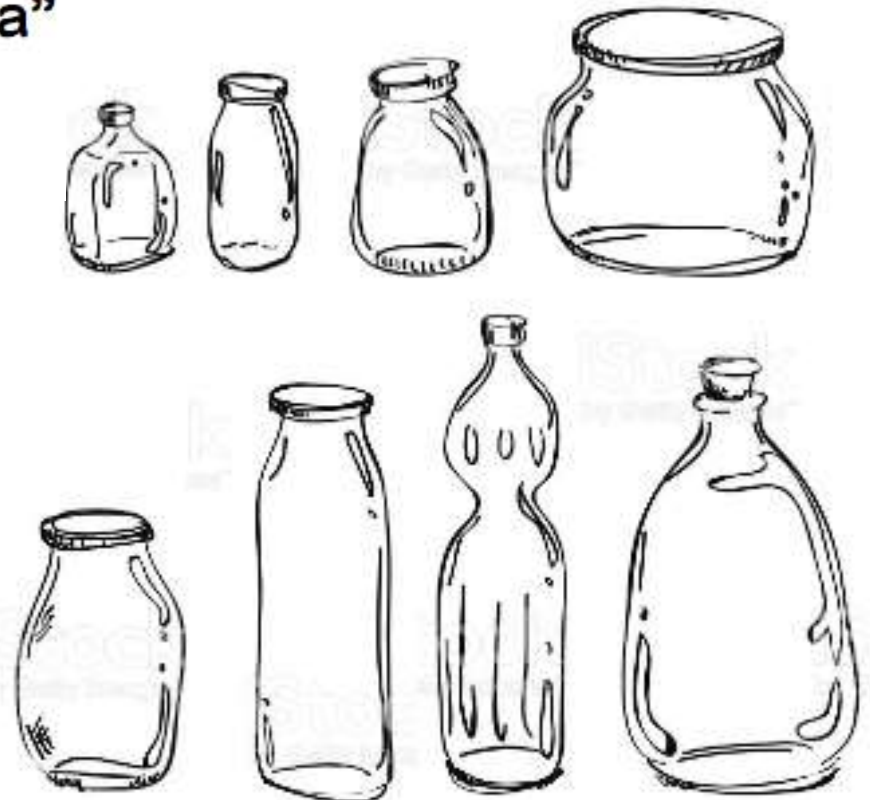
# *dati primitivi e oggetti contenitori*

# Tipi primitivi in java

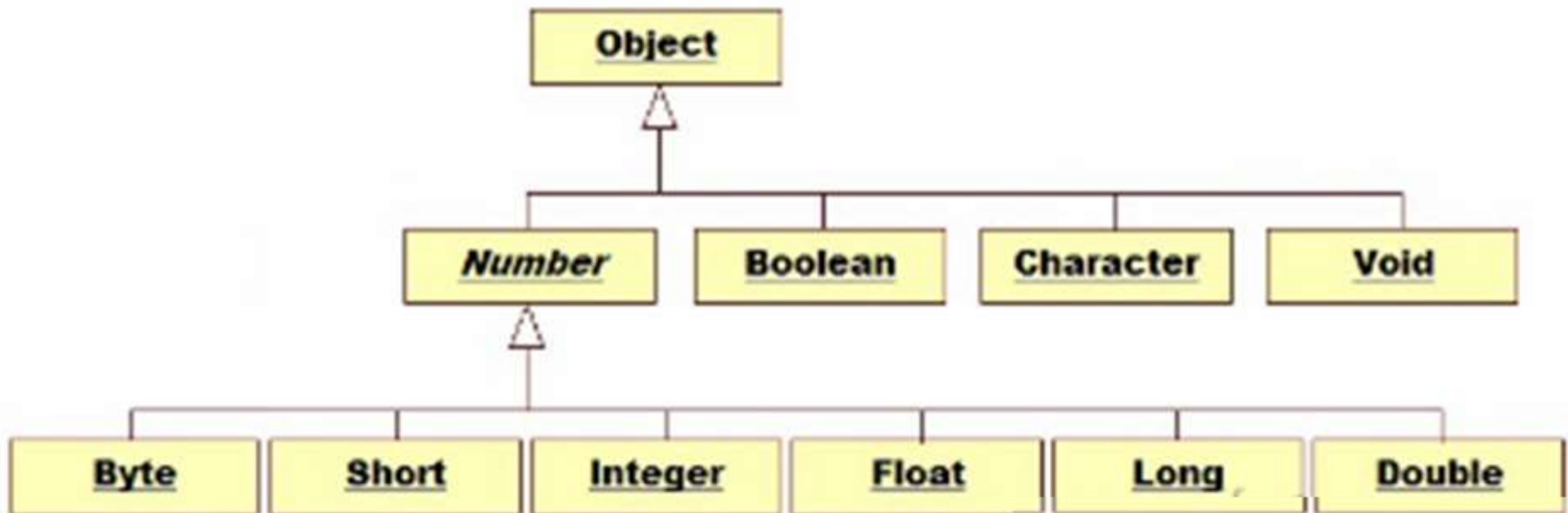
| tipo | default | mem. | da | a |
|---|---|---|---|---|
| boolean | *false* | 1 bit | true | false |
| char | *'\u0000'* | 16 bit | 0 | 65535 |
| byte | *0* | 8 bit | -127 | 128 |
| short | *0* | 16 bit | -32768 | 32767 |
| int | *0* | 32 bit | -2^31 | 2^31-1 |
| long | *0L* | 64 bit | -2^63 | 2^63-1 |
| float | *0.0f* | 32 bit | e,E | |
| double | *0.0d* | 64 bit | esp$10^x$ | |

