



Metodologie di Programmazione

Lezione 6: Concetti fondamentali della programmazione orientata agli oggetti

Lezione 6:

Sommario



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Classi e oggetti
- Campi e metodi
- Costruttori
- Variabili locali e campi

Classi e oggetti

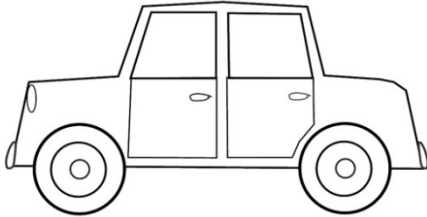
- Possono:
 - Modellare gli oggetti del **mondo reale**
 - Rappresentare **oggetti grafici**
 - Rappresentare **entità software** (file, immagini, eventi, ecc.)
 - Rappresentare **concetti astratti** (regole di un gioco, posizione di un giocatore)
 - Rappresentare **stati** di un processo, di esecuzione, ecc.

Classi e oggetti

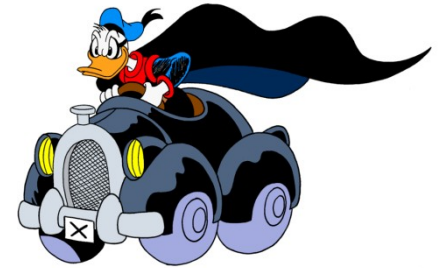
- Una **classe** è un pezzo del codice sorgente di un programma che **descrive** un particolare tipo di oggetti
- Le classi vengono definite dal programmatore (**definizione di classe**)
- La classe fornisce un **prototipo astratto** per gli oggetti di un particolare tipo
- Ne definisce la struttura in termini di:
 - **Attributi** (stato) degli oggetti
 - **Metodi** (comportamenti) degli oggetti
- Un **oggetto** è un'**istanza** (un esemplare) di una classe
- Un **programma** può creare e usare **uno o più oggetti** (istanze) della **stessa classe**

Classe vs. oggetto

- **Classe:** automobile



- **Oggetto:** una certa automobile



- **Attributi:**

- String modello;
- Color colore;
- int numPasseggeri;
- double benzina;

- **Metodi:**

- Aggiungi/togli passeggero
- Riempi serbatoio
- Segnala quantità benzina

- **Attributi:**

- String modello = "X";
- Color colore = Color.BLACK;
- int numPasseggeri = 1;
- double benzina = 50;

- **Metodi:**

- Come la classe

Classe vs. oggetto

- **Classe:**
 - Definita mediante parte del codice sorgente del programma
 - Scritta dal programmatore
- **Oggetto:**
 - Un'entità all'interno di un programma in esecuzione
 - Creato quando un programma "gira" (dal metodo main o da un altro metodo)

Classe vs. oggetto

- **Classe:**
 - Specifica la struttura (ovvero numero e tipi) degli attributi dei suoi oggetti
 - Specifica il comportamento dei suoi oggetti mediante il codice dei metodi
- **Oggetto:**
 - Contiene specifici valori degli attributi; i valori possono cambiare durante l'esecuzione
 - Si comporta nel modo prescritto dalla classe quando il metodo corrispondente viene chiamato a tempo di esecuzione

