

Java: fondamenti di Object-oriented programming

Liceo G.B. Brocchi

Classi quarte Scientifico - opzione scienze applicate

Bassano del Grappa, Settembre 2022

Prof. Giovanni Mazzocchin

Classi

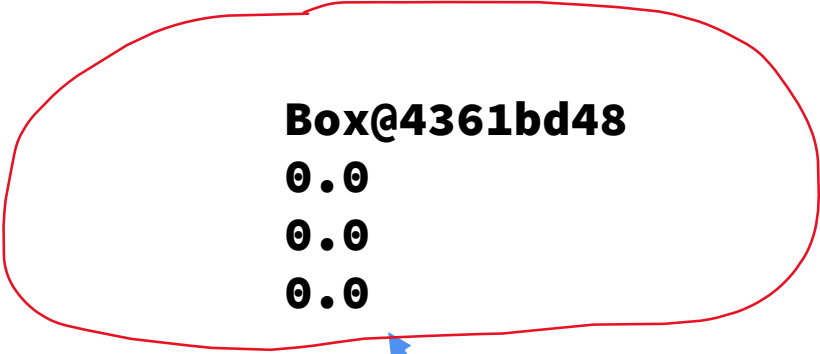
- **Domanda:** spiegare la relazione tra classi e tipi
- La definizione della classe **Box** comporta l'allocazione in memoria di un oggetto?

```
class Box {  
    double width;  
    double height;  
    double depth;  
}
```

- **Domanda:** scrivere l'istruzione che permette di creare un'istanza di **Box**

Riferimenti e oggetti

```
class Box {  
    double width;  
    double height;  
    double depth;  
}  
  
class Lecture1 {  
    public static void main (String[] args) {  
        Box b = new Box();  
        System.out.println(b);  
        System.out.println(b.width);  
        System.out.println(b.height);  
        System.out.println(b.depth);  
    }  
}
```



Box@4361bd48
0.0
0.0
0.0



Spiegare questo output

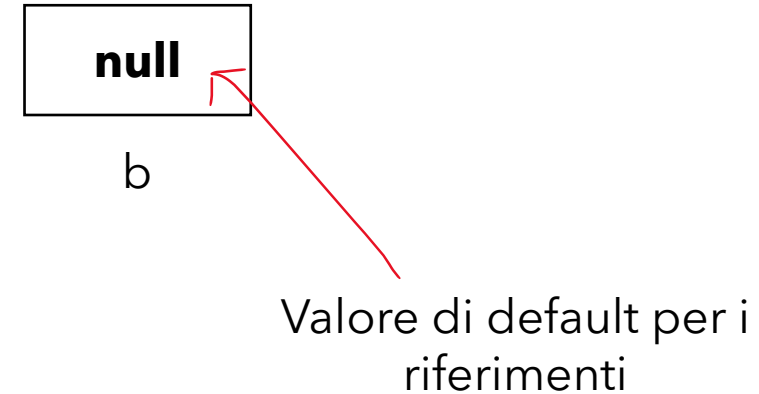
Riferimenti e oggetti

```
class Box {  
    double width;  
    double height;  
    double depth;  
}
```

```
class Lecture1 {  
    public static void main (String[] args) {  
        Box b;  
    }  
}
```

NB: **b** è un riferimento, non un oggetto
(per chi ha un po' di esperienza con il C++,
stiamo parlando sostanzialmente di un
puntatore)