# Wrapper (involucro)

- In varie situazioni, può essere comodo poter trattare i tipi primitivi come oggetti.
- Una classe wrapper (involucro) incapsula una variabile di un tipo primitivo
- In qualche modo "trasforma" un tipo primitivo in un oggetto equivalente
  - la classe Boolean incapsula un boolean
  - la classe Double incapsula un double
  - la classe Integer incapsula un int
  - .....



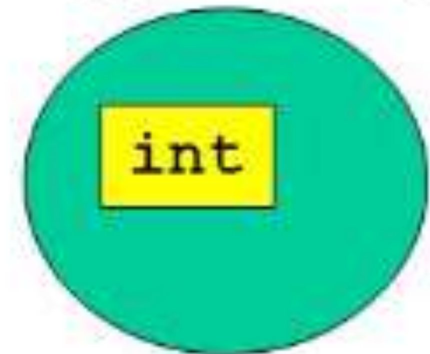
# Nella gerarchia di classi




- La classe wrapper ha nome (quasi) identico al tipo primitivo che incapsula, ma con l'iniziale maiuscola

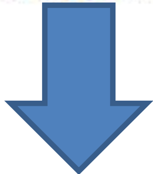
# Tipi primitivi e classi *wrapper*

| Tipo primitivo | Classe "wrapper" corrispondente |
|---|---|
| boolean | Boolean |
| char | Character |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |

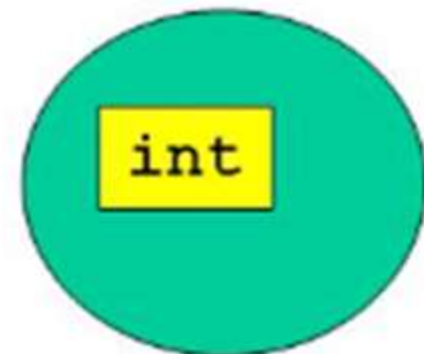


# Funzionamento

- **Ogni classe wrapper ha come stato semplicemente un attributo del tipo che incapsula: Integer avrà un attributo di tipo int, Double un attributo di tipo double e così via.**
- **Le classi wrapper sono state costruite per essere immutabili: assumono un valore al momento della creazione e non lo cambiano mai più.**



come gli oggetti *String*,
***non hanno metodi***
***che permettono di modificarli***
e di conseguenza
sono anche di tipo ***final***



# Funzionamento

- Quindi per ogni classe esiste un costruttore che prende come parametro una variabile del tipo incapsulato e lo memorizza nello stato.

`Integer i = new Integer(`*valore int*`)`

- Esiste poi un metodo che consente di leggere in modo protetto il valore dello stato.