Oggetto 1

Oggetto 2

Oggetto 3

Oggetto 4

Oggetto 3

Oggetto 1

Oggetto 2

1P-

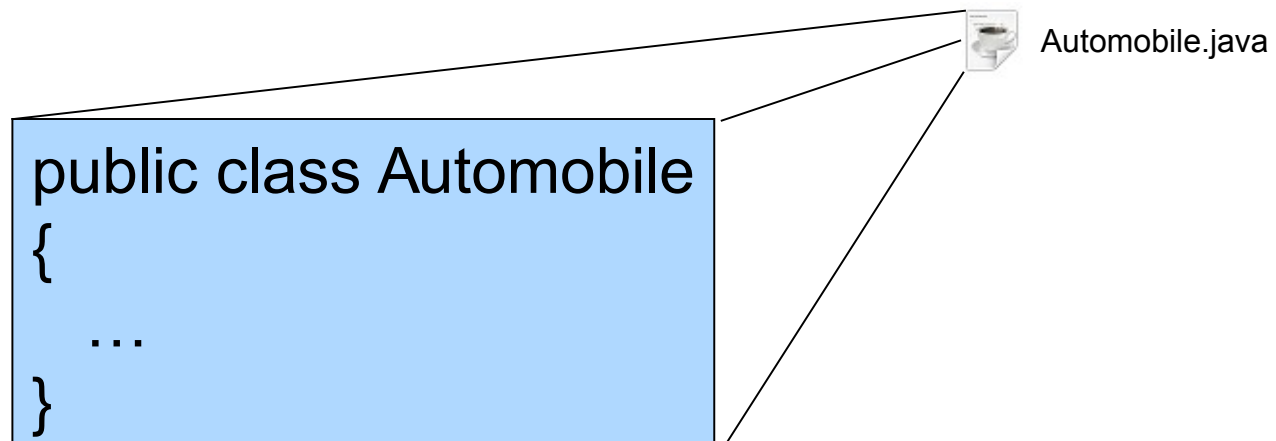
BEAM 74300

HI-

174500

Classi e file sorgenti

- Ogni **classe** è memorizzata in un file separato
- Il **nome del file** DEVE essere lo **stesso** della **classe**, con estensione **.java**
- I nomi di classe iniziano sempre con una maiuscola (es. **Automobile**, non **automobile**)
- I nomi in Java sono **case-sensitive**!



- I programmi Java normalmente non sono scritti **da zero**
- Esistono migliaia di **classi di libreria** per ogni esigenza (sia standard scritte da Sun, sia sul Web scritte da centinaia di sviluppatori)
- Le **classi** sono organizzate in **package**
- Alcuni esempi:
 - **java.util** – classi di utilità
 - **java.awt** – classi per la grafica e le finestre
 - **javax.swing** – sviluppo di interfacce GUI
- Un package “**speciale**” è **java.lang**: contiene le classi fondamentali per la programmazione in Java (es. **String**, **System**, ecc.)