Stato della memoria



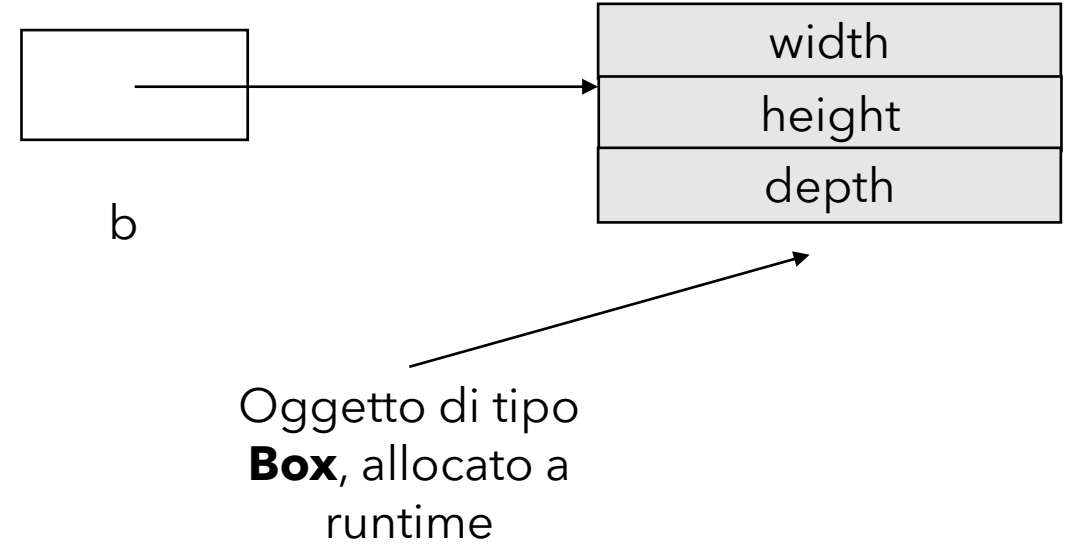
Il riferimento **b** punta a
qualcosa in memoria?

Riferimenti e oggetti

```
class Box {  
    double width;  
    double height;  
    double depth;  
}
```

```
class Lecture1 {  
    public static void main (String[] args) {  
        Box b = new Box();  
    }  
}
```

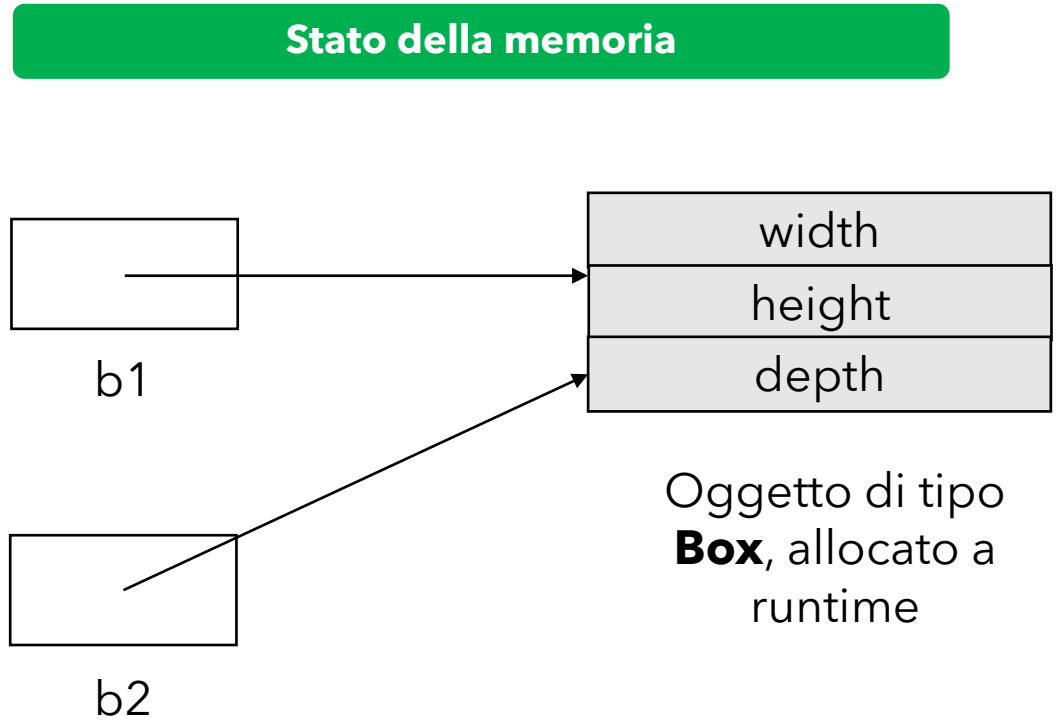
Stato della memoria



Cosa potremmo scrivere dentro la scatoletta che rappresenta `b`?