`Integer` fornisce il metodo `intValue()`

**Ogni classe wrapper definisce metodi per estrarre il valore della variabile incapsulata e viceversa.**

- Per estrarre il valore incapsulato:
  - `Integer` fornisce il metodo `intValue()`
  - `Double` fornisce il metodo `doubleValue()`
  - `Boolean` fornisce il metodo `booleanValue()`
  - `Character` fornisce il metodo `charValue()`
  - `...`
- Per creare un oggetto da un valore primitivo:
  - `Integer i = new Integer(`*valore int*`)`
  - `Double d = new Double(`*valore double*`)`
  - `Boolean b = new Boolean(`*valore boolean*`)`
  - `Character c = new Character(`*valore char*`)`
  - `...`

# Semplice esempio

```
public class EsempioWrapper {
 public static void main(String args[]){

  int x = 35;

  Integer ix = new Integer(x);

  x = 2*ix.intValue();


  System.out.println("ix =" + ix);
  System.out.println("x =" + x);
 }
}
```

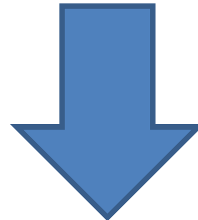
Costruisce un oggetto `Integer` a partire da un valore `int`

Estrae da un oggetto `Integer` il valore `int` incapsulato al suo interno, mediante `intValue()`

```
ix = 35
x = 70
```

# Funzionalità in doppia versione

- **La classe wrapper contiene anche *funzioni che operano sul tipo primitivo corrispondente***
  - **in particolare, conversione stringa / numero ...**
  - **... e viceversa**
- **Attenzione: le funzioni che operano sul tipo primitivo sono *funzioni statiche, di classe***
  - infatti, non esiste alcun oggetto su cui invocare metodi
- **Quindi, varie funzionalità sono disponibili in *doppia versione:***
  - funzione statica (per variabili di tipo primitivo)
  - metodi (per oggetti che siano istanze del tipo wrapper)



*esempio toString(..)*

## Esempio: *`toString()`* *nella classe* *`Integer`*

- **versione statica:**

```
public static String toString(int x);
```

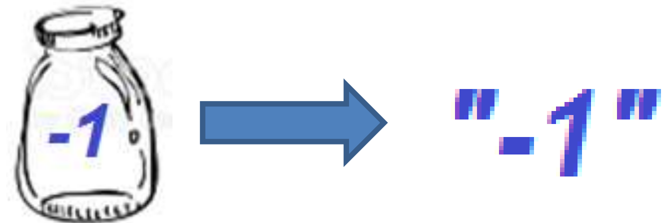
  – prende un valore `int` e ne produce la rappresentazione sotto forma di stringa

- **versione metodo:**

```
public String toString();
```

  – è implicitamente invocato su un oggetto Integer

  – ne recupera il valore e ne produce la rappresentazione sotto forma di stringa.



# Convertire da tipo numerico a tipo String: altri modi

- metodo **valueOf()** della classe **String**
- **conversione implicita**

### *Esempio di metodo (da tipo intero)*

```
public static void intToString(int i){
    System.out.println(i);
    System.out.println("Conversione implicita: " + i);  // conversione implicita
    String s = Integer.toString(i);
    System.out.println("Converto con Integer.toString: " + s);
    String s1 = String.valueOf(i);
    System.out.println("Converto con String.valueOf: " + s1);
}
```

"-1"

# *Appendice: dati primitivi vs oggetti*

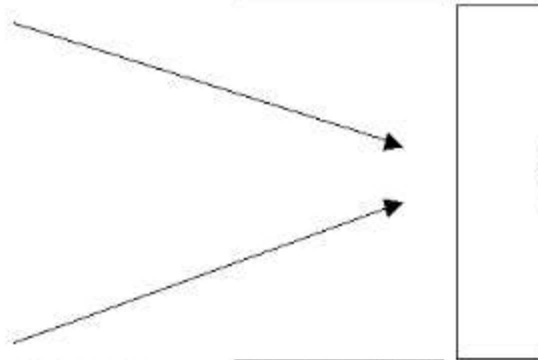
RAM

| | | | |
|---|---|---|---|
| | | | |
| | intVar | | |
| | | | |

unNumero 5.0

unAltroNumero 5.0

unGiorno 0xabcd2346



unAltroGiorno 0xabcd2346

# VARIABILI NEI LINGUAGGI IMPERATIVI

Una *variabile* in un linguaggio imperativo

- *non è solo un sinonimo per un dato* come in matematica
- **è un'astrazione della cella di memoria**
- **associata a due diverse informazioni:**
  - il contenuto (R-value)
  - l'indirizzo a cui si trova (L-value)