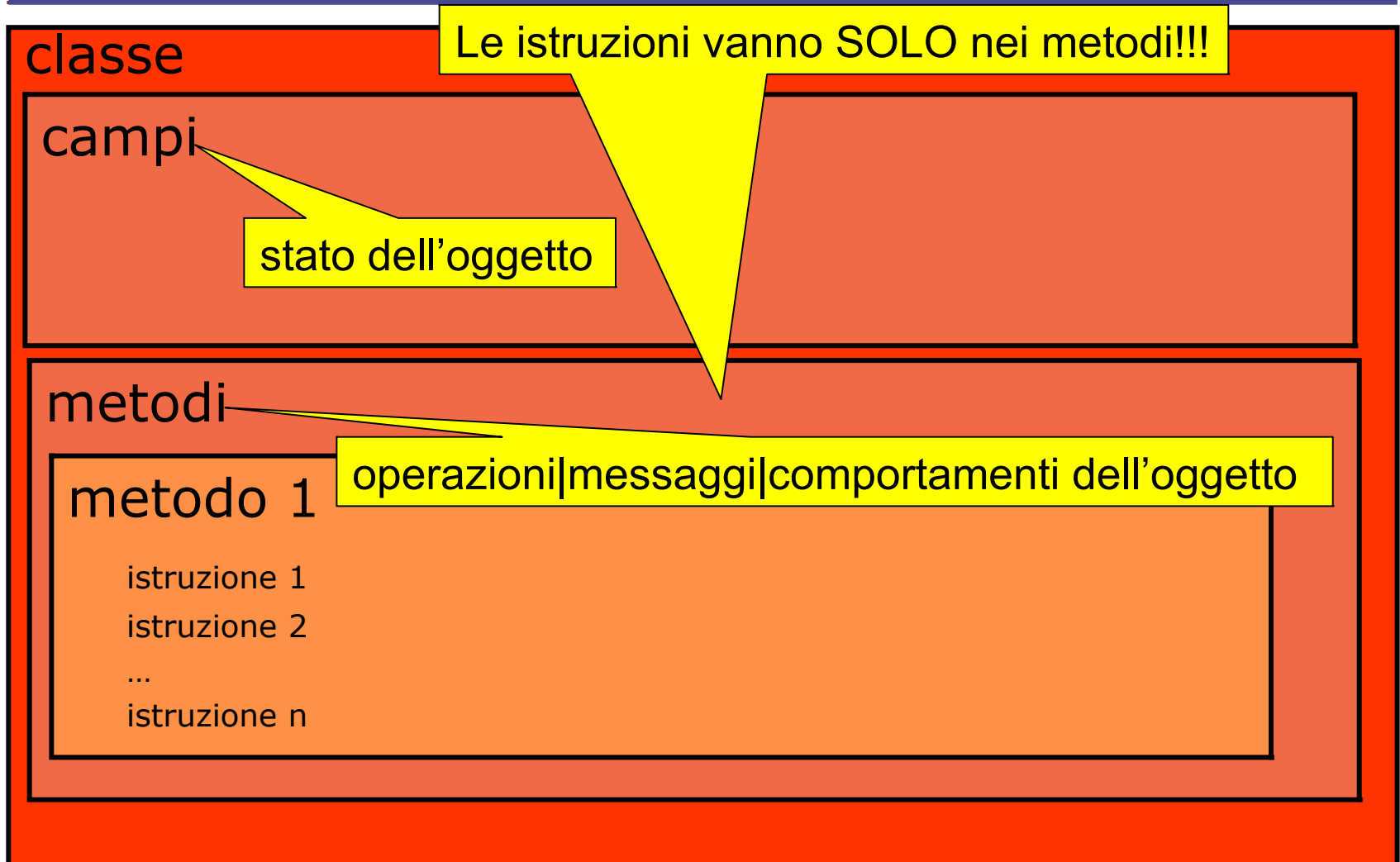
Esercizio: un contatore



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Vogliamo realizzare una classe che rappresenta un contatore
- Il contatore permette di:
 - **Incrementare** il conteggio attuale
 - **Ottenere** il conteggio attuale
 - **Resettare** il conteggio a 0 (o a un altro valore)

Come strutturare il codice di una classe?



Counter.java

```
public class Counter  
{
```

Campi

```
/**  
 * Valore intero del contatore  
 */  
private int value;
```

Commento Javadoc

Dichiarazione di un campo

Tipicamente i campi sono privati

Costruttore

```
/**  
 * Costruttore della classe  
 */  
public Counter()  
{  
    value = 0;  
}
```

Costruttore degli oggetti della classe

Inizializza il campo **value**

I metodi della classe sono pubblici

Metodi

```
/**  
 * Incrementa il contatore  
 */  
public void count()  
{  
    value++;  
}
```

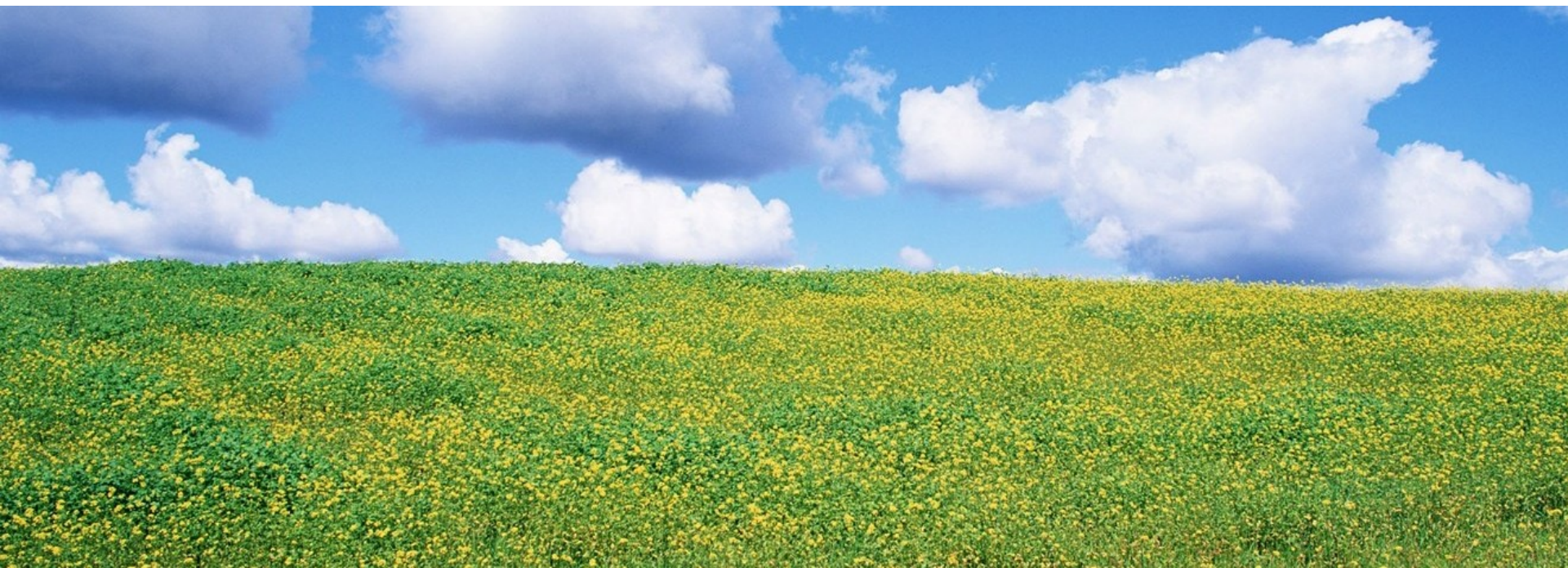
Incrementa il valore di **value**

```
/**  
 * Ottiene il valore corrente del contatore  
 * @return valore intero del contatore  
 */  
public int getValue() { return value; }
```