Riferimenti e oggetti

```
class Box {  
    double width;  
    double height;  
    double depth;  
}  
  
class Lecture1 {  
    public static void main (String[] args) {  
        Box b1 = new Box();  
        Box b2 = b1;  
  
        System.out.println("Object identifier - hexadecimal: " + b1);  
        System.out.println("Object identifier - hexadecimal: " + b2);  
        System.out.println("Object identifier - decimal:\t" + b1.hashCode());  
        System.out.println("Object identifier - decimal:\t" + b2.hashCode());  
    }  
}
```




Object identifier - hexadecimal: Box@2a3046da
Object identifier - hexadecimal: Box@2a3046da
Object identifier - decimal: 707806938
Object identifier - decimal: 707806938

Classi

```
class Box {  
    double width;  
    double height;  
    double depth;  
}  
  
class Lecture1 {  
    public static void main (String[] args) {  
        Box b1 = new Box();  
        Box b2 = new Box();  
  
        System.out.println("Object identifier - hexadecimal: " + b1);  
        System.out.println("Object identifier - hexadecimal: " + b2);  
        System.out.println("Object identifier - decimal:\t" + b1.hashCode());  
        System.out.println("Object identifier - decimal:\t" + b2.hashCode());  
    }  
}
```

**Descrivere lo stato della
memoria dopo la seconda
assegnazione**

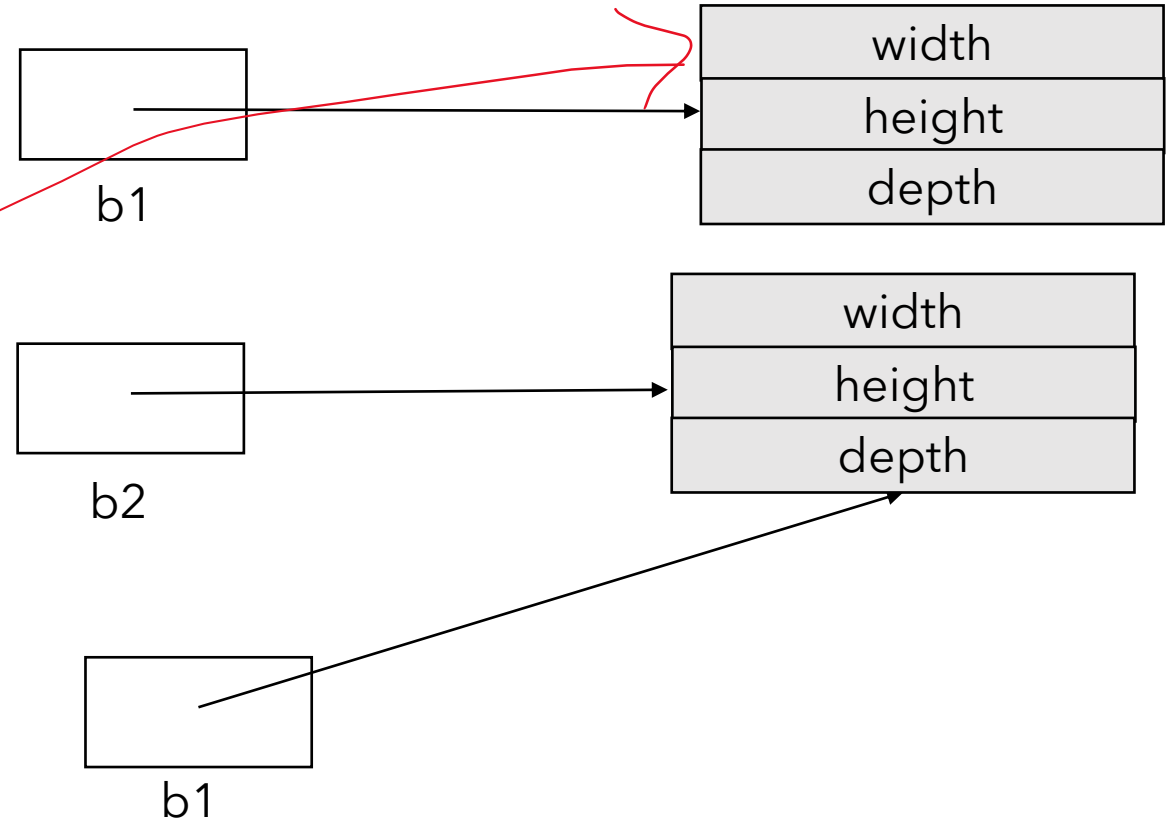


**Object identifier - hexadecimal: Box@2a3046da
Object identifier - hexadecimal: Box@5cbc508c
Object identifier - decimal: 707806938
Object identifier - decimal: 1555845260**

Riferimenti e oggetti

```
class Box {  
    double width;  
    double height;  
    double depth;  
}
```

```
class Lecture1 {  
    public static void main (String[] args) {  
        Box b1 = new Box();  
        Box b2 = new Box();  
        b1 = b2;  
    }  
}
```



Non ci sono più riferimenti
a questo oggetto. Che fine
farà?

Ogni tanto uno «spazzino» interno alla
JVM passa a raccogliere queste aree di
memoria «sporche» (si chiama **garbage
collector**)

Costruttori, metodi, il riferimento *this*

```
class Box {  
    double width;  
    double height;  
    double depth;  
  
    //constructor with 3 parameters  
    Box(double w, double h, double d) {  