# I tipi primitivi

| Nome | Tipo di valore | Memoria usata | Range di valori |
|---|---|---|---|
| byte | intero | 8 bit = 1 byte | -128 : +127 |
| short | | 16 bit = 2 byte | -32.768 : +32.767 |
| int | | 32 bit = 4 byte | -2.147.483.648 : +2.147.483.647 |
| long | | 64 bit = 8 byte | -9.223.372.036.854.775.808 : +9.223.374.036.854.775.808 |
| float | floating point | 32 bit = 4 byte | +/- 3,4028... x 10+38 : +/- 1,4023... x 0-45 |
| double | | 64 bit = 8 byte | +/- 1,767... x 10+308 : +/- 4,940... x 0-324 |
| char | singolo carattere | 16 bit = 2 byte | Tutti i caratteri Unicode |
| boolean | true o false | 1 bit | *true o false* |

# TIPI DI DATO PRIMITIVI IN JAVA

- caratteri
  - `char` (2 byte) codifica UNICODE
  - coincide con ASCII sui primi 127 caratteri
  - e con ANSI / ASCII sui primi 255 caratteri
  - *costanti char anche in forma* `'\u2122'`
- interi (con segno)
  - `byte` (1 byte) -128 ... +127
  - `short`(2 byte) -32768 ... +32767
  - `int` (4 byte) -2.147.483.648 ... 2.147.483.647
  - `long` (8 byte) $-9\ 10^{18}$ ... $+9\ 10^{18}$

  *NB: le costanti long terminano con la lettera L*

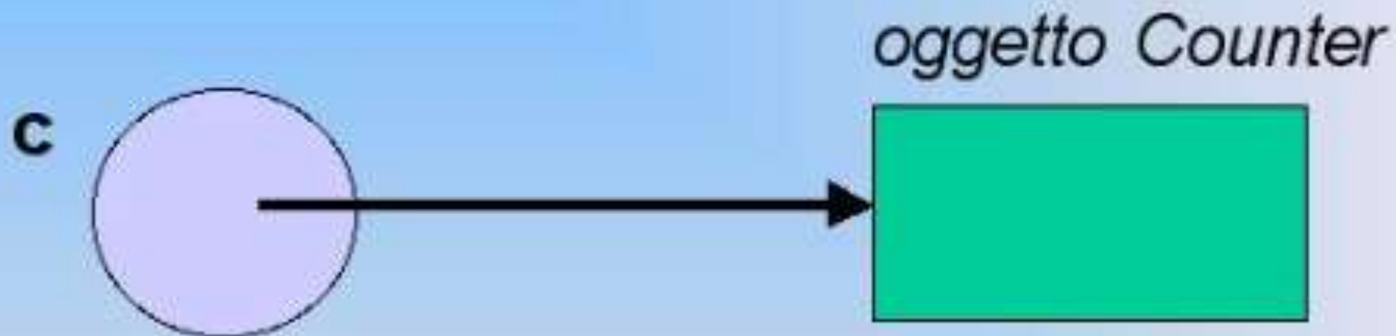
# TIPI DI DATO PRIMITIVI IN JAVA

- **reali** *(IEEE-754)*
  - `float` **(4 byte)** $-10^{45}$ ... $+10^{38}$
    *(6-7 cifre significative)*
  - `double` **(8 byte)** $-10^{328}$ ... $+10^{308}$
    *(14-15 cifre significative)*
- **boolean**
  - **`boolean`** (1 bit) **`false`** e **`true`**
  - tipo autonomo *totalmente disaccoppiato dagli interi:* non si convertono boolean in interi e viceversa, *neanche con un cast*
  - tutte le espressioni relazionali e logiche danno come risultato un `boolean`, non più un `int`!