Restituisce il valore di **value**

Campi

- Un **campo** (detto anche **variabile di istanza**) costituisce la **memoria privata** di un oggetto
- Ogni campo ha un **tipo di dati** (es. il valore del contatore è intero)
- Ogni campo ha un **nome** fornito dal programmatore (es. value nella diapositiva precedente)



Dichiarare un campo

- La dichiarazione di un campo avviene come segue:

```
private [static] [final] tipo_di_dati nome;
```

Tipicamente ad accesso privato

Se specificato indica che il campo è condiviso da tutti gli oggetti della classe

Se specificato indica che il campo è una costante

Es. int, double, String, ecc.

Esempi di campo

```
public class Hotel
{
    /**
     * Da evitare l'uso di una variabile "di comodo" come campo di una classe
     */
    private int k;

    /**
     * Nome dell'hotel
     */
    private String nome;

    /**
     * Numero di stanze
     */
    private int numeroStanze;

    /**
     * Superficie totale
     */
    private double superficie;

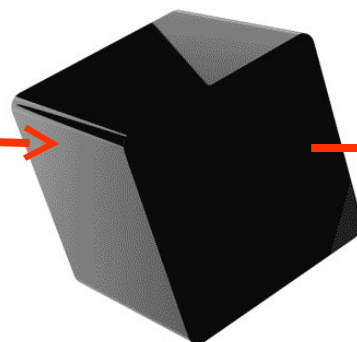
    /**
     * Sulla Guida Michelin
     */
    private boolean bGuidaMichelin;
}
```

Metodi

- Un **metodo** è tipicamente **pubblico**, ovvero visibile a tutti
- Il **nome di un metodo** per convenzione inizia con una lettera minuscola, mentre le parole seguenti iniziano con lettera maiuscola (es. `dimmiTuttoQuelloCheSai()`)
 - Convenzione detta **CamelCase**

tipo1 nomeParam1,
tipo2 nomeParam2,
...
tipon nomeParamn

OK anche
nessun
parametro



mioMetodo

valore restituito
di un certo tipo

OK anche
nessun valore
restituito

Definizione di un metodo

- La definizione di un metodo avviene come segue:

```
public tipo_di_dati nomeDelMetodo(tipo_di_dati nomeParam1, ..., tipo_di_dati nomeParamn)
```

```
{  
    istruzione 1;  
    :  
    :  
    istruzione m;  
}
```

Tipicamente ad accesso pubblico

Tipo del valore restituito (void se non viene restituito nulla)

Nome del metodo

Corpo del metodo
(racchiuso da parentesi graffe)

Lista (eventualmente vuota) dei
nomi di parametri con relativo
tipo

- Esempi `public int getValue() { return value; }`

```
public void reset(int newValue) { ..... }
```

Metodi e valori restituiti



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Un metodo può **restituire** un valore al **chiamante**

```
public int getValue() { return value; }
```

- La parola chiave **void** nell'**intestazione** ("**signature**" o "**header**") del metodo indica che il metodo non restituisce alcun valore:

```
public void reset(int newValue) { ..... }
```

Costruttori

- **Metodi** (funzioni) per la **creazione** degli oggetti di una classe
- Possiedono **sempre** lo **stesso nome** della classe
- **Inizializzano** i **campi** dell'oggetto
- Possono prendere **zero, uno o più parametri**
- **NON** hanno **valori di uscita** (**MA** non specificano **void**)
- Una classe può avere anche **più costruttori** che differiscono nel numero e nei tipi dei parametri