        width = w;  
        height = h;  
        depth = d;  
    }  
  
    //method that computes box' volume  
    double volume() {  
  
    }  
}
```


Spiegare perché non ci sono parametri formali

Creare 2 oggetti Box nel main

Completare la definizione del metodo

Costruttori, metodi, il riferimento *this*

```
class Box {  
    double width;  
    double height;  
    double depth;  
  
    //constructor with 3 parameters  
    Box(double width, double height, double depth) {  
        this.width = width;  
        this.height = height;  
        this.depth = depth;  
    }  
}
```



Cos'è **this**? Perché l'abbiamo usato?

Costruttori, metodi, il riferimento *this*

Creare una classe che rappresenti un punto nel piano cartesiano in 2 dimensioni.

La classe deve fornire un metodo per calcolare la distanza tra l'oggetto di invocazione e un secondo punto.

Creare un array di punti appartenenti alla bisettrice del primo e del terzo quadrante.

Costruttori, metodi, il riferimento *this*

```
class Point {  
    double x;  
    double y;  
    Point (double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    double distance (Point p1) {  
        double xDistanceSquared = Math.pow(p1.x - this.x, 2);  
        double yDistanceSquared = Math.pow(p1.y - this.y, 2);  
  
        return Math.sqrt(xDistanceSquared + yDistanceSquared);  
    }  
}
```

Costruttori, metodi, il riferimento *this*

```
class Lecture2 {  
    public static void main (String[] args) {  
        Point points[] = new Point[5];  
        points[0] = new Point(0.0, 0.0);  
        points[1] = new Point(1.0, 1.0);  
        points[2] = new Point(2.0, 2.0);  
        points[3] = new Point(3.0, 3.0);  
        points[4] = new Point(4.0, 4.0);  
  
        double distance;  
        for (int i = 0; i < points.length - 1; i++) {  
            distance = points[i].distance(points[i + 1]);  
            System.out.println("Distance between point " + i + " and " + (i+1) + " is " + distance);  
        }  
  
        double distance_first_last = points[0].distance(points[points.length - 1]);  
        System.out.println("Distance between first and last point is " + distance_first_last);  
    }  
}
```