# RIFERIMENTI A OGGETTI

- **La frase** `Counter c;`

  ***non definisce una variabile*** `Counter`**,**
  **ma solo un *riferimento a*** `Counter`



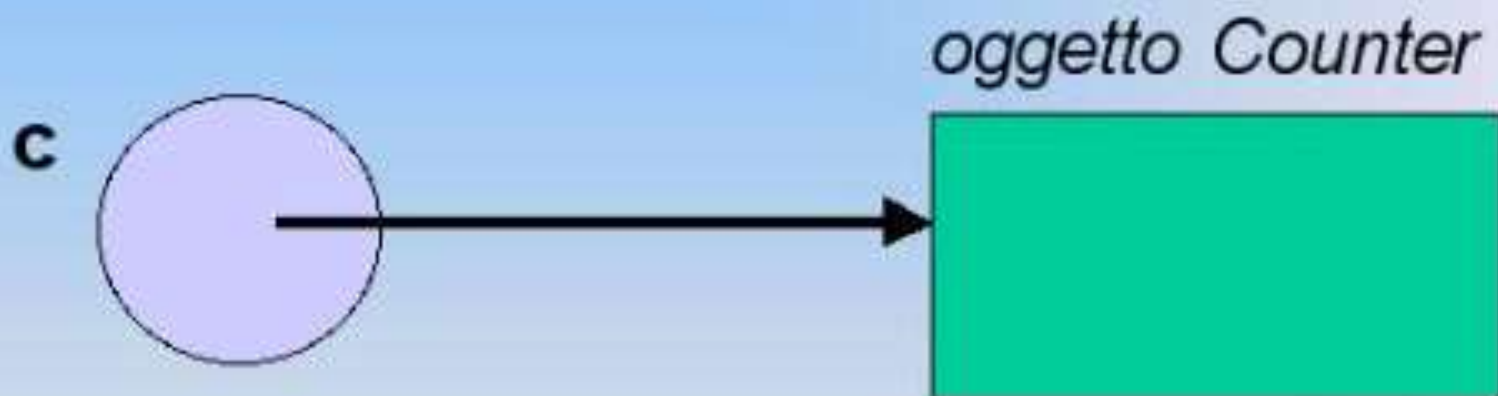

- **L'oggetto `Counter` viene poi creato dinamicamente, quando opportuno, con `new`**

```
c = new Counter();
```

# RIFERIMENTI vs. PUNTATORI

**Analogie...**

- **a livello fisico,** ***un riferimento è di fatto un puntatore...***



# RIFERIMENTI vs. PUNTATORI

## ... e Differenze

- **un riferimento, invece, *contiene* un indirizzo, ma *non consente di vederlo o manipolarlo***
  - *niente accesso diretto alla macchina sottostante!*
  - non occorre dereferencing *esplicito*
    *basta la notazione puntata*

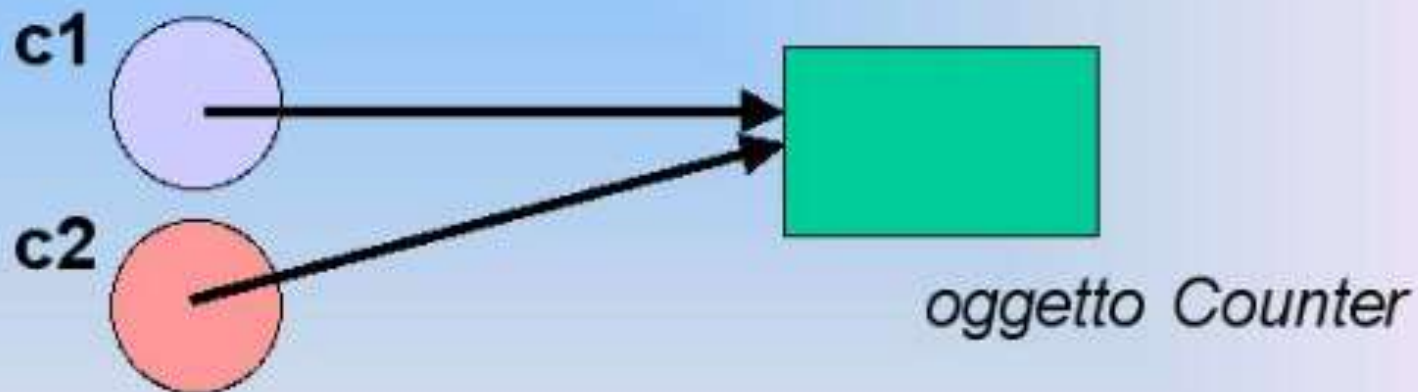
Un riferimento è un *"puntatore controllato"*
- non può spostarsi - *Niente aritmetica dei puntatori!*
- non può invadere aree non proprie

*Mantiene la potenza dei puntatori, ma riduce i pericoli legati al loro abuso*

# UGUAGLIANZA FRA OGGETTI

***Quale significato per*** `c1==c2`***?***

- **`c1` e `c2` sono due riferimenti**
  → ***uguali se puntano allo stesso oggetto***

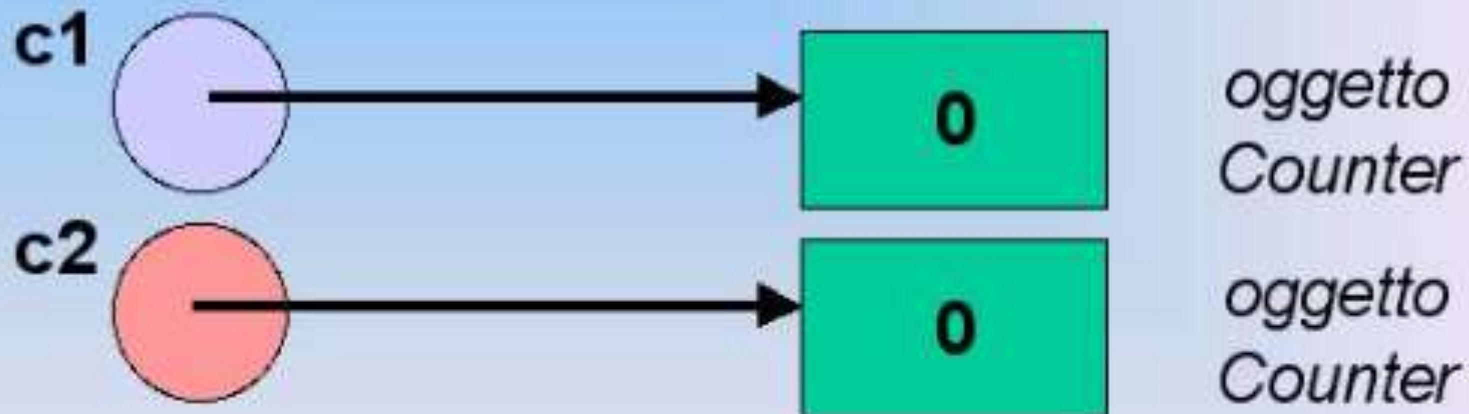



- **qui, `c1==c2` è `true`**

# UGUAGLIANZA FRA OGGETTI

*E se creo due oggetti identici?*

```
Counter c1 = new Counter();
Counter c2 = new Counter();
c1.reset(); c2.reset();
```

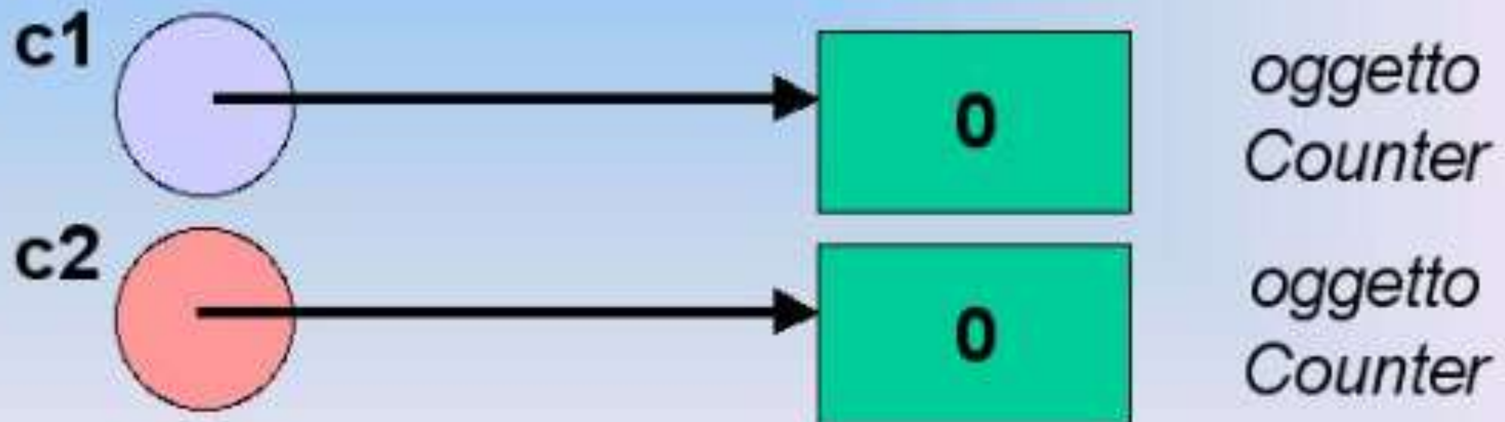



- il contenuto *non conta*: `c1==c2` è ***false***!

# UGUAGLIANZA FRA OGGETTI

*Per verificare l'uguaglianza fra i valori di due oggetti si usa il metodo* **`equals`**

```
Counter c1 = new Counter();
Counter c2 = new Counter();
c1.reset(); c2.reset();
```




- **contenuto uguale: `c1.equals(c2)` è `true`**

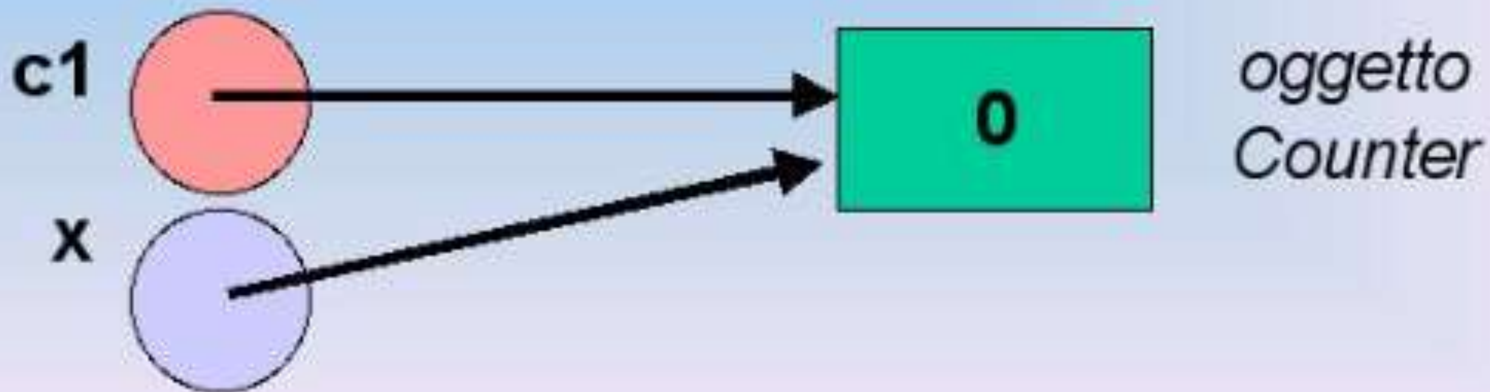
# PASSAGGIO DEI PARAMETRI

**Esempio:**

```
void f(Counter x) { ... }
```

**Il cliente:**

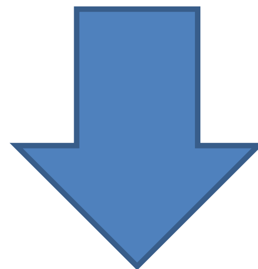
```
Counter c1 = new Counter();
f(c1);
```



# PASSAGGIO DEI PARAMETRI

Quindi:

- *un parametro di tipo primitivo* viene copiato, e la funzione riceve la copia
- *un riferimento* viene *pure copiato*, la funzione riceve la copia, ma con ciò accede all'oggetto originale!



- *un riferimento* viene *pure copiato*, la funzione riceve la copia, ma con ciò **accede all'oggetto originale!**

```
public class UsaVoti {
        private   Voto voti[]      ;
        ...
        public void stampa( Voto v[]){
                        ....
        }
        public static void main(String[] args) {
                UsaVoti o = new UsaVoti( )  ;
                ...
                o.stampa( o. voti  );
                ...
        }
}
```

# *sitografia*

Dispense e slides

Dati primitivi