Costruttori: l'esempio del contatore

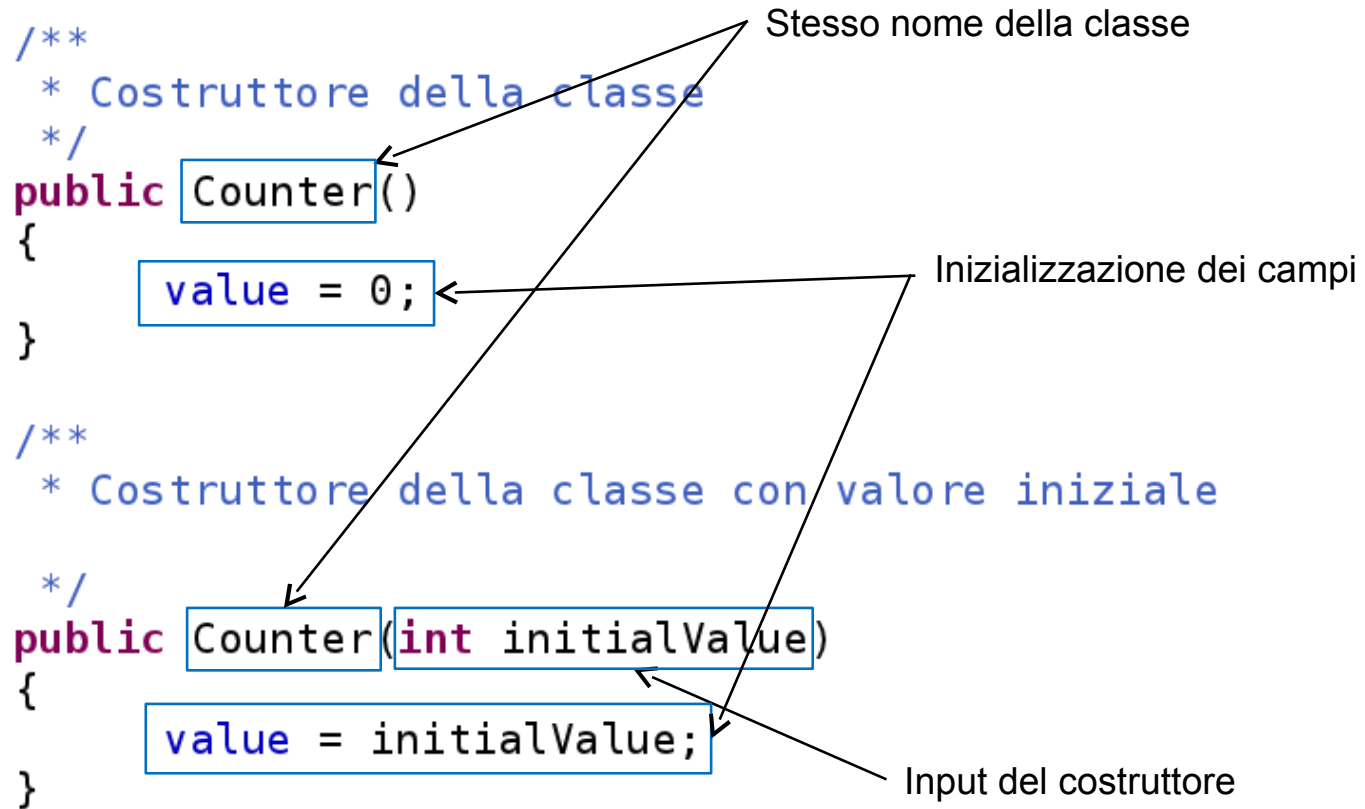
```
/**
 * Costruttore della classe
 */
public Counter()
{
    value = 0;
}

/**
 * Costruttore della classe con valore iniziale
 */
public Counter(int initialValue)
{
    value = initialValue;
}
```

Stesso nome della classe

Inizializzazione dei campi

Input del costruttore



Costruttori: creazione dell'oggetto

- Un oggetto viene creato con l'operatore **new**:

```
static public void main(String[] args)
{
    Counter contatore1 = new Counter();
    Counter contatore2 = new Counter(42);

    System.out.println("Valore del contatore1: "+contatore1.getValue());
    System.out.println("Valore del contatore2: "+contatore2.getValue());
}

/**
 * Costruttore della classe con valore iniziale
 */
public Counter(int initialValue)
{
    value = initialValue;
}
```

operatore new

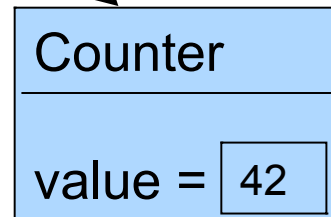
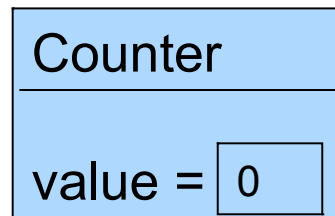
Il numero, l'ordine e i tipi dei parametri **devono corrispondere**