Costruttori, metodi, il riferimento this

```
class Lecture2 {  
    public static void main (String[] args) {  
        Point points[] = new Point[5];  
        points[0] = new Point(0.0, 0.0);  
        points[1] = new Point(1.0, 1.0);  
        points[2] = new Point(2.0, 2.0);  
        points[3] = new Point(3.0, 3.0);  
        points[4] = new Point(4.0, 4.0);  
  
        double distance;  
        for (int i = 0; i < points.length - 1; i++) {  
            distance = points[i].distance(points[i + 1]);  
            System.out.println("Distance between point " + i + " and " + (i+1) + " is " + distance);  
        }  
  
        double distance_first_last = points[0].distance(points[points.length - 1]);  
        System.out.println("Distance between first and last point is " + distance_first_last);  
    }  
}
```

Costruttori, metodi, il riferimento *this*

```
class Lecture2 {  
    public static void main (String[] args) {  
        Point points[] = new Point[5];  
        points[0] = new Point(0.0, 0.0);  
        points[1] = new Point(1.0, 1.0);  
        points[2] = new Point(2.0, 2.0);  
        points[3] = new Point(3.0, 3.0);  
        points[4] = new Point(4.0, 4.0);  
  
        for (int i = 0; i < points.length; i++) {  
            System.out.print(points[i] + " ");  
        }  
    }  
}
```



Cosa stampa?

Costruttori, metodi, il riferimento *this*

```
class Point {  
    double x;  
    double y;  
    Point (double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    void printThisIdentifier(){  
        System.out.println("Current object's identifier is: " + this);  
    }  
}  
  
class Lecture2 {  
    public static void main (String[] args) {  
        Point p = new Point(2.0, 3.5);  
        System.out.println("p's identifier is: " + p);  
        p.printThisIdentifier();  
    }  
}
```

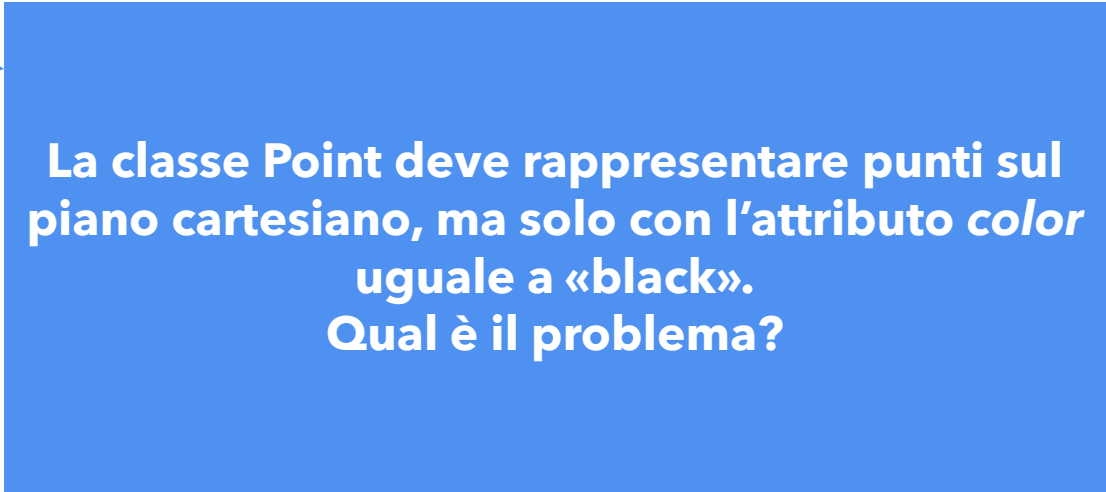


```
14 Point@2a0d3932 Point@22d71001 Point@3330013d Point@0231  
15 p's identifier is: Point@387c703b  
16 Current object's identifier is: Point@387c703b  
17
```


Access control

- I linguaggi orientati agli oggetti permettono l'**incapsulamento** attraverso meccanismi di *access control*
- L'accesso ai membri (attributi e metodi) di una classe può essere controllato grazie ai *modificatori d'accesso*

```
class Point {  
    double x;  
    double y;  
    String color;  
  
    Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
        this.color = "black";  
    }  
}
```



La classe Point deve rappresentare punti sul piano cartesiano, ma solo con l'attributo *color* uguale a «black». Qual è il problema?

Access control

```
class PointFixedBlack {  
    double x;  
    double y;  
    String color;  
  
    Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
        this.color = "black";  
    }  
}  
  
class Lecture1 {  
    public static void main (String[] args) {  
        PointFixedBlack p = new Point(5.0, 6.0);  
        p.color = "green";  
    }  
}
```

Abbiamo modificato l'attributo *color* dell'oggetto *p* dall'esterno della classe *Point*. Non possiamo più affermare che la classe *Point* rappresenta solo punti «black».

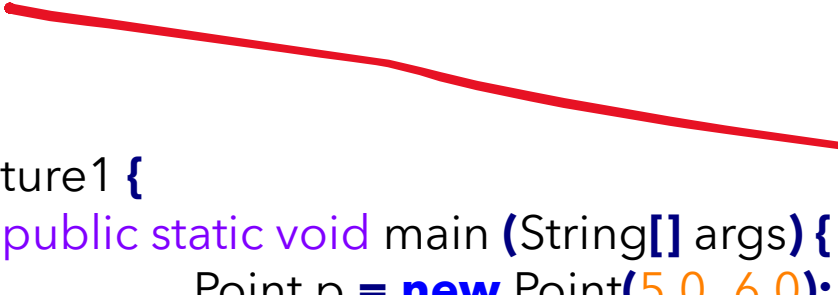
Access control

```
class PointFixedBlack {  
    double x;  
    double y;  
    private String color;  
  
    Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
        this.color = "black";  
    }  
}  
  
class Lecture1 {  
    public static void main (String[] args) {  
        PointFixedBlack p = new Point(5.0, 6.0);  
    }  
}
```

Ora per *color* viene specificato il modificatore di accesso *private*.
Sarà accessibile solo dall'interno della classe *PointFixedBlack*

Access control

```
class Point {  
    private double x;  
    private double y;  
  
    Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
class Lecture1 {  
    public static void main (String[] args) {  
        Point p = new Point(5.0, 6.0);  
    }  
}
```

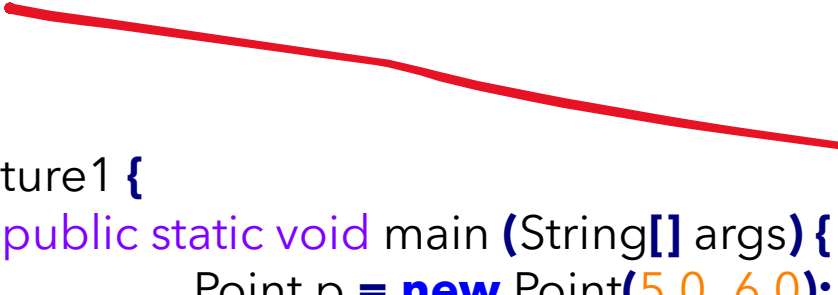


Generalmente si specifica l'accesso *private* per gli attributi e *public* per alcuni metodi. Spesso è utile accedere ad alcuni attributi in sola lettura tramite dei metodi detti *getter*.

scrivere i metodi *getter* per gli attributi x e y. Quale modificatore di accesso utilizzeresti?

Access control

```
class Point {  
    private double x;  
    private double y;  
  
    public Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
class Lecture1 {  
    public static void main (String[] args) {  
        Point p = new Point(5.0, 6.0);  
    }  
}
```



Spesso è utile scrivere il valore di alcuni attributi dall'esterno, tramite dei metodi detti *setter*

scrivere i metodi *setter* per gli attributi x e y. Quale modificatore di accesso utilizzeresti?

Access control

```
class Point {  
    private double x;  
    private double y;  
  
    public Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
class Lecture1 {  
    public static void main (String[] args) {  
        Point p = new Point(5.0, 6.0);  
    }  
}
```

Spiegare perché il metodo costruttore di *Point* e il metodo *main* di *Lecture1* sono *public*

Attributi static

```
class Point {
    private double x;
    private double y;
    public static int pointCounter;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
        pointCounter++;
    }
}

class Lecture1 {
    public static void main (String[] args) {
        System.out.println(Point.pointCounter + " points have been created so far");
        Point p_5_6 = new Point(5.0, 6.0);
        Point p_4_3 = new Point(4.0, 3.0);
        System.out.println(Point.pointCounter + " points have been created so far");
        Point p1_7_6 = new Point(7.0, 6.0);
        Point p2_1_8 = new Point(1.0, 8.0);
        Point p1_2_9 = new Point(2.0, 9.0);
        Point p2_5_6 = new Point(5.0, 6.0);
        System.out.println(Point.pointCounter + " points have been created so far");
    }
}
```

Una classe può avere al suo interno dei membri che possono essere utilizzati indipendentemente dalle istanze della classe stessa.

Per questi membri va specificata la keyword *static*

Attributi static

```
class Point {  
    private double x;  
    private double y;  
    public static int pointCounter;  
    public Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
        pointCounter++;  
    }  
}
```

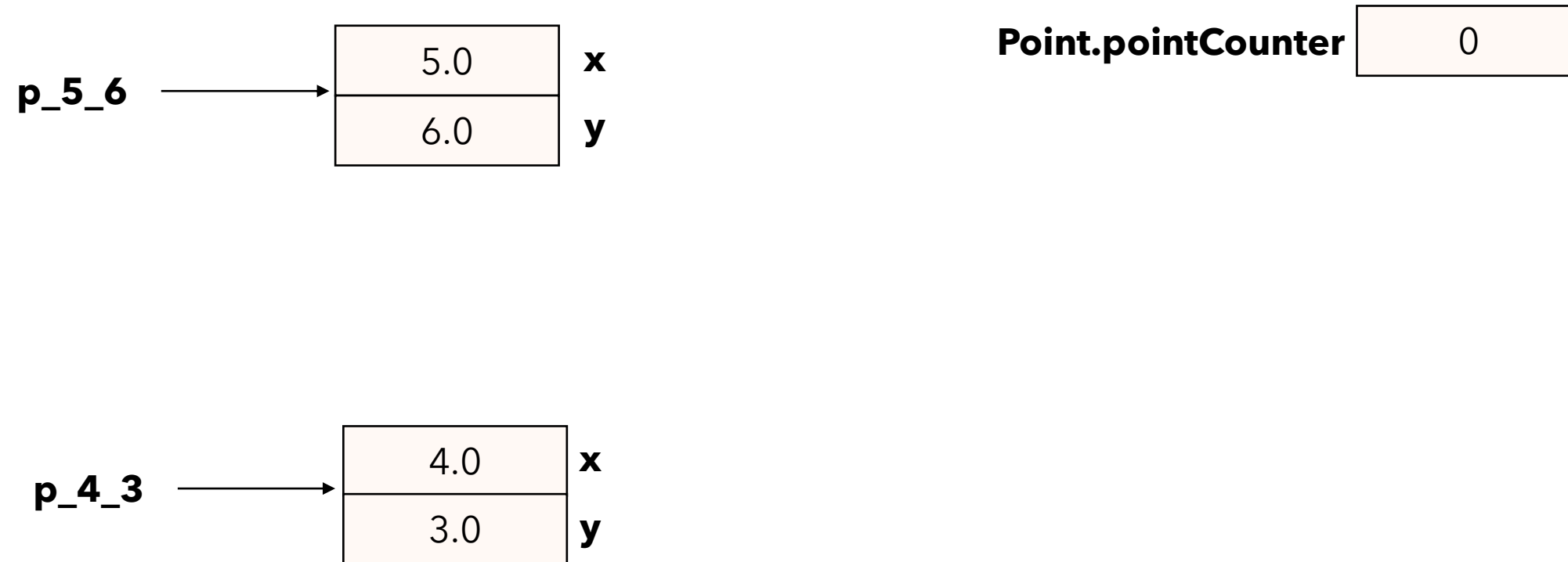
```
class Lecture1 {  
    public static void main (String[] args) {  
        System.out.println(Point.pointCounter + " points have been created so far");  
        Point p_5_6 = new Point(5.0, 6.0);  
        Point p_4_3 = new Point(4.0, 3.0);  
        System.out.println(Point.pointCounter + " points have been created so far");  
    }  
}
```

Cosa c'è di nuovo rispetto agli accessi a membro che abbiamo sempre fatto?

```
: 0 points have been created so far  
: 2 points have been created so far
```


Attributi static

- Tutte le istanze di una classe condividono lo stesso membro dichiarato **static**, che viene allocato in memoria al momento del caricamento della classe, prima della creazione degli oggetti della classe!



Attributi static

```
class Point {  
    private double x;  
    private double y;  
    public static int pointCounter;  
    public Point(double x, double y) {  
        this.x = x;  
        this.y = y;  
        pointCounter++;  
    }  
}  
class Lecture1 {  
    public static void main (String[] args) {  
        System.out.println(Point.pointCounter + " points have been created so far");  
        Point p_5_6 = new Point(5.0, 6.0);  
        Point p_4_3 = new Point(4.0, 3.0);  
        System.out.println(Point.pointCounter + " points have been created so far");  
    }  
}
```

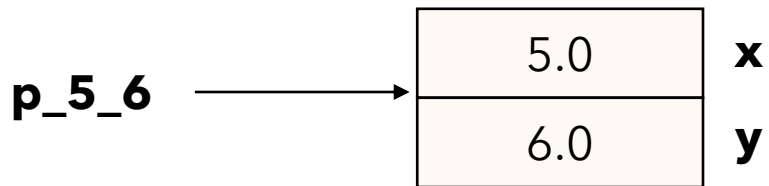
Attributi static

- Situazione della memoria prima di creare oggetti di tipo **Point**:

Point.pointCounter

0

- Situazione della memoria dopo l'istruzione `Point p_5_6 = new Point(5.0, 6.0);`



Point.pointCounter

1

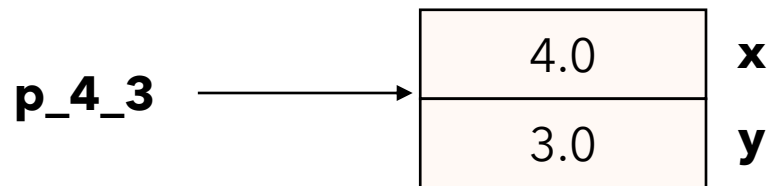
pointCounter viene incrementata nel costruttore di Point

Attributi static

- Situazione della memoria prima di creare alcun oggetto di tipo **Point**:

Point.pointCounter 0

- Situazione della memoria dopo l'istruzione `Point p_4_3 = new Point(4.0, 3.0);`



Point.pointCounter 2

pointCounter viene incrementata nel costruttore di Point