Costruttori: creazione dell'oggetto

- Un oggetto viene creato con l'operatore **new**:

```
static public void main(String[] args)
{
    Counter contatore1 = new Counter();
    Counter contatore2 = new Counter(42);

    System.out.println("Valore del contatore1: "+contatore1.getValue());
    System.out.println("Valore del contatore2: "+contatore2.getValue());
}
```



Ancora sui costruttori



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Non è obbligatorio specificare un costruttore
- Se non ne viene specificato uno, Java crea per ogni classe un **costruttore di default** "vuoto" (senza parametri)
- Inizializza le variabili d'istanza ai **valori di default**

Implementazione del metodo reset()

- Versione 1: semplicemente azzera il contatore

```
public void reset() { value = 0; }
```

- Versione 2: reimposta il contatore a un determinato valore

```
public void reset(int newValue) { value = newValue; }
```



Chiamate di metodi

- Vengono chiamati su un particolare oggetto

```
static public void main(String[] args)
{
    Counter contatore1 = new Counter();
    Counter contatore2 = new Counter(42);

    contatore1.count();
    contatore2.count();

    contatore2.reset();
    contatore1.reset(10);

    System.out.println("Valore del contatore1: "+contatore1.getValue());
    System.out.println("Valore del contatore2: "+contatore2.getValue());
}
```

Chiamate di metodi

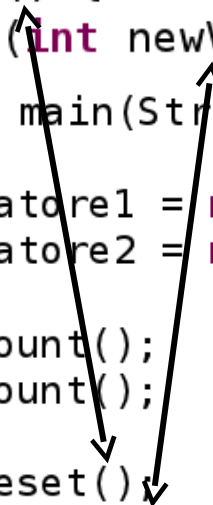
- Il numero e i tipi dei parametri (argomenti) passati a un metodo deve **coincidere** con i parametri formali del metodo

```
public void reset() { value = 0; }
public void reset(int newValue) { value = newValue; }
static public void main(String[] args)
{
    Counter contatore1 = new Counter();
    Counter contatore2 = new Counter(42);

    contatore1.count();
    contatore2.count();

    contatore2.reset();
    contatore1.reset(10);

    System.out.println("Valore del contatore1: "+contatore1.getValue());
    System.out.println("Valore del contatore2: "+contatore2.getValue());
}
```



Variabili locali vs. campi



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- I **campi** sono variabili dell'oggetto
 - Sono visibili **almeno** dall'interno dell'oggetto stesso
 - Esistono per tutta la vita di un oggetto
- Le **variabili locali** sono variabili definite all'interno di un metodo
 - Come parametri del metodo o all'interno del corpo del metodo
 - Esistono dal momento in cui sono definite fino al termine dell'esecuzione della chiamata al metodo in questione

Esercizio: la classe Rettangolo



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Progettare una classe **Rettangolo** i cui oggetti rappresentano un rettangolo e sono costruiti a partire dalle coordinate x , y e dalla lunghezza e altezza del rettangolo
- La classe implementa i seguenti metodi:
 - **trasla** che, dati in input due valori x e y , trasla le coordinate del rettangolo dei valori orizzontali e verticali corrispondenti
 - **toString** che restituisce una stringa del tipo `"(x1, y1)->(x2, y2)"` con i punti degli angoli in alto a sinistra e in basso a destra del rettangolo
- Implementare una classe di test **TestRettangolo** che verifichi il funzionamento della classe **Rettangolo** sul rettangolo in posizione $(0, 0)$ e di lunghezza 20 e altezza 10, traslandolo di $(10, 5)$ (ovvero 10 verso destra e 5 in basso)

Esercizio:

la classe ContoCorrente



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Si progetti la classe **ContoCorrente** che rappresenta il conto corrente di un cliente. Un oggetto della classe è costruito a partire dall'identificativo del cliente e, opzionalmente, dal valore iniziale del conto corrente (altrimenti pari a 0). La classe implementa i seguenti metodi:
 - **svuota**: il valore del conto viene azzerato e il quantitativo di denaro presente viene restituito in output all'utente;
 - **getImporto**: restituisce l'attuale importo del conto;
 - **getIdUtente**: restituisce l'identificativo dell'utente;
 - **preleva**: preso in input un importo da prelevare, modifica l'importo del conto in modo da considerare tale prelievo di denaro;
 - **versa**: incrementa l'importo del conto della quantità di denaro passata in ingresso